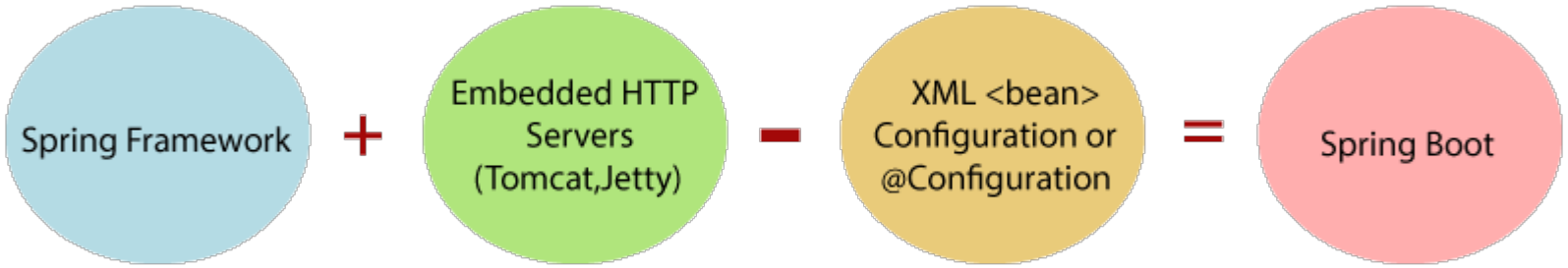
Spring Boot Overview
+
Introduction to
Auto Configuration & Spring Data JPA

By

Pichet Limvajiranan

What is Spring Boot?

- Spring Boot is a project that is built on the top of the Spring Framework.
- It provides an easier and faster way to set up, configure, and run both simple and web-based applications.
- It allows us to build a stand-alone application with minimal or zero configurations.
- It is better to use if we want to develop a simple Spring-based application or RESTful services.



Spring Boot Features

- Create stand-alone Spring applications
- Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files)
- Provide opinionated 'starter' dependencies to simplify your build configuration
- Automatically configure Spring and 3rd party libraries whenever possible
- Provide production-ready features such as metrics, health checks, and externalized configuration
- Absolutely no code generation and no requirement for XML configuration

Why should we use Spring Boot Framework?

We should use Spring Boot Framework because:

- The dependency injection approach is used in Spring Boot.
- It contains powerful database transaction management capabilities.
- It simplifies integration with other Java frameworks like JPA/Hibernate ORM, Struts, etc.
- It reduces the cost and development time of the application.

Spring Boot: Auto Configuration

- The problem with Spring and Spring MVC is the amount of configuration that is needed

```
<bean class="org.springframework.web.servlet.view.InternalResourceViewResolver">
    <property name="prefix"><value>/WEB-INF/views/</value></property>
    <property name="suffix"><value>.jsp</value> </property>
</bean>

<mvc:resources mapping="/webjars/**" location="/webjars/"/>
```

- Spring Boot solves this problem through a combination of **Auto Configuration** and **Starter Projects**.
  - Spring Boot looks at Frameworks available on the CLASSPATH then Existing configuration for the application.
  - Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks.
  - This is called Auto Configuration.

Spring Boot: Starter Projects

- Starters are a set of convenient dependency descriptors that you can include in your application.
- You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy paste loads of dependency descriptors.
- example starter - Spring Boot Starter Web.

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

Spring Boot Starter Project Options

- spring-boot-starter-web-services - SOAP Web Services
- **spring-boot-starter-web - Web & RESTful applications**
- spring-boot-starter-test - Unit testing and Integration Testing
- spring-boot-starter-jdbc - Traditional JDBC
- spring-boot-starter-hateoas - Add HATEOAS features to your services
- **spring-boot-starter-security - Authentication and Authorization using Spring Security**
- **spring-boot-starter-data-jpa - Spring Data JPA with Hibernate**
- spring-boot-starter-cache - Enabling Spring Framework's caching support
- spring-boot-starter-data-rest - Expose Simple REST Services using Spring Data REST

Auto Configuration & Property Default Value example

1) Add com.h2database dependency to classicmodels-service project & reload maven
2) Comment all properties in application.properties

```
#spring.datasource.driver-class-name=com.mysql.
#spring.datasource.username=root
#spring.datasource.password=143900
#spring.datasource.url=jdbc:mysql://localhost:
```

```
<dependency>
   <groupId>com.h2database</groupId>
   <artifactId>h2</artifactId>
   <version>2.2.224</version>
   <scope>runtime</scope>
</dependency>
```

3) Run/Test all end-point with postman
   - /api/offices POST
   - /api/offices GET
   - /api/offices/{id} GET
   - /api/offices/{id} PUT
   - /api/offices/{id} DELETE

