3 : Component Based Application Development

IT4206 – Enterprise Application Development

Level II - Semester 4

Overview

- After completing this section, students should be able to describe the basic concepts and benefits of enterprise applications.

Intended Learning Outcomes

- At the end of this lesson, you will be able to;
  - Learn about what are enterprise applications
  - Multi-tiered application architecture
  - Learn how to deploy an application

List of sub topics

3.1 Enterprise Applications
- 3.1.1 Introduction
- 3.1.2 Features of Enterprise Applications
- 3.1.3 Benefits of Java EE Enterprise Applications
- 3.1.4 Comparison of Java EE and Java SE
- 3.1.5 Java EE 7 Profiles
- 3.1.6 Building, Packaging and Deploying Java EE Applications

3.2 Multi-tiered application Architecture
- 3.2.1 Introduction
- 3.2.2 Types of Multi-tier Architecture

3.3 Deploying an Application

3.1 Enterprise Applications

- Software applications typically used in large business organizations.
- **Java Enterprise Edition** *(Java EE)* is a specification for developing enterprise applications using Java.
- It is a platform-independent standard that is developed under the guidance of the Java Community Process (JCP).

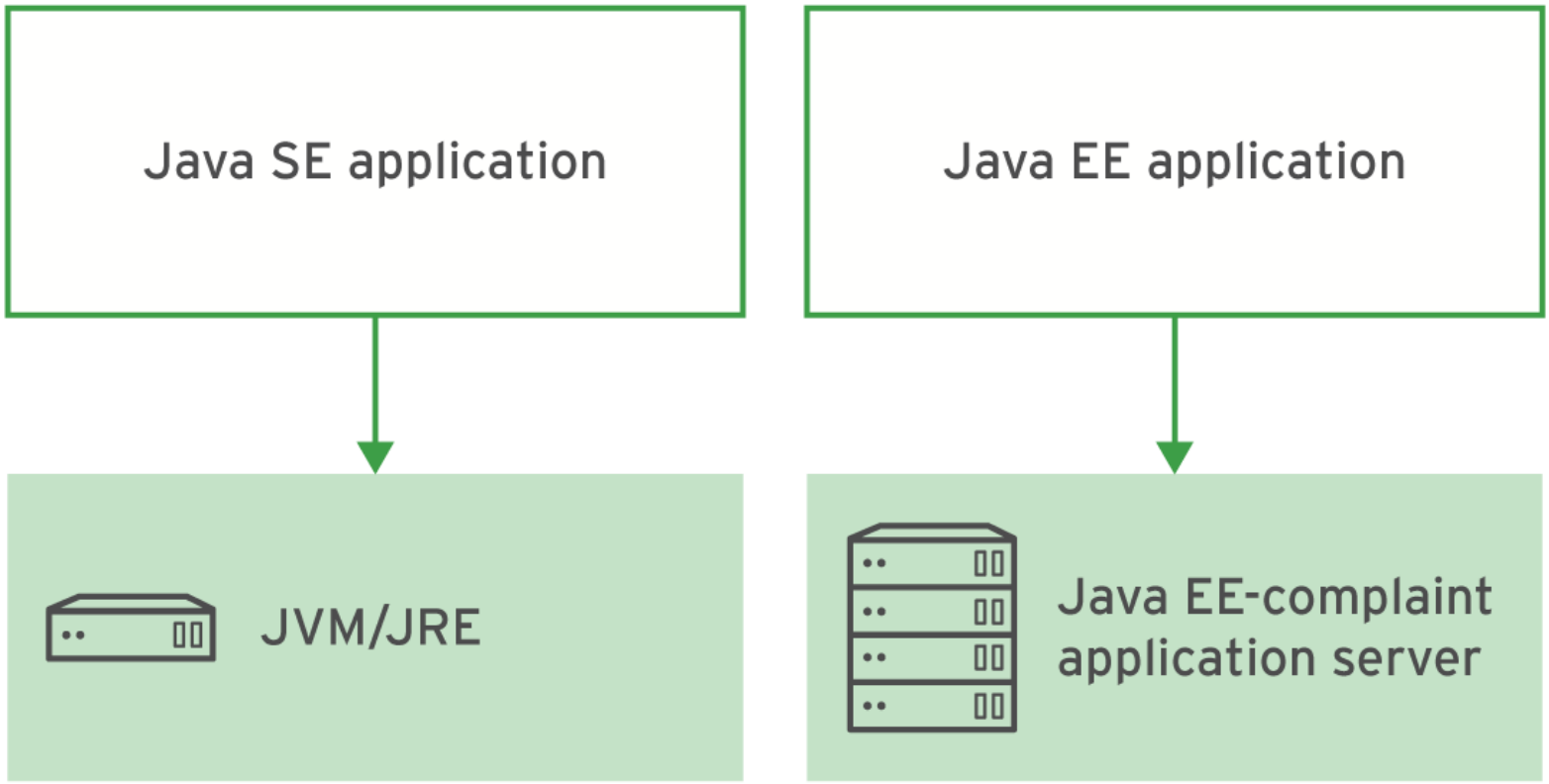
Features of Enterprise Applications

- Support for concurrent users and external systems.
- Support for synchronous and asynchronous communication using different protocols.
- Ability to handle transactional workloads.
- Support for scalability to handle future growth.
- A resilient and distributed platform to ensure high availability.
- Support for highly secure access control for different types of users.
- Ability to integrate with back-end systems and web services.

