Java EE Fundamentals & Structure of a Web Application

GÖTEBORGS UNIVERSITET

DAT076/DIT126 Web Applications

Adam Waldenberg

Jakarta / Java EE in a nutshell

A set of specifications for extending Java with features for enterprise software

- Enterprise software is computer software used to satisfy the needs of an organization rather than individual users
- Vague concept. The exact definition depends on the context
- In the case of Java Enterprise we talk about the underlying system *providing a tool set and an API for fulfilling the needs of an enterprise* (large business)
- Consequently scalability, maintainability, security and reliability are very important factors

Quotes about enterprise software

- *“Software products designed to integrate computer systems that run all phases of a businesses' operations to increase internal coordination of work and cooperation across an enterprise”* (p5ee, 2005)

- *“Enterprise applications are about the display, manipulation, and storage of large amounts of often complex data and the support or automation of business processes with that data.”* - Martin Fowler

- *“Software whose failure everyone notices quickly.”* - Brian D. Foy

Some Features of Jakarta / Java EE

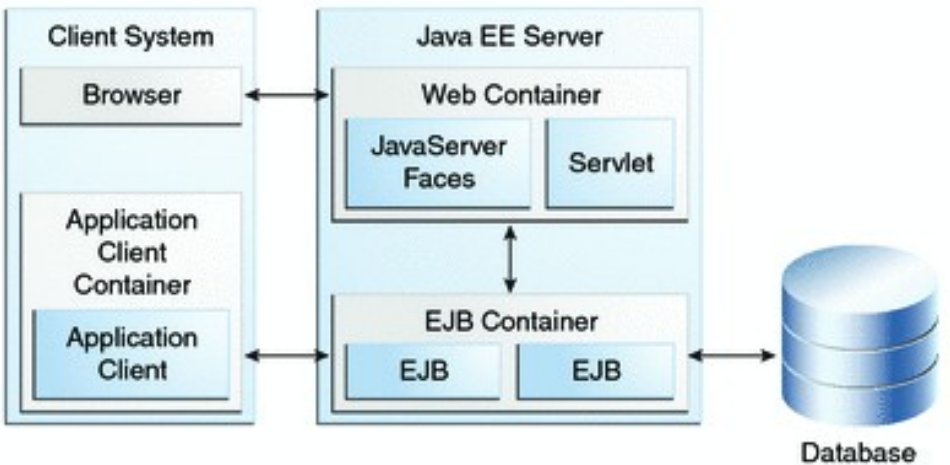
- Java Persistence API
- Java Transaction API
- Enterprise Java Beans (EJB)
- Contexts and Dependency Injection (CDI)
- Java Server Faces (JSF) and Facelets
- Java API for Restful Web Services (JAX-RS)
- Java Security API

… and much, much more ...

Anatomy of an Application Server

- **Servlets** Handling the parsing of incoming requests and the sending of responses

- **Enterprise Java Beans (EJB)** Handling the business logic and micro services of the application