Creating Spring Boot Projects

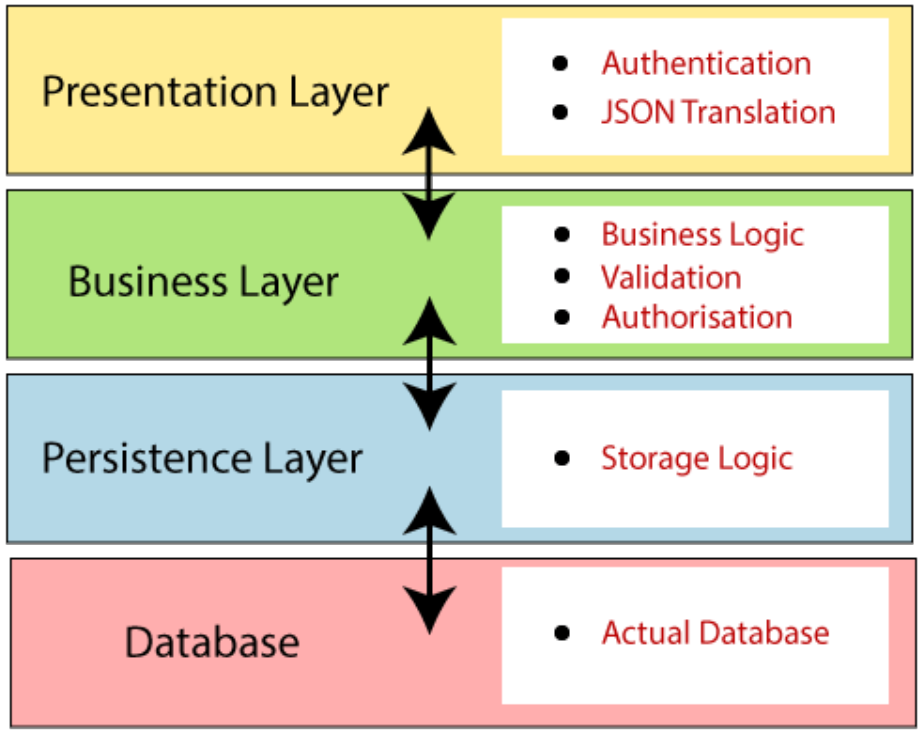
- Using Spring Initializr
  - A great web to bootstrap your Spring Boot projects.
  - https://start.spring.io
- Using the Spring Tool Suite (STS)
  - The Spring Tool Suite (STS: https://spring.io/tools/sts) is an extension of the Eclipse IDE with lots of Spring framework related plugins.
- Using IDE Bundled tool.

Maven Wrapper:

```
mvnw dependency:tree
mvnw spring-boot:run
```

Spring Boot Architecture

- Spring Boot follows a layered architecture in which each layer communicates with the layer directly below or above (hierarchical structure) it.
  - Presentation Layer
  - Business Layer
  - Persistence Layer
  - Database Layer

