



**SPRING II**

## @SPRINGBOOTAPPLICATION

- `@EnableAutoConfiguration`: enable Spring Boot's auto-configuration mechanism
- `@ComponentScan`: enable `@Component` scan on the package where the application is located
- `@Configuration`: allow to register beans in the context or import additional configuration classes

All of your application components (`@Component`, `@Service`, `@Repository`, `@Controller` etc.) are automatically registered as Spring Beans

## @EnableAutoConfiguration

- ❑ Spring Boot auto-configuration attempts to automatically configure our Spring application based on the jar dependencies Added by the programmer
- ❑ if **HSQLDB** is on your classpath, and you have not manually configured any database connection beans then
  - ❑ Spring Boot auto-configures an in-memory database.

Spring Boot adds @EnableWebMvc automatically when it sees spring-webmvc on the classpath. This flags the application as a web application and activates key behaviors such as setting up a DispatcherServlet.

- ❑ Auto-configuration is non-invasive.
- ❑ At any point, you can start to define your own configuration to replace specific parts of the auto-configuration.
- ❑ For example, if you add your own **DataSource** bean, the default embedded database support backs away.

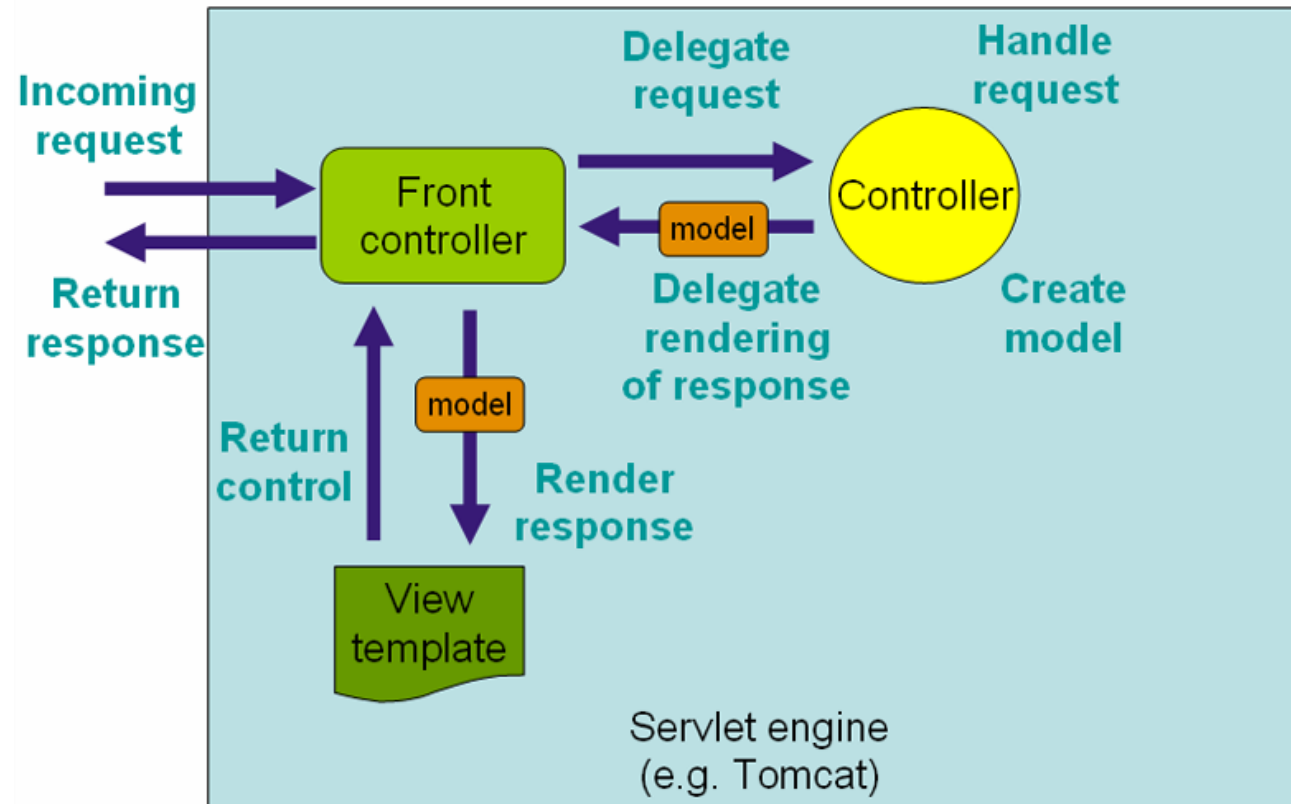
## @EnableAutoConfiguration

- ❑ these four annotations on this configuration class go and set up an entire web container
- ❑ they create the DispatcherServlet that we need to route requests to our controllers.
- ❑ They automatically scan the appropriate packages that we want, and discover our controllers,
- ❑ they'll automatically configure our controllers with any dependencies that we want them to have.

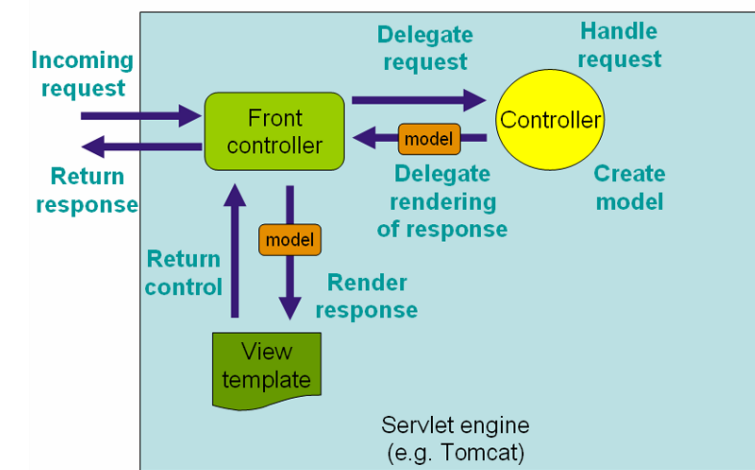
# APPLICATIONCONTEXT

- ❑ If Dependency Injection is the core concept of Spring, then the ApplicationContext is its core object.
- ❑ The interface `org.springframework.context.ApplicationContext` represents the Spring IoC container
- ❑ It extends the BeanFactory interface, in addition to extending other interfaces to provide additional functionality in a more *application framework-oriented style*
- ❑ The container gets its instructions on what objects to instantiate, configure, and assemble by reading configuration metadata
- ❑ Several implementations of the ApplicationContext interface are supplied out-of-the-box with Spring
  - ❑ such as ContextLoader that automatically instantiates an ApplicationContext as part of the normal startup process