Benefits of Java EE Enterprise Applications

- Can be developed and will run on many different types of operating systems
- Applications are portable
- The Java EE specification provides a number of APIs
- A number of sophisticated tools such as IDEs, monitoring systems, enterprise application integration (EAI) frameworks, and performance measurement tools are available

Comparison of Java EE and Java SE



Comparison of Java EE and Java SE

Java SE	Java EE
• Java SE is generally used to develop stand-alone programs, tools, and utilities that are mainly run from the command line, GUI programs, and server processes.	• The Java EE specification is a set of APIs built on top of Java SE. It provides a runtime environment for running multi-threaded, transactional, secure and scalable enterprise applications.

Java EE 7 Profile

Java EE includes support for multiple *profiles*, or subsets of APIs. For example, the Java EE 7 specification defines two profiles: the **full profile** and the **web profile**.

- **Full Profile:** Contains all Java EE technologies, including all APIs in the web profile as well as others.
- **Web Profile:** Contains a full stack of Java EE APIs for developing dynamic web applications.

Building, Packaging and Deploying Java EE Applications

- Java EE applications can be packaged in different ways for deployment to a compliant application server.
- Depending on the application type and the components it contains, applications can be packaged into different deployment types
- The three most common deployment types are:
    - **JAR files**
    - **WAR files**
    - **EAR files**

Directory structure of Java EE application

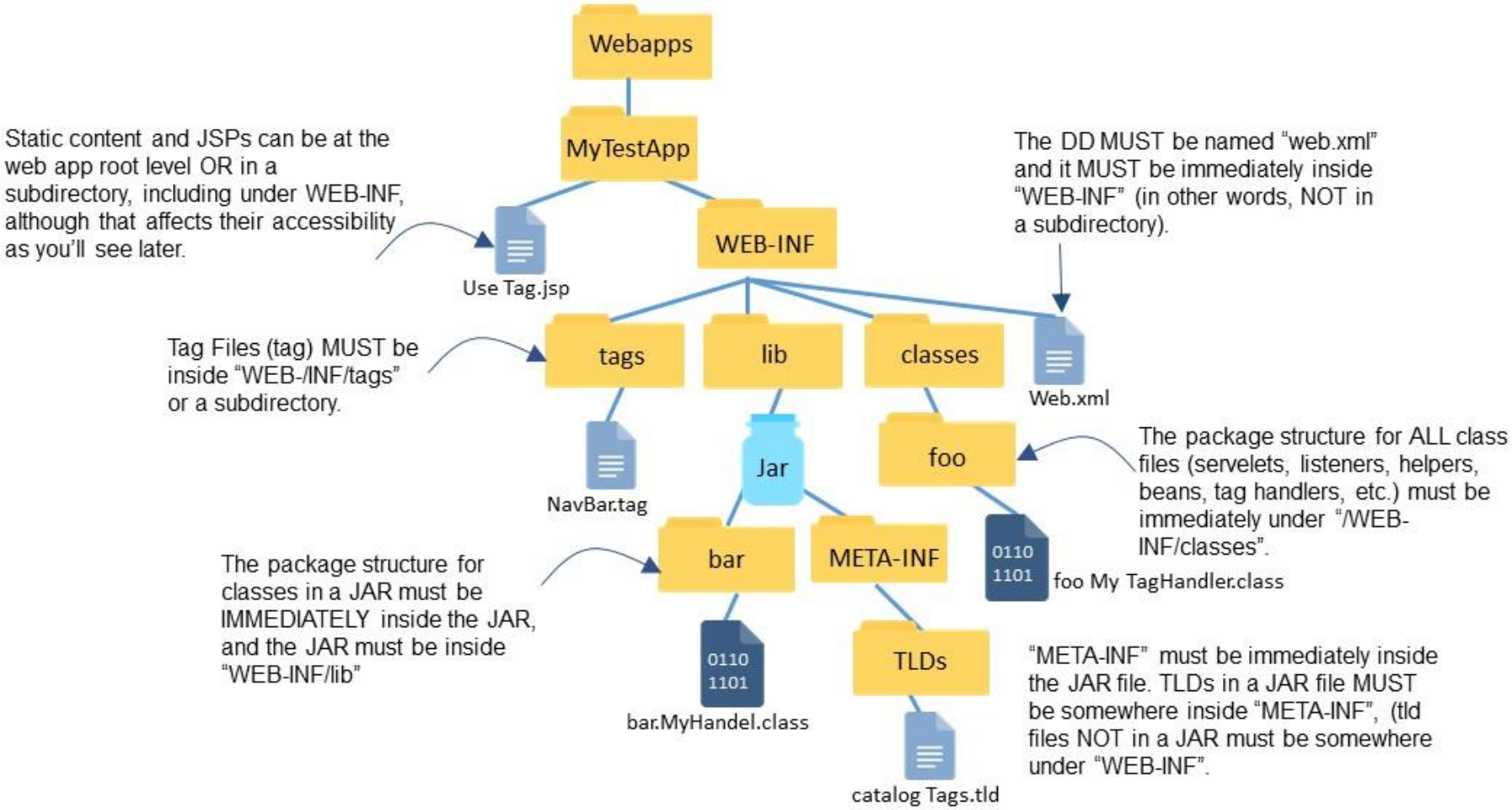
- Directory structure of a web application may contain;
  - Static content
  - JSP pages
  - Servlet classes
  - Deployment descriptor
  - Tag libraries
  - JAR files
  - Java class files

Directory Structure

To deploy a web app successfully you must follow this structure.