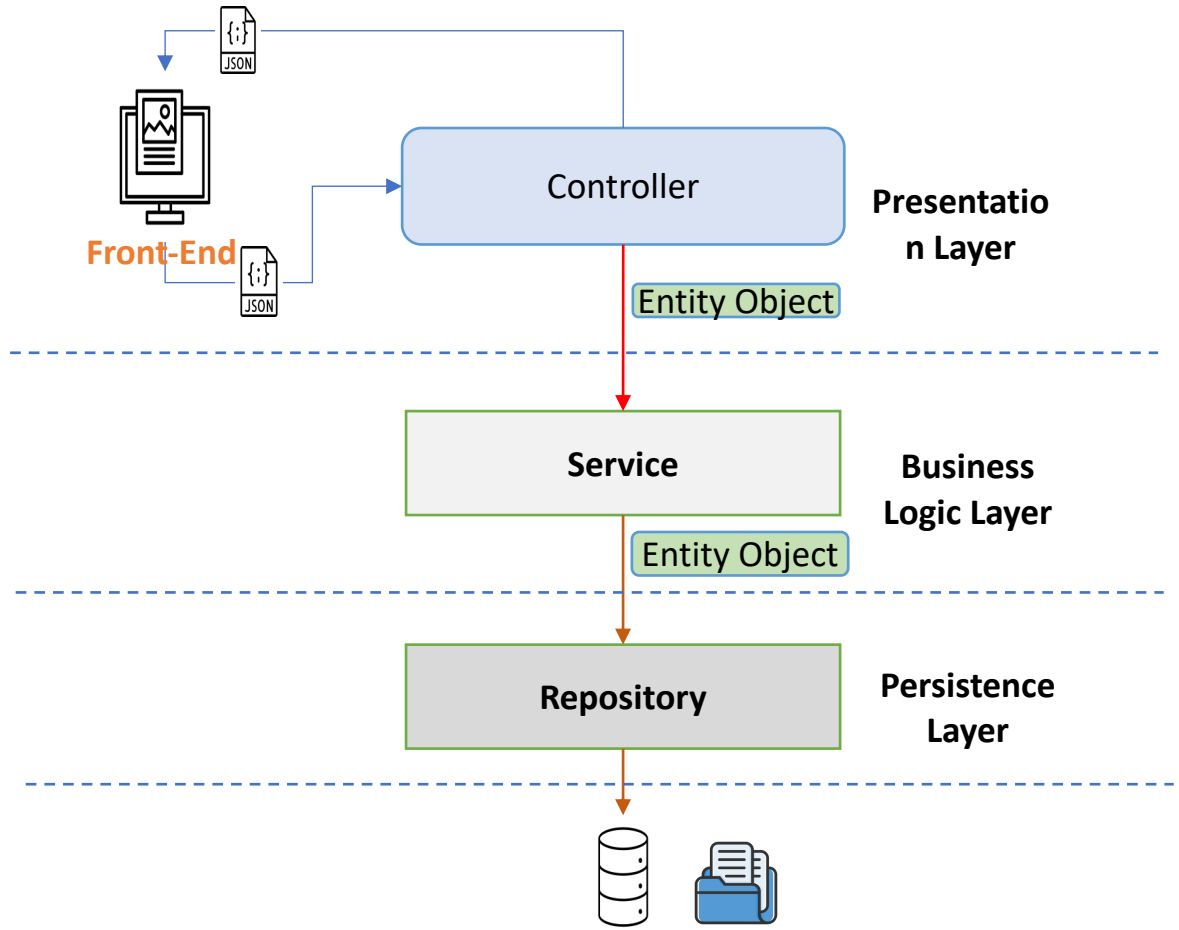


Spring Boot Layers

- Presentation Layer:
  - Handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of views i.e., frontend part.
- Business Layer:
  - Handles all the business logic. It consists of service classes and uses services provided by data access layers. It also performs authorization and validation.
- Persistence Layer:
  - Contains all the storage logic and translates business objects from and to database rows.
- Database Layer:
  - Perform CRUD (create, retrieve, update, delete) operations.

Spring Boot Flow Architecture

- Spring Boot uses all the modules of Spring-like Spring MVC, Spring Data, etc.
- Creates a data access layer and performs CRUD operation.
- The client makes the HTTP requests (GET or POST).
- The request goes to the controller, and the controller maps that request and handles it. After that, it calls the service logic if required.
- In the service layer, all the business logic performs. It performs the logic on the data that is mapped to JPA with model classes.
- A HTTP Response is returned to the user if no error occurred.



Spring Framework Annotations

- Basically, there are 6 types of annotation available in the whole spring framework.
  1. Spring Web Annotations
  2. Spring Core Annotations
  3. Spring Boot Annotations
  4. Spring Scheduling Annotations
  5. Spring Data Annotations
  6. Spring Bean Annotations

1) Spring Web Annotations

- Present in the org.springframework.web.bind.annotation
- Some of the annotations that are available in this category are:
  - **@RequestMapping**
  - **@RequestBody**
  - **@PathVariable**
  - @RequestParam
  - Response Handling Annotations
    - @ResponseBody @ExceptionHandler @ResponseStatus
  - @Controller
  - **@RestController**
  - @ModelAttribute
  - @CrossOrigin

Spring Web Annotation example

```
@RestController
@RequestMapping("/api/offices")
public class OfficeController {
    :
    :
   @GetMapping("/{officeCode}")
   public Office getOfficeById(@PathVariable String officeCode) {
      return service.getOffice(officeCode);
   }


   @PostMapping("")
   public Office addNewOffice(@RequestBody Office office) {
       return service.createNewOffice(office);
  }
```

Practice

- Create API service for calculate student grade as following requirements:
  - input

```
{ "id" : 1001,
  "name" : "Somchai",
   "score" : 78
}
```

  - output

```
{ "id" : 1001,
  "name" : "Somchai",
   "score" : 78,
   "grade" : "B"
}
```

