**Component Diagram:** It gives a static view of the system. It serves as a transition from design to implementation. Its internals are hidden and inaccessible. It is a autonomous unit within a system.

A component is encapsulated. Its dependencies are designed such that it can be treated as independently as possible.

**Component:**

* **Representation:** A rectangle with two small rectangles stacked on its left side. You can also use a more simplified rectangle with the <<component>> stereotype.
* **Definition:** A modular part of a system that encapsulates its content (which can be classes, interfaces, and other components) and whose manifestation is replaceable within its environment.
* **Characteristics:**
  + **Modularity:** Represents a cohesive unit of functionality.
  + **Encapsulation:** Hides its internal implementation details.
  + **Replaceability:** Can be substituted with other components that conform to the same interfaces.
  + **Deployability:** Often corresponds to deployable artifacts like JAR files, DLLs, executables, or web services.

**Interface:**

* **Provided Interface (Lollipop):** Represented by a circle (lollipop) attached to a component. It signifies an interface that the component **implements** and makes available to other components. The name of the interface can be written next to the lollipop.
* **Required Interface (Socket):** Represented by a half-circle (socket) attached to a component. It signifies a specific interface that the component **needs** from other components to fulfill its functionality. It explicitly names the interface that the component requires.

**Port:**

* **Representation:** A small rectangle attached to the boundary of a component.
* **Purpose:** Defines a specific interaction point between a component and its environment. Ports can specify provided and required interfaces. They allow for a more detailed specification of how a component interacts. A port can have a name and a type (interface).

**Dependency:**

* **Representation:** A dashed arrow pointing from the component that **requires** a service to the component that **provides** that service. The arrow can be labelled with a stereotype like <<uses>>. A more **general reliance** of one component on another, but it doesn't necessarily specify a particular interface.
* **Definition:** Indicates that one component relies on another component for its proper functioning.

**Assembly Connector:**

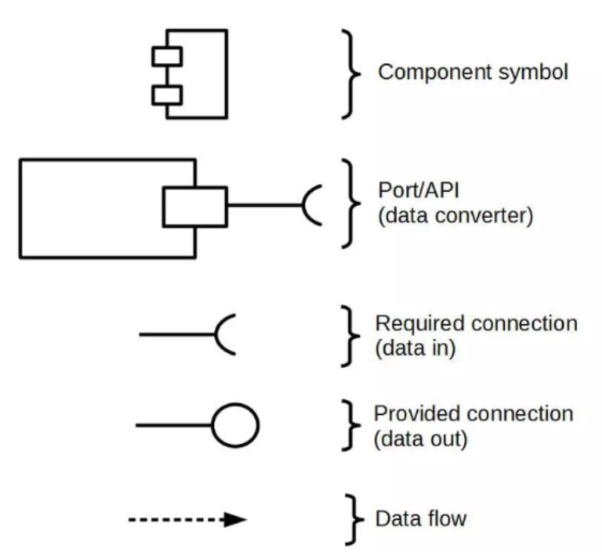
* **Representation:** A solid line connecting a required interface (socket) of one component to a provided interface (lollipop) of another component.
* **Purpose:** Shows the actual wiring or connection between components, indicating that the required interface of one is satisfied by the provided interface of the other.
* **They can be used as:**

