Top of Form

**JDBC Driver**

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:JDBC-ODBC bridge driverNative-API driver (partially java driver)Network Protocol driver (fully java driver)Thin driver (fully java driver)

**1) JDBC-ODBC bridge driver**

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

**In Java 8, the JDBC-ODBC Bridge has been removed.**

Oracle does not support the JDBC-ODBC Bridge from Java 8. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

**Advantages:**

* easy to use.
* can be easily connected to any database.

**Disadvantages:**

* Performance degraded because JDBC method call is converted into the ODBC function calls.
* The ODBC driver needs to be installed on the client machine.

**2) Native-API driver**

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

**Advantage:**

* performance upgraded than JDBC-ODBC bridge driver.

**Disadvantage:**

* The Native driver needs to be installed on the each client machine.
* The Vendor client library needs to be installed on client machine.

**3) Network Protocol driver**

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

**Advantage:**

* No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

**Disadvantages:**

* Network support is required on client machine.
* Requires database-specific coding to be done in the middle tier.
* Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

**4) Thin driver**

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

**Advantage:**

* Better performance than all other drivers.
* No software is required at client side or server side.

**Disadvantage:**

* Drivers depend on the Database.

Bottom of Form

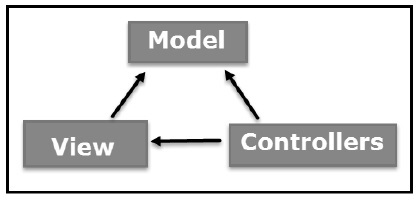
Top of Form

**MVC Framework**

The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

**MVC Components**

Following are the components of MVC −

**Model**

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

**View**

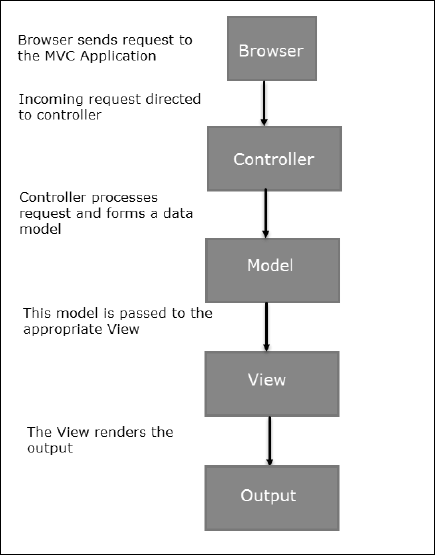
The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

**Controller**

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

Now let us take a look at how the execution of an MVC application takes place when there is a certain request from the client. The following diagram illustrates the flow.

**MVC Flow Diagram**

****

**Flow Steps**

Step 1 − The client browser sends request to the MVC Application.

Step 2 − Global.ascx receives this request and performs routing based on the URL of the incoming request using the RouteTable, RouteData, UrlRoutingModule and MvcRouteHandler objects.

Step 3 − This routing operation calls the appropriate controller and executes it using the IControllerFactory object and MvcHandler object's Execute method.

Step 4 − The Controller processes the data using Model and invokes the appropriate method using ControllerActionInvoker object

Step 5 − The processed Model is then passed to the View, which in turn renders the final output.

Bottom of Form

Top of Form

**Reference Links:**

https://www.tutorialspoint.com/mvc\_framework/mvc\_framework\_architecture.htm

https://www.tutorialspoint.com/mvc\_framework/mvc\_framework\_models.htm

https://www.guru99.com/mvc-tutorial.html

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

https://www.educative.io/blog/mvc-tutorial

Bottom of Form