# MVC WORKFLOW



# DISPATCHERSERVLET

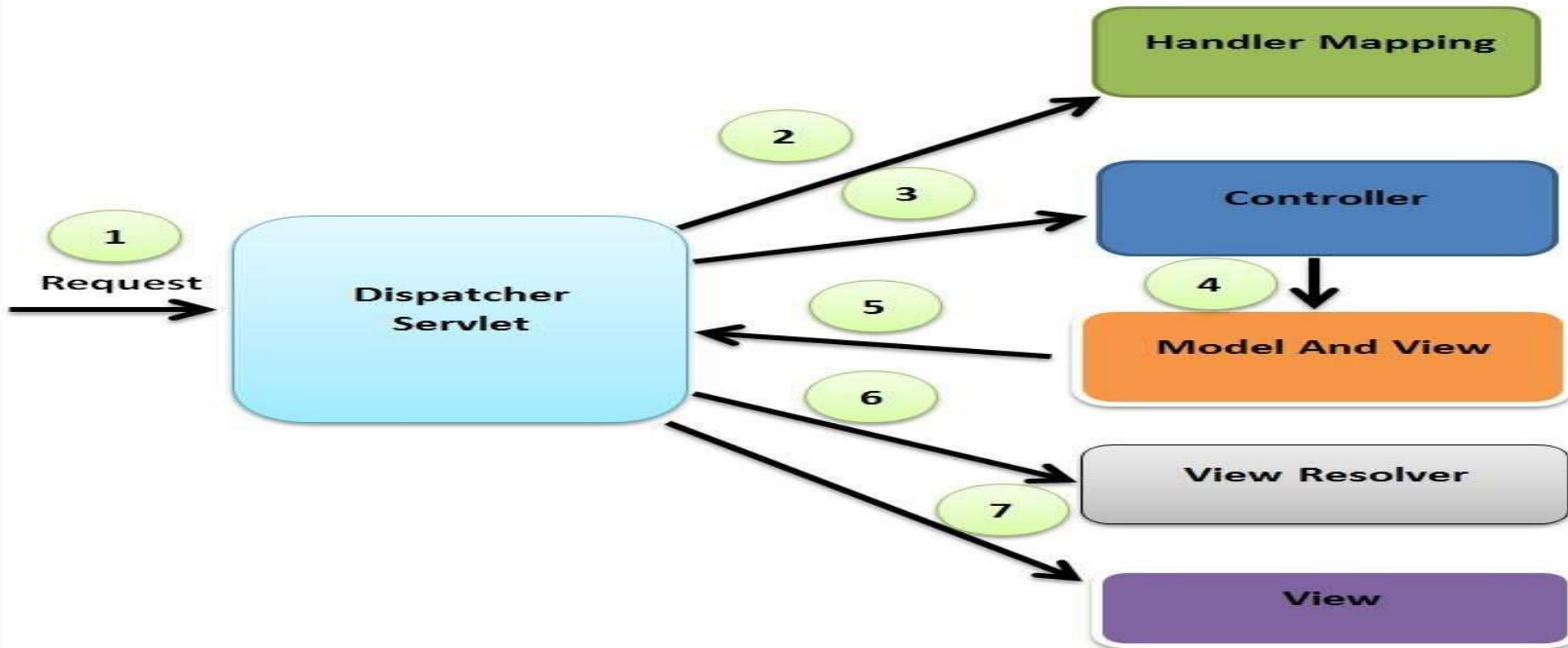


It gets its name from the fact that it dispatches the request to many different components, each an abstraction of the processing pipeline

1. Discover the request's Locale; expose for later usage.
2. Locate which request handler is responsible for this request (e.g., a Controller).
3. Locate any request interceptors for this request. Interceptors are like filters, but customized for Spring MVC.
4. Invoke the Controller.
5. Call `postHandle()` methods on any interceptors.
6. If there is any exception, handle it with a `HandlerExceptionResolver`.
7. If no exceptions were thrown, and the Controller returned a `ModelAndView`, then render the view. When rendering the view, first resolve the view name to a View instance.

## SPRING 3.0

The `DispatcherServlet` is an expression of the “Front Controller” design pattern the `@Controller` mechanism also allows you to create RESTful Web sites and applications, through the `@PathVariable` annotation







**WITHOUT HIBERNATE**

CRUD

```

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;

public class CRUDEXample {

    // Set up the database connection
    private static final String DB_URL =
        "jdbc:mysql://localhost/mydatabase";
    private static final String DB_USER = "myuser";
    private static final String DB_PASSWORD = "mypassword";

    public static void main(String[] args) {
        try (Connection conn = DriverManager.getConnection(DB_URL,
            DB_USER, DB_PASSWORD)) {
            // Create a new record
            String insertQuery = "INSERT INTO users (name, email) VALUES
                (?, ?)";
            PreparedStatement pstmt =
                conn.prepareStatement(insertQuery);

