Medienprojekt

Website für die Frauenfussballmannschaft Este06/70



Ellen Schwartau, Minf9888  
Julia Menzel, Minf9950

Gliederung

Anforderungsbeschreibung

Aufgabenstellung unseres Medienprojektes war eine Website für die Frauenfussballmannschaft des Este06/70.

Diese sollte unter Anderem zur besseren Organisation der Mannschaft selbst, der vereinfachten Präsentation der sportlichen Aktivitäten und Ergebnisse sowie dem Werben neuer Teammitglieder dienen.

Die Website sollte sich aus einem öffentlichen, sowie einem mannschaftsinternen Bereich zusammensetzen. Die genauen Anforderungen waren dabei Folgende:

**Allgemein**

* Login und Registrierungsfunktion

**Öffentlicher Bereich**

* Allgemeine Informationen

Besucher sollen über die Seite aktuelle Informationen zu der Mannschaft, den Trainern und dem Training abrufen können.

* Weiterführende Links

Wichtige Seiten sollen über die Mannschaftswebsite schnell zugreifbar gemacht werden.

* Kontaktaufnahme

Es sollen Möglichkeiten geschaffen werden direkt Kontakt zur Mannschaft oder zu den Trainern aufzunehmen (beispielsweise über E-Mail Versand oder Facebook).

* Bildergalerie

Eine Bildergalerie zum Upload von Eindrücken aus dem Mannschaftsgeschehen.

* Spielberichte

Über die Website sollen Spielberichte zu vergangenen Spielen eingesehen werden können.

**Mannschaftsinterner Bereich**

* Forum

Als Grundlage zur einheitlichen Kommunikation soll ein Forum dienen, in denen Threads gepostet und kommentiert werden können.

* Terminplaner

Zur besseren Organisation zukünftiger Aktivitäten soll ein Terminplaner, bzw. ein Zeitplan zum Eintragen von Ab-/Anwesenheit, Krankheit oder längerfristigen Verletzungen dienen.

* Dokumente Manager

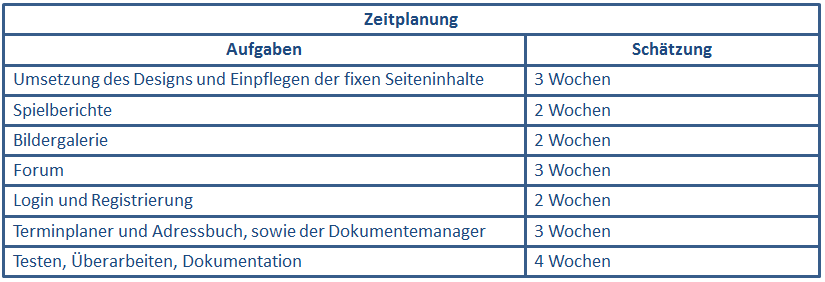
Dateien wie eine Mannschaftskassenübersicht oder Formulare sollen über einen Manager hoch- und heruntergeladen werden können.

* Adressbuch

Die Adressen der Mannschaftsmitglieder sollen konsistent abruf- und aktualisierbar gemacht werden.

Zeitplanung

Ausgehend von einer Entwicklung der Webanwendung mit PHP erstellten wir zu beginn des Projektes folgende Zeitplanung:



In den einzelnen Bereichen enthalten war auch die Feinplanung der Themen und das Testen der Implementierung. Im Anschluss an die oben genannten Phasen war außerdem noch einmal Zeit zum finalen Testen, Überarbeiten bzw. Optimieren sowie für die Dokumentation eingeplant.

Bei der Schätzung per Wochenschritten waren die weiteren Semesteraktivitäten während des Semesters bereits mit Einkalkuliert, sodass ein Zeitraum von einer Woche in etwa als eine aktive Projektarbeitszeit von 15 Stunden entspricht.

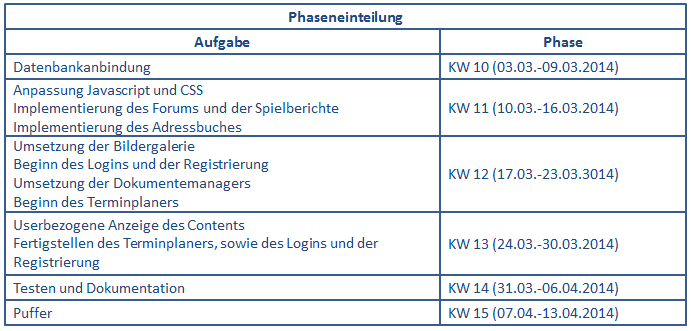
Da wir uns nach der Umsetzung des Designs allerdings für eine Portierung auf Java entschieden, konnten wir den vorherigen Zieltermin mitte Februar nicht einhalten. Grund hierfür war die sehr zeitaufwendige Einarbeitung in das Verwendete Framework (siehe „Freamwork und wichtige Komponenten“) und das zeitintensive Einrichten der Arbeitsumgebung, aufgrund derer eine Weiterarbeit an dem Projekt lange Zeit nicht möglich war.

Daher erstellten wir Ende Februar eine aktualisierte Zeitplanung, mit den zu dem Zeitpunkt noch offenen, detailiierter geplanten, Aufgaben:



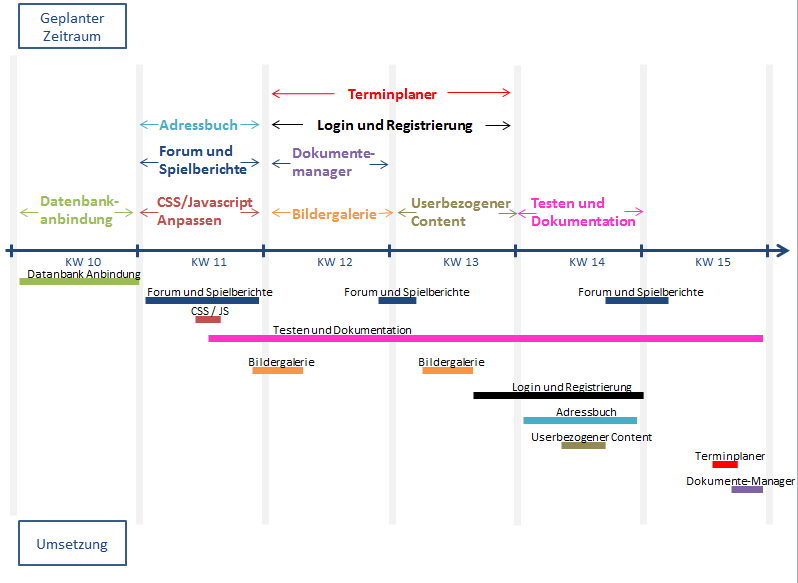
Die Zeitschätzungen waren hier mit Absicht sehr ehrgeizig gewählt, um eine Überschreitung des neuen Abgabetermins zu vermeiden.