**Delegate Connector:**

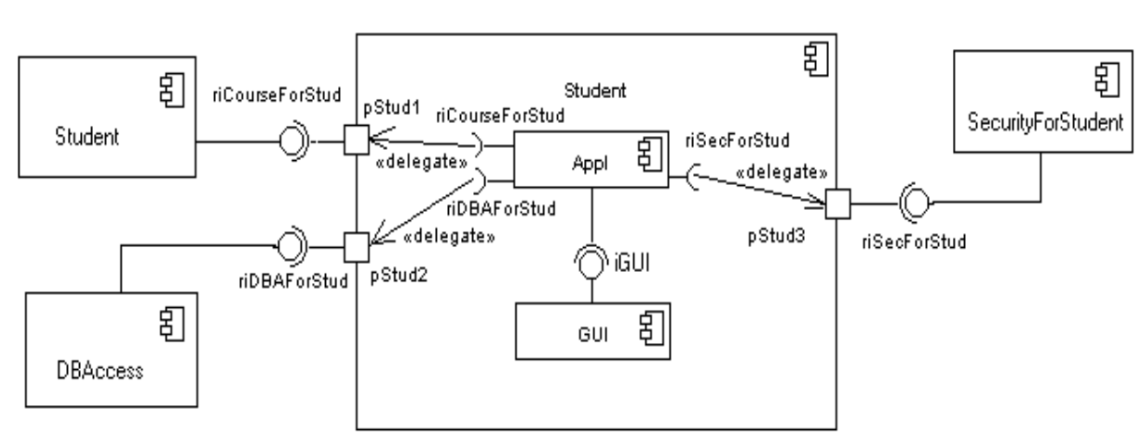
* **Representation:** A solid line connecting an external port of a component to an internal part (e.g., an internal component or an interface realization).
* **Purpose:** Shows that the responsibility for an interface exposed by the component is delegated to an internal element.

**Stereotypes:**

* **Representation:** Text enclosed in double angle brackets (<< >>), used to extend the meaning of UML elements. Common stereotypes for components include <<library>>, <<executable>>, <<service>>.

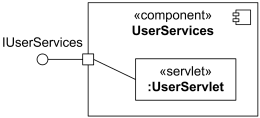


Using **dependency arrows** **or sockets** when the source component itself is using the required interface from the port:

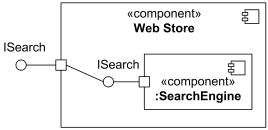


Using **solid line** when the source component delegates the interface to an internal class or component, which is from the port to the port.

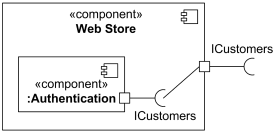
Another way of **delegating interface** when the component itself is not going to use the interface is:



* "Delegation connector from the delegating port to the UserServelet part"
* **Delegation Connector (Solid Line):** The solid line connects the IServices port of UserServices to the :UserServlet component. This **delegate connector** signifies that when an external entity interacts with the IServices interface of UserServices, the calls are actually being delegated to the :UserServlet component for implementation.