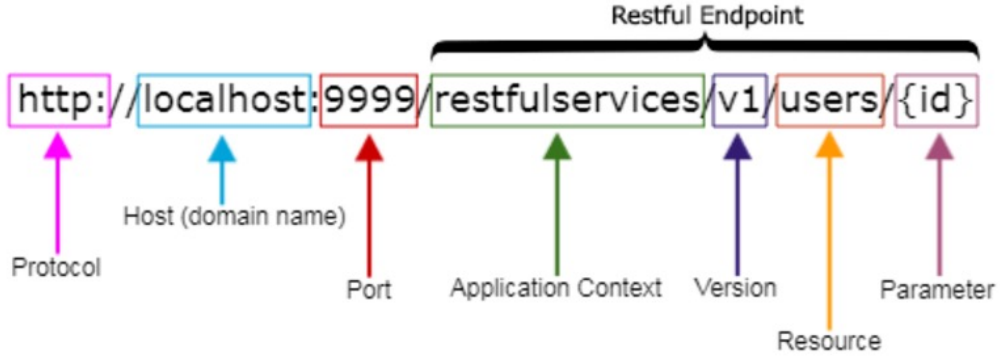
TO DO:

- Specify URI & Method
- Create Entity
- Create Service
- Create Controller & method

Business rule :

- Score is 80 and up, A
- Score 70 to 79, B
- Score 60 to 69, C
- Score 50 to 59, D
- Score is 49 and lower, F

REST API URI Naming Conventions and Best Practices



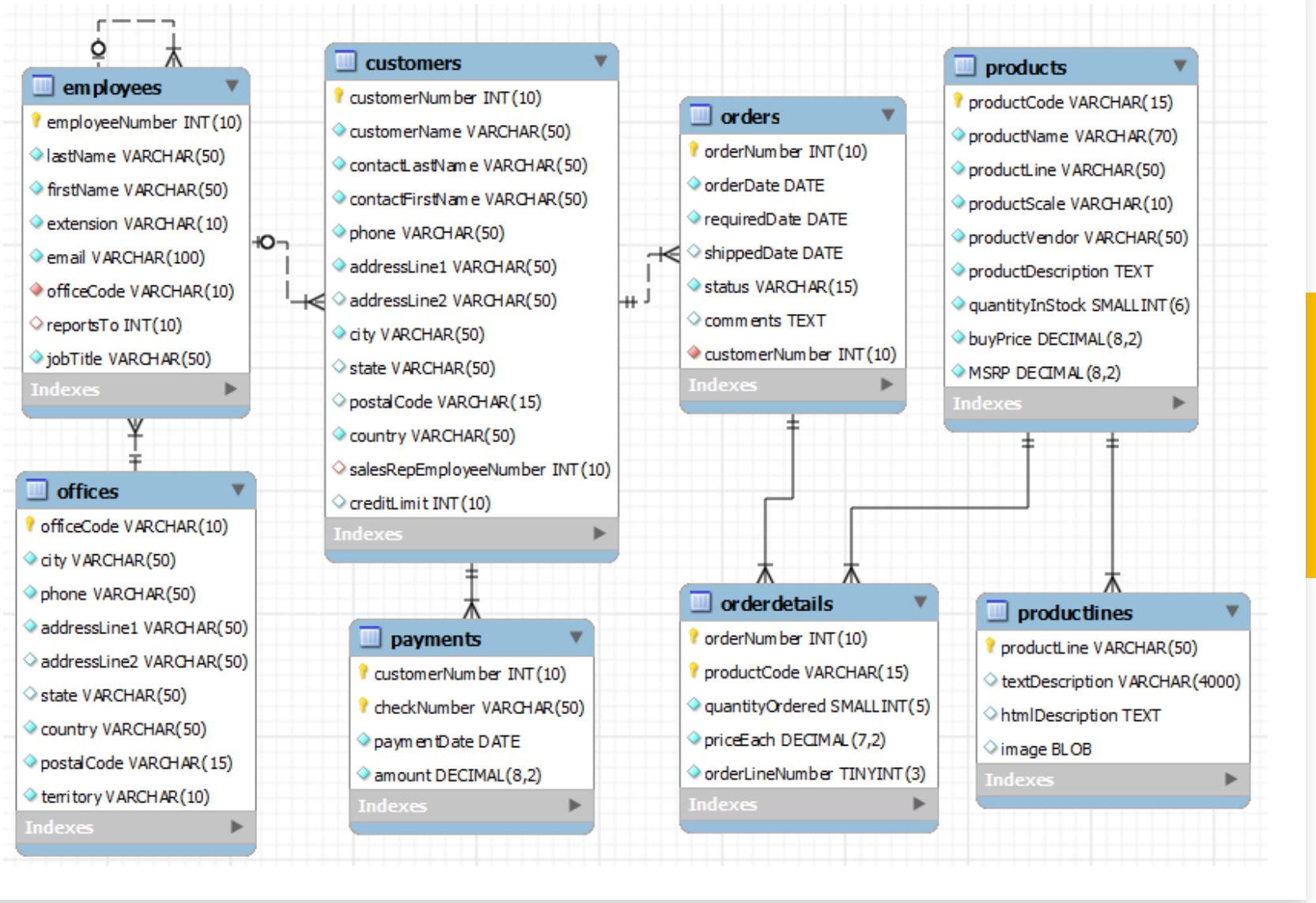
- Singleton and Collection Resources

```
/customers                  // is a collection resource
/customers/{id}             // is a singleton resource
```

