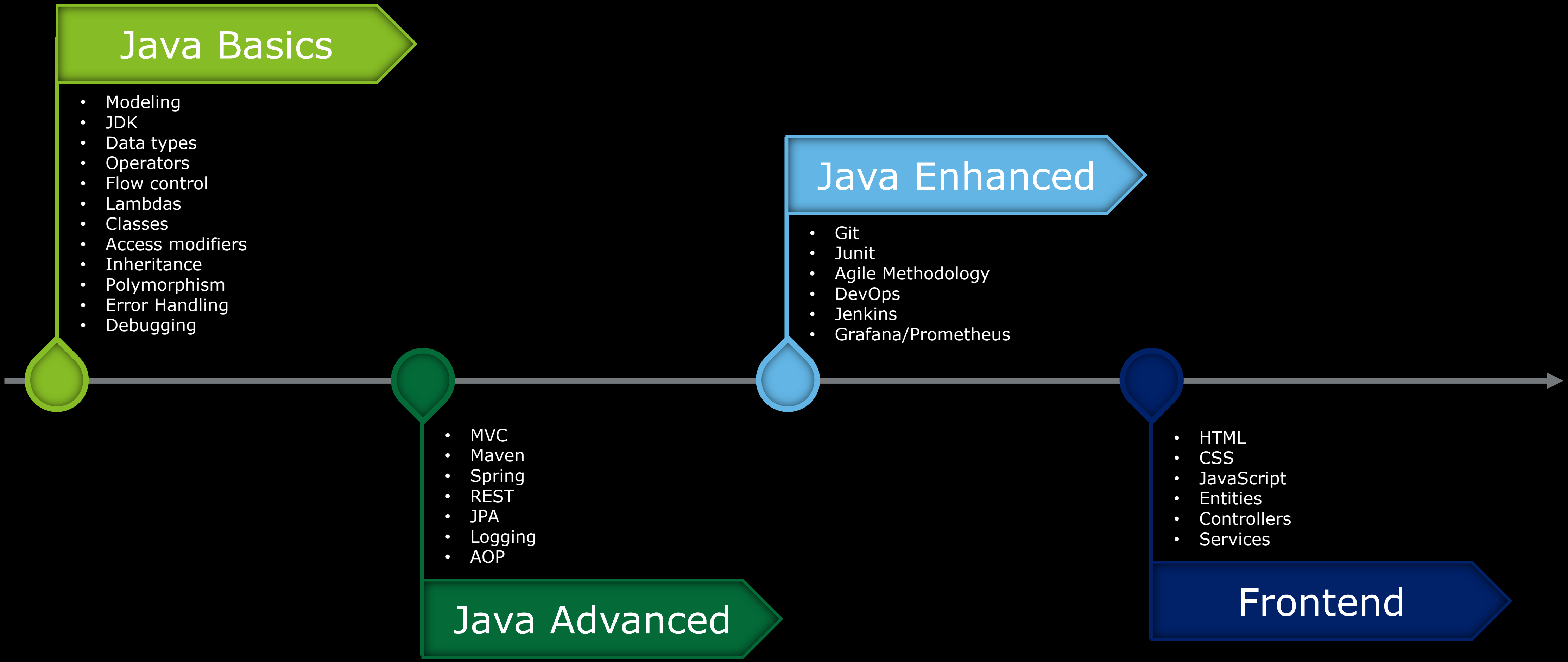
Deloitte.

