Backend Java with Spring(boot)

Matthias Colin

Spring

Framework Java with components to develop Backend Applications

A component Spring is often a wrapper to a JEE component

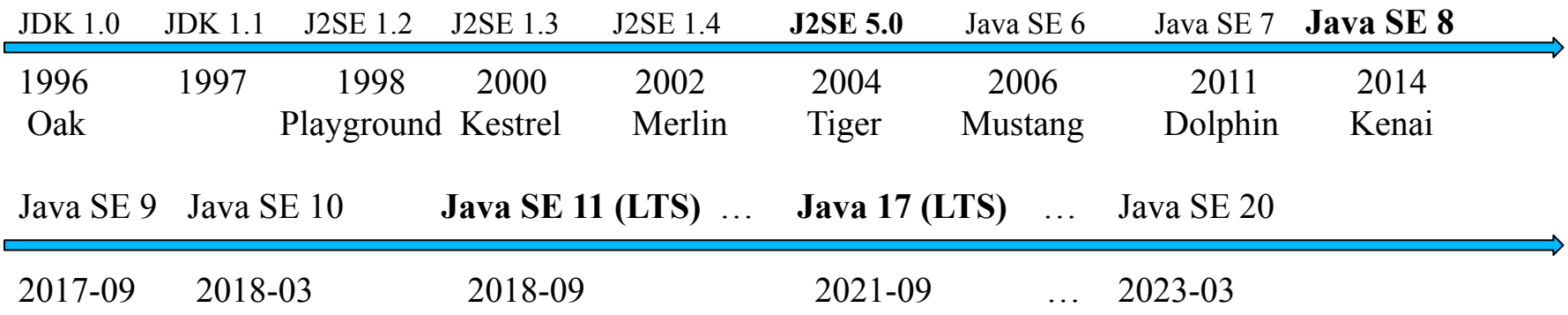
- Controller Web to deal with HTTP: Rest, MVC
- Spring Security: authentication, cryptography, CORS
- Spring Data: JPA, JDBC, MongoDB
- Reactive: asynchronous
- Tests
- Batch, Messaging
- Micro Services

Spring

Framework 2 in 1

- Spring 6: Web Application to deploy in a Java Application Server (Tomcat, Wildfly)
- Spring Boot 3: standalone application containing
  - All dependencies: JEE, Spring, others
  - Application Server: Tomcat, Jetty