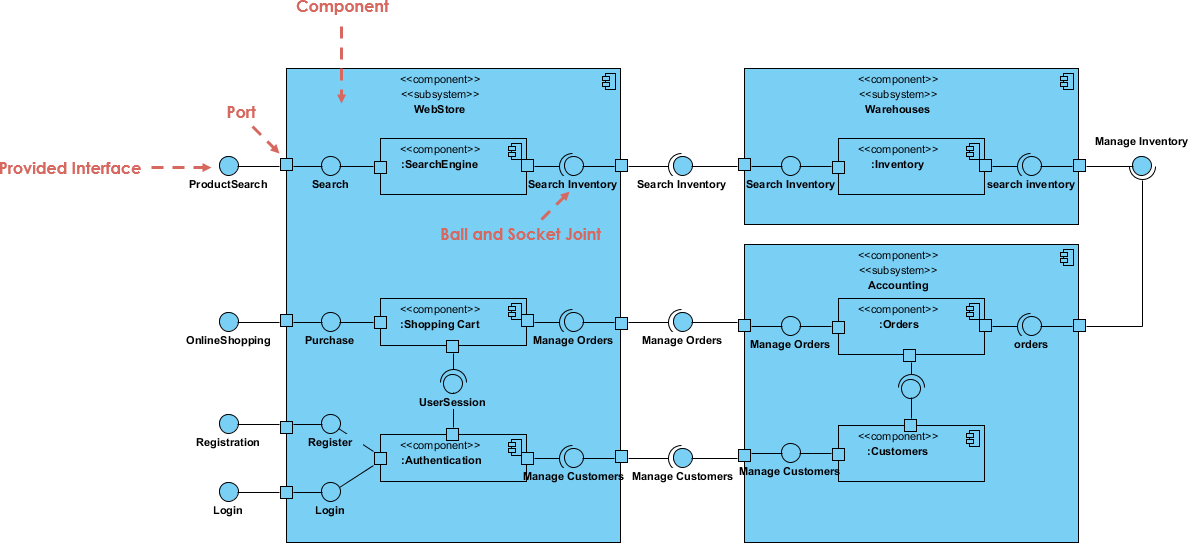


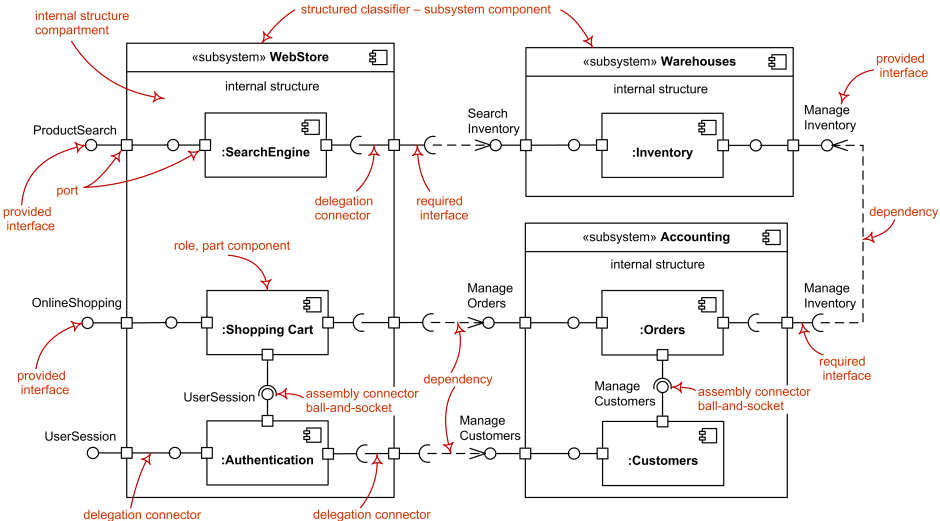
* "Delegation connector from the delegating port to the simple port of SearchEngine"
* **Delegation Connector (Solid Line):** The solid line connects the ISearch port of Web Store to the ISearch port of the internal :SearchEngine component. This indicates that when an external entity calls the ISearch interface of Web Store, the call is delegated to the ISearch interface provided by the internal :SearchEngine component.



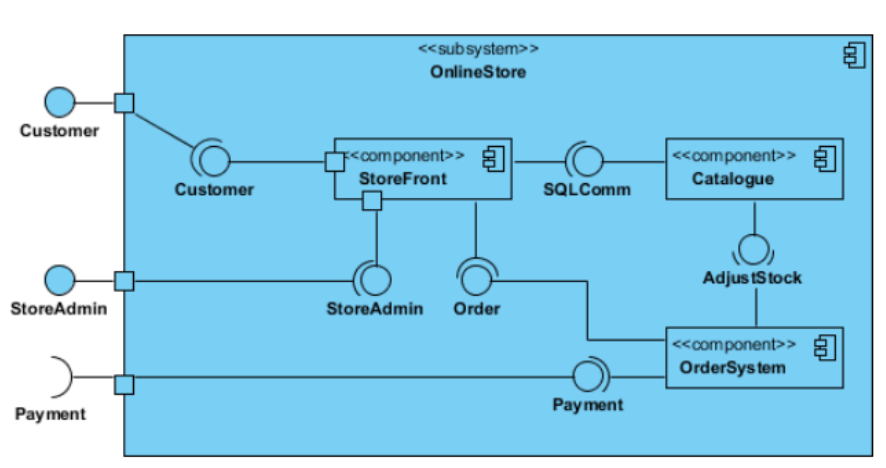
* "Delegation connector from the simple port of Authentication component to the delegating port"
* **Delegation Connector (Solid Line):** The solid line connects the ICustomers port of the internal :Authentication component to the ICustomers port of the Web Store component. In this case, the internal :Authentication component is requiring the ICustomers functionality, and the Web Store is exposing it as its own required interface. This might seem counterintuitive at first, but it could represent a scenario where the Web Store needs customer information for its internal operations, and the :Authentication component is the sink of that information within the Web Store. The Web Store then might expose this need to an external system that ultimately provides the customer data.