- WEB – INF must be immediately under the application context.
- The “classes” directory must be immediately inside “WEB - INF”.
- The package structure for the classes must be immediately inside “classes”.
- The “lib” directory must be immediately inside “WEB - INF” and the JAR file must be immediately inside “lib”.
- The “META - INF” directory must be immediately inside the JAR, and TLD files in a JAR must be somewhere under “META - INF”.
- TLDs that are not in JAR must be somewhere under “WEB - INF”.
- Tag files must be somewhere under “WEB – INF/tags”.

- In the next slide you can see the directory structure more in detail.



Java Archive (JAR) Files

- Individual modules of an application and Enterprise Java Beans (EJBs) can be deployed as separate JAR files.
- Third-party libraries and frameworks are also packaged as JAR files. If the application depends on these libraries, the library JAR files should be deployed on the application server.
- JAR files have a **.jar** extension.

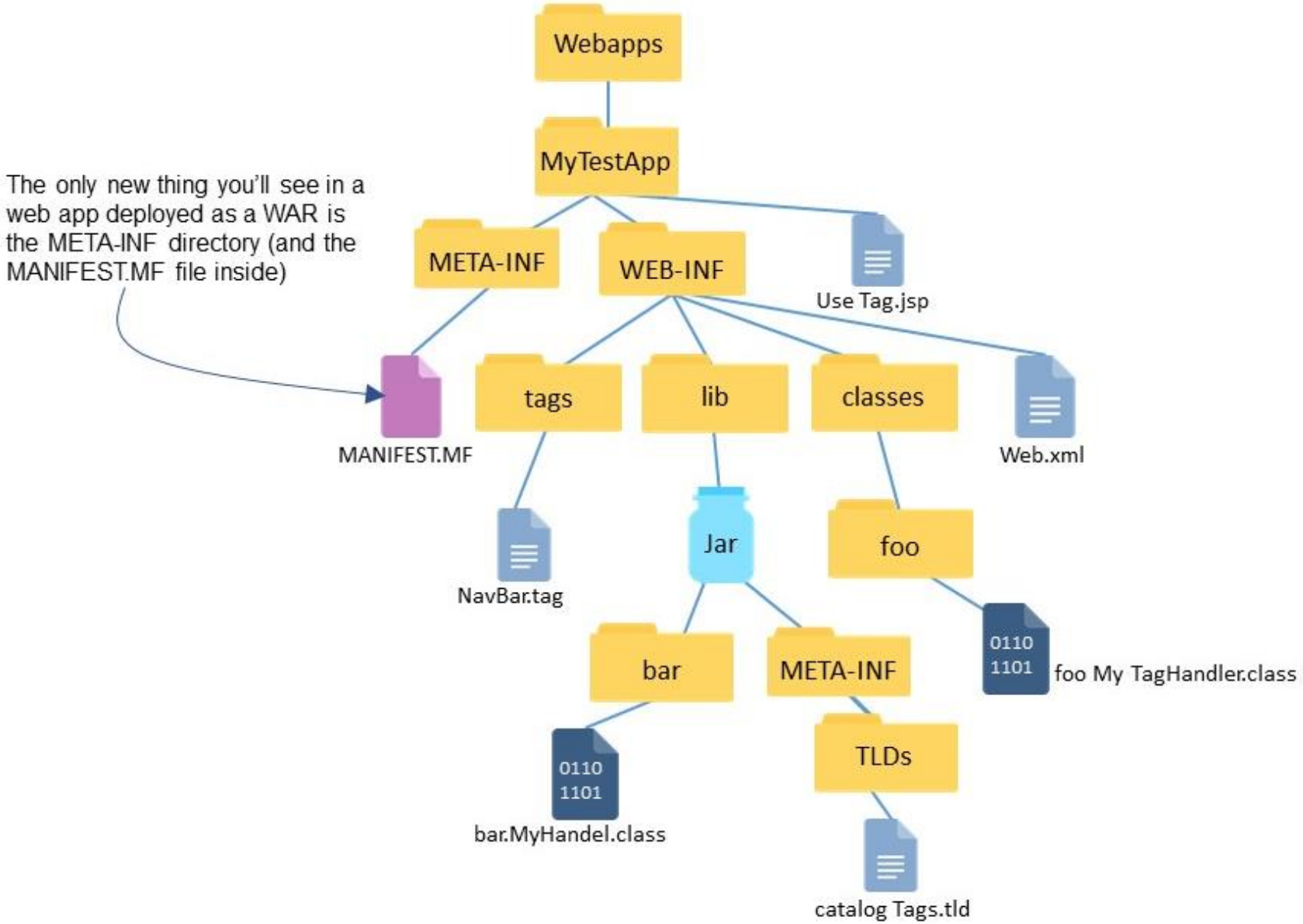
Enterprise Archive (EAR) Files

- An EAR file has an extension of **.ear** and is essentially a compressed file with one or more WAR or JAR files and some XML deployment descriptors inside it.
- It is useful in scenarios where the application contains multiple WAR files or reuses some common JAR files across modules. In such cases, it is easier to deploy and manage the application as a single deployable unit.

Web Archive (WAR) Files

- A WAR file is simply a snapshot of your web app structure in a nice portable, compressed form.
- If the Java EE application has a web-based front end or is providing RESTful service endpoints, then code and assets related to the web front end and the services can be packaged as a WAR file.
- A WAR file has a **.war** extension and is essentially a compressed file containing code, static HTML, images, CSS, and JS assets, as well as XML deployment descriptor files along with dependent JAR files packaged inside it.