Java SE

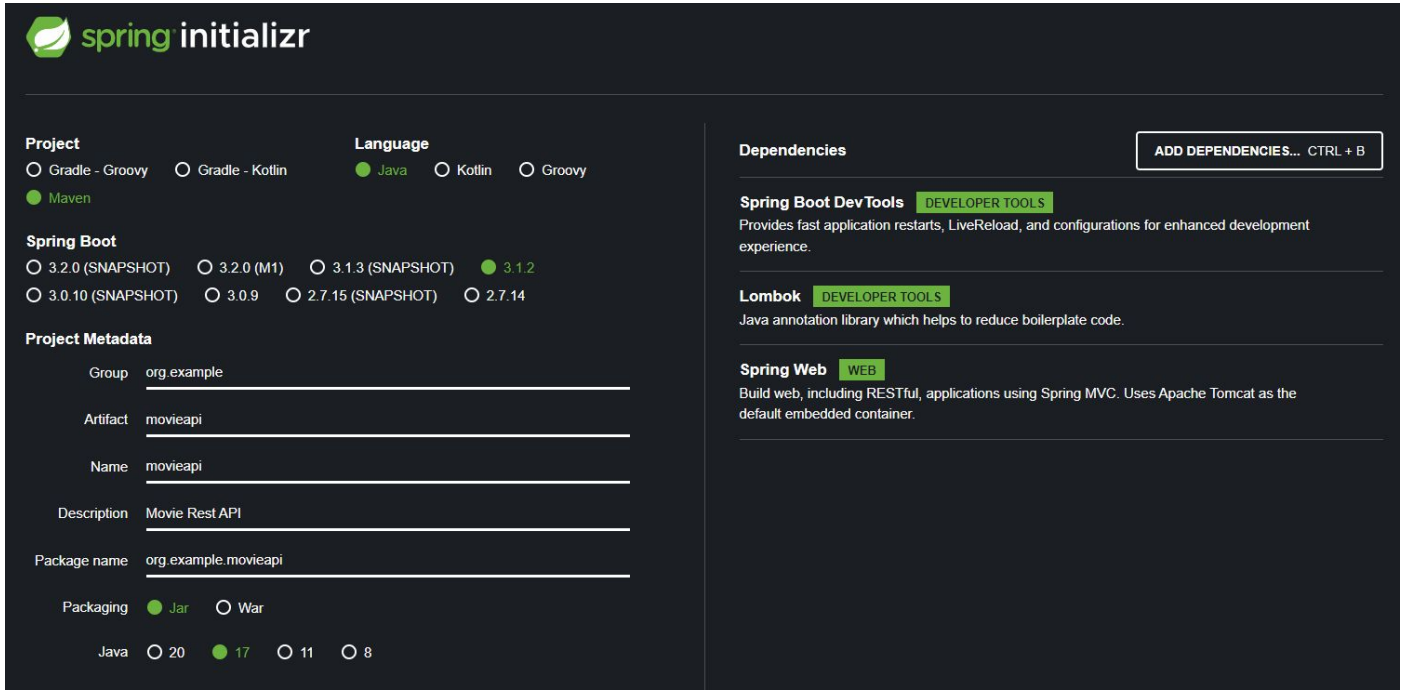

JDK 1.0
JDK 1.1
J2SE 1.2
J2SE 1.3
J2SE 1.4
J2SE 5.0
Java SE 6
Java SE 7
Java SE 8
1996
Oak
1997
1998
Playground
2000
Kestrel
2002
Merlin
2004
Tiger
2006
Mustang
2011
Dolphin
2014
Kenai
Java SE 9
Java SE 10
Java SE 11 (LTS)
…
Java 17 (LTS)
…
Java SE 20
2017-09
2018-03
2018-09
2021-09
…
2023-03

Spring and JEE

- Java EE 8 by Oracle: Spring 5, Spring Boot 2.7, Java **8-11**-17
  - javax.persistence.Entity (JPA)
  - javax.validation.constraints.NotNull (Bean Validation)
  - Tomcat 9
- Jakarta EE 9 (10) by Eclipse Foundation: Spring 6, Spring boot 3+, Java **17**
  - jakarta.persistence.Entity (JPA)
  - jakarta.validation.constraints.NotNull (Bean Validation)
  - Tomcat 10
- All Spring components are JEE Bean components
  - DI: Dependency Injection

Spring(boot)

- Main site: spring.io
- Spring Initializr: https://start.spring.io/



HTTP(S)

- Protocole de communication Client-Server
- Domaines
  - Web: HTML + CSS + images + …
  - Web Service : SOAP, WSDL (XML) by W3C
  - API Rest : HTTP + JSON (ou XML)
- Elements
  - Headers
    - Method: GET, POST, PUT/PATCH, DELETE
    - Url: http[s]://www.example.org/api/movies
    - Status:
      - 1xx: Information
      - 2xx: OK
      - 3xx: redirection
      - 4xx: mauvaise demande du client
      - 5xx: erreur serveur
    - Other Headers: Accept, User-Agent, Content-Type, Content-Length, Date
  - Body: HTML, CSS, IMG, JSON, XML, …

HTTP(2)

- Headers for Request/Response
  - Content-Type: MIME-TYPE (image/png, text/html, application/json, …, */*)
  - Accept: MIME-TYPE(s)
  - Content-Length
  - Date
  - Server

Spring Boot Rest Controller

(De)Serialization default Content-Type:

- Simple data (String, int, double, boolean, …) : text/plain
- Object: application/json

For other media types: XML, CSV, … Spring just need a converter

Routing

2 ways

- A rest controller class (@RestController + @RequestMapping) with methods:
  - Entry point: method annotated with @GetMapping, @PostMapping, …
- A routing class : FunctionalRoute
  - Entry point: route => function

Example: controller running in tomcat listening at address localhost:8080

GET http://localhost:8080/api/movies/byTitle?t=Batman

```
@RestController @RequestMapping(“/api/movies”)
class  MovieController {
        @GetMapping(“byTitle”)
        Movie getByTitle(String title)
}
```