Auf dieser Grundlade entwarfen wir außerdem eine ungefähre Phasenplanung:



Als neuer angestrebter Abgabetermin ergab sich hieraus der 14.04.2014.

Folgende Übersicht zeigt die geplanten Phasen, zu den in Wirklichkeit vorgenommenen Arbeitsschritten:



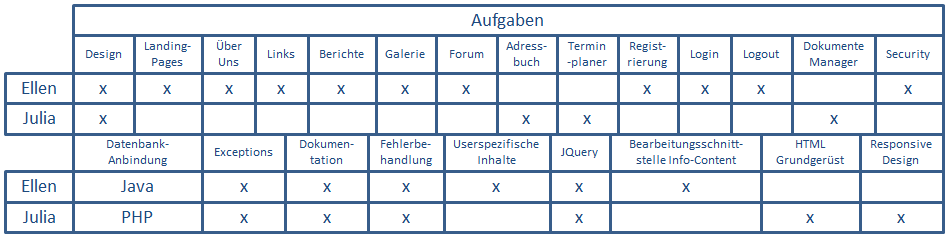
Der Verzug um eine Woche lässt sich auf im Voraus nicht eingeplante, weitere Aufgabenfelder, bzw. Verbesserungen, bzw. Ausweitung der zuvor geplanten Aufgaben (z.B. das Implementieren einer Seitenansicht für das Forum und die Spielberichte).

Tabelle vergleich SOLL IST

Projektplanung und Strukturierung

Aufgabenteilung

Die unten stehende Grafik zeigt die Aufteilung der Aufgaben:



Framework und wichtige Komponenten

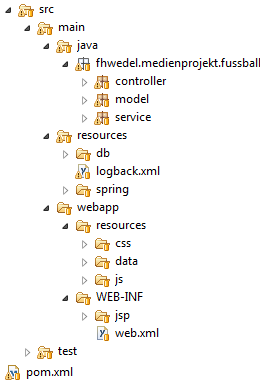
**Programmiersprache und Framework**

Zu Beginn des Projektes entschieden wir uns dazu, die Website mit PHP umzusetzen, da uns diese Sprache sowohl aus der Uni, als auch aus dem Arbeitsumfeld bekannt war.

Schließlich nahmen wir jedoch eine Portierung auf Java vor, um unsere Kenntnisse auf eine neue Programmiersprache zu erweitern. Außerdem verwendeten wir das Framework Spring, das einige Features zur Webentwicklung wie Dependency Injection und Datenbanken Templates mitbringt und gestützt auf Java Server Pages (im Folgenden *JSP* genannt) unterstützt.

**Projektstruktur**

Zum besseren Verständnis der Struktur des Programmes, gehen wir hier grob auf den Aufbau des Projektes ein:

Das Projekt gliedert sich in die nebenstehende Ordnerstruktur auf.

Unter src/main/java sind die Java Dateien zu finden, die sich in Controller, Model und Service gliedern lassen.

Die Controller behandeln die Web Requests und ermitteln den logischen Namen der View, die zurückgeliefert werden soll, unter Model sind die verschiedenen Datenstrukturen der Anwendung zusammengefasst, bzw. hierin sind die darzustellenden Daten enthalten und die Services beinhalten Klassen, die die Logik ermöglichen (z.B. Datenbankzugriffe).

Src/main/resources umfasst die Aufsetzung, bzw. Konfiguration der Datenbank und Spring Konfigurationen.

Weitere statische Ressourcen (wie Css, JavaScript oder Dateien wie Bilder und Icons) sind unter src/main/webapp/resources zu finden.

Die eigentliche View in Form von JSPs liegen unter src/main/webapp/WEB-INF/jsp. Entsprechende Web-Konfigurationen (z.B. DispatcherServlet) sind ebenfalls im WEB-INF Ordner in der web.xml zu finden.

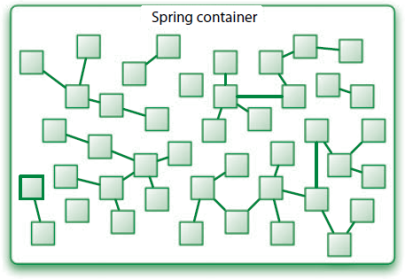
Für Tests steht der Ordner src/test zu Verfügung.

Dependencies für das Framework können in der pom.xml festgelegt werden, über die die benötigten Ressourcen bereitgestellt werden.

**Wichtige Komponenten**

Das Spring Framework bringt einige mächtige Werkzeuge mit sich, mit denen eine Webanwendung entwickelt werden kann.

Im Folgenden werden kurz einige wichtige Komponenten, die wir verwendet haben genannt und deren Aufgabe erläutert.

In Spring-basierten Anwendungen, werden Objekte über den Spring Container verwaltet. Dieser kümmert sich um das Erzeugen und Verwalten, sowie das gegenseitige Verdrahten und organisiert diese während ihres gesamten Lebenszyklus.

Quelle:

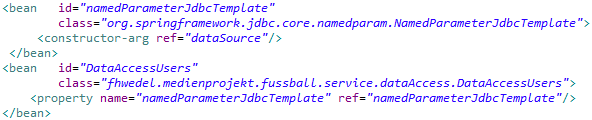
„Spring in Action” Third Edition,

Craig Walls

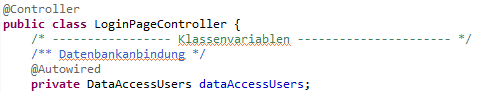
Zum Definieren der Objekte in diesem Container wird die DependencyInjection verwendet. Diese basiert auf den Deklarationen, die überwiegend in der src/main/resources/spring/applicationContext.xml gemacht werden.

Im ApplicationContext können unter Anderem Objekte (im weiteren Bean, gestützt auf die JavaBeans, genannt) deklariert, in andere Objekte gebunden und deren Lebenszyklus beeinflusst werden.

Das grundsätzliche Vorgehen kann an diesem Beispiel verdeutlicht werden:

Über das <bean>-Tag können Objekte mit den Attributen id und class definiert werden.