What a deployed WAR File look like



Examples of Servlet mapping in the Deployment Descriptor

- Every servlet mapping has two parts; the <servlet> element and the <servlet-mapping> element.
- The <servlet> defines a servlet name and the class.
- The <servlet-mapping> defines the URL pattern that maps to a servlet name defined somewhere else in the Deployment Descriptor.

Example

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/webapp_2_4.xsd"
         version= "2.4">

      <servlet>

         <servlet-name>Car</servlet-name>

          <servlet-class> com.example.CarSelect </servlet-class>

      </servlet>

      <servlet-mapping>

         <servlet-name>Car</servlet-name>

          <url-pattern>/Car/SelectCar.do</url-pattern>

       </servlet-mapping>

</web-app>
```

Rules of servlet mappings

- It looks first for an exact match. If it can't find an exact match, it looks for a directory match. If it can't find a directory match, it looks for an extension match.
- An example is in the next slide.

Mappings:

```
  <servlet>
          <servlet-name>One</servlet-name>
          <servlet-class>foo.DeployTestOne</servlet-class>
      </servlet>
      <servlet-mapping>
          <servlet-name>One</servlet-name>
           <url-pattern>*.do</url-pattern>
       </servlet-mapping>
       <servlet>
          <servlet-name>Two</servlet-name>
           <servlet-class>foo.DeployTestTwo</servlet-class>
      </servlet>
      <servlet-mapping>