Java Essentials

Java Academy

Deloitte, September 2019

Agenda
Detailed



Java Basics

- Modeling
- JDK
- Data types
- Operators
- Flow control
- Lambdas
- Classes
- Access modifiers
- Inheritance
- Polymorphism
- Error Handling
- Debugging

Java Advanced

- MVC
- Maven
- Spring
- REST
- JPA
- Logging
- AOP

Java Enhanced

- Git
- Junit
- Agile Methodology
- DevOps
- Jenkins
- Grafana/Prometheus

Frontend

- HTML
- CSS
- JavaScript
- Entities
- Controllers
- Services

Day 2

Java Advanced

MVC

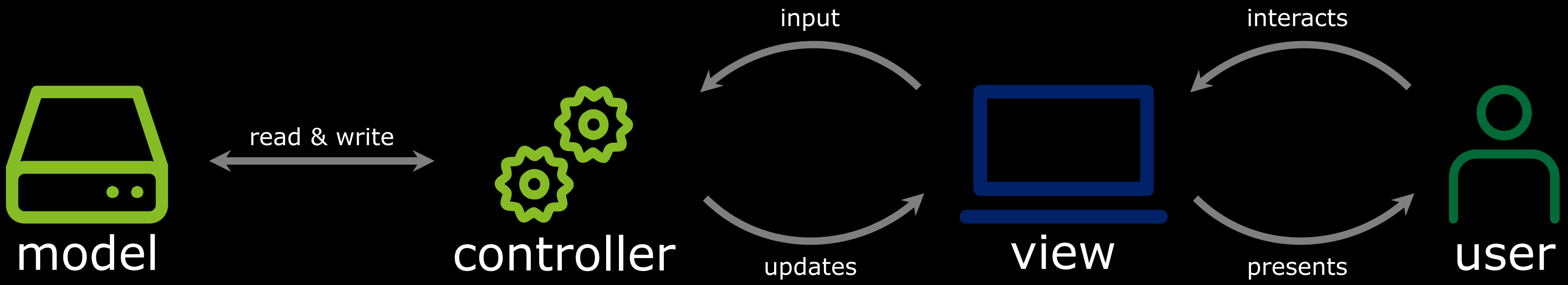
Java Advanced
MVC

- A software architectural pattern that allows to separate the code that handles business logic from the code that handles presentation and event handling

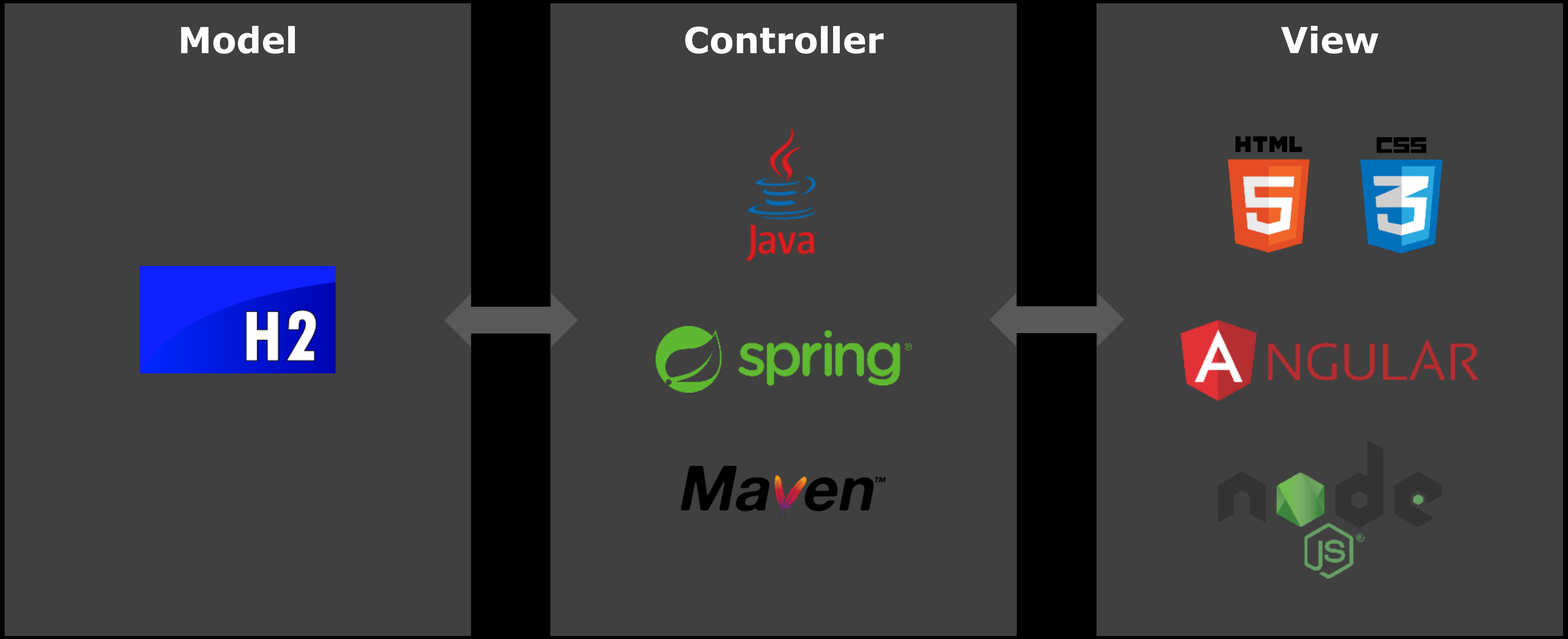
- The purpose is to have 3 interconnected components in a software application:
  - **Model:** represents the data and the rules that govern access to and updates of this data
  - **View:** renders the contents of a model, specifying how the model data should be presented
  - **Controller:** translates the user interactions with the view into actions that the model will perform