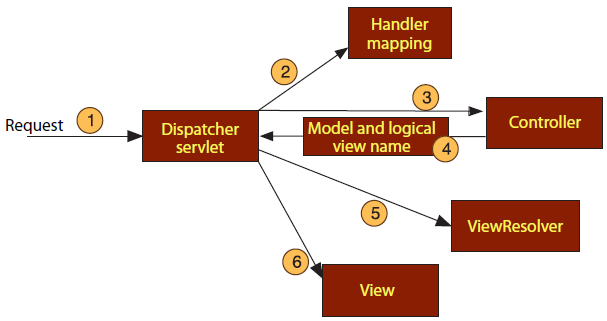
Des Weiteren können zum Beispiel Konstruktorvariablen übergeben oder Properties der Klasse gesetzt werden.

Auch über Annotationen wie @Controller können Beans automatisch generiert werden.

@Autowired sorgt dafür, dass Beans aus dem Container über die ID ohne weitere Konfiguration in Properties gemappt werden.

Alternativ kann über die Funktion getBean(<id>) auf die Beans zugegriffen werden. Näheres kann z.B. unter <http://docs.spring.io/spring/docs/2.5.6/reference/beans.html> nachgelesen werden.

Auch beim Bearbeiten von Request und dem Auflösen in JSPs kommen ebenfalls einige Komponenten zum Einsatz. Das unten stehende Diagramm veranschaulicht, wie ein Request mit Spring verarbeitet wird:

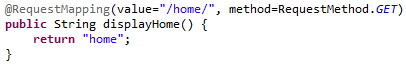


Quelle:

„Spring in Action” Third Edition,

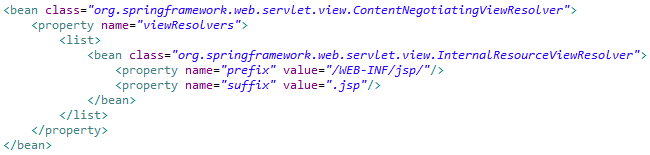
Craig Walls

Ein Request wird zunächst vom DispatcherServlet entgegengenommen, dessen Aufgabe es dann ist, den Request an den entsprechenden Controller weiterzuleiten. Hierfür sind die Controller mit den URLs gekennzeichnet, auf die sie hören sollen. Dies geschieht über die Angabe des @RequestMapping am Funktionskopf des Controllers. Zur Verdeutlichung ein Beispiel für einen Home-Controller:



Als Ergebnis liefert die Controller-Methode einen String als Indikator für die JSP die geladen werden soll.

Um nun auf die tatsächliche JSP, die geladen werden soll, schließen zu können, können über den ViewResolver Regeln festgelegt werden, wie mit der Antwort des Controllers umgegangen werden soll. In unserem Fall dient ein einfacher Post- und Präfix dazu, den Pfad der JSP zu definieren:



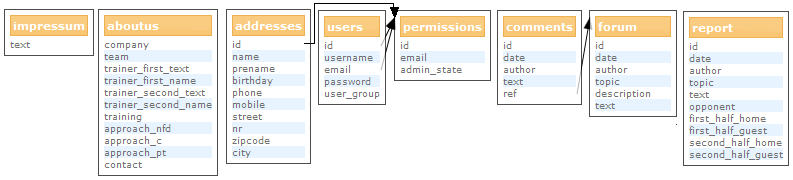
Der gelieferte String „home“ wird demnach zu /WEB-INF/jsp/home.jsp aufgelöst und verweist somit auf die eigentliche Ressource und kann zur Anzeige gebracht werden.

Für die Arbeit mit JSPs lassen sich erweiternde Tag-Libraries importieren, mit denen das Arbeiten mit diesen Vereinfacht wird. Es stehen beispielsweise Tags zur Verfügung, mit denen z.B. Schleifen oder Verzweigungen umgesetzt oder Formulareingaben in Objekte gebunden werden können. Ein weiterer Vorteil ist, dass so Java Befehle in prinzipieller Schreibweise von HTML-Elementen dargestellt werden können, was den Aufbau von JSPs auch optisch einheitlicher macht. Die Tag-Libraries, die in unserer Anwendung Gebraucht gefunden haben sind unter src/main/webapp/WEB-INF/jsp/includes/taglibs\_variables.jspf zu finden.

Weiteres zu Spring kann unter <http://spring.io/> bzw. <http://spring.io/>docs nachgelesen werden.

Datenbankarbeit

Unsere Wahl bezüglich des Datenbankverwaltungsssystems fiel auf MySQL, da dieses uns sowohl aus der Vorlesung als auch aus dem Arbeitsumfeld bereits bekannt war. Auch wegen weiterer Vorteile wie Betriebssystemunabhängigkeit, Verfügbarkeit als Open Source System und häufige Anwendung als Basis für dynamische Webanwendungen und eine somit lebhafte Community, war MySQL für unser Projekt gut geeignet. Zudem sind Hosts verfügbar, die Webanwendungen mit Java in Zusammenarbeit mit MySQL unterstützen.

Unsere Datenbank gliedert sich in folgende Tabellen auf:

**Impressum**

Diese Datenbanktabelle enthält den Text für das Impressum.

**Aboutus**

Hier werden die Bestandteile der Über-Uns-Seite gespeichert. Dazu gehört ein Text über den Verein, das Team, die beiden Trainer, das Training, Texte zur Anfahrt und eine Beschreibung zum Kontakt.

**Report**

Zu den Spielberichten werden hier eine Identifikationsnummer, ein Datum, der Author, sowie ein Titel, der Text, der Name des Gegners und verschiedene Spielstände gespeichert.

**Forum**

Zu den Foreneinträgen wird ebenfalls eine Identifikationsnummer, sowie Datum, Author, Titel und Text gespeichert. Außerdem die forenspezifischen Texte zur Kurzbeschreibung .

**Comment**

Die Kommentare besitzen genauso eine Nummer zur Identifikation, ein Datum, sowie Author und Text und referenzieren eine ID der Foreneinträge. Sollte ein Foreneintrag gelöscht werden, werden auch die Kommentare, die die ID diesee Foreneintrag im Attribut „ref“ enthalten, ebenfalls entfernt.

**Permissions**

Die Permission-Tabelle dient dazu, E-Mail-Adressen zu speichern, die zur Registrierung zugelassen sind. Außerdem ist hier ein Attribut vorhanden, das Aufschluss über den Administrator-Status eines potenziellen Users angibt. Als Primary Key dient auch hier die ID.

**Users**

Die Usertabelle referenziert als Foreign Key die Identifikationsnummer der Permissiontabelle und enthält außerdem den Usernamen und das Passwort, über die sich ein Benutzer einloggen kann. Zudem wird die E-Mail Adresse und eine Benutzer-Gruppe, die entsprechend dem Attribut „admin\_state“ in der Permission-Tabelle entweder den Wert „USER\_GROUP\_ADMIN“oder „USER\_GROUP\_NO\_ADMIN“ annimmt. Der User Eintrag wird gelöscht, sollte die referenzierte Permission verworfen werden.

