Sehr geehrte Professoren, liebe Mitschüler ich darf Sie heute recht herzlich zu meinem Referat über Java Webtechnologien begrüßen. Kurz zur Agenda ich werde ihnen Heute kurz etwas über die verschiedenen Ansätze der Java Webtechnologien erzählen und anschließend werde ich Ihnen die Technologien JSF und Quarkus Qute näherbringen.

Ansätze bei Java Webtechnologien

Fangen wir also direkt mit den verschiedenen Ansätzen der Java Webtechnologien an. Grundsätzlich gibt 2 Ansätze, zum einen den komponentenbasieren Ansatz und zum anderen den aktionsbasierten Ansatz.

Komponentenbasierter Ansatz

Der komponentenbasierte Ansatz ist eher schwergewichtig und bringt viel mit sich. Er wird vor allem repräsentiert durch JSF oder ASP.NET (Vaadin). Wie der Name schon sagt lebt er davon, dass die Web UI aus Komponenten zusammengebaut wird. Diese können dann klarerweise öfter in der Applikation verwendet werden. Wichtig ist es jedoch zu verstehen, dass diese Komponenten Serverseitig liegen. Das heißt zur Laufzeit liegt die gesamte Logik nicht am Client, sondern am Server. Als Nachteil muss jedoch so bei jeder Benutzerinteraktion der Server benachrichtigt werden. Ein Vorteil dieser Frameworks ist es, dass man Boilerplate-Code (Code der mit minimalen Änderungen immer wieder einfügen) minimiert und man sich mehr wirklich mit Webtechnologien auseinandersetzen muss, weil die Frameworks die gesamte HTTP-Kommunikation als auch die Generierung von JavaScript, HTML und CSS übernehmen. Das heißt man schreibt lediglich beispielsweise in JSF einen XHTML Code mit Java und der Code wird von dem Framework in Webtechnologien übersetzt und man braucht sich um nichts kümmern.

Ein Nachteil des komponentenbasierten Ansatzes ist es, dass neue Technologien immer erst später in die Frameworks eingebaut werden da dies klarerweise Zeit benötigt (Websockets ab V 2.3).

Empfohlen wird vor allem komponentenbasierte Frameworks zu verwenden, wenn man schnell einen Prototyp bauen möchte, und in der UI viele Validierungen benötigt werden.

Aktionsbasierter Ansatz

Der aktionsbasierte Ansatz wird repräsentiert durch Spring MVC, Java EE MVC, Struts oder auch ASP.NET MVC. Er ist vor allem leichtgewichtiger und man programmiert näher am Web.

Bei dem aktionsbasierten Ansatz will man im Gegensatz zu dem komponentenbasierten Ansatz nicht das Web abstrahieren, sondern die klassische Webprogrammierung anzuwenden. Das heißt man ist wieder für die HTTP-Kommunikation verantwortlich und man muss die UI wieder selbst mit Webtechnologien erstellen. Dadurch sind klarerweise die Frameworks um einiges leichtgewichtiger, weil es weniger zur Verfügung stellt.

Vorteile hier liegen vor allem in der enormen Flexibilität, da man sofort neue Technologien einsetzen kann, wenn diese auf den Markt kommen und man nicht warten muss, bis diese in das Framework eingebaut werden.

Nachteile sind vor allem, dass im Vergleich zum komponentenbasierten Frameworks viel mehr Aufwand betrieben werden muss. Außerdem muss man für die UI Webtechnologien beherrschen, da man diese selbst schreiben muss.

JSF – Java Server Faces

Ich selbst habe mich mit JSF oder auch Java Server Faces auf den komponentenbasierten Ansatz spezialisiert. JSF ist ein Framework-Standard zur Entwicklung von grafischen Benutzeroberflächen für Webapplikationen. JSF gehört zu den Webtechnologien von Java EE. Mit Hilfe von JSF kann der Entwickler auf einfache Art und Weise Webseiten erstellen und muss sich dabei nicht um eine Kommunikation zwischen Server und Client kümmern. Außerdem schreibt man die View mit .xhtml Dateien, von denen aus man quasi auf den Java Code zugreifen kann.