**Online Webstore Component Diagram**





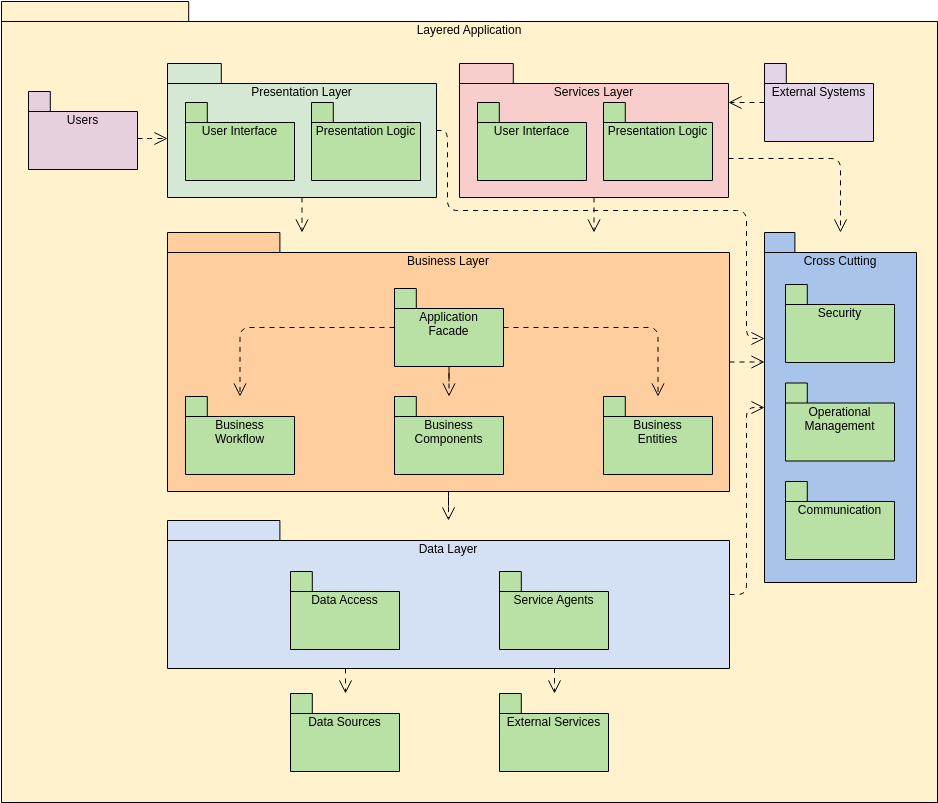
**Another syntax of the Component Diagram:**