```

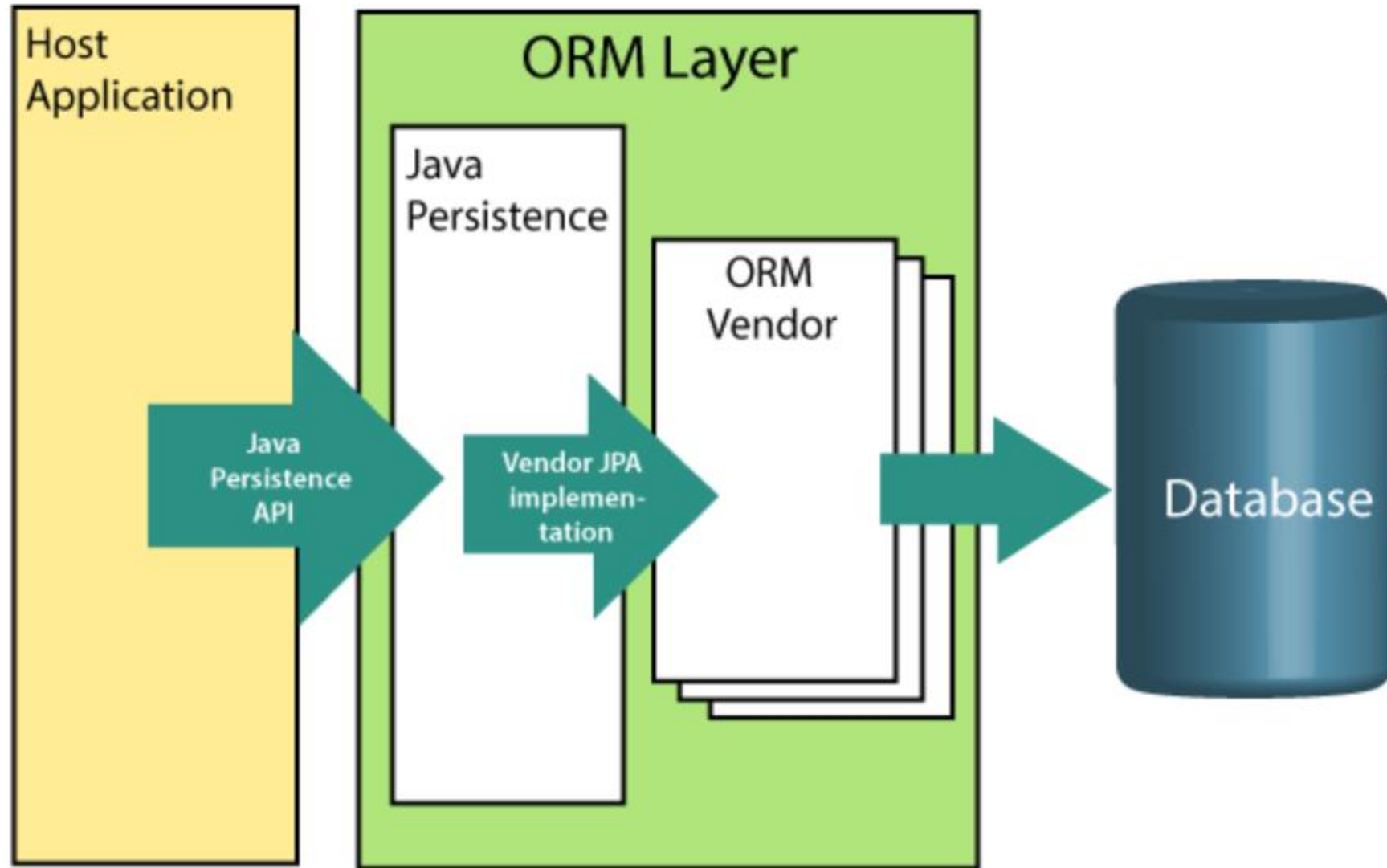
```


        pstmt.setString(1, "John Doe");
        pstmt.setString(2, "johndoe@example.com");
        int rowsAffected = pstmt.executeUpdate();
        System.out.println("Inserted " + rowsAffected + " rows.");
        // Retrieve a record
        String selectQuery = "SELECT * FROM users WHERE id = ?";
        pstmt = conn.prepareStatement(selectQuery);
        pstmt.setInt(1, 1);
        ResultSet rs = pstmt.executeQuery();
        while (rs.next()) {
            int id = rs.getInt("id");
            String name = rs.getString("name");
            String email = rs.getString("email");
            System.out.println(id + ": " + name + " (" + email + ")");
        }
        // Update a record
        String updateQuery = "UPDATE users SET email = ? WHERE
            ?";
        pstmt = conn.prepareStatement(updateQuery);
        pstmt.setString(1, "newemail@example.com");
        pstmt.setInt(2, 1);
        rowsAffected = pstmt.executeUpdate();
        System.out.println("Updated " + rowsAffected + " rows.");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