- Sub-collection Resources

```
/customers/{id}/orders                  // is a sub-collection resource
```

- Best Practices
  - https://medium.com/@nadinCodeHat/rest-api-naming-conventions-and-best-practices-1c4e781eb6a5
  - https://restfulapi.net/resource-naming



Excercise

Design REST Resources representation for the Classicmodels

Spring Core annotations

- Spring Core annotations are present in the 2 packages
  - org.springframework.beans.factory.annotation
  - org.springframework.context.annotation
- We can divide them into two broad categories:
  - DI-Related Annotations
    - **@Autowired**
    - @Bean
    - @Value
    - @Lookup, etc.
    - @Qualifier
    - @Lazy
    - @Scope
    - @Primary
    - @Required
  - Context Configuration Annotations
    - @Profile
    - @ImportResource
    - @Import
    - @PropertySource, etc.

@Autowired

- We use @Autowired to mark the dependency that will be injected by the Spring container.
- Applied to the fields, **setter** methods, and **constructors**. It injects object dependency implicitly.

```
public class OfficeService {
    @Autowired
    private OfficeRepository repository;
```

```
public class OfficeService {
    private final OfficeRepository repository;
    @Autowired
    public OfficeService(OfficeRepository repository) {
        this.repository = repository;
    }
```

```
public class OfficeService {
    private final OfficeRepository repository;
    @Autowired
    public void setRepository(OfficeRepository repository) {
        this.repository = repository;
    }
```

