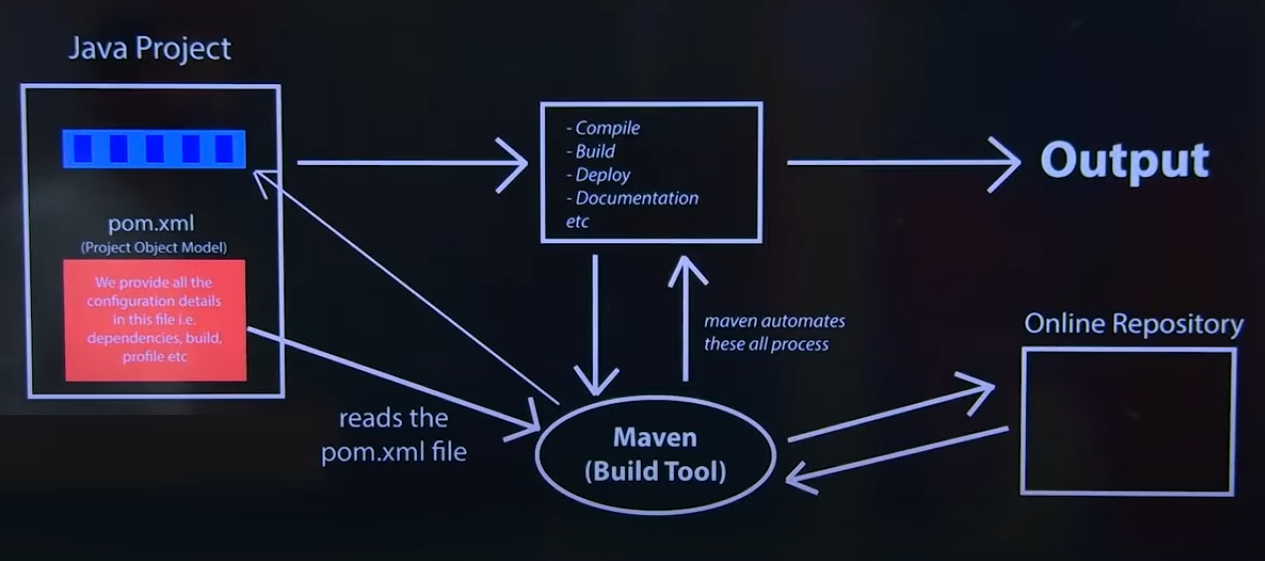
Maven is a build automation and dependency management tool primarily used for Java projects, though it can also be applied to other languages. Developed by the Apache Software Foundation, Maven simplifies project setup and management by automating processes like compiling code, running tests, and packaging applications.

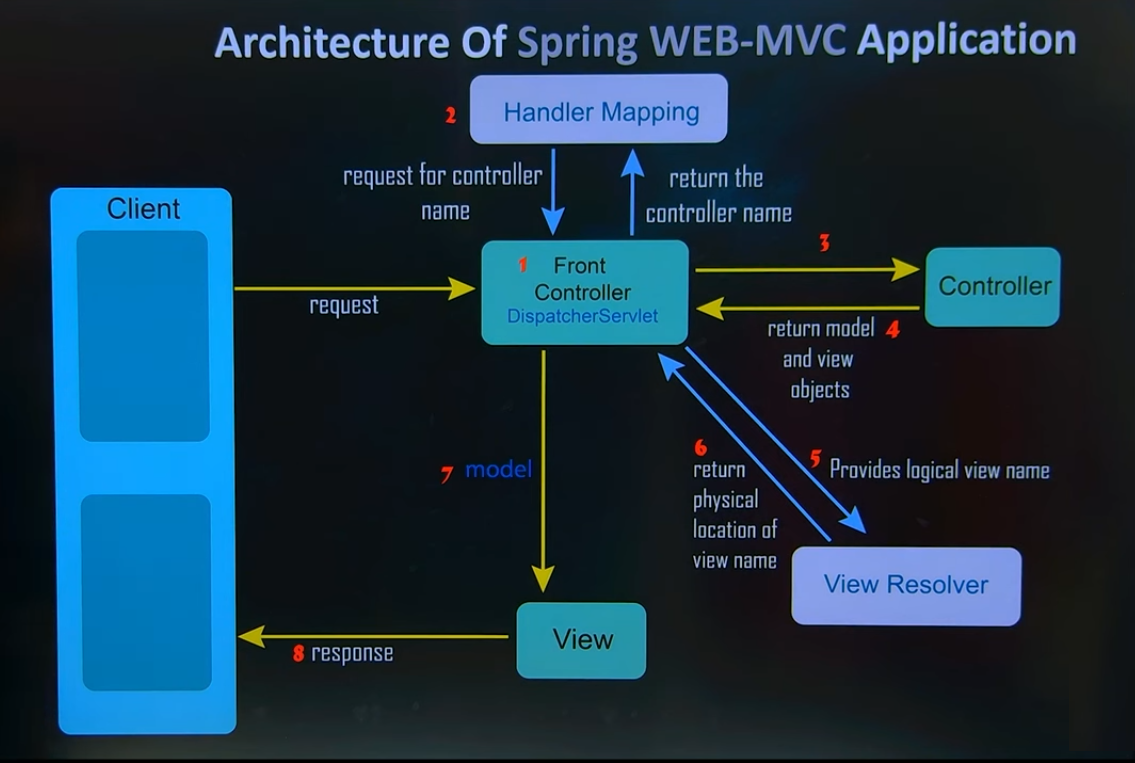
* Project Structure
* Download Dependencies
* Prepare the Documentation
* Compiles Source code
* Start or stops the server
* Packaging the project in jar or ear file.

