



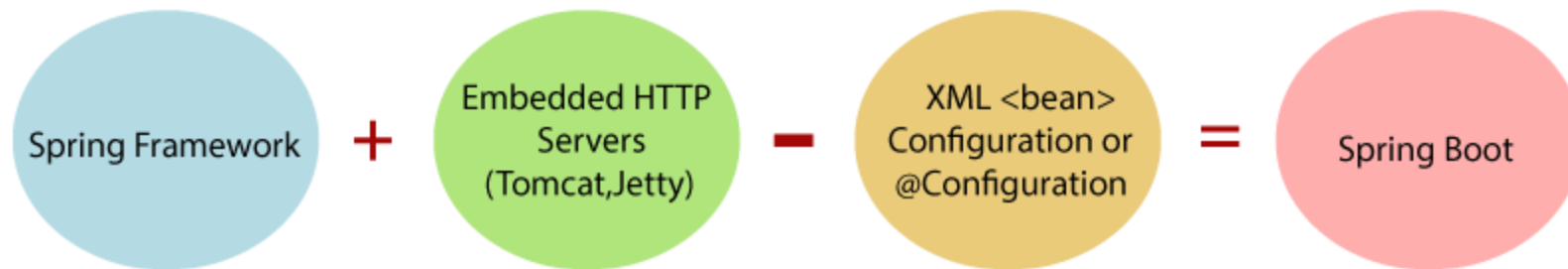
Spring Boot Overview

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

Creating Spring Boot Projects

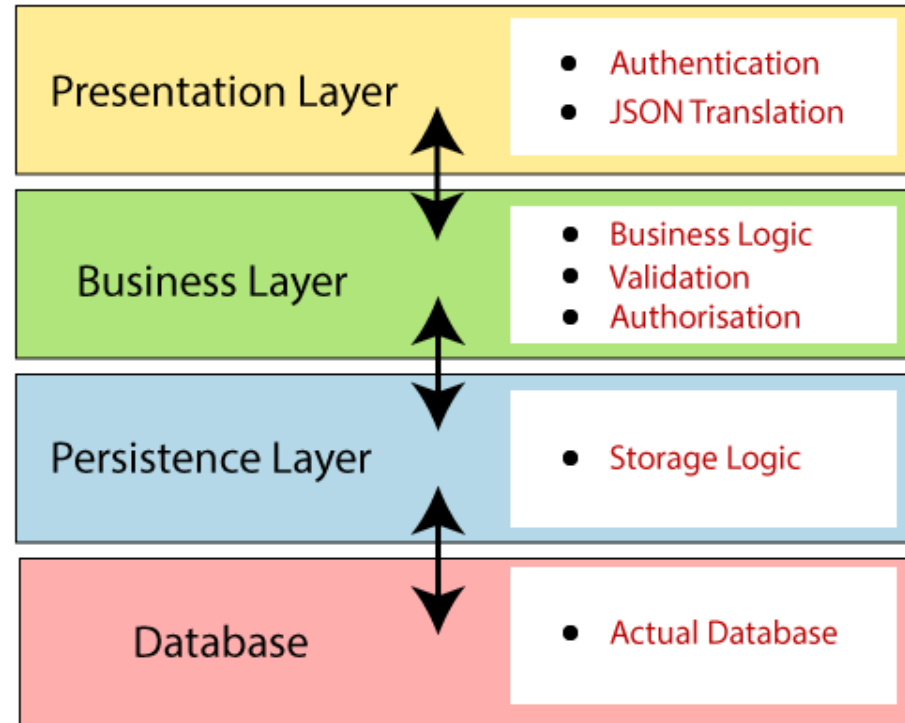
- Using Spring Initializr
 - A great web to bootstrap your Spring Boot projects.
 - <https://start.spring.io/>
- Using the Spring Tool Suite (STS)
 - The Spring Tool Suite (STS: <https://spring.io/tools/sts>) is an extension of the Eclipse IDE with lots of Spring framework related plugins.
- Using IDE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