Classic Routing

Resource: /api/movies

method	path	semantic
GET		Read All Movies
GET	123	Read Movie #123
POST		Add new Movie
PUT/PATCH	123	Update Movie #123
DELETE	123	Delete Movie #123

Rest controller: params

- query param: url?t=Batman&y=2022
  - 1st param: name=t value=Batman
  - 2nd param: name=y value=2022
  - @RequestParam with validation:
    - Required
    - Conversion String => boolean, int, double, …, LocalDate, …
- path param: /api/movies/{id}
  - Example: /api/movie/123
  - @PathVariable
- body param: complex data (Collection, Custom Object)
  - @BodyRequest

Rest Controller: Custom Data

- Java class with default constructor and getters/setters
- Java 17 immutable Records
- Converter JSON: Jackson

Rest Controller Development

1. Define a service interface: MovieService
2. Write Unit Tests for all methods of controller: MovieController
3. Correct code of the methods tested until success

Bean Validation

- Specification JEE, provider Hibernate Validator
- Constraints:
  - Builtins: @NotNull, @NotBlank, @Min, @Size, …
  - Custom constraint
- Spring Rest Controller
  - @Valid request body
  - simple constraint on request params

Controller + error

- throw ResponseStatusException with status + message
- throw custom exception
  - annotated with @ResponseStatus
- throw already defined or custom exception + controller advice
  - HTTP status + message ou problem dto
  - Example: DataAccessException => CONFLICT, BAD REQUEST, …

DI: Dependency Injection

What can be injected: @Bean (java/jakarta EE)

Spring beans:

- @Component
- @Controller, @RestController
- @Service
- @Repository

Spring Data

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

- JPA
- JDBC
- MongoDB
- others …

Java with RDBMS (2)

- JDBC
- specification java JEE: JPA (Java Persistence API) :
  - main provider Hibernate
  - ORM: Object Relational Mapper
- Spring Data: JPARepository

Java application with Relational Database

- communication appli Java <-> RDBMS
- langage commun de communication SQL
- JDBC : Java Database Connectivity (inclus Java SE)
  - Comment gérer des requêtes (insert, update, delete, select)
  - package java.sql et javax.sql
    - Driver : spécification d'un driver éditeur
    - Connection : établir une connexion avec la base de données
      - host, port, dbname, user (, password)
    - DataSource : pool de connexion(s)
    - Statement : exécuter une requête
      - select * from movies where year = 2020
    - PreparedStatement : exécuter une requête préparée
      - select * from movies where year = ?
      - paramètre #1 pourra être 2020, 2021, …
    - ResultSet : résultat d'une requête
  - Driver JDBC apporté par l'éditeur ou la communauté
    - postgresql-42.2.20.jar
- JNDI : externaliser les settings JDBC de l'appli => serveur appli

ORM in Object Oriented Languages

- Java: Java/Jakarta EE JPA + provider Hibernate ORM (or Eclipse Link)
- Python: Django ORM, SQLAlchemy
- .NET: Entity Framework
- php: symfony doctrine

ORM JPA

JPA = Java Persistence API

- J2EE, Java EE, JEE: JPA 1 and JPA 2
  - Hibernate 1 to 5
  - package javax.persistence.*
- Jakarta EE: JPA 2 and 3 (3.1)
  - Hibernate 6
  - package jakarta.persistence.*

ORM

ORM = Object Relational Mapper

- entity: class Java: Movie <-> table DB: movies
  - attribute: String title <-> column title
    - **id** <-> column id (Primary Key)
- association:
  - Movie-Person director <-> column director_id (FK)
  - Movie-Person; actors <-> table play
- crud
  - save(movie: Movie) <-> insert into movies …
  - List<Movie> res = read(...) <-> select … from movies where …

Spring JPA Repository

- CRUD
  - save, saveAll, saveAndFlush, saveAllAndFlush
  - (update)
  - delete, deleteById, …
  - findById
  - findAll
  - findAll(Sort)
  - findAll(Pageable)
  - findAll(Example)
- Add business methods
  - with method name only: SQL query automatically generated
  - with JPA JPQL query
  - with JPA entity graph
  - with JPA api criteria
  - with native SQL (vendor dependant)

