Comprehensive Summary: How Spring MVC Works, Resolves, and Injects Dependencies

---

1. What is Spring MVC?

Spring MVC (Model-View-Controller) is a framework within the Spring ecosystem for building web applications. It separates the application into three interconnected components:

- **Model:** Holds application data and business logic.
- **View:** Renders the data (HTML, JSON, etc.) for the client.
- **Controller:** Handles incoming HTTP requests, processes user input, and returns responses.

---

2. Spring MVC Request Processing Flow

1. **Client sends HTTP request** → hits a DispatcherServlet (front controller).
2. **DispatcherServlet**:
   - Finds the appropriate controller using handler mappings.
   - Passes the request to the matched controller.
3. **Controller**:
   - Processes the request (e.g., fetches data via service layer).
   - Returns a ModelAndView (view name + data) or ResponseEntity.
4. **View Resolver**:
   - Resolves the logical view name to an actual view (e.g., JSP, Thymeleaf, JSON).
5. **View**:
   - Renders the response and returns it to the client.

---

3. Dependency Resolution and Injection in Spring MVC

a. Dependency Injection (DI)

Spring uses **Dependency Injection** to manage object creation and wiring. This means that you define the dependencies (collaborators) of a class, and Spring provides them at runtime.

Types of Dependency Injection

- **Constructor Injection:** Dependencies are provided via class constructors.
- **Setter Injection:** Dependencies are provided via setter methods.
- **Field Injection:** Dependencies are injected directly into fields (not recommended for testability).

Example

```
@Controller
public class MyController {
    private final MyService myService;

    @Autowired // Constructor injection (recommended)
    public MyController(MyService myService) {
        this.myService = myService;
    }
}
```

```
@Controller
public class MyController {
    @Autowired // Default constructor injection (recommended)
    private final MyService myService;
}
```

b. How Does Spring Resolve and Inject Dependencies?

1. Component Scanning and Bean Creation

- Spring scans packages for classes annotated with stereotypes: `@Component`, `@Controller`, `@Service`, `@Repository`.
- It creates a bean (object) for each detected component and manages its lifecycle in the ApplicationContext.

2. Dependency Resolution

- When creating a bean (e.g., a controller), Spring looks at its dependencies (constructor parameters, fields, or setters).
- For each dependency (e.g., a service), Spring finds a matching bean in the context.

- If only one bean matches, Spring injects it automatically. If multiple beans match, you may need to use `@Qualifier` to specify which one to use.

3. Injection

- Spring injects the resolved dependencies using the chosen method (constructor, setter, or field).

4. Request Handling

- When an HTTP request hits the controller, Spring ensures all dependencies are already injected and ready for use.

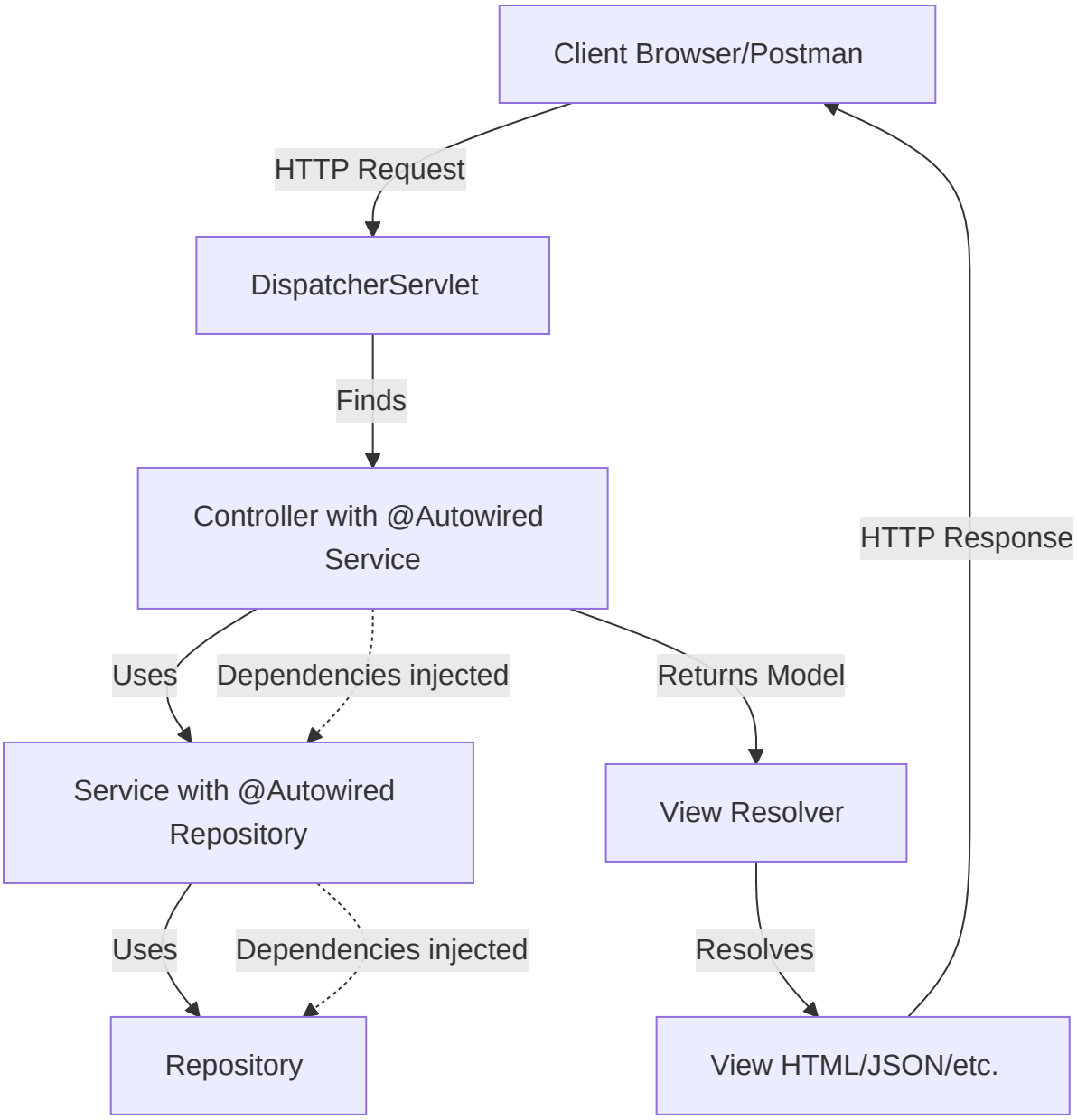
---

4. Key Annotations

- `@Controller`: Marks a class as a Spring MVC controller.
- `@Service`: Marks a service-layer bean.
- `@Repository`: Marks a DAO/data access bean.
- `@Autowired`: Tells Spring to inject a dependency automatically.
- `@Component`: Generic stereotype for a Spring-managed bean.
- `@RequestMapping`, `@GetMapping`, etc.: Map HTTP requests to controller methods.

---

5. Example Diagram



---

6. Summary Table

Concept	Description
DispatcherServlet	Central front controller that handles all web requests.
Controller	Handles HTTP requests, business logic, and returns response.
Service	Contains business logic, reusable methods.
Repository	Handles data persistence (database access).
Dependency Injection	Spring automatically wires dependencies into beans.
Bean	An object managed by Spring’s IoC container.

In summary:
Spring MVC uses a layered architecture and the principle of dependency injection to automatically resolve and supply dependencies (services, repositories, etc.) to controllers and other beans. This fosters loose coupling, easier testing, and maintainable code.