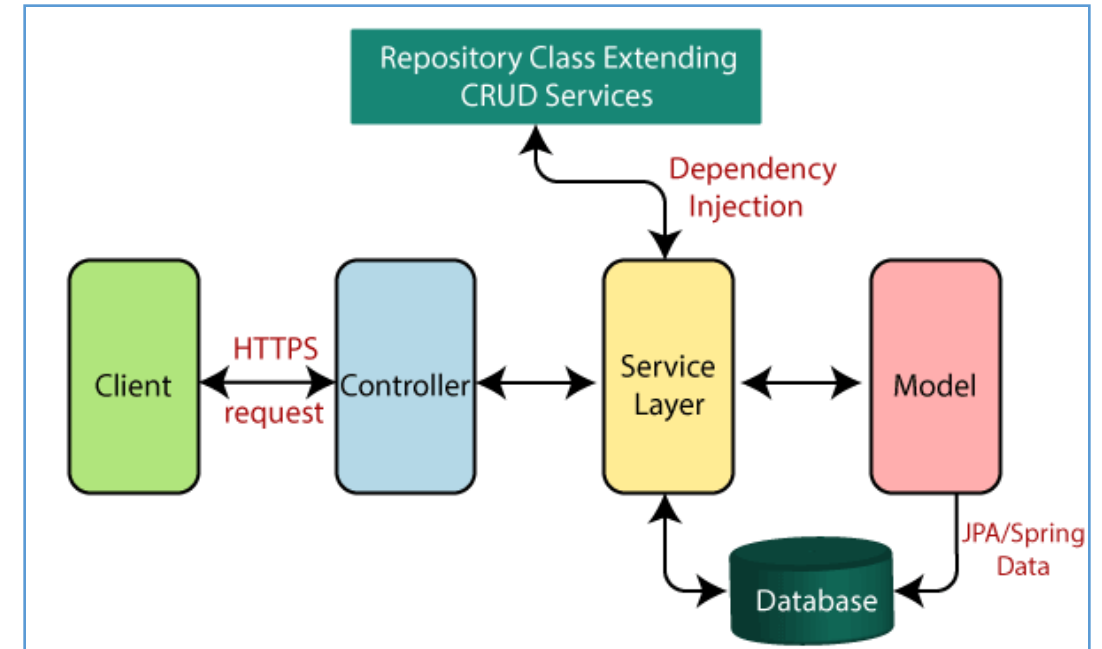


Spring Boot Layers

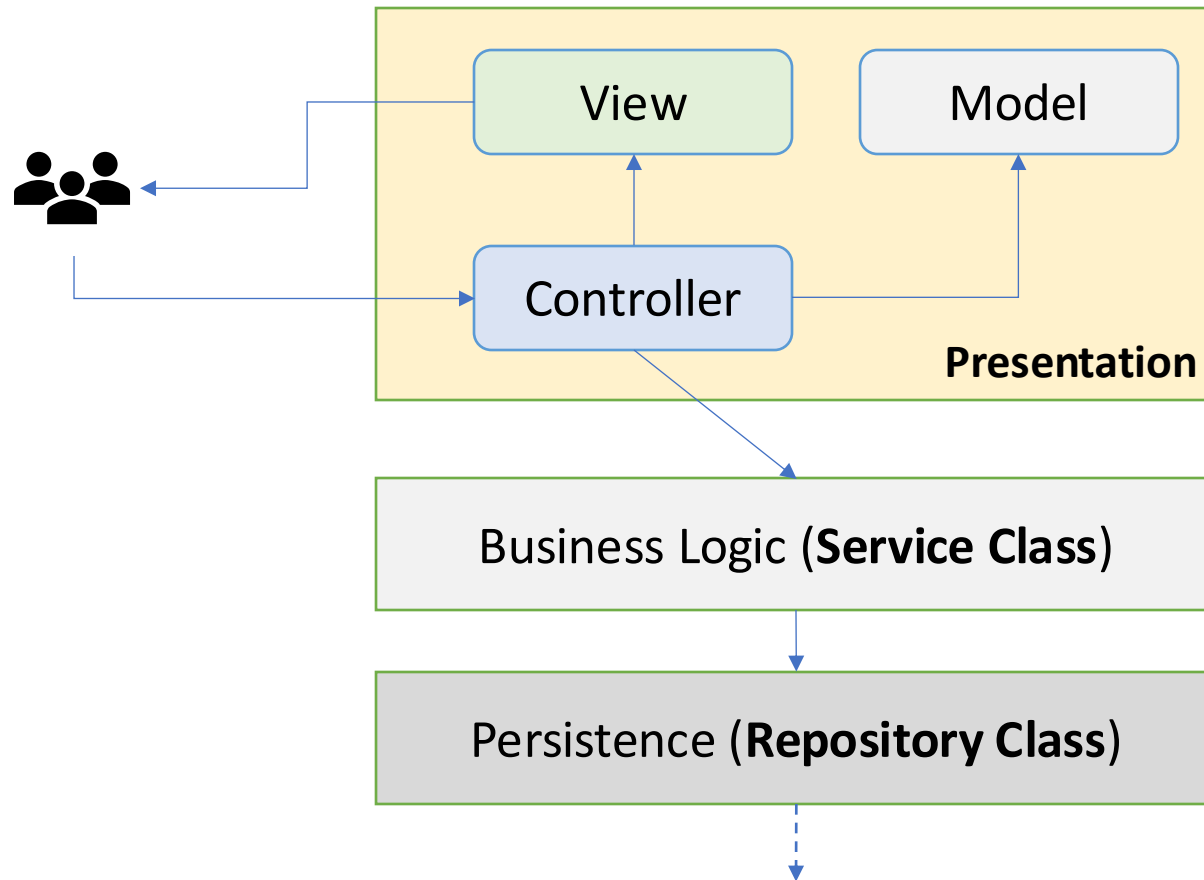
- Presentation Layer:
 - Handles the HTTP requests, translates the JSON parameter to object, and authenticates the request and transfer it to the business layer. In short, it consists of views i.e., frontend part.
- Business Layer:
 - Handles all the business logic. It consists of service classes and 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 Boot Layer Architectures vs MVC