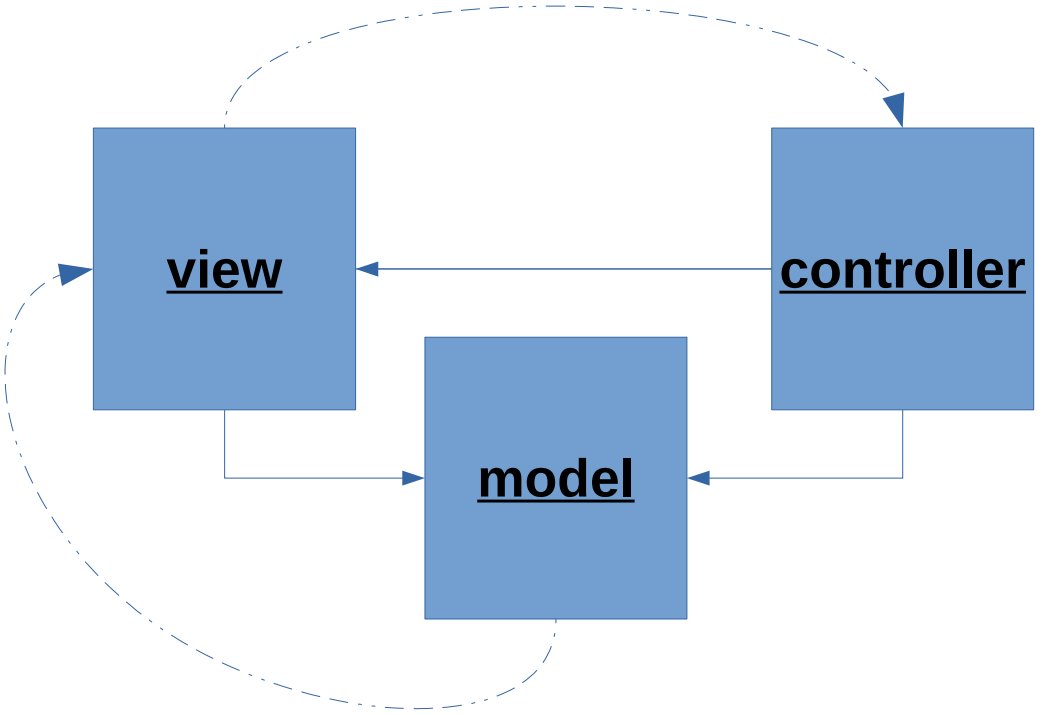


Servlets

- Servlets are an integral part of Java EE. However, most of the time we don’t use them when developing a modern Java Server Faces application.
- Java Server Faces is implemented via a servlet called *FacesServlet*
- Sometimes used for specific tasks. Some examples;
  - A servlet that, given a name, fetches and returns an image from a database
  - A servlet that, given a string, generates and returns a bar code image
  - A servlet that, given a social security number, returns a name
- These days, web services replace most of these use cases – thus, other than making you aware of them, we won’t be covering servlets at all

Model-View-Controller

- The model contains the logic and mental representation
- The view is the visual output and representation
- The controller handles input from the user and the view
- The controller updates the model
- The view shows the new state of the model



MVC, MVP, MVVM and the list goes on...

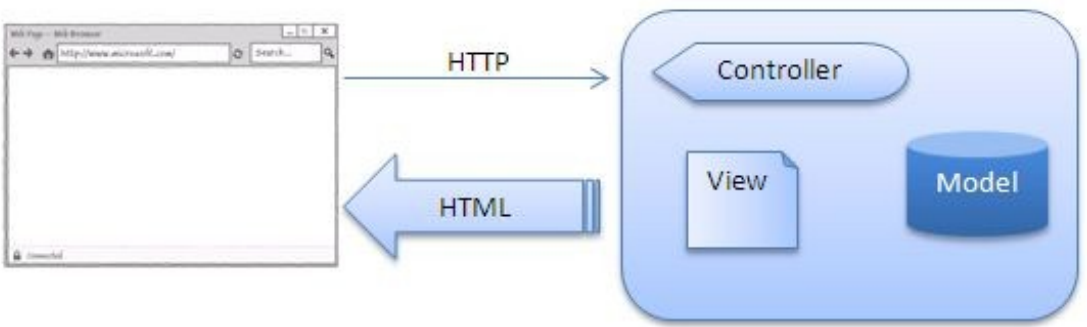
Many variations of MVC have been defined, slightly *tweaking* how the core idea is defined

- For example – JSF requires a backing bean as a placeholder for view values. Does this make it a MVCVM framework (can we make our own definition)?
- The only way to know the exact meaning is to locate the original source of each definition. Everybody on the Internet (including in articles and books) seem to have their own idea
- Many different interpretations and variations of each exist
- In the end, ***it's all about separating the view and input handling from the model and business logic***

Server-side MVC

The whole application and state lives on the server-side

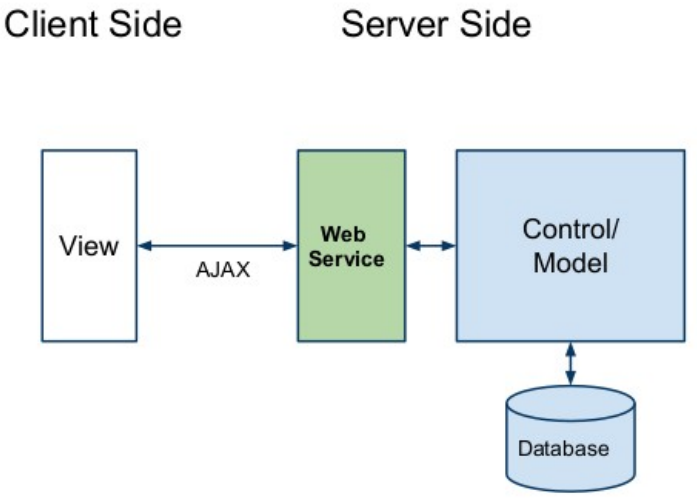
- Browser is mostly a thin client. Modern JSF component frameworks do however have some model manipulation and state stored on the client-side because of the *richness* of the component frameworks
- At some point this is pushed up to the server-side to persist the state
- Some examples include Java Server Faces, React and Vue.js