          <servlet-name>Two</servlet-name>
           <url-pattern>/fooStuff/bar</url-pattern>
       </servlet-mapping>
<servlet>
          <servlet-name>Three</servlet-name>
          <servlet-class>foo.DeployTestThree</servlet-class>
      </servlet>
      <servlet-mapping>
          <servlet-name>Three</servlet-name>
           <url-pattern>/fooStuff/*</url-pattern>
       </servlet-mapping>
```

Requests:

http://localhost:8080/MapTest/blue.do
Container choice: DeployTestOne
(matched the *.do *extension* pattern)

http://localhost:8080/MapTest/fooStuff/bar
Container choice: DeployTestTwo
(*exact* match with /fooStuff/bar pattern)

http://localhost:8080/MapTest/fooStuff/bar/blue.do
Container choice: DeployTestThree
(matched the /fooStuff/* *directory* pattern)

http://localhost:8080/MapTest/fooStuff/blue.do
Container choice: DeployTestThree
(matched /fooStuff/* *directory* pattern)

http://localhost:8080/MapTest/fred/blue.do
Container choice: DeployTestOne
(matched the *.do *extension* pattern)

http://localhost:8080/MapTest/fooStuff
Container choice: DeployTestThree
(matched the /fooStuff/* *directory* pattern)

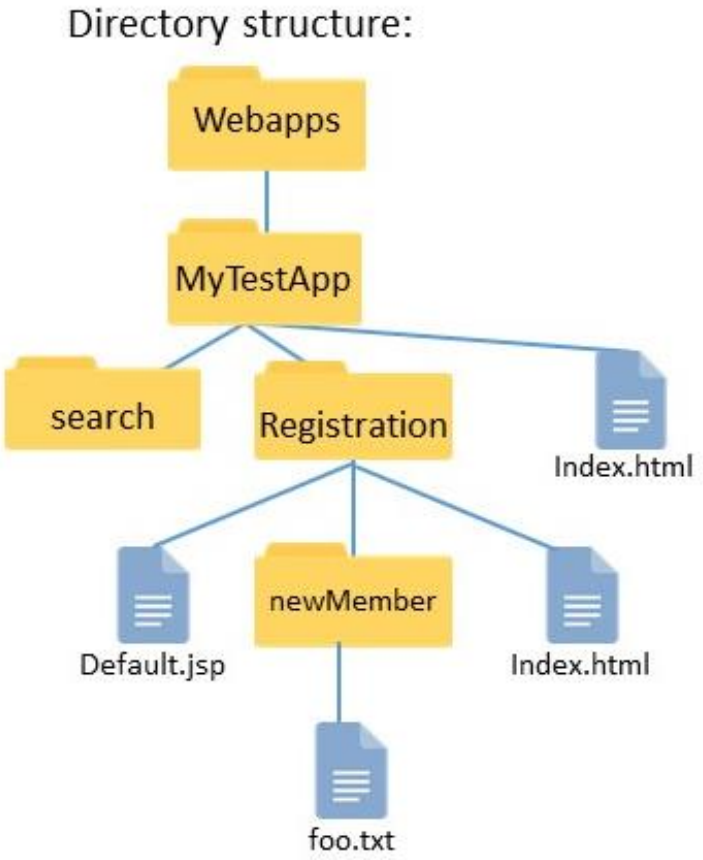
http://localhost:8080/MapTest/fooStuff/bar/foo.fo
Container choice: DeployTestThree
(matched the /fooStuff/* *directory* pattern)

http://localhost:8080/MapTest/fred/blue.fo
Container choice: 404 NOT FOUND
(doesn't match ANYTHING)

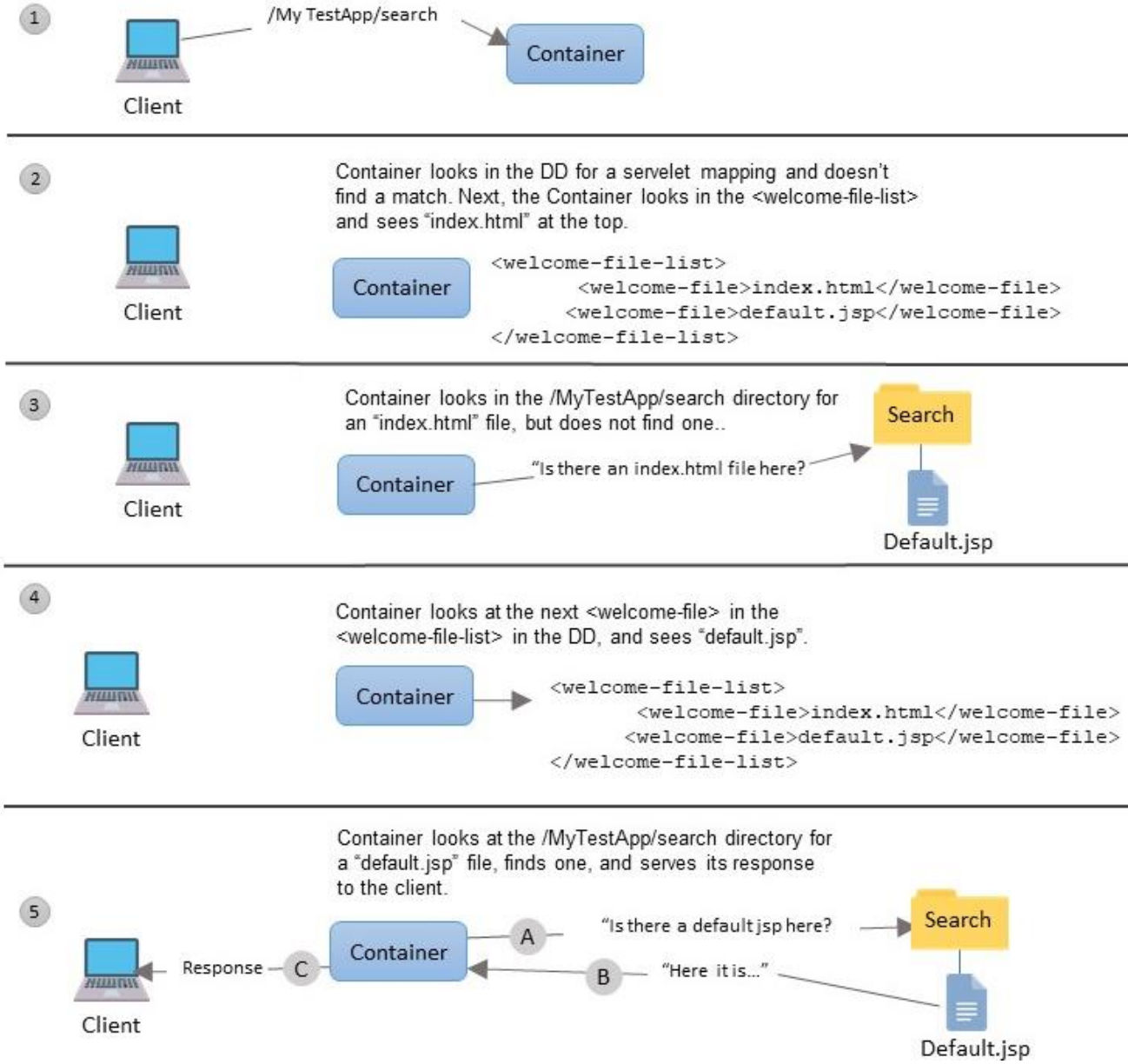
Configure welcome files in Deployment Descriptor

- The DD:

```
<welcome-file-list>
        <welcome-file>
                index.html
        </welcome-file>
        <welcome-file>
                default.jsp
        </welcome-file>
</welcome-file-list>
```



How the container chooses a welcome file



3.2 Multi-tiered application Architecture

- Java EE applications are designed with a multi-tier architecture in mind.
- The application is split into components, each serving a specific purpose.
- Each component is arranged logically in a tier. Some of the tiers run on separate physical machines or servers.
- The application's business logic can run on application servers hosted in one data center, while the actual data for the database can be stored on a separate server.

In a classic web-based Java EE application architecture, there are four tiers:

- ***Client Tier***: This is usually a browser for rendering the user interface on the end-user machines, or an applet embedded in a web page.

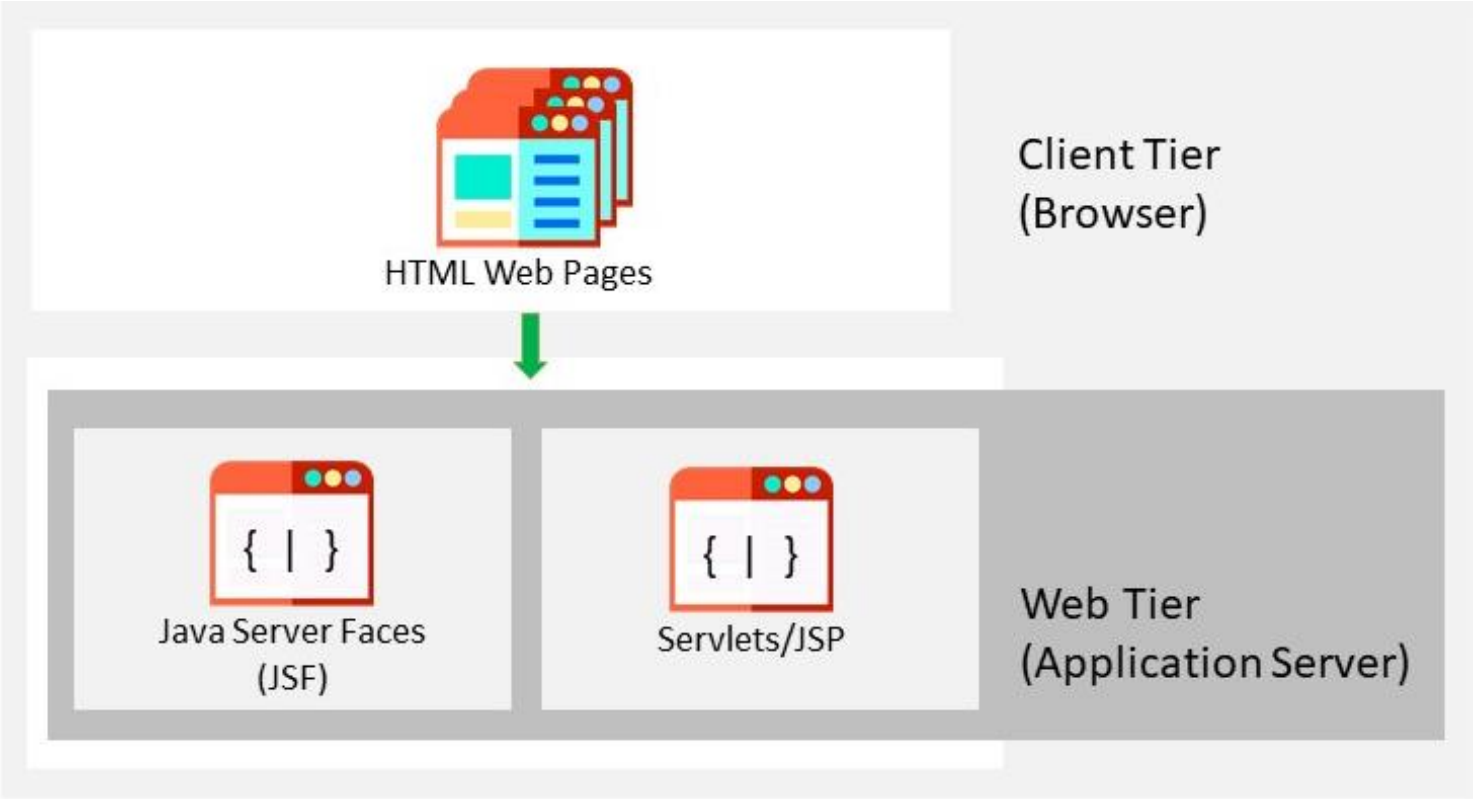