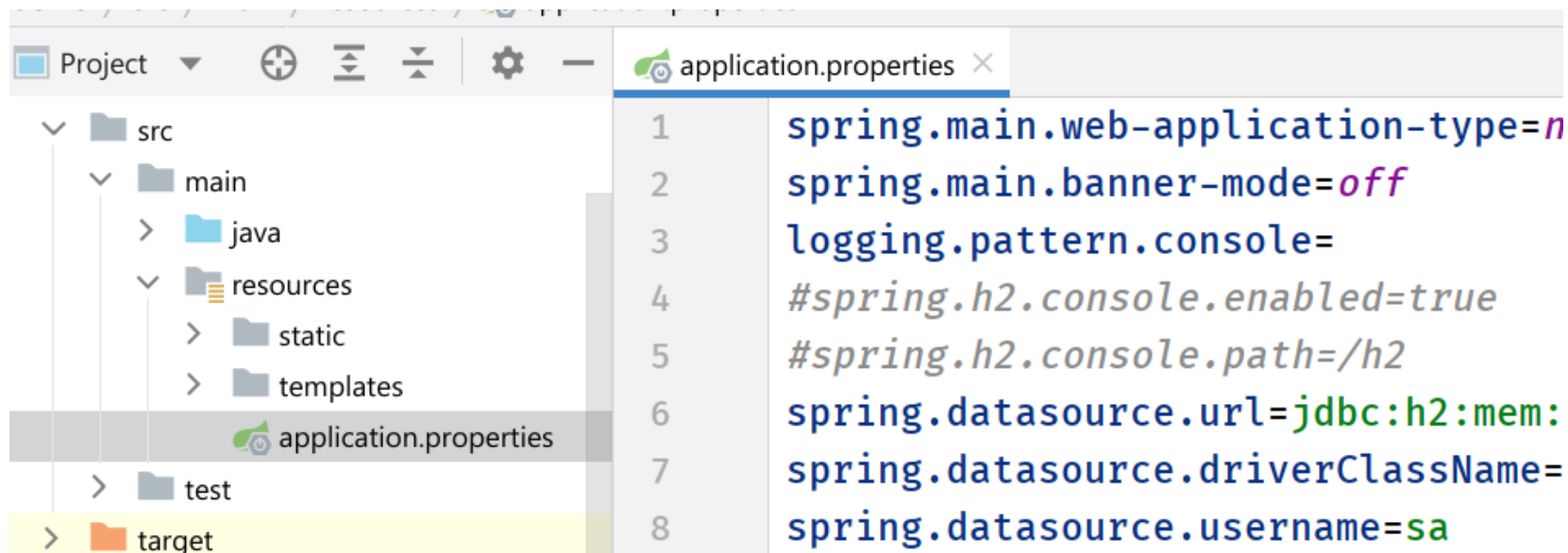


Spring Boot Annotations

- `@SpringBootApplication`
 - Combination of three annotations `@EnableAutoConfiguration`, `@ComponentScan`, and `@Configuration`.
- Core Spring Framework Annotations
 - `@Required` `@Autowired` `@Configuration` `@Bean`
- Spring Framework Stereotype Annotations (class-level annotation).
 - `@Component`
 - Used to mark a Java class as a bean
 - `@Controller`
 - It marks a class as a web request handler. It is often used to serve web pages
 - `@Service`
 - It tells the Spring that class contains the business logic.
 - `@Repository`
 - The repository is a DAOs (Data Access Object) that access the database directly.