**Addresses**

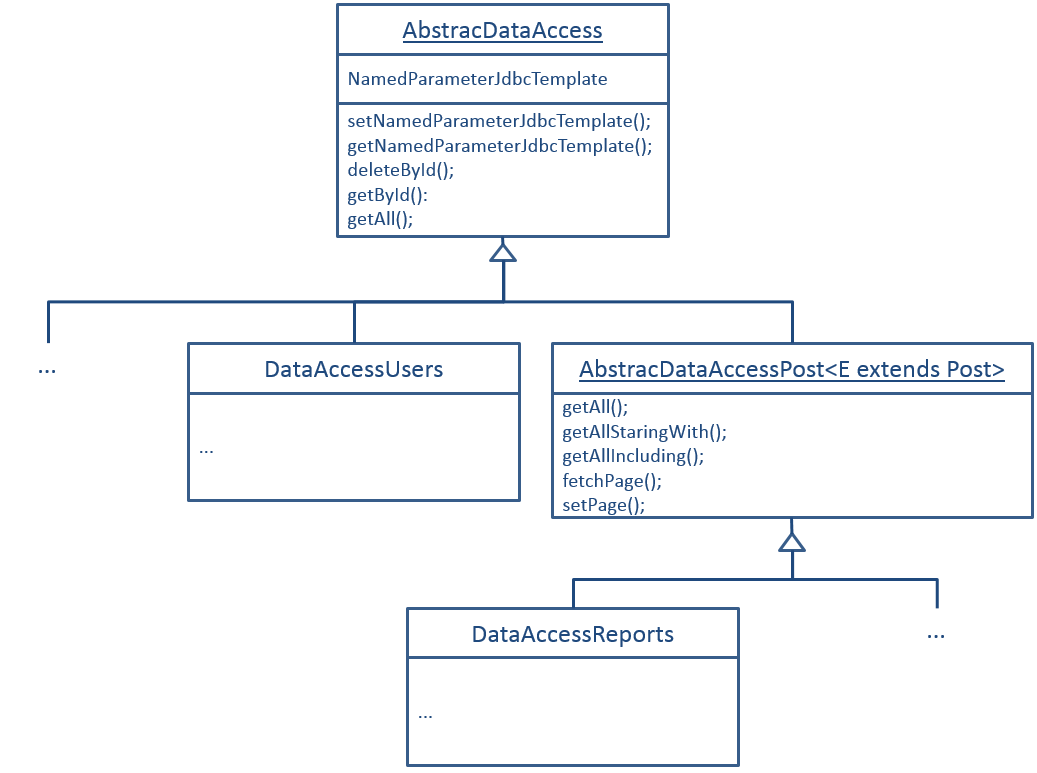
Auch diese Tabelle referenzieren die ID der Permissions und enthält die Adressinformationen der User. Dazu gehören Vor- und Nachname, Straße und Hausnummer, Wohnort und Postleitzahl, sowie Telefonnummern. Sollte die Permission gelöscht werden, wird auch hier der entsprechende Eintrag entfernt.

Zur Datenbankarbeit mit Java stehen im Package service.dataAccess sowie service.dataErrors Klassen zum Datenbankzugriff sowie dem Abfangen und Behandeln von Fehlern in den Benutzereingaben zur Verfügung.

Hierbei greifen wir vor allem auf das NamedParameterJdbcTemplate zurück, bei dem im Vorteil gegenüber dem SimpleJdbcTemplate den Vorteil hat, dass die Abfrageparameter über einen zugewiesenen Namen zugreifbar gemacht werden können, die Reihenfolge der Parameter spielt somit keine Rolle.

Die JdbcTemplates arbeiten mit der Templating Methode, das heißt, das nur die eigentlichen Datenbankabfragen geschrieben werden müssen, Verbindungsauf- und -abbau, sowie das Behandeln von Exceptions werden von den Templates übernommen, was die Datenbankarbeit vereinfacht und übersichtlicher macht.

Um nicht in jeder DataAccessKlasse das NamedParameterJdbcTemplate setzen zu müssen, erweitern alle DataAccess-Klassen die AbstractDataAccess.java, die sowohl das JdbcTemplate, als auch die Set- und Get-Methode hierfür implementiert. Auch die AbstractDataAccessPost<Post> fasst einige Aufgaben für die erbenden Klassen zusammen:



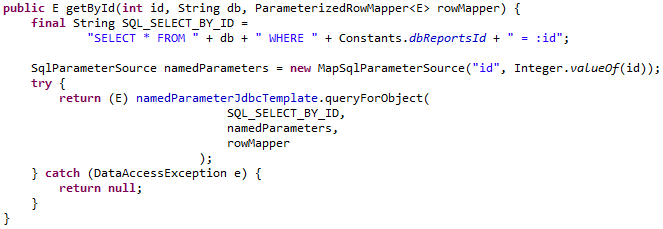
Abstrakte

Klassen

der

Datenbank

-arbeit

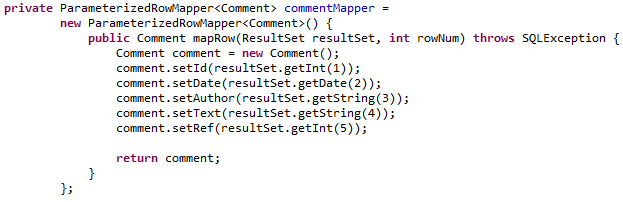
Um die Funktionsweise zu verdeutlichen, im Folgenden ein Beispiel zur Abfrage eines Datensatzes über dessen ID (AbstractDataAccess.java):

Die eigentliche SQL Abfrage wird als String repräsentiert, wobei Variablen, die mit „:“ beginnen, über die HashMap ein Wert zugewiesen wird.

Zur Abfrage wird in diesem Beispiel außerdem ein RowMapper benötigt, der den Attributen der Klasse E (z.B. User.java oder ForumEntry.java) Spalten des Ergebnisses zuweist.

Die RowMapper sind in den entsprechenden DataAccess-Klassen als anonyme inner Klasse deklariert, da sie nur hier benötigt werden und die Komplexität betreffend noch relativ überschaubar sind.