Java Advanced

MVC



Java Advanced
Architecture



Maven

Java Advanced
Maven

- It is a tool for managing the JAVA application projects

- Based on the concept of a project object model (POM)

- Makes operational tasks easier for the project life cycle:
  - Builds
  - Documentation
  - Reporting
  - Dependencies
  - Releases

Java Advanced
Maven

- Project Object Model (POM) consists of a XML file that contains information and configuration details about the project

- It defines all the necessary configuration to be used by Maven when performing tasks

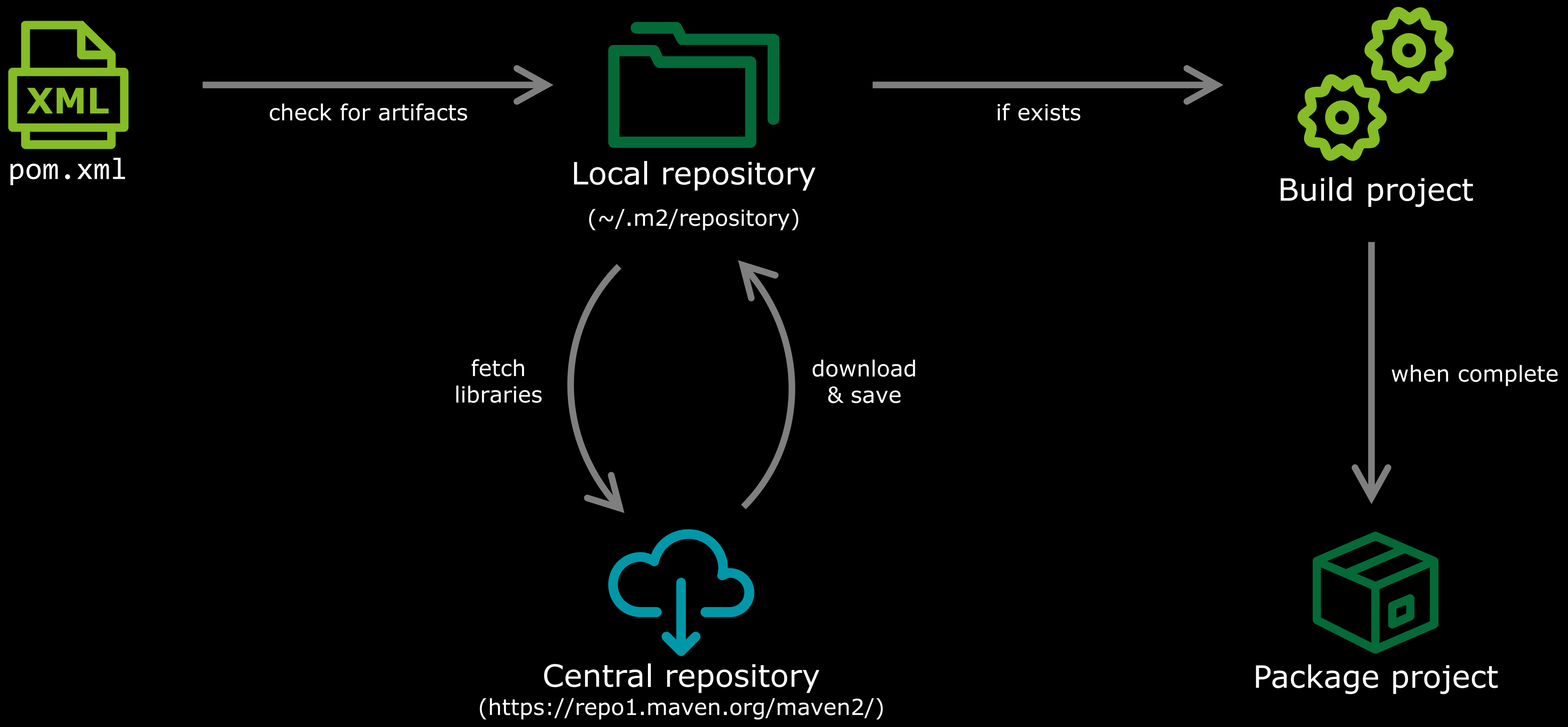
- A Maven project can be divided in modules, each with its own POM