- ***Web Tier***: The web tier components run inside an application server and generate HTML or other markup that can be rendered or consumed by components in the client tier. This tier can also serve non-interactive clients such as other enterprise systems via protocols such as **Simple Object Access Protocol (SOAP)** or **Representational State Transfer (REST)** web services.

- ***Business Logic Tier***: The components in the business logic tier contain the core business logic for the application. These are usually a mix of Enterprise Java Beans (EJB), Plain Old Java Objects(POJO), Entity Beans, Message Driven Beans, and Data Access Objects (DAO), which interface with persistent storage systems such as RDBMS, LDAP, and others.

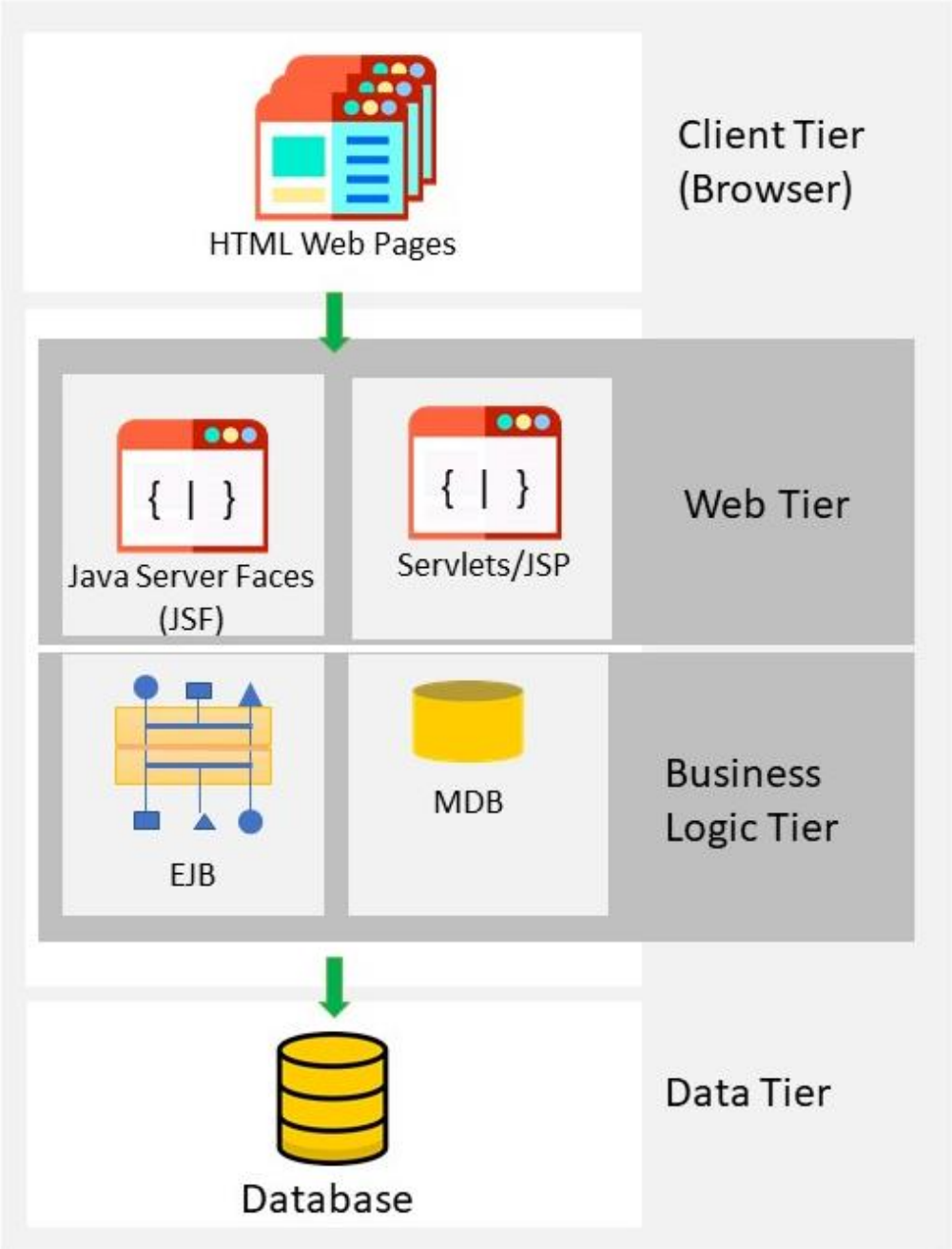
- ***Enterprise Information Systems (EIS) Tier***: Many enterprise applications store and manipulate persistent data that is consumed by multiple systems and applications within an organization. Examples are relational database management systems (RDBMS), Lightweight Directory Access Protocol (LDAP) directory services, NoSQL databases, in-memory databases, mainframes, or other back-end systems that store and manage an organization's data securely.

Types of Multi Tier Architecture

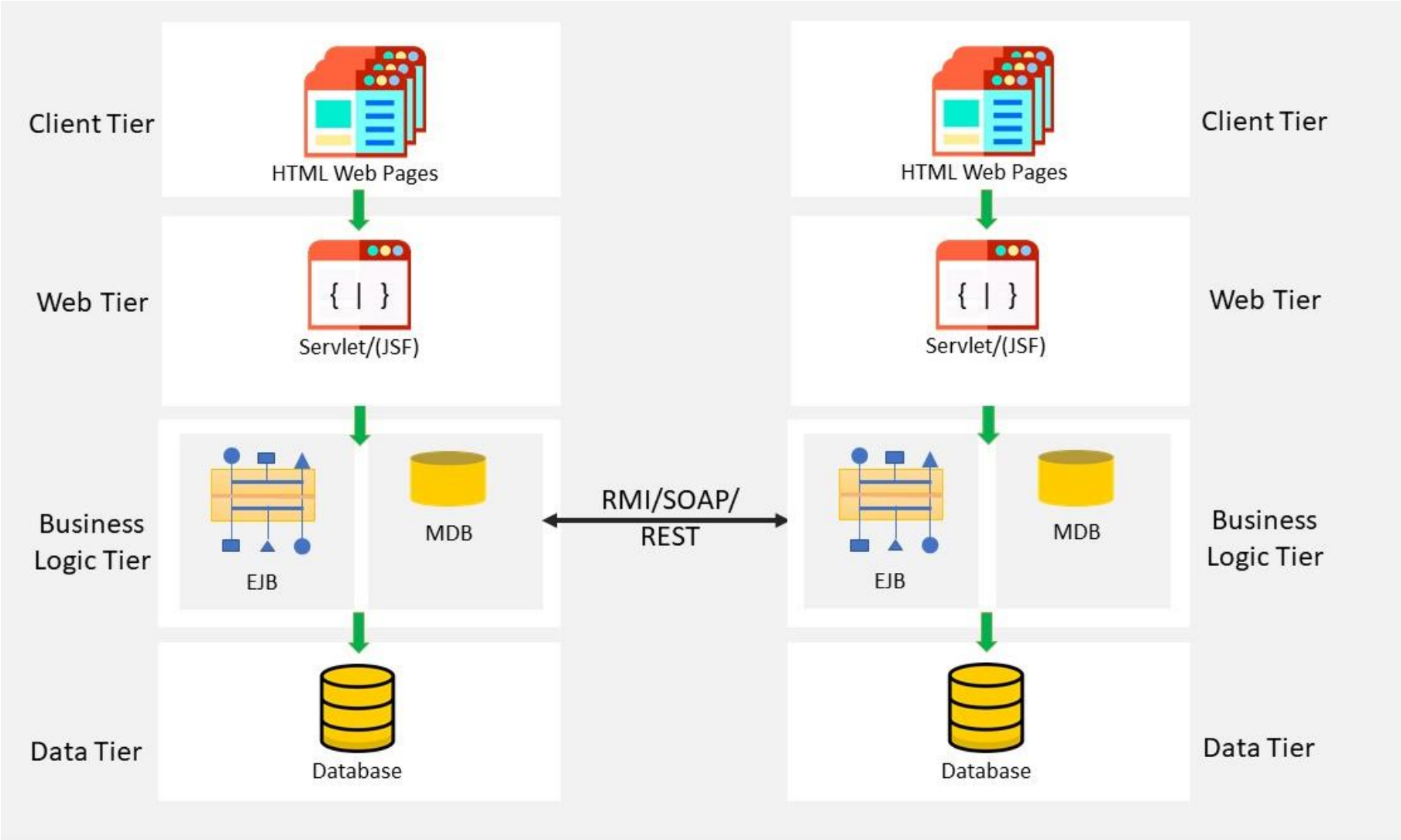