JSF wird vor allem im Umgang mit HTML-Formularen zur Dateneingabe genutzt, da die JSF API eine serverseitige Validierung zur Verfügung stellt. Zusammengefasst lässt sich sagen, dass sich JSF für die Synchronisation der Daten zwischen Front- und Backend kümmert.

MVC

JSF baut grundsätzlich auf dem MVC Pattern auf. Das Model-View-Controller Pattern schlägt drei Hauptkomponenten vor, die Daten, Ansicht und die Logik trennen. Zum einen hätten wir

- Models,
  die die zugrunde liegende logische Struktur von Daten in einer Anwendung repräsentieren. Models enthalten keinerlei Informationen über die Benutzeroberfläche.
- Views
  die die Elemente in der Benutzeroberfläche darstellen also alle Dinge, die Benutzer sieht wie Buttons, Tabellen usw. Die Präsentation ist für die Darstellung der benötigten Daten aus dem Modell und die Entgegennahme von Benutzerinteraktionen zuständig. In JSF entspricht die View den Facelets, welche XHTML Dateien sind.
- Und zu guter Letzt den Controller
  der das Modell und die View verbindet. Der Controller nimmt die Benutzeraktionen der View entgegen, und reagiert entsprechend. Der Controller wird in JSF durch Faces-Servlets dargestellt und ist in das Framework eingebettet.

Ziel des Musters ist ein flexibler Programmentwurf, der eine spätere Änderung oder Erweiterung erleichtert und eine Wiederverwendbarkeit der einzelnen Komponenten möglich macht.

Wie schaut dieses Pattern jetzt in JSF aus?

- Die Modell-Schicht umfasst in JSF nicht nur die Entitätsklassen sondern auch die Managed Beans, die quasi die Code-Behind Datei sind, von der man in der View aus zugreifen kann und Methoden aufrufen kann.
- Die View Schicht wird durch .xhtml (Faclets) Dateien realisiert, wo man verschiedene Komponenten einbinden kann wie beispielsweise von Primefaces, um die UI schöner zu gestalten.
- Der Controller (FacesServlet) wird in JSF bereits von dem Framework zur Verfügung gestellt und fälschlicherweise wird dieser oft mit den Managed Beans also den Code-Behind Dateien verwechselt.

Lebenszyklus

Jetzt komme ich noch zu dem Lebenszyklus eines Requests in JSF, dabei werde ich mich jedoch relativ kurzhalten, da er komplett von JSF gemanagt wird und man ihn zum Erstellen von kleinen Webanwendungen nicht ins genaueste Detail verstehen muss. Dieser Lebenszyklus beinhaltet alle Phasen, die bei einem Request durchlaufen werden. Zu Beginn wir von dem Client eine Request geschickt und am Ende sollte wieder durch die Response eine fertige HTML-Seite zurückgeschickt werden. Die 1. Phase ist die

1. Restore View Phase
   In dieser Phase wird die View aufgebaut und Ereignishändler, Validatoren und Converter mit der View verknüpft.
2. Apply Request Values
   und hier werden Benutzereingaben werden aus der http-Anfrage ausgelesen und in die entsprechenden Variablen geschrieben
3. Process Validations
   Hier werden alle registrierten Validierungen aufgerufen und so werden die neuen Daten der Benutzereingaben validiert. Falls hier ein Error aufkommt werden alle nachfolgenden Phasen bis zur Render Response Phase übersprungen.
4. Update Model Values
   Hier werden die Benutzereingaben in das entsprechende Model übertragen. Hier kommen Converter zum Einsatz. Tritt ein Fehler auf so werden wieder alle nachfolgenden Phasen bis zur Render Response Phase übersprungen.
5. Invoke Application
   Aufruf zusätzlicher Methoden, wie z.B. die OnClickListener (Button klick Methode) Methoden oder Navigierungsmethoden.
6. Render Response
   In der Render Response Phase wird die HTML-Seite generiert. Falls bei den anderen Phasen ein Error aufgetreten ist, so wird die Seite zurückgegeben, die beim Request hereingekommen ist.

Viele Entwickler beschwerten sich, dass dieser Lebenszyklus sehr starr ist und man nicht eingreifen kann und deshalb wurden sogenannte Events zwischen den Phasen eingeführt. Diese Events können genutzt werden, um den eigenen Code zwischen den Phasen laufen zu lassen. Dafür muss man sich in der View auf das jeweilige Event registrieren. Ich habe das jedoch bei meinem Beispiel nicht benötigt.