MVC - Client-side view

No view representation on the server side

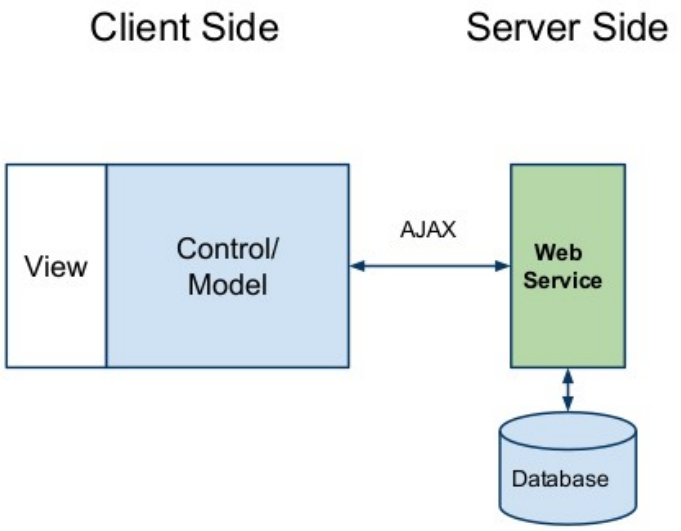
- The view representation is stored completely on the client side
- A web service with endpoints separates the view from the rest of the application logic
- Examples include React and Vue.js



Client-side MVC

The whole application lives on the client side

- Calculations and state updates originate and are performed on the client-side
- The server-side is used for security, validation and persistence
- Examples include Angular and Ember



Pros and cons of the different approaches

Server-side MVC

- Consider the advantages of server-side MVC
  - **Less reliance on JavaScript**
  - Cleaner implementation, allowing you to implement almost all you code on the server-side
  - Tightened security and control
- Consider the disadvantages of server-side MVC
  - **Bigger load on the server**

Pros and cons of the different approaches

Client-side MVC

- Consider the advantages of client-side MVC
  - **Less load on the server**
  - For many (most?), a simpler implementation
- Consider the disadvantages of client-side MVC
  - **More reliance on JavaScript**
  - Security concerns with business logic on the client-side

A final word on the different approaches

- Each approach has advantages and disadvantages
- You don’t have to strictly stick to one. It’s OK to use a mixture to offset the advantages and disadvantages of each approach
- Even with Java Server Faces you can potentially get the best of both worlds by using a component framework like *AngularFaces* (*angularfaces.net*)

Sessions

- As we previously covered, HTTP is a stateless protocol. What are sessions?
- Defining a session on the server-side is one way of keeping track of a client between HTTP requests
- The server can track session via cookies, URL rewriting or hidden form fields. This is used to identify a single client across different requests
- Session tracking using cookies is the primary mechanism. JavaEE containers can fall back to one of the other methods if the browser does not support cookies or has cookies disabled
- To access the and store data into the session we can use the CDI session scope to tell the container to store the data in the session

Controlling the session in Java EE

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app ...>
    ...
    <session-config>
       <session-timeout>
           10
       </session-timeout>
    </session-config>
</web-app>
```

```
HttpSession session = request.getSession();
session.setMaxInactiveInterval(10 * 60);
```

- In Java EE, we can control the session timeout by configuring editing the *web.xml* file
- Payara supports persistence types such as *memory*, *file* or *hazelcast*
- We can also programmatically modify the session

Resource and dependency injection

- Dependency injection allows us to turn regular classes into managed objects and to inject those managed objects into other managed objects
- Application or the framework has to ensure we it’s providing the correct managed object at the right time
- Makes it easier to mock and test objects
- In some frameworks, parameters sent during instance creation can be defined in seperate configuration files
- In Java EE the framework is called *Contexts and Dependency Injection (CDI)*

Dependency injection and instance management

Utilizing dependency injection is a complete shift in the way you program in Java

- You generally never instantiate instance variables yourself
- You allow the container of the application server to completely control the life cycle of your instances
- Makes it very important that you annotate your classes with the correct *scope*
- Different scopes control how long a specific object instance is kept active and under which conditions it’s destroyed

Dependency injection scopes

Many scopes exist in Java EE, some are backend-specific, while others are tightly bound to the view model

- *@ApplicationScoped* Active during the life cycle of the application
- *@SessionScoped* Active during the life cycle of the session
- @RequestScoped Active during the life cycle of a a servlet request
- *@ViewScoped* Active during the life cycle of the view
- *@ConversationScoped* Manually controlled scope
- *@Clustered @ApplicationScoped*

Apache DeltaSpike extension scopes

DeltaSpike (*deltaspike.apache.org*) consists of a number of portable CDI extensions that provide useful features for Java application developers

- *@WindowScoped* — Active during the life cycle of a tab
- *@GroupedConversationScoped* — Allows you to sequence and combine conversations in one JSF view

Injections and interceptors (Example)