```

# SPRING DATA JPA

- ❑ The JPA (Java Persistence API) provides a POJO persistence model for object-relational mapping
- ❑ It follows the *aggregate root* concept
  - ❑ Support for repositories (a concept from *Domain-Driven Design*)
- ❑ Implementing data access can be a hassle because we need to deal with connections, sessions, exception handling, and more, even for simple CRUD operations
- ❑ That's why the Spring Data JPA provides an additional level of functionality: creating repository implementations directly from interfaces and using conventions to generate queries from method names
- ❑ Pagination, sort, dynamic query execution support.
- ❑ Support for @Query annotations
- ❑ JavaConfig based repository configuration by using the @EnableJpaRepositories annotation.





Hibernate is an ORM framework that provides a high-level API for interacting with databases, JPA is a specification that defines a common API for ORM frameworks like Hibernate, and Spring Data JPA is a part of the Spring Framework that provides a higher-level, easier-to-use API for working with JPA.

# REPOSITORY CONCEPT

```
@NoRepositoryBean
public interface CrudRepository<T, ID> extends Repository<T, ID> {
    <S extends T> S save(S entity);
    <S extends T> Iterable<S> saveAll(Iterable<S> entities);
    Optional<T> findById(ID id);
    boolean existsById(ID id);
    Iterable<T> findAll();
    Iterable<T> findAllById(Iterable<ID> ids);
    long count();
    void deleteById(ID id);
    void delete(T entity);
    void deleteAll(Iterable<? extends T> entities);
    void deleteAll();
}
```

- ❑ The most compelling feature of Spring JPA is the ability to create repository implementations automatically, at runtime, from a repository interface.
- ❑ We only need to create an interface that extends from a `Repository<T,ID>`, `CrudRepository<T,ID>`, or `JpaRepository<T,ID>`
- ❑ The `JpaRepository` interface offers not only what the `CrudRepository` does, but also extends from the `PagingAndSortingRepository` interface that provides extra functionality

the T means the entity (your domain model class) and the ID, the primary key that needs to implement Serializable.

# JAVA PERSISTENCE API

In a simple Spring app, you are required to use the `@EnableJpaRepositories` annotation that triggers the extra configuration that is applied in the life cycle of the repositories defined within of your application

The class that should persist in the database

`@Entity`



```
public class Person {  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private long id;  
  
    private String firstName;  
    private String lastName;  
    Getter and setter methods, constructors both versions }  
}
```

# APPLICATION PROPERTIES

Spring Boot provides properties that allow you to override defaults when using the Spring Data JPA. One of them is the ability to create the DDL (data definition language), which is turned off by default, but you can enable it to do reverse engineering from your domain model.

`hibernate.ddl.auto (create, create-drop, update) spring.jpa.hibernate.ddl-auto=update`

- `none`: The default for `MySQL`. No change is made to the database structure.
- `update`: Hibernate changes the database according to the given entity structures.
- `create`: Creates the database every time but does not drop it on close.
- `create-drop`: Creates the database and drops it when `SessionFactory` closes

`spring.jpa.show-sql=true`

The simplest way is to dump the queries to standard out



## CREATE THE REPOSITORY

```
public interface PersonRepository extends CrudRepository<Person, Long> {  
    List<Person> findByFirstName(String firstName);  
}
```

Invoke the database through dependency injection

```
@RepositoryRestResource(collectionResourceRel = "people", path = "people")
```

- At runtime, Spring Data REST automatically creates an implementation of this interface.
- collectionResourceRel -The rel value to use when generating links to the collection resource.
- Path-The path segment under which this resource is to be exported.