Beispiel

Jetzt möchte ich auch schon zu meinem Beispiel zu JSF kommen.

[HERZEIGEN DER UI]. VALIDATION, PREMIUM ACCOUNT UND NORMAL

Grundsätzlich lauft das Projekt auf einem WIldfly und dahinter ist eine Derby Datenbank.

[POM.XML] Zum Erstellen des Projekts braucht man grundsätzlich nur die javaee-api dependency einbinden und aufgrund dessen, dass ich noch Komponenten von PrimeFaces benutzt habe um eine schönere UI erstellen zu können habe ich ebenfalls dependencies von Primefaces.

[PERSISTENCE.XML] Dann gibt es noch die Persistence-xml zum Persistieren der Daten das sollten wir jedoch alle kennen.

[WEB.XML] Dann gibt es noch das Web.xml File. Hier werden jsf spezifische Einstellungen wie das Primefaces Theme oder die Startseite und Userrollen sowie deren Berechtigungen definiert.

[TODO]Unter dem Model Package haben wir unsere Entität. Im Prinzip ist es eine normale Entitätsklasse wie wir sie von JPA kennen. Um die Validierungen, die wir vorher gesehen haben zu gewährleisten, muss man Annotationen wie NotEmpty oder NotNull über die jeweiligen Fields schreiben.

[INIT BEAN] In der Init bean fügen wir einfach unsere Daten ein die beim Start der Applikation in der Datenbank sein sollen.

[TODO BEAN] Und im Java Ordner haben wir unsere ganzen Beans die für die Logik verantwortlich sind. Diese müssen mit @Named und @ViewScoped annotiert werden.

[NEWTODOCOMPONENT] Unter dem Ordner Webapp haben wir dann die ganzen Views bzw. Komponenten. Und hier können wir jetzt auf unsere Beans zugreifen. Hier ist es wichtig am Beginn des Files die Namespaces zu definieren sprich, dass der p – namespace auf Primefaces verweist und so können die Primefaces Komponenten verwendet werden. Hier sieht man auch schon wie einfach die Navigation in JSF funktioniert. Man muss lediglich in der action die View eintragen, die als nächstes angezeigt werden sollte.

[REPOSITORY] Das Repository kümmert sich dann eben um die Kommunikation zur Datenbank.

Authentication Thema

[WEB.XML] Grundsätzlich kann man in der web.xml Userrollen eintragen und die Berechtigung auf gewisse Views für die jeweiligen Userrollen geben.

[APPLICATION CONFIG] Um zu definieren, dass man standartmäßig zur Login Seite kommt muss man diese Annotation über die ApplicationConfig schreiben.

[CUSTOMINMEMORYIDENTITYSTORE] Aufgrund von Zeitmangel habe ich es so implementiert, dass die User derzeit nicht in einer Datenbank inMemory gespeichert werden und so überprüft wird ob man die richtige Email und Passwort eingegeben hat. Dann bekommt der User die jeweiligen Credentials. Und so schaut die Implementierung von Authentication aus.

Quarkus – Qute

Jetzt komme ich zu Quarkus Qute. Qute selbst bezeichnet sich als Engine, die speziell für Quarkus entwickelt wurde, um HTML Seiten serverseitig zu rendern. Qute ist speziell auf gute Performance ausgerichtet (Reflection wurde minimiert. Unter Reflection versteht man die Fähigkeit einer Java-Klasse von sich aus zu untersuchen und zu manipulieren z.B. Methoden über den String Namen zur Laufzeit aufzurufen: getClass gibt Klasse zurück, obwohl sie zum Kompilierungszeitpunkt noch nicht klar ist wenn man sie z.B. als Object deklariert, Junit durchsucht mit Reflection Methoden die mit @Test gekennzeichnet sind). Derzeit befindet sich Qute noch im experimentellen Modus sprich es ist eine relativ neue Technologie und es können möglicherweise Bugs auftreten. Aufgrund dessen, dass die Technologie relativ neu ist, findet man eher bis auf den Quarkus Guide relativ wenig Informationen. Auch in Qute habe ich dieses ToDo Beispiel vereinfacht implementiert.