Im folgenden ein beispielhafter RowMapper, der eine Reihe aus der Comments-Tabelle auf ein Objekt der Klasse Comments.java zuweist:

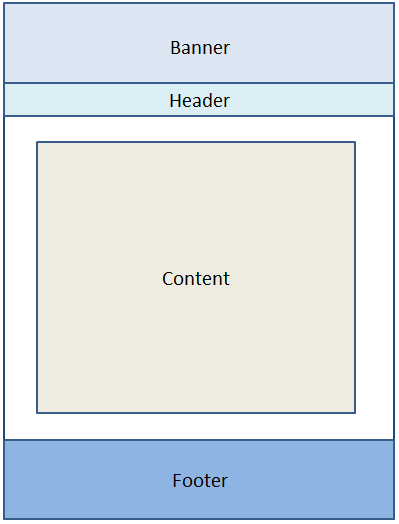


Das ResulSet beinhaltet eine Liste von Ergebnissen aus der Datenbank. Über verschiedene get-Funktionen können Spalten der Datenbank als ein bestimmter Typ (z.B. Int, String oder Date) ausgelesen und anschließend über die Set-Methoden der Comment.java auf Variablen dieser Instanz zugewiesen werden.

Konzept und Implementierung

**Design**

Das Aussehen der Website musste sowohl gut bedienbar, als auch übersichtlich und ansprechend sein. Da im Fußballsport sowohl junge als auch ältere Männer und Frauen tätig sind, musste sie dafür ausgelegt sein, eine breite Zielgruppe anzusprechen.



Wir entschieden uns für einen Grundaufbau mit Folgenden Bestandteilen:

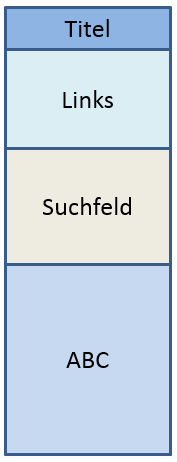
Am Kopf der Seite sollte ein Banner mit einem Bild passend zum Thema Fussball stehen.

Mit dem Header folgt darauf die Navigation, die je nachdem, ob ein Besucher ein- oder ausgeloggt ist, andere Links zur Verfügung stellt.

Am Saum der Seite erscheint der Footer mit Kontaktdaten, wichtigen Shortcuts, sowie Login und Registrierfunktion, sollte der Besucher nicht eingeloggt sein.

Zwischen Header und Footer wird der seitenspezifische Content dargestellt.

Um die verschiedenen Seiten mit einem konsistenten Design auszustatten, dieses aber dennoch an die individuellen Bedürfnisse anpassen zu können, entwarfen wir verschiedene Grundlayouts, die dann übergreifend Anwendung finden konnten.

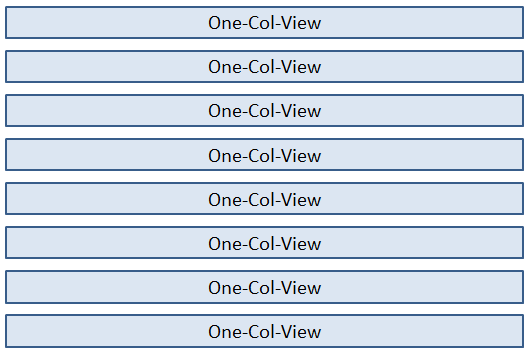
Das Grunddesign lässt sich in zwei Teile gliedern: Der Sidebar und dem eigentlichen Content.

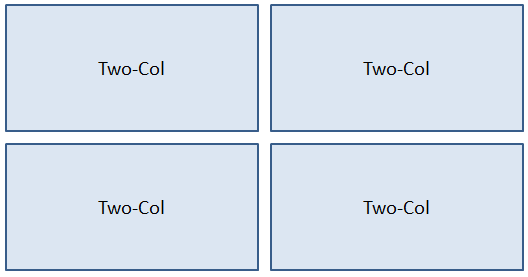
Die Sidebar setzt sich zusammen aus dem Titel der Seite, darunter folgend weitere Links, ein Suchfeld und eine alphabetische Schaltfläche zur Selektion von Content zu einem bestimmten Buchstaben (Beispielsweise zum Suchen nach Namen im Adressbuch).

Die einzelnen Bestandteile sind einzeln in- bzw. exkludierbar, je nach Bedarf kann diese dann entsprechend bereitgestellt werden.

Der Content der Seite steht entweder allein oder neben, bzw. unter der Sidebar.

Der Content selbst kann in drei verschiedenen Layouts organisiert werden – einer ein-, zwei- oder dreispaltigen Einteilung.





Hierbei besteht bei dem einspaltigen Design außerdem die Möglichkeit, den Content per Mausklick ein- oder auszublenden, um für mehr Übersicht zu sorgen.

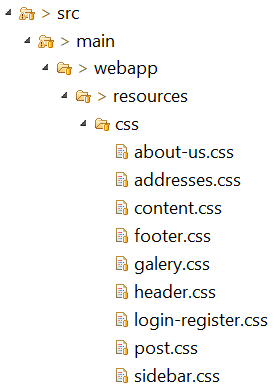
Alle JSPs (ausgenommen die galery.jsp) haben einen ähnlichen Aufbau:

Zunächst werden die gemeinsamen Tag-Libraries und Variablen (z.B. Linknamen) über <%include%> eingebunden. Auch die gemeinsamen <head>-Daten sind in eine head.jsp ausgelagert, wobei Titel und Description der Seite innerhalb des <jsp:include>-Elements über das <jsp:param> Element gesetzt und in der head.jsp mit Hilfe der ${<name>}-Annotation ausgelesen werden.

Auch Header und Footer sind in eigene Dateien ausgelagert und werden ebenfalls eingebunden.

Um den Content der Seite formatieren und positionieren zu können sind verschiedene Wrapper-Container vorhanden. Zum Umfassen des gesamten Contents dienen die <div>-Elemente mit den Klassen main-container, container, main sowie main-inner, die optional direkt den Content (z.B. das Imressum) oder die Sidebar, sowie den weiteren Content enthalten, wobei dieser wiederum über die div-Elemente mit der ID main-content-small und den Klassen outer, inner und content-list umschlossen werden.

Die Container dienen neben der Positionierung der Elemente auch dem Einstellen von weiteren Stylesheets, wie Padding, Margin oder text-align.

**Stylesheets**

Die CSS-Dateien, in denen die Stylesheet-Angaben getätigt werden, befinden sich in dem Ordner src/main/webapp/resources/css. Die Dateien lassen sich einteilen in allgemeine Stylesheets, die websiteübergreifend greifen (content.css), und spezifischere Stylesheets zu einzelnen Elementen oder Seiten, wie z.B. die addresses.css oder footer.css mit Styleangaben zum Addressbuch oder dem Footer.

Statt verschiedene CSS-Dateien für die spezifischen Endgeräte (wie z.B. mobile oder screen) zu erstellen, haben wir uns dazu entschieden, für alle Geräte das gleiche CSS auszuliefern und dieses über die @media-Angaben an die unterschiedlichen Auflösungen anzupassen.