Spring Boot Application Properties

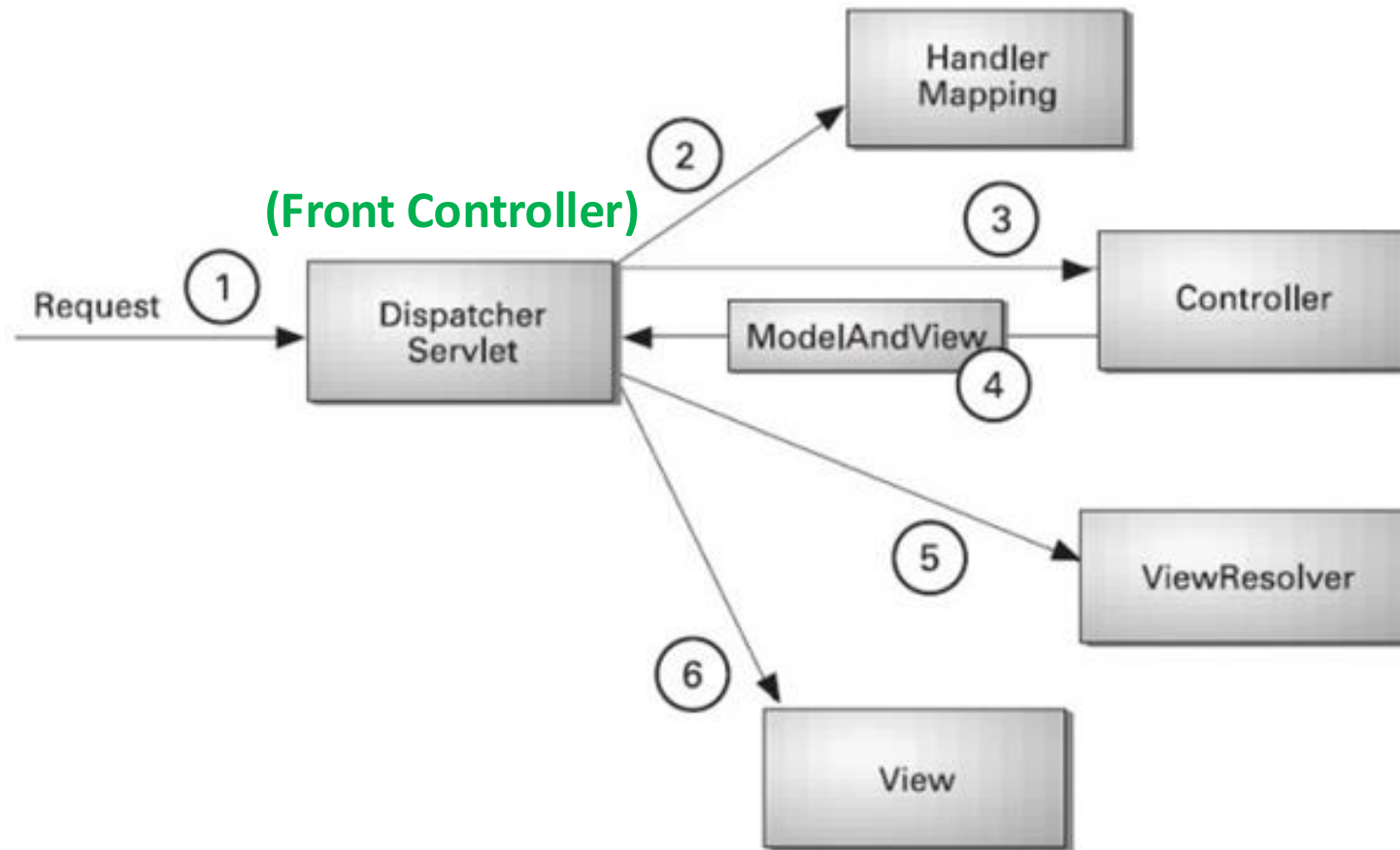
- Spring Boot Framework comes with a built-in mechanism for application configuration using a file called application.properties.
- It is located inside the src/main/resources folder.
- The properties have default values.
- We can set a property(s) for the Spring Boot application.