JPA

- class tagged with @Entity
  - default constructor
  - getter/setter for each persistent field
  - by default all fields are persistent
  - primary key: @Id, @GeneratedValue
    - strategy: IDENTITY, SEQUENCE, AUTO, TABLE, UUID
  - tuning names, constraints
    - class: @Table
    - fields: @Column

JPA Associations

- Kind
  - One to one
  - Many to one
  - One to many
  - Many to many
- Navigability
  - Unidirectional
    - Queries: add query to cross association backward
    - Update: +
  - Bidirectional
    - Queries: +
    - Update: -

SQL vocabulary

CRUD: DML = **Data Manipulation** Language

- INSERT
- UPDATE
- DELETE
- SELECT

DDL = **Data Definition** Language: table, view, index, user, …

- CREATE
- ALTER
- DROP

Hibernate settings

- dialect: H2, MariaDB, MySQL, PostgreSQL, …

  https://docs.jboss.org/hibernate/orm/6.2/userguide/html_single/Hibernate_User_Guide.html#database-dialect

- hbm2ddl.auto: DDL (JPA: jakarta.persistence.schema-generation.database.action)
  - none: no ddl generation (production)
  - update: create new table, alter existing table
  - create: drop previous version of all tables, then create all tables
  - create-drop: idem create + drop all tables when shutting down hibernate
- show_ql: show DDL and DML SQL queries
- format_sql: pretty print SQL

JPA Queries

- Traduction to native SQL according to chosen dialect (MariaDB, PostgreSQL, …)
  - Advantage: Java code independent from DB vendor
  - Techniques
    - JPQL: pseudo SQL with entities (not tables)
    - API Criteria: Java Code with methods .from(), .where(), .join()
- Native SQL
  - Drawback: Java code dependent from DB vendor

Spring AOP

AOP = Aspect Oriented Programming

adding additional behavior to existing code without modifying the code itself.

- Include AspectJ from Eclipse

  https://eclipse.dev/aspectj/doc/released/progguide/index.html

  https://www.digitalocean.com/community/tutorials/spring-aop-example-tutorial-aspect-advice-pointcut-joinpoint-annotations

  https://howtodoinjava.com/spring-aop/aspectj-pointcut-expressions/

- Spring boot starter

AOP Pointcuts and Advices

- Pointcuts:
  - execution
  - within
  - args
  - @annotable
  - custom
- Advices
  - @Before
  - @After
  - @AfterReturning
  - @AfterThrow
  - @Around

Spring Security

- Authentication
- Client-Server
  - CORS
  - CSRF
  - …

Reactive

Asynchronous in Java:

- ThreadPool, ForkJoinPool, Future

Spring

- Reactor project: Publisher, Mono, Flux
- WebFlux: rest controller
- Reactive Repository:
  - NoSQL: MongoDB, Redis, Cassandra, …
  - SQL: R2DBC repo

Hibernate: Hibernate Reactive ORM

Article comparatif:
https://medium.com/geekculture/spring-data-jpa-spring-data-r2dbc-hibernate-reactive-bcc43e321566

Examples

- WebFlux + R2DBC Reactive Repository
- WebFlux + MongoDB Reactive Repository

Association handling example:

https://medium.com/pictet-technologies-blog/reactive-programming-with-spring-data-r2dbc-ee9f1c24848b

https://gokhana.dev/spring-r2dbc-for-reactive-relational-databases-in-reactive-programming/

Micro Services

Examples

Registry

https://spring.io/guides/gs/service-registration-and-discovery/

API Gateway

https://spring.io/guides/gs/gateway/

Run / Deploy

- Target bootRun from maven or gradle
- java -jar movieapi.jar (fat jar or classpath defined)
  - Optional: application.properties to add or override properties from jar
  - Option -Dsomeproperty=value
- docker

Miscellaneous

- WebSocket
  https://spring.io/guides/gs/messaging-stomp-websocket/
- Batch
  https://spring.io/projects/spring-batch
- Messaging (JMS)
  https://spring.io/guides/gs/messaging-jms/
- GraphQL
  https://spring.io/projects/spring-graphql
- Vault
  https://spring.io/projects/spring-vault