- **Web-centric architecture**



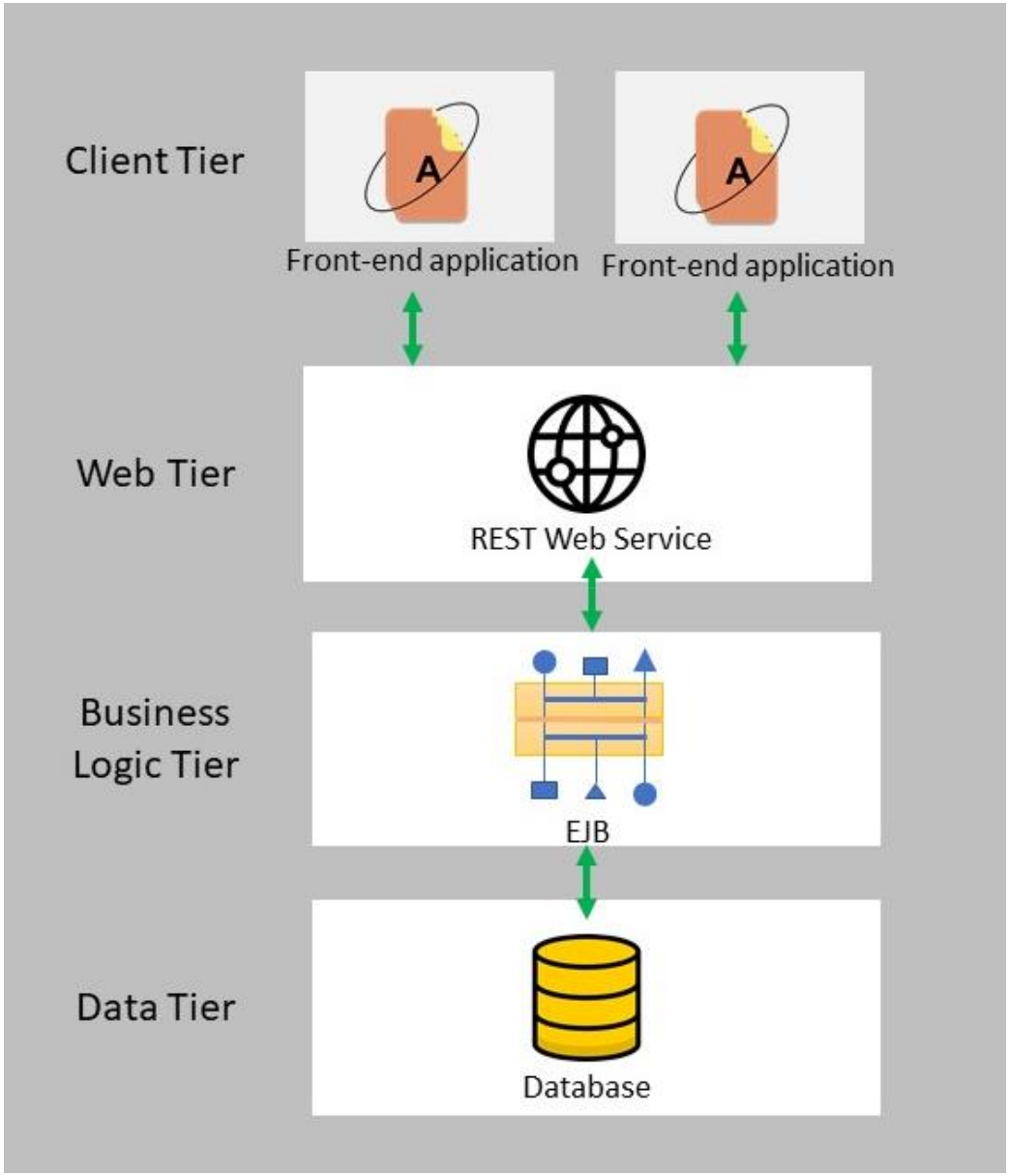
- **Combined web and business logic component-based architecture**



- **Business-to-Business architecture (B2B)**



- **Web service application architecture**



3.3 Deploying an Application

- The current best practice for developing, testing, building, packaging, and deploying Java SE and Java EE applications is to use *Apache Maven*.
- Maven is a project management tool that uses a declarative approach to specify how to build, package, execute, and deploy applications together with dependency information.

Maven has a small core and has a large number of plug-ins that extend the core functionality to provide features such as:

- Predefined build life cycles for end products, called artifacts, like WAR, EAR, and JAR.
- Built-in best practices such as source file locations and running unit tests for each build.
- Dependency management with automatic downloading of missing dependencies.
- Extensive plug-in collection including plug-ins specific to JBoss development and deployment.
- Project report generation including Javadocs, test coverage, and many more.