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);
    }
}
```

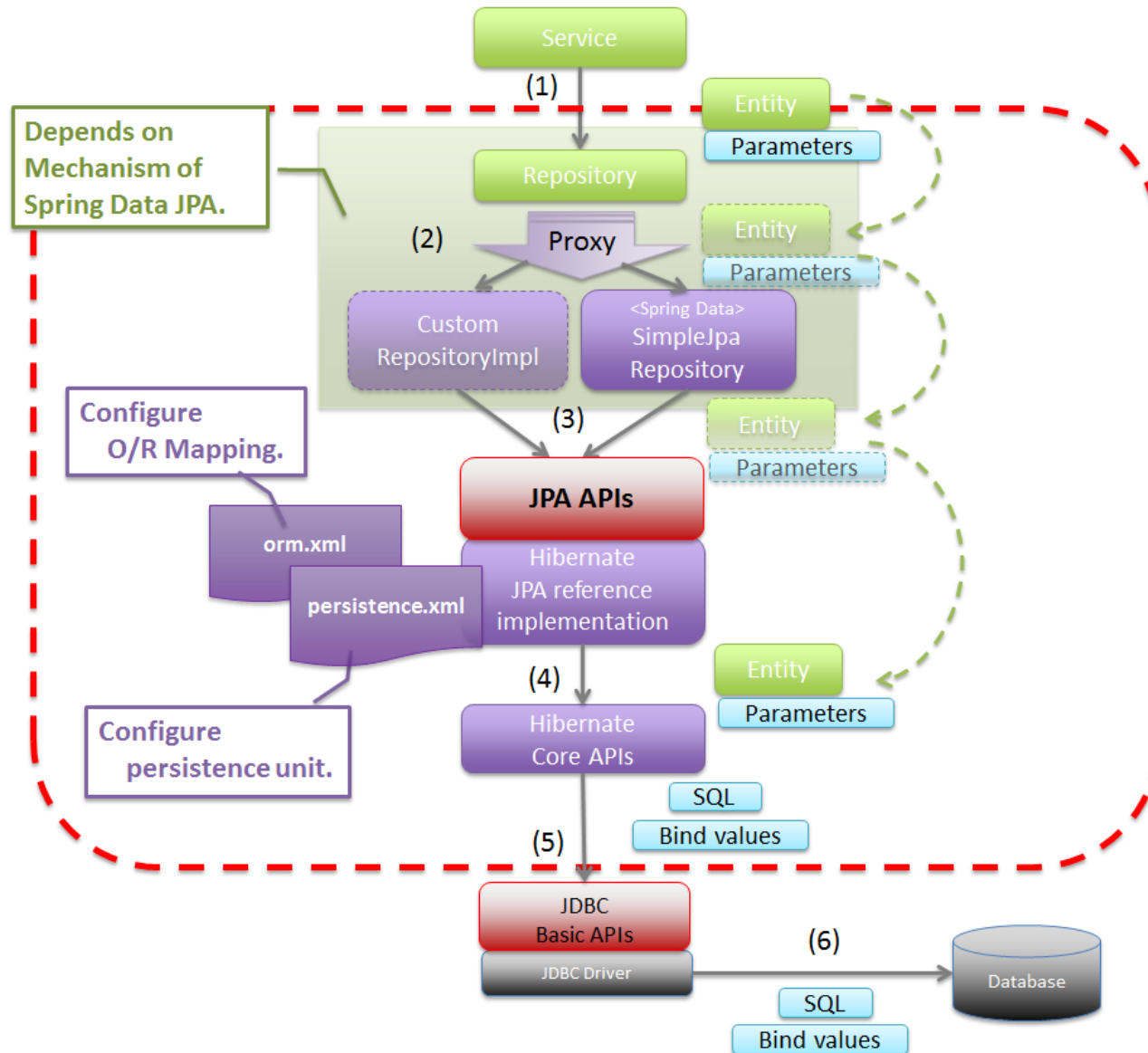
Spring View Technology

- 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

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.**

Basic Spring Data JPA Flow



JPA Repository Example

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

```
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);
}
```

Query methods

Jpa Repository methods

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

```
(m) count()  
(m) count(Example<S> example)  
(m) delete(Student entity)  
(m) deleteAll()  
(m) deleteAll(Iterable<? extends Student> entities)  
(m) deleteAllById(Iterable<? extends Student> ids)  
(m) deleteAllByIdInBatch(Iterable<Integer> ids)  
(m) deleteAllInBatch()  
(m) deleteAllInBatch(Iterable<Student> entities)
```

```
(m) deleteById(Integer id)  
(m) exists(Example<S> example)  
(m) existsById(Integer id)  
(m) findAllById(Iterable<Integer> ids)  
(m) findBy(Example<S> example, Function<String, Boolean> matcher)  
(m) findById(Integer id)  
(m) findOne(Example<S> example)  
(m) flush()  
(m) saveAll(Iterable<S> entities)
```

```
(m) saveAndFlush(S entity)  
(m) getById(Integer id)  
(m) findAll()  
(m) save(S entity)  
(m) findAll(Sort sort)  
(m) findAll(Example<S> example)  
(m) findAll(Example<S> example, Sort sort)  
(m) findAll(Pageable pageable)
```