So enthalten die CSS-Dateien zunächst die allgemeinen Stylesheet-Angaben und anschließend die auflösungsspezifischen Angaben.

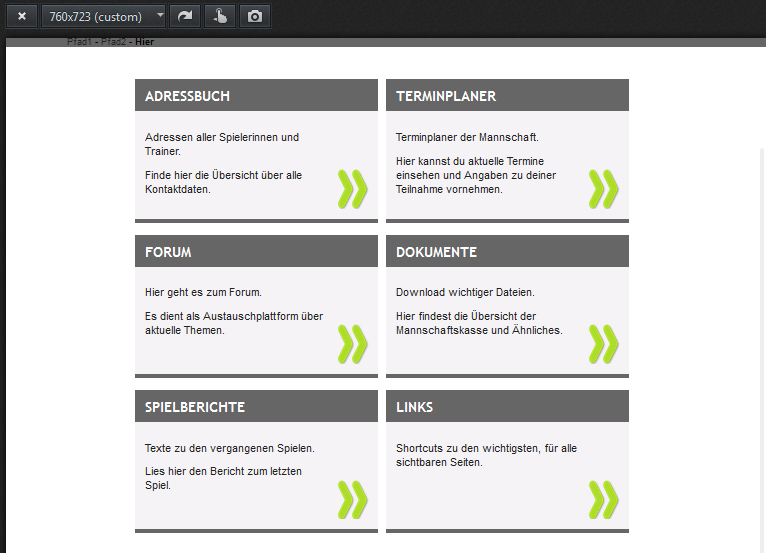
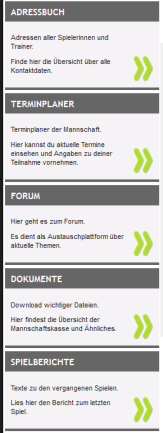
**Farbgebung**

Als Grundlage zum Design der Seite wählten wir verschiedene Grautöne und die Farbe Grün, zum setzen bestimmter Akzente als Anspielung auf das Grün des Fußballrasens.

**Responsive Design**

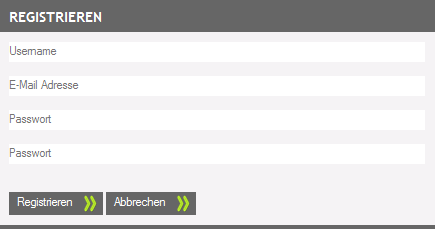
Aufgrund der Vielzahl an verschiedenen Devices und den damit einhergehenden unterschiedlichen Display-Maße und Auflösungen war eine weitere Anforderung an das Design die Kompatibilität mit möglichst vielen Formaten, auf denen die Website angezeigt werden könnte.

Hierfür setzten wir vor allem relative Größenangaben und Media-Queries ein, mit denen wir zum Beispiel auf das Floating Verhalten, das Ein- oder Ausblenden von Bildern oder Größen- und Abstandsverhältnisse der Elemente Einfluss nahmen.

Im Folgenden ein Beispiel des Verhaltens des Three-Col-Views in Abhängigkeit zur Bildschirmauflösung (von links nach rechts 850px Breite, 760px Breite, 605px Breit, 400px Breite):

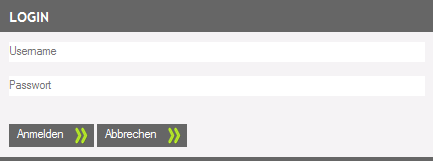
Zunächst wird das Design wie gewollt dreispaltig angezeigt, anschließend werden die Bilder weggelassen und auf ein zwei-spaltiges Design geswitcht, bis der Inhalt nur noch einspaltig in gleicher Höhe, anschließend einspaltig mit automatischer Höhe dargestellt wird.

**Login und Registrierung**

Die Registrierung läuft in einem mehrstufigen Verfahren ab - Zunächst muss die E-Mail-Adresse, mit der ein User sich registrieren möchte, durch einen Administrator zugelassen werden. Erst anschließend kann sich ein User mit entsprechender E-Mail einmalig anmelden. Bei Vergabe der Registrierungserlaubnisse können Administratorrechte vergeben werden. Diese Rechte können über einen Administrator auch später bearbeitet werden.

Das Registrierungsformular besteht aus vier Pflichtfeldern für den Usernamen, die E-Mail-Adresse, welche der abgespeicherten in Datenbank entsprechen muss sowie zwei Passwortfeldern, die gleich sein müssen.

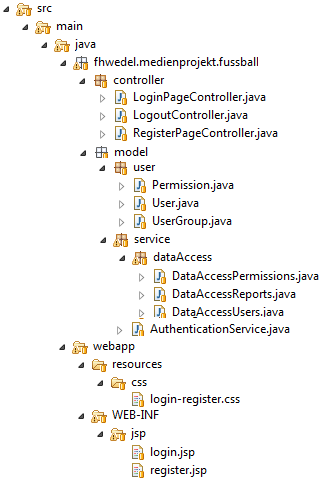
Nach der Registrierung ist der User automatisch angemeldet.

Der Login erfolgt nach Registrierung über den angegebenen Usernamen und das Passwort.

Nach erfolgreicher Anmeldung wird der User auf die Landing-Page für eingeloggte Besucher weitergeleitet (diese enthält Shortcuts zu wichtigen Seiten für angemeldete Benutzer, wie Adressbuch, Forum usw.).

Bei beiden Formularen wird auf Labels verzichtet und stattdessen mit Placeholdern gearbeitet, die angeben, was in das jeweilige Feld einzutragen ist.

Zu erreichen sind Registrierung und Login über Schaltflächen im Footer, sowie im Header. Ist ein User eingeloggt, erscheint im Header ein Link für den Logout.



Die Logik ist auf drei grundlegenden Klassen aufgebaut:

* Permission.java repräsentieren die

Registrierungszulassungen

* User.java bilden die Grundlage für die User
* Und UserGroup.java beinhaltet die möglichen

Rollen, die ein User innehaben kann (genaueres

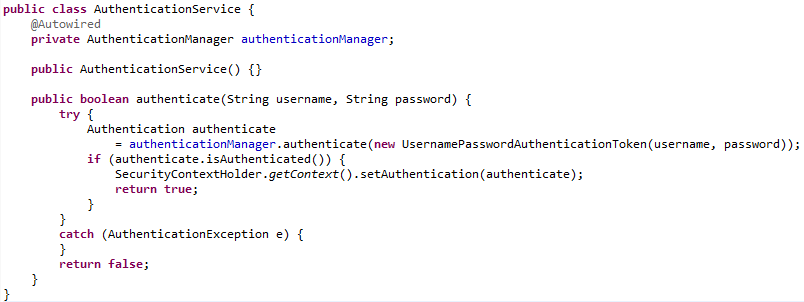
siehe: Zugriffsrechte und Editieren von

Content)

Die Datenbankarbeit wird übernommen von den DataAccess-Klassen DataAccessPermissions.java und DataAccesUser.java.