**Key Features of Maven:**

1. **Dependency Management**: Maven automates dependency resolution. Instead of manually adding libraries, you specify dependencies in a pom.xml file (Project Object Model), and Maven downloads them from a central repository.
2. **Project Standardization**: Maven enforces a standard project structure, making it easy for developers to understand the layout of a project. The structure includes separate directories for source code, tests, resources, etc.
3. **Build Automation**: With Maven, you can automate complex build tasks like compilation, testing, packaging, and deployment by using predefined goals and plugins.
4. **Plugin-Based Architecture**: Maven's functionality can be extended using plugins. There are plugins for compiling Java code, running tests, packaging JARs/WARs, and deploying applications.
5. **Repository Management**: Maven connects with repositories (central or custom) to download project dependencies and plugins. It can also push built packages to these repositories, enabling artifact sharing across projects.
6. **Lifecycle Management**: Maven follows a lifecycle consisting of phases like *validate*, *compile*, *test*, *package*, *verify*, *install*, and *deploy*. Each phase represents a stage in the build process and can be customized.



**MVC**



**Key Components**

1. **Client**: Represents the user or front-end interface that initiates the HTTP request to the server.
2. **DispatcherServlet (Front Controller)**: This is the central component in the Spring MVC framework. It acts as a front controller that receives all incoming HTTP requests and coordinates the request processing by forwarding it to the appropriate handlers, views, and other components.
3. **Handler Mapping**: This component is responsible for mapping incoming requests to the correct controller. It checks the URL of the request and decides which controller should handle it.
4. **Controller**: Controllers are responsible for processing user requests and managing data interactions with the model. They interpret user inputs, fetch data if needed, and determine the response view.
5. **View Resolver**: The view resolver determines the appropriate view for a given request. Based on the view name provided by the controller, the view resolver finds the actual view file and returns it to the DispatcherServlet.
6. **View**: Represents the user interface that displays the data. Views are typically HTML, JSP, or other templated files that present the model data to the user in a readable format.

**Flow of the Request**

1. **Step 1 - Initial Request to Front Controller (DispatcherServlet)**:
   * The client sends a request to the application. All requests are initially captured by the DispatcherServlet, which acts as the front controller.
2. **Step 2 - Handler Mapping**:
   * The DispatcherServlet consults the Handler Mapping component to determine which Controller should handle the request based on the request’s URL or other attributes.
3. **Step 3 - Forwarding to Controller**:
   * After identifying the appropriate controller, the DispatcherServlet forwards the request to that specific Controller.
4. **Step 4 - Controller Processing and Returning Model**:
   * The controller processes the request, interacts with the model layer (e.g., fetching data from a database), and prepares the response data. It returns a **model** and a **view name** to the DispatcherServlet.
5. **Step 5 - View Resolver Provides Logical View Name**:
   * The DispatcherServlet passes the view name to the View Resolver, which maps the logical view name to a physical view file (such as an HTML or JSP file).
6. **Step 6 - View Resolver Returns Physical View Location**:
   * The View Resolver returns the physical location of the view file to the DispatcherServlet.
7. **Step 7 - Model and View to the View**:
   * The DispatcherServlet sends the model data and view location to the View component.
8. **Step 8 - Sending Response to Client**:
   * The view generates the final output (e.g., an HTML page) based on the model data and sends it as a response back to the client.

**ORM**