Spring Boot Annotations

- @SpringBootApplication Combination of three annotations
  - @EnableAutoConfiguration
  - @ComponentScan
  - @Configuration.

```
@SpringBootApplication

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

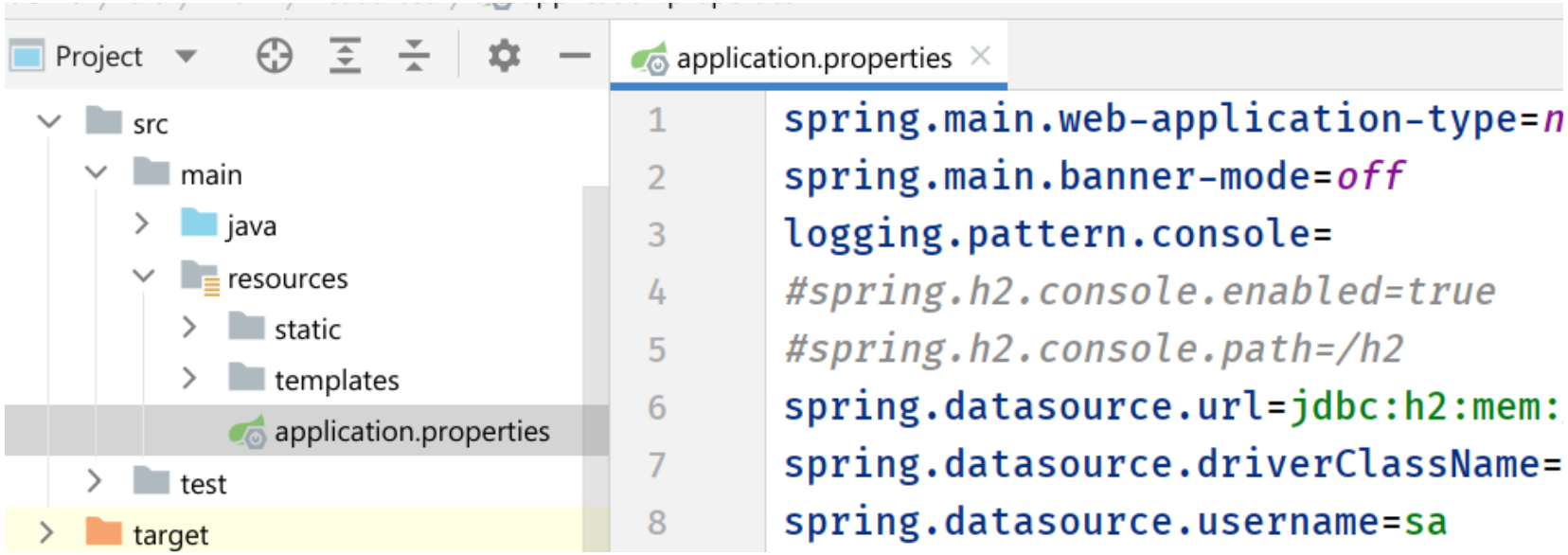
Spring Bean Annotations

- Some of the annotations that are available in this category are:
  - @ComponentScan
  - @Configuration
  - Stereotype Annotations
    - @Component
      - **@Service**
      - **@Repository**
      - **@Controller**

@Service: We specify a class with @Service to indicate that they're holding the business logic. Besides being used in the service layer, there isn't any other special use for this annotation. The utility classes can be marked as Service classes.
@Repository: We specify a class with @Repository to indicate that they're dealing with **CRUD operations**, usually, it's used with DAO (Data Access Object) or Repository implementations that deal with database tables.
@Controller: We specify a class with @Controller to indicate that they're front controllers and responsible to handle user requests and return the appropriate response. It is mostly used with REST Web Services.

Spring Boot Application Properties

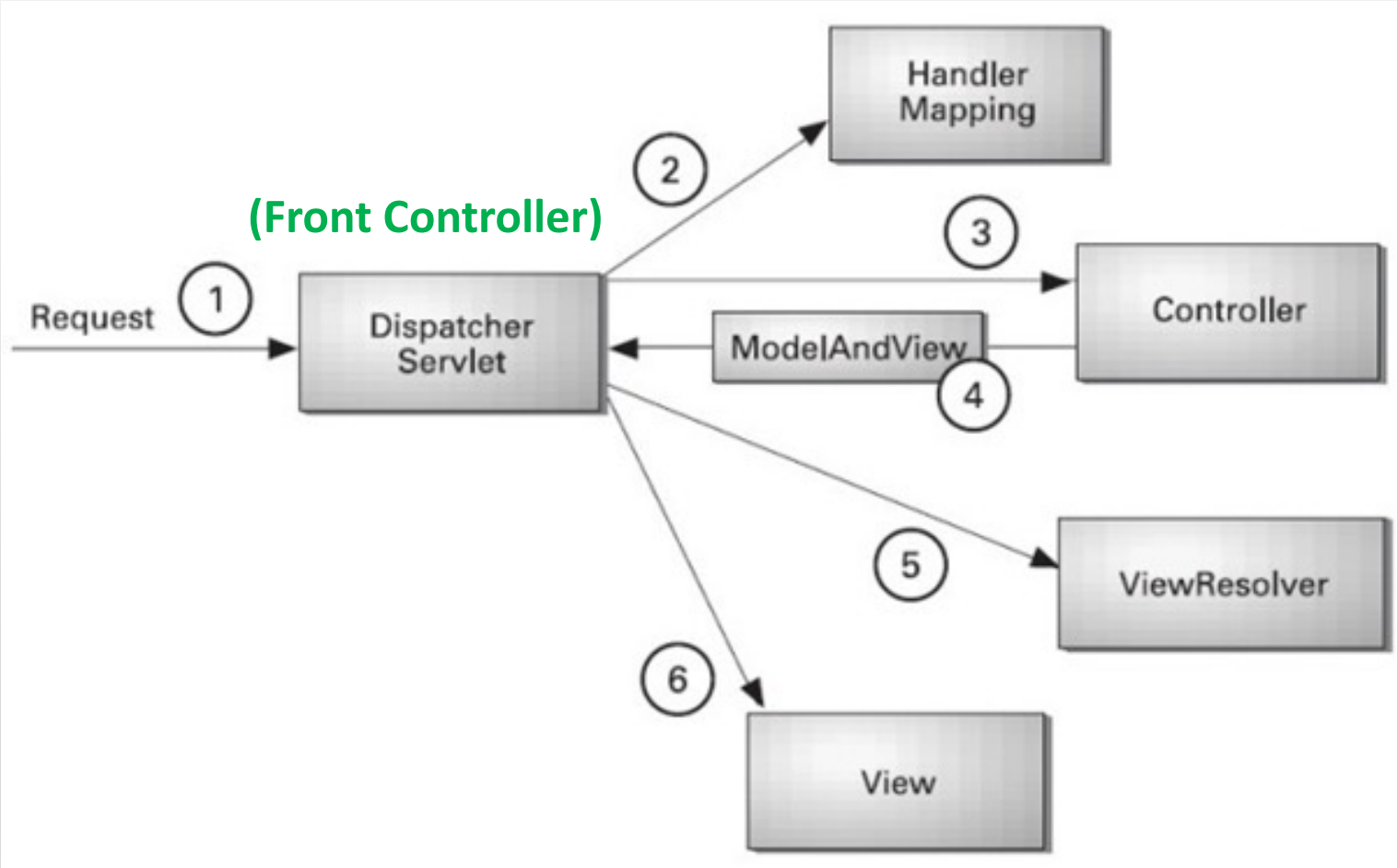
- Spring Boot Framework comes with a built-in mechanism for application configuration using a file called application.properties.
- It is located inside the src/main/resources folder.
- The properties have default values.
- We can set a property(s) for the Spring Boot application.



Spring Web MVC

- The Spring Web MVC framework provides Model-View-Controller (MVC) architecture and ready components that can be used to develop flexible and loosely coupled web applications.
- The MVC pattern results in separating the different aspects of the application (input logic, business logic, and UI logic), while providing a loose coupling between these elements.
- A Spring MVC provides an elegant solution to use MVC in spring framework by the help of DispatcherServlet.
  - In Spring Web MVC, the **DispatcherServlet** class works as the **front controller**. It is responsible to manage the flow of the Spring MVC application.

The DispatcherServlet and Flow of Spring Web MVC



Defining a Controller

- The DispatcherServlet delegates the request to the controllers to execute the functionality specific to it.
- The **@Controller** annotation indicates that a particular class serves the role of a controller.
- The @**RequestMapping** @**GetMapping** @**PostMapping** annotation is used to map a URL to either an entire class or a particular handler method.