[HERZEIGEN DES PROJEKTS]

[POM.XML] Zu Beginn erstellt man sich eine gewöhnliche Quarkus Anwendung. Anschließend fügt man die Quarkus-Resteasy-Qute Dependency hinzu.

Wie wir es auch schon von dem anderen Beispiel kennen haben wir auch hier wieder unsere InitBean, das Repository und die Model-Klasse Todo.

[TODO.HTML] Hier haben wir noch unsere View wo wir die ganzen Daten anzeigen sowie eingeben können. Bei Quarkus Qute ist es wichtig die Views immer in dem Folder src/main/resources/templates zu speichern, ansonsten findet er sie nicht.

[TODORESSOURCE] Bei Qute muss man sich jedoch selbst um die Rest – Kommunikation kümmern und immer die vom Server gerenderten HTML zurückgeben und deshalb haben wir hier unsere TodoRessource. Interessant ist auf jeden Fall, dass wir uns die jeweilige View in der TodoRessource als Template injecten und dann mit .data die Daten setzen können. Mit dem Key können wir wiederrum in der View zugreifen.

Schlusswort

Zu guter Letzt möchte ich noch einmal zusammenfassen, und anmerken was ich denke wann es sinnvoll ist welche Technologien einzusetzen. Dadurch dass man sich durch JSF die kompletten Rest – Schnittstellen als auch Validierung etc. erspart und es schnell zum Erstellen geht, denke ich ist es sinnvoll bei vor allem Eingabeformularen bzw. kleinen Prototypen JSF einzusetzen. Auch im Internet wird JSF vor allem für schnell gebaute Prototypen empfohlen. Eine JSF Applikation lässt sich, wenn man es kann sehr schnell erstellen, was zu den größten Vorteilen von JSF gehört.

Quarkus Qute ist grundsätzlich cool zum Entwickeln gewesen, weil es einfach Quarkus ist, dennoch denke ich ist es derzeit noch mit mehr Aufwand verbunden als JSF, weil sich das Ganze noch in den Startlöchern bzw. von ihnen selbst bezeichnet im experimentellen Modus befindet. Zu Auftretenden Problemen findet man im Internet relativ wenige Informationen. Außerdem muss man im Gegensatz zu JSF die Rest Schnittstellen selbst entwickeln, was für schnelle Prototypen meiner Meinung nach ein Nachteil ist.

Obwohl JSF ziemlich cool ist wird es heute hauptsächlich als legacy Framework bezeichnet. Daher würde ich nicht so viel Zeit investieren, um JSF zu lernen da es kaum eingesetzt wird. JSF kann aber auf jeden Fall hilfreich sein, wenn man wenig Front-End-Kentnisse hat und dennoch eine gutaussehende UI basteln möchte. Wenn ich jedoch in nächster Zeit ein Projekt realisieren möchte werde ich auf die klassische Server Client Architektur mit selbst implementierter Rest Schnittstelle zurückgreifen und mit Hilfe von Webframeworks wie Angular einen eigenen Client schreiben.

Unterschied JSP und JSF: Der Lebenszyklus wird bei JSP für jedes einzelnes Eingabefeld ausgeführt. Bei JSF wird die gesamte Seite mitgegeben und der Lebenszyklus nur einmal für die Seite ausgeführt

Die serverseitige Validierung stellt sicher dass auch bei einem Rest-Zugriff die Validierungen ausgeführt werden im Gegensatz zu clientseitigen Validierungen. Hier werden bei einem direkten Zugriff auf die Rest-Schnittstelle die Validierungen nicht ausgeführt.

JSP bzw. JSF werden in Servlets umgewandelt. Auch mit Servlets können serverseitig gerenderte HTML Seiten zur Verfügung gestellt werden. Hier hat man einen Stream, wo man die gesamte HTML Seite in einem String mitgibt.

Bei den Beans: View Scope heißt dass die Bean für die View gültig ist. Es gibt auch noch andere Scopes wie SessionScoped, ConversationScoped (in Quarkus nicht unterstützt), etc.

```
@ViewScoped//Scope = Gültigkeitsbereich
```