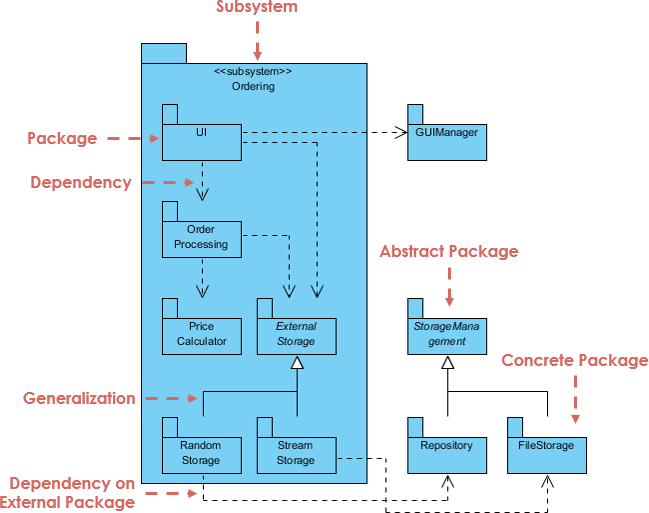


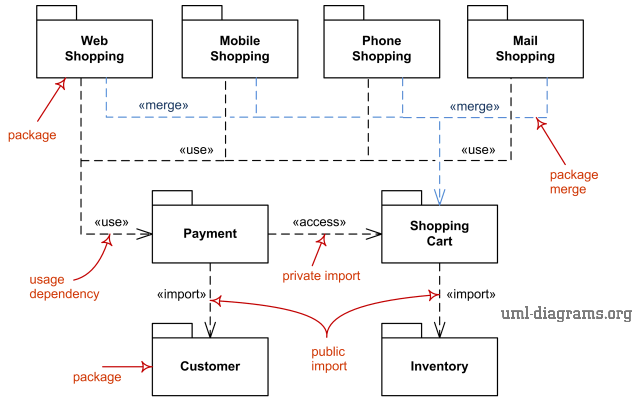
**Package Diagram**

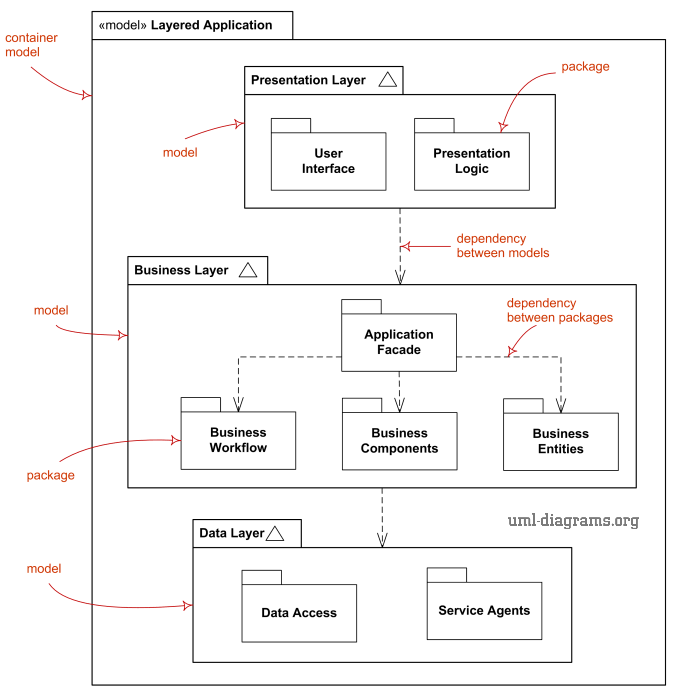
A **Package Diagram** in the Unified Modelling Language (UML) is a structural diagram used to **organize the elements of a model into logical groups called packages**. Think of packages as containers or namespaces that help manage complexity and provide a high-level view of the system's architecture.

A keyword is shown near the dashed arrow to identify which kind of package import is intended. The predefined keywords are **«import»** for a **public** package import, and **«access»** for a **private** package import. By default, the value of visibility is **public**, so it is the same as **«import»**.









**Deployment Diagram**

**Node:**

* **Representation:** A three-dimensional box (cube).
* **Definition:** Represents a physical or virtual computing resource capable of hosting and executing artifacts. Examples include servers, workstations, devices (like mobile phones or sensors), virtual machines, and even network devices.
* **Stereotypes:** Nodes are often stereotyped to indicate their specific type, such as <<server>>, <<database server>>, <<web server>>, <<application server>>, <<workstation>>, <<mobile device>>, <<virtual machine>>.

**Artifact:**

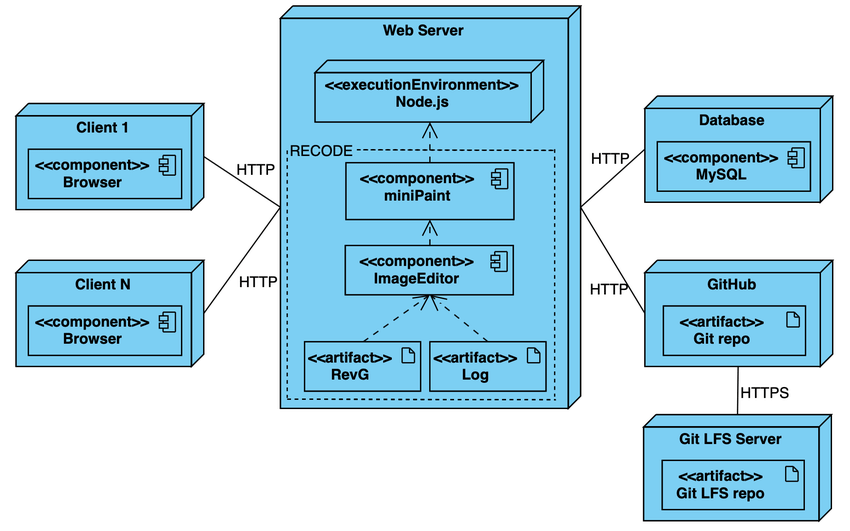
* **Representation:** A rectangle with the stereotype <<artifact>> and the artifact's name. It can also contain a list of the files it comprises.
* **Definition:** Represents a concrete physical element that is the result of the development process. Examples include executable files (.exe, .jar), libraries (.dll, .so), configuration files, data files, HTML files, and scripts.
* **Deployment:** Artifacts are shown as being deployed onto nodes, indicating that the node hosts or runs that artifact. This is typically represented by a dashed arrow pointing from the artifact to the node.