```
public class Email { … }

@ApplicationScoped
public class ApplicationBean {
    @Inject
    private Email email;

    @PostConstruct
    private void init() {
    }
}
```

What is actually happening here?

- An instance is injected and instantiated by the container when you use the ***@Inject*** annotation

- ***@PostConstruct*** and ***@PreDestroy*** are interception annotations to do initializations and de-initializations

Factory classes and CDI

- When creating certain objects, we might need to initialize them in different ways before any injection
- ***@Produces*** allows us to implement factory classes whose responsibility is the creation of fully-initialized services
- Can be defined together with a CDI scope

```
public class LoggerFactory {
    @Produces
    public Logger createLogger(InjectionPoint ip) {
        return Logger.getLogger(
            ip.getMember().getDeclaringClass().getName()
        );
    }
}
```

Factory classes – an example

```
@Qualifier
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD, ElementType.FIELD})
public @interface Global { }

public class LoggerFactory {
    @Produces
    public Logger createLogger(InjectionPoint ip) {
        return Logger.getLogger(
            ip.getMember().getDeclaringClass().getName()
        );
    }

    @Produces @Global
    public Logger createGlobalLogger(InjectionPoint ip) {
        return Logger.getLogger(Logger.GLOBAL_LOGGER_NAME);
    }
}
```

What is happening here?

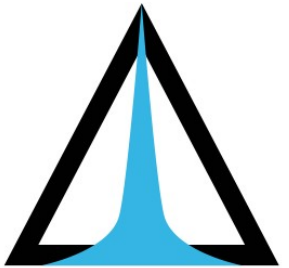
Factory classes and CDI continued

- *@Produces* can be used with *CDI qualifiers* to modify the behavior
- Qualifiers can even have parameters/arguments
- What if we want to inject a ***@Global*** Logger? How would we do that?

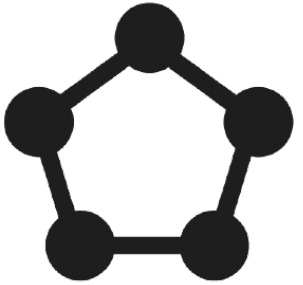
```
@Inject @Global
private Logger logger;
```

OmniFaces & Apache DeltaSpike

- DeltaSpike consists of a number of portable CDI extensions that provide useful features for Java application developers
- OmniFaces aim is to make JSF life easier by providing a set of artifacts meant to improve the functionality of the JSF framework
- OmniFaces was created in response to seeing the same questions and the same example and utility code posted over and over again



DELTASPIKE



OmniFaces

Enterprise Java Beans

A server-side software component that encapsulates business logic of an application

- Allows for the creation of micro-services and modules with a very specific target functionality
- Supports and grants transaction control
- Integrates with JPA (Java Persistence API)
- Concurrency control *(@Lock(LockType))*
- Asynchronous method invocations *(@Asynchronous)*
- Job scheduling and timers *(@Schedule and @Timeout)*

Enterprise Java Beans continued

Support for *@Stateless* and *@Stateful* EJB beans

- The server can maintain a variable amount of stateless session bean instances in a pool. Each time a client requests such a stateless bean a random instance is chosen to serve that request
- A stateful session bean is closely connected to the client. Each instance is created and bound to a single client and serves only requests from that particular client. Similar to a ***@SessionScoped*** CDI bean
- Service layers that are a collection of enterprise beans can be packaged in an EJB package an distributed as a collection to a web application (war) or an enterprise archive (ear)

Enterprise Java Beans usage

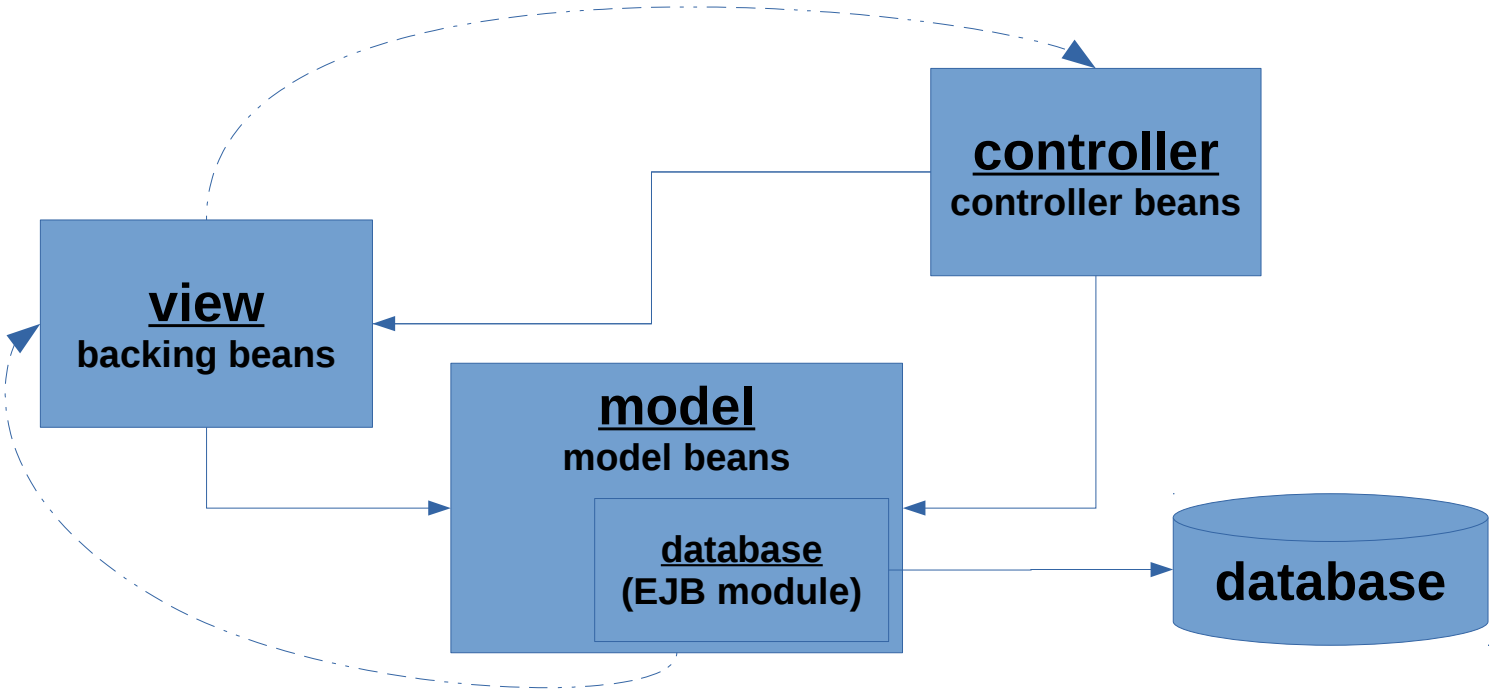
- Usage of Enterprise Java Beans is similar to CDI beans. However, instead of using ***@Inject***, we inject them with ***@EJB***