```
@Controller
public class HelloController {
   @RequestMapping("/hello")
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "hello";
   }
}
```

Spring Boot Controller example

```
@Controller
public class AppController {
    @Autowired
    private final StudentRepository studentRepository;

    @RequestMapping("/home")
    public String home() {
        return "home";
    }
    @GetMapping("/student-listing")
    public String students(Model model) {
        model.addAttribute("students", studentRepository.findAll());
        return "student-list";
    }
    @GetMapping("/student-list-plain-text")
    public ResponseEntity<String> students_list(Model model) {
        return new ResponseEntity<>(studentRepository.findAll()
                .toString(), HttpStatus.OK);
    }
```

Spring View Technology

- The Spring web framework is built around the MVC (Model-View-Controller) pattern, which makes it easier to separate concerns in an application.
- This allows for the possibility to use different view technologies, from the well established JSP technology to a variety of template engines.
  - Java Server Pages
  - Thymeleaf
  - FreeMarker
  - Groovy Markup Template Engine

Thymeleaf Template Engine example

```
<!DOCTYPE html>
<html lang="en“  xmlns=http://www.w3.org/1999/xhtml
          xmlns:th="http://www.thymeleaf.org">
<body>
<div class="container p4 m4">
    <h2>Student List:</h2><hr>
    <div class="row">
        <div class="col-2">Student Id</div>
        <div class="col-4">Name</div>
        <div class="col-2">GPAX</div>
    </div>
    <div class="row" th:each="student : ${students}">
        <div class="col-2" th:text="${student.id}"/>
        <div class="col-4" th:text="${student.name}"/>
        <div class="col-2" th:text="${student.gpax}"/>
    </div>
</div>
```

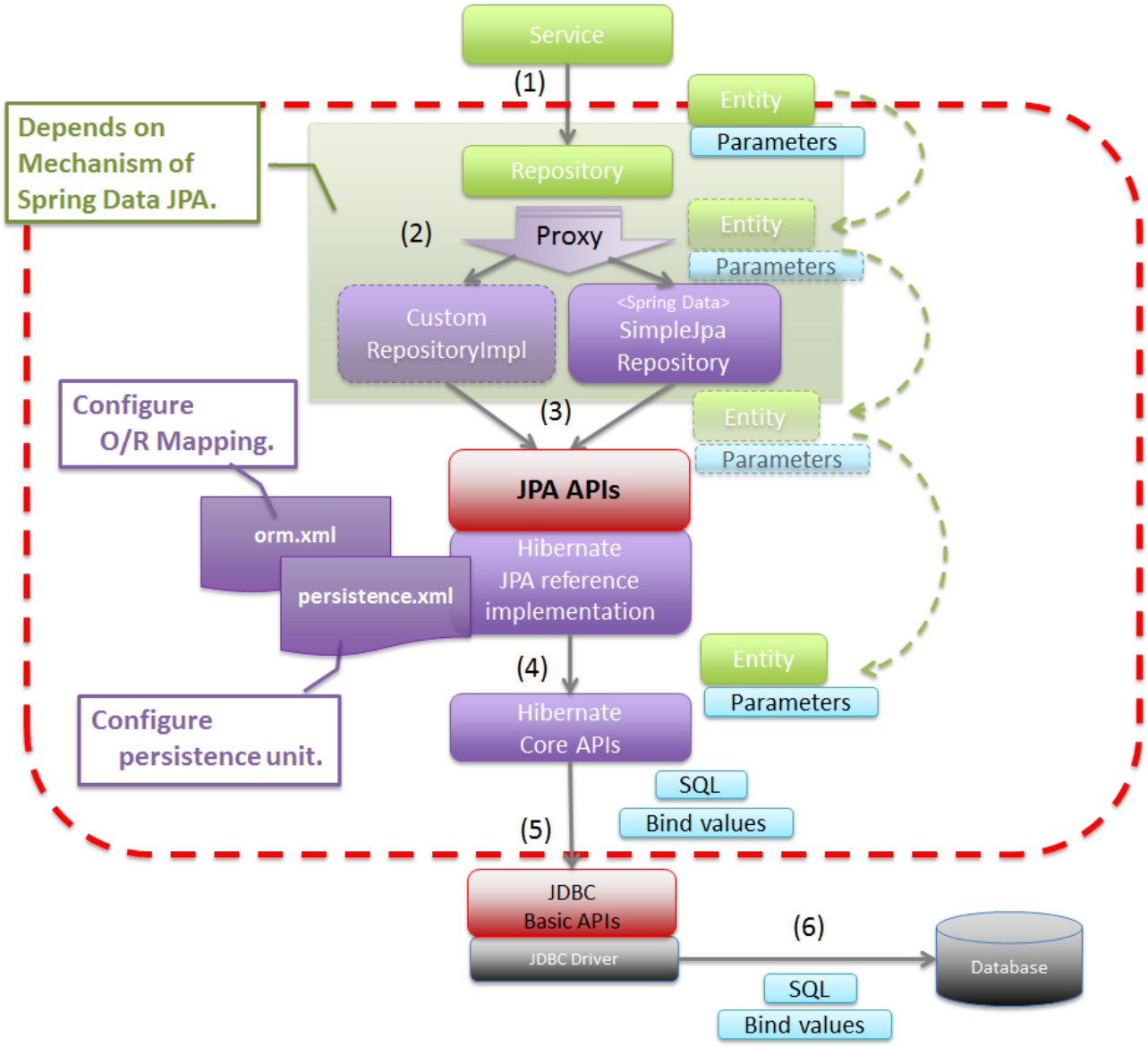
Spring Data JPA

- Managing data between java classes or objects and the relational database is a very cumbersome and tricky task.
- The DAO (Data Access Object) layer usually contains a lot of **boilerplate code** that should be simplified in order to reduce the number of lines of code and make the code reusable.
- Spring Data JPA:
  - This provides spring data repository interfaces which are implemented to create JPA repositories.
  - Spring Data JPA provides a solution to reduce a lot of boilerplate code.
  - Spring Data JPA provides an **out-of-the-box** implementation for all the required CRUD operations for the JPA entity so we don't have to write the same boilerplate code **again and again**.

```
public class CustomerRepository {
    private static final int PAGE_SIZE = 10;
    private EntityManager getEntityManager()
    public List<Customer> findAll() {..}
    public void save(Customer c) {..}
    public Product find(Integer cid) {..}
}
```

```
public class ProductRepository {
    private static final int PAGE_SIZE = 50;
    private EntityManager getEntityManager()
    public List<Product> findAll() {..}
    public void save(Product p) {..}
    public Product find(String pid) {..}
}
```

Basic Spring Data JPA Flow

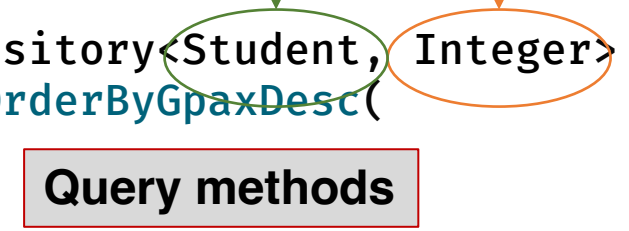


JPA Repository Example

```java
@Getter @Setter @NoArgsConstructor
@AllArgsConstructor @ToString
@Entity
public class Student {
    @Id
    private Integer id;
    private String name;
    private Double gpax;
}
```

```java
import org.springframework.data.jpa.repository.JpaRepository;
import sit.int204.demo.entities.Student;

public interface StudentRepository extends JpaRepository<Student, Integer> {
    List<Student> findByNameContainsOrGpaxBetweenOrderByGpaxDesc(
            String name, double low, double high);
}
```



Jpa Repository default methods

```
public class AppController {
    @Autowired
    private final StudentRepository
studentRepository;
```