Die involvierten Controller sind der LoginPageController.java, LogoutController.java und der RegisterPageController.java.

Die Darstellung für den Login und die Registrierung ist über die login.jsp und die register.jsp definiert. Die register.jsp enthält außerdem die Schaltflächen zum Hinzufügen neuer, bzw. eine Übersicht über die aktuellen Permissions, die dort auch bearbeitet werden können. Sichtbar sind die zuletzt genannten Bestandteile nur für Administratoren.

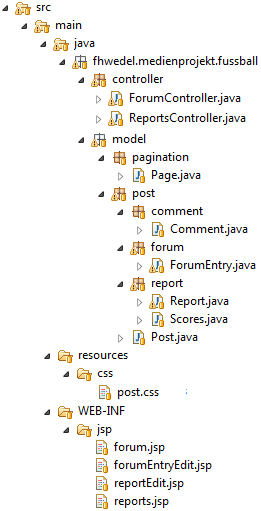
Für die eigentliche Authentifizierung des Users ist der AuthenticationService zuständig, auf den im Folgenden näher eingegangen wird. Der Service ist wie folgt aufgebaut:

Wichtigster Bestandteil ist hier der AuthenticationManager, der die Authentifikation von Usern ermöglicht und diese organisiert. Die Funktion authenticate() übernimmt hier die Authentifizierung eines Users über ein UsernamePasswordAuthenticationToken, wodurch anschließend im Context der Webanwendung mit Hilfe von <security>-Tags der Login-Status und die User-Gruppen des eingeloggten Users abgefragt werden können (näheres siehe unter dem Punkt „Sicherheit“).

**Forum und Spielberichte**

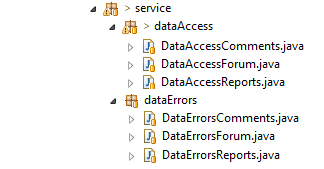
Die Website sollte sowohl die Möglichkeit bieten, Spielberichte zu verfassen und zu bearbeiten, als auch die Führung, Bearbeitung und Verwaltung eines Forums zu ermöglichen. Außerdem sollte es innerhalb des Forums möglich sein, Kommentare unter dem eigenen Username zu verfassen, bzw. zu löschen.

Die bei der Umsetzung beteiligten Komponenten sind folgende:

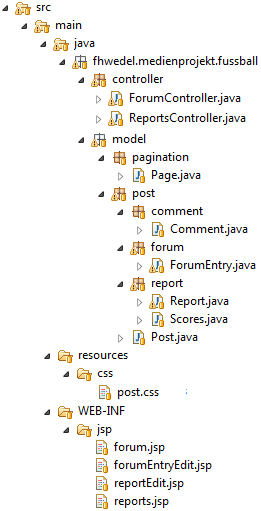


Zuständige Controller sind der ForumController.java und der ReportsController.java.

Die Datenstruktur stützt sich im wesentlichen auf die Klassen Post.java, Comment.java, ForumEntry.java, sowie Report.java im package fhwedel.medienprojekt.fussball.post.



Services zur Datenbankarbeit und der Fehlerbehandlung sind die DataAccessComments.java, DataAccessForum.java und DataAccessReports.java, bzw. die Data DataErrorsComments.java, DataErrorsForum.java und DataErrorsReports.java.

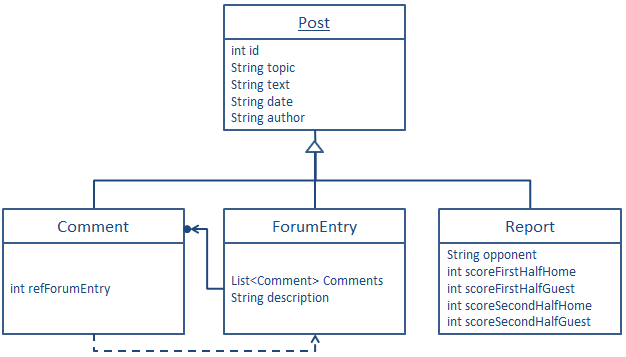
Die allgemeinen CSS Angaben der content.css werden durch spezifische Angaben zur Gestaltung der Spielberichte, den Kommentaren und der Seitennavigation in der post.css ergänzt.

JSPs, die zur Visualisierung der Thematik dienen sind die forum.jsp, forumEntryEdit.jsp, reports.jsp und die reportEdit.jsp.

Aufgrund des ähnlichen Charakters der Einträge im Forum und der Spielberichte, wie z.B. ähnliche benötigte Variablen, wie Titel und Text, sind bei der Implementierung beider Teile die gemeinsamen Variablen und Methoden in abstrakten Klassen zusammengefasst.

Zur Repräsentation der Datenstrukturen sind die Klassen Post.java, ForumEntry.java, Report.java und Comment.java im Package src/main/java/model/post zur Repräsentation der Datenstrukturen (Spielbericht, Foreneintrag und Kommentar) vorhanden, wobei die abstrakte Klasse Post.java gemeinsame Variablen, sowie deren Getter- und Setter Methoden implementiert, und von den weiteren Klassen beerbt wird.

Zur Übersicht ein Klassendiagramm, das die Zusammenhänge zwischen den eben genannten Klassen verdeutlichen soll:



Struktur des Package post

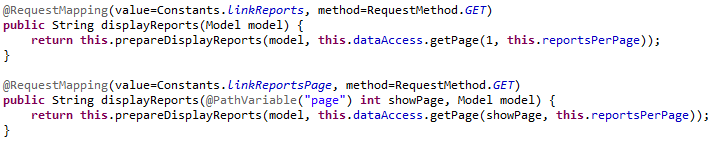
Außerdem ist zur Anzeige der Beiträge eine Seitenanzeige implementiert, die durch die Klasse Page.java repräsentiert wird, denn zur übersichtlicheren Anzeige der Einträge, werden nicht direkt alle Berichte oder Forenbeiträge auf einer Seite angezeigt, sondern über eine Seitenansicht strukturiert. Hierbei werden stets maximal 10 Einträge pro Seite angezeigt, vorherige oder weitere Einträge können dann über eine Seitennavigation unterhalb des Contents erreicht werden. Dazu werden in der Klasse Page.java die aktuelle Liste der Einträge, sowie angezeigte, nächste, vorherige und Gesamtzahl der Seiten gespeichert.