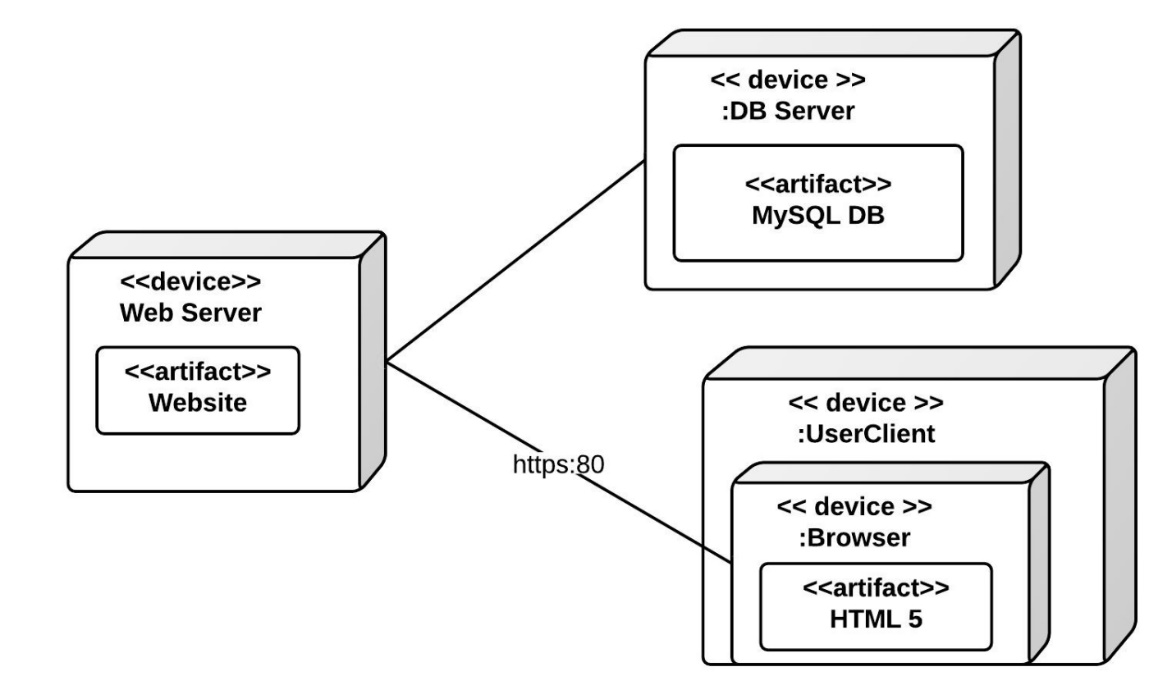
**Communication Path:**

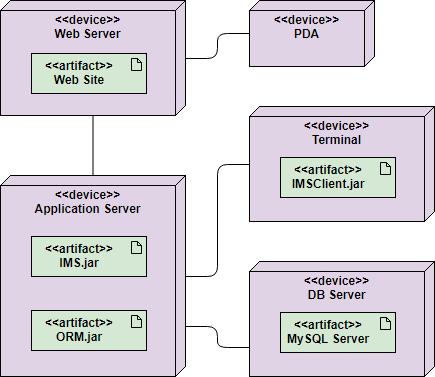
* **Representation:** A solid line connecting two nodes. It can be stereotyped to indicate the nature of the communication, such as <<TCP/IP>>, <<HTTP>>, <<wireless>>.
* **Definition:** Represents the physical or logical connections and communication protocols between nodes.

**4 Devices:** Client, Web Server, Application Server, Database Server and other devices.

Client







A diagram of a cloud service

AI-generated content may be incorrect.