- Provides a notion of group to define groups of modules

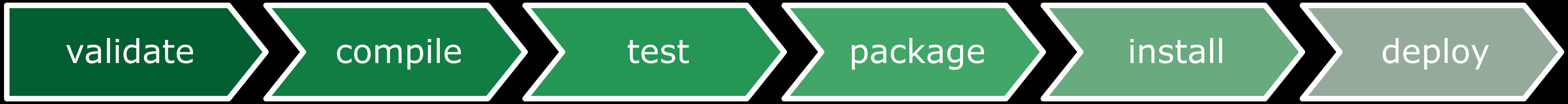
- Each POM should have a unique artifactId to identify it within a group

Java Advanced

Maven



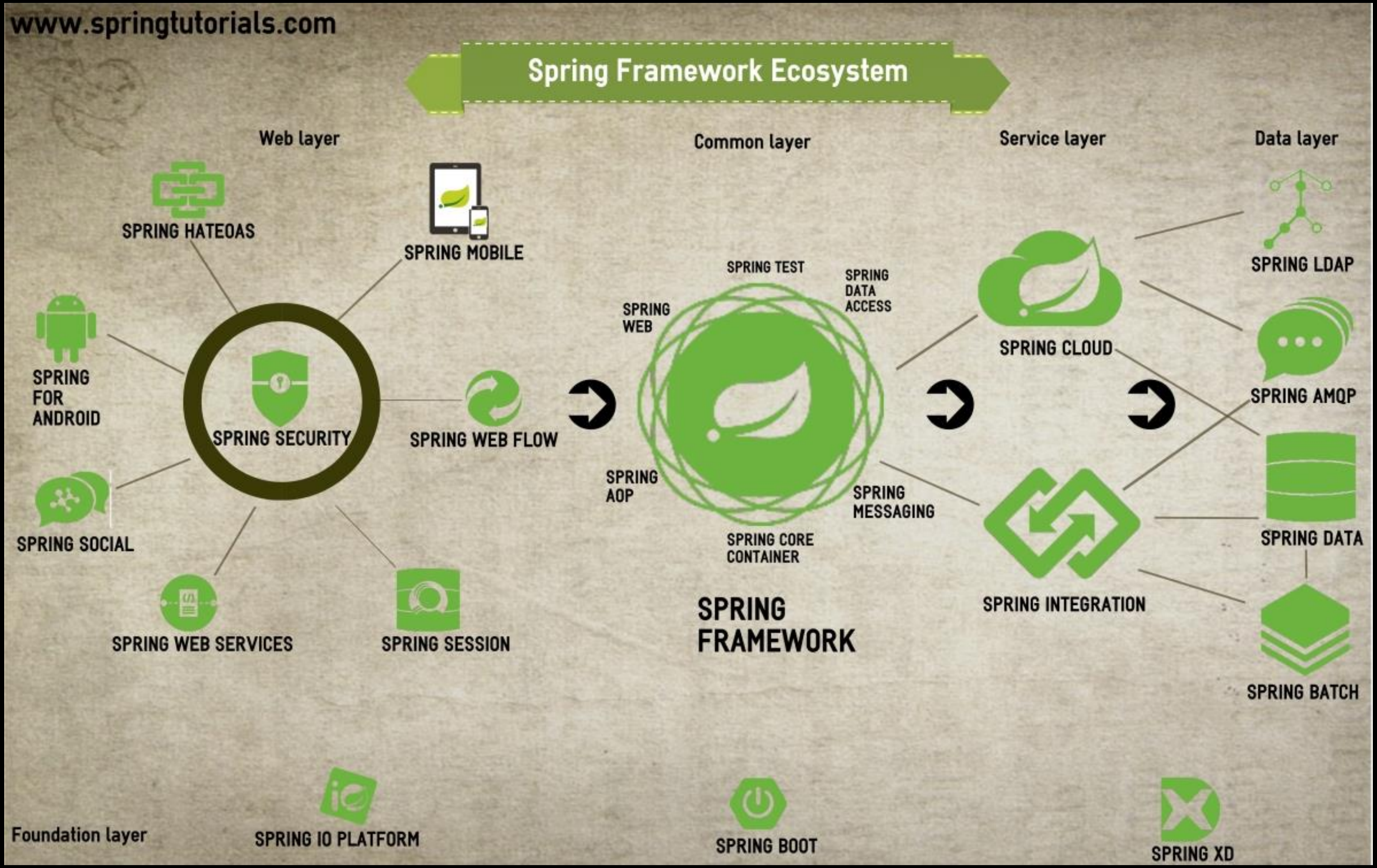
Java Advanced
Maven lifecycle



Spring

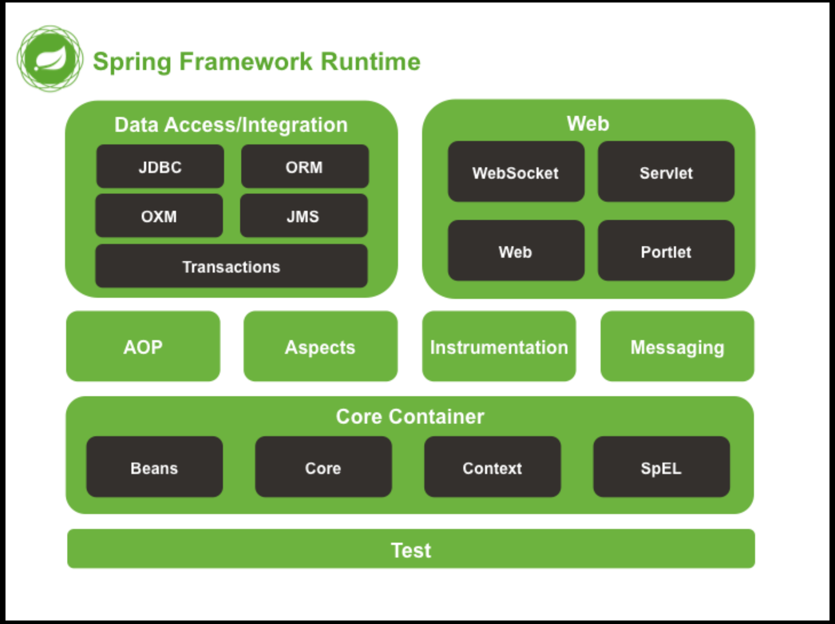
Java Advanced

Spring



Java Advanced
Spring Framework

- The Spring Framework is an application framework for the Java platform (building web applications but not only)
- Although the framework does not impose any specific programming model, it has become popular in the Java community as an addition to, or even replacement for the Enterprise JavaBeans (EJB) model.

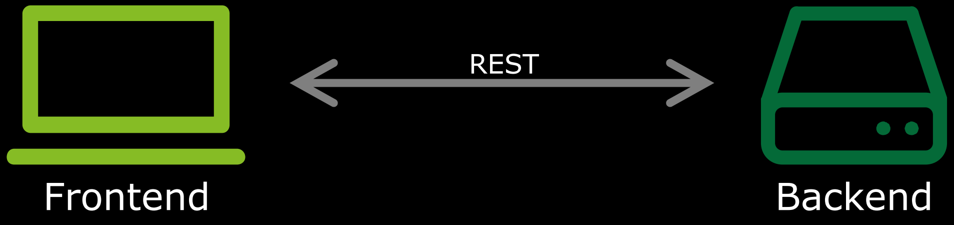


REST

Java Advanced
REST

- REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other

- REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server



```
@RestController
class Controller {

    @GetMapping("/patients")
    List<Patient> all() {
        // ...
    }

    @PostMapping("/patients")
    Patient newPatient(@RequestBody Patient newPatient) {
    }

    @GetMapping("/patients/{id}")
    Patient one(@PathVariable Long id) {