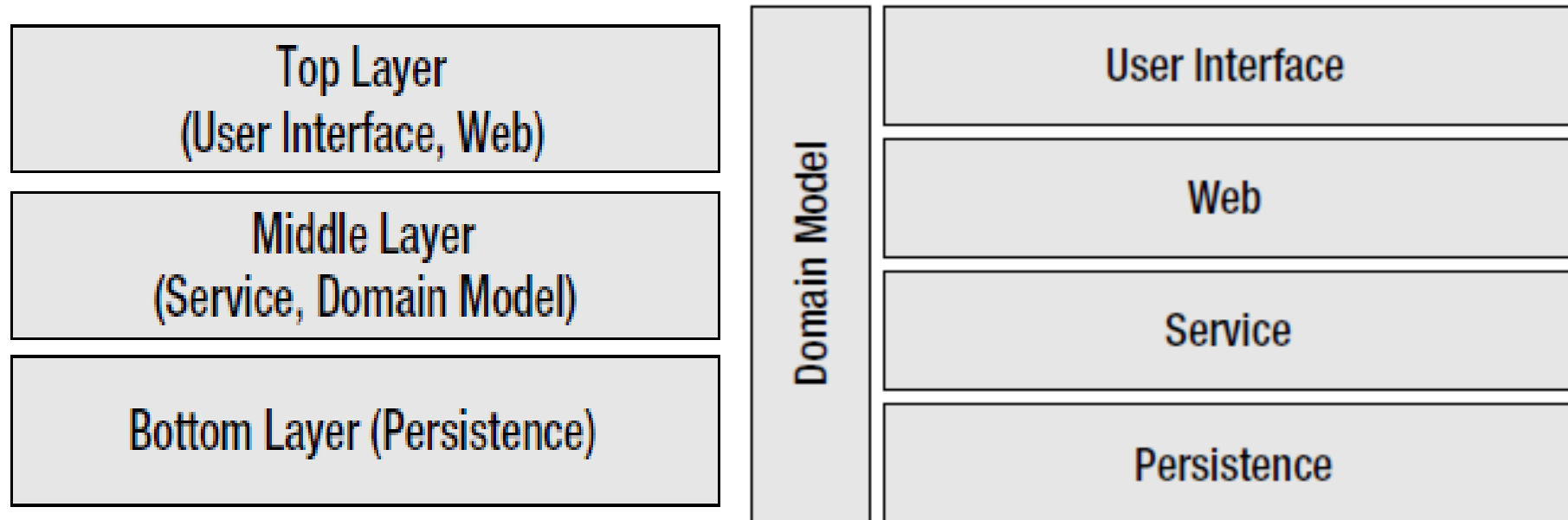
Then it uses the `@RepositoryRestResource` annotation to direct Spring MVC to create RESTful endpoints at `/people`

# APPLICATION.PROPERTIES

```
spring.datasource.url=jdbc:mysql://localhost:3306/testconnect
spring.datasource.username=root
spring.datasource.password=1234
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5InnoDBDialect
```

- Advanced translation of PersistenceExceptions to Spring DataAccessException
- Applying specific transaction semantics such as custom isolation level or transaction timeout
- Spring Data Jdbc

# ARCHITECTURE OF SPRING MVC



Isolating problem domains, such as persistence, web navigation, and user interface, into separate layers creates a flexible and testable application

If we find that a layer has permeated throughout many layers, consider if that layer is itself an aspect of the system.

The interface is a contract for a layer, making it easy to keep implementations and their details hidden while enforcing correct layer usage.