```
@Stateless
public class UserService {
    public String getPrimaryEmail(User user) {
        return user.getEmails().first();
    }
}
```

```
@EJB
private UserService userService;
```

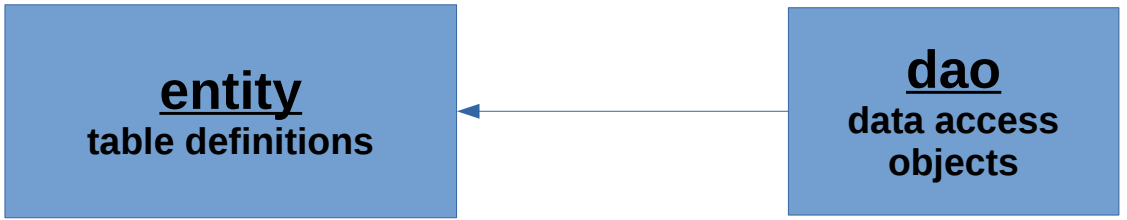
An example layout of a EE application

Using an EJB module, a simplest EE application could look something like this



What about the design of the database module ?

The database service is extremely simple



- Data access objects have all the connections to the underlying database and typically connect to the Java Persistence API
- Entities just describe the tables and their relationships

The file structure of an EE application

When developing a EE application, you typically follow a strict file structure enforced upon you by Maven and the requirements from the EE standard

- webapp/ — All the main files on the client side of the application
- webapp/WEB-INF/ — Application server configuration files and hidden HTML files/resources loaded by Facelets
- resources/META-INF/ — Configuration files related to persistence and EJB
- setup/ — Actual setup files for the application server