    }

    @DeleteMapping("/patients/{id}")
    void deletePatient(@PathVariable Long id) {
    }
}
```

JPA

Java Advanced
JPA

- Java Persistence API is a mechanism by which Java objects outlive the application process that created them

- The JPA specification lets you define which objects should be persisted, and how those objects should be persisted in your Java applications

```
@Entity
class Patient {

  private @Id Long id;
  private String name;

  Patient() {}

  Patient(String name, String role) {
    this.name = name;
    this.role = role;
  }
}
```

Java Advanced

JPA

```
import org.springframework.data.jpa.repository.JpaRepository;

interface PatientRepository extends JpaRepository<Patient, Long> {

}



@Controller
class PatientController() {
    private @Autowired PatientRepository repository;

    void doSomething() {
        repository.findAll();
    }
}
```

```
repository.findAll()

repository.save(newPatient)

repository.findById(id)

repository.findById(id)

repository.deleteById(id)
```

Logging

Java Advanced
Logging

- Provides the interface that applications should code to and provides the adapter components required for implementers to create a logging implementation

```
Logger LOGGER = Logger.getLogger("Health");

logger.info();
logger.debug();
```

```
ConfigurationBuilder<BuiltConfiguration> builder = ConfigurationBuilderFactory.newConfigurationBuilder();

AppenderComponentBuilder console = builder.newAppender("stdout", "Console");
builder.add(console);

Configurator.initialize(builder.build());
```

AOP

Java Advanced

AOP

- Aspect-oriented programming is a programming paradigm that aims to increase modularity by allowing the separation of cross-cutting concerns

- It does so by adding additional behavior to existing code (an advice) without modifying the code itself, instead separately specifying which code is modified via a "pointcut" specification

- @Around, @Before, @After

```
@Target(ElementType.METHOD)
@Retention(RetentionPolicy.RUNTIME)
public @interface AspectAnnotation {}


@Aspect
@Component
public class ExampleAspect {
    @Around("@annotation(AspectAnnotation)")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) {
        // do something or joinPoint.proceed()
    }
}


@AspectAnnotation
public void action() {
    // ...
}
```

Deloitte.