Unterhalb der Einträge erscheint dann folgende Navigation:



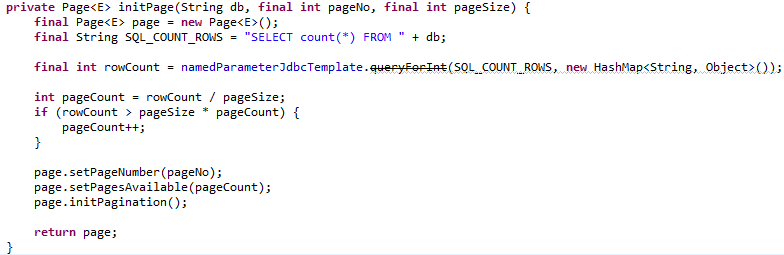
Hierbei stellen die Wort <vorherige> und <weitere> Links zu den jeweiligen Seiten dar, die beispielsweise die URLs der Form „.../seite-2/“ aufrufen, auf die ein entsprechender Controller hört und anschließend die Einträge der Seite lädt.

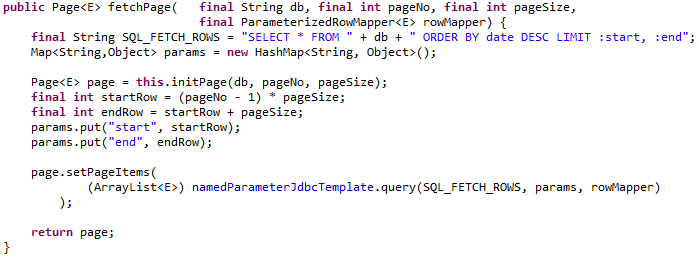
Die eben genannte Controller-Funktion ist dabei überladen:



Mit Seitenangabe in der URL wird diese ausgelesen und die gewünschte Seite mit Hilfe der getPage()-Methode, die die fetchPage()-Funktion aus der AbstractDataAccessPost.java aufruft, erstellt und angezeigt. Sollte keine explizite Angabe erfolgt sein, wird die erste Seite geladen.

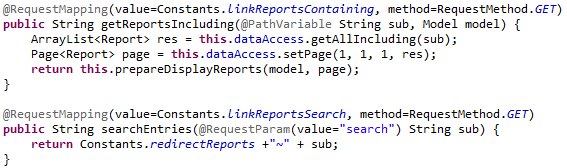
Die zuletzt genannten Funktionen sind wie folgt aufgebaut:

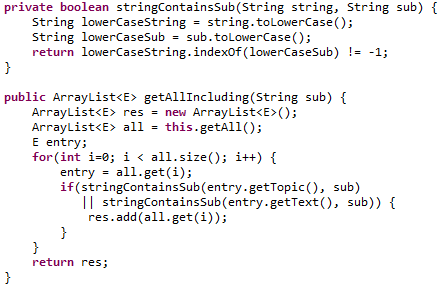




FetchPage() liefert eine Page.java mit einer Liste von Einträgen des generischen Datentyps E (welcher hierbei die Klasse Post beerben muss) zurück. Dazu wird zunächst über die initPage()-Methode, die Gesamtzahl der Einträge bestimmt und ausgehend von pageSize (Anzahl der pro Seite anzuzeigenden Elemente) die Gesamtseitenzahl, sowie die vorherige und nachfolgende Seite bestimmt. Ausgehend von der anzuzeigenden Seitenzahl werden dann Start- und Endreihe der Datenbanktabelle berechnet, aus der Datenbank ausgelesen, sowie in der Page gespeichert und in der JSP über das <c:forEach>-Tag angezeigt.

Es besteht die Möglichkeit, nach bestimmten Einträgen zu suchen. Zunächst war unsere Überlegung, eine Schaltfläche zu implementieren, mit Hilfe derer Einträge zu einem bestimmten Datum selektiert werden könnten. Es erschien uns nun jedoch sinnvoller, gezielt nach Inhalten in den Einträgen suchen zu können, da so direkt nach Berichten zu bestimmten Mannschaften oder Themengebieten aus dem Forum gesucht werden kann, ohne dass das Datum bekannt sein muss. Das Suchen ist zum einen über das Suchfeld in der Sidebar möglich, zum Anderen kann eine URL der Form „.../~<gesuchter-String>“ verwendet werden. In den jeweiligen Controllern sind dann Methoden vorhanden, die die gewünschten Einträge auslesen und auf einer Seite anzeigen:



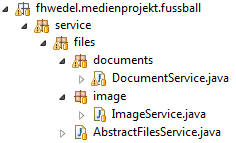


Sollte über das Suchfeld gesucht werden, werden die Angaben aus dem GET-Request über die @RequestParam-Annotation herausgefiltert und auf eine URL entsprechend dem Pattern der Such-URL redirectet.

Um entsprechende Einträge herauszufiltern, wird die Methode getAllIncluding() aus der AbstractDataAccessPost.java aufgerufen. Diese selektiert aus allen Einträgen über die Funktion String.indexOf(<sub>) diejenigen Einträge heraus, die den gesuchten String im Titel oder dem Text enthalten. Groß- und Kleinschreibung wird hierbei ignoriert.

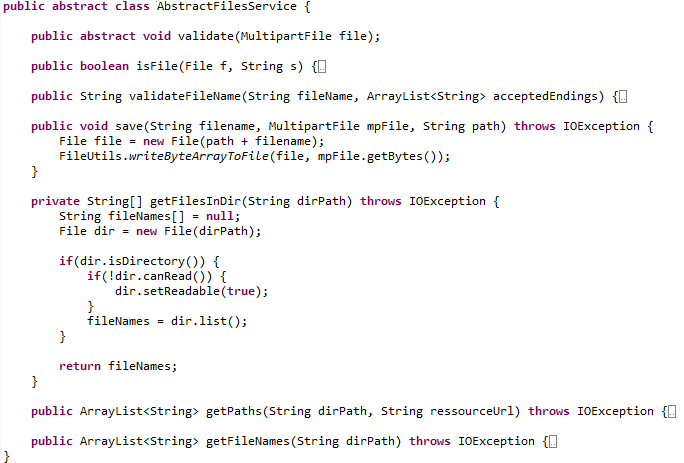
**Adressbuch**

**...**

**Hochladen und Auslesen von Dateien**

Da auf der Website der Fussballmannschaft sowohl die Bilder der Bildergalerie, als auch Dokumente für den Dokumente Manager hochgeladen werden müssen, stellt der AbstractFilesService im Package fhwedel.medienprojekt.fussball.service.files gemeinsame Funktionalität für Upload und Anzeige bereit und wird von dem DocumentService, sowie dem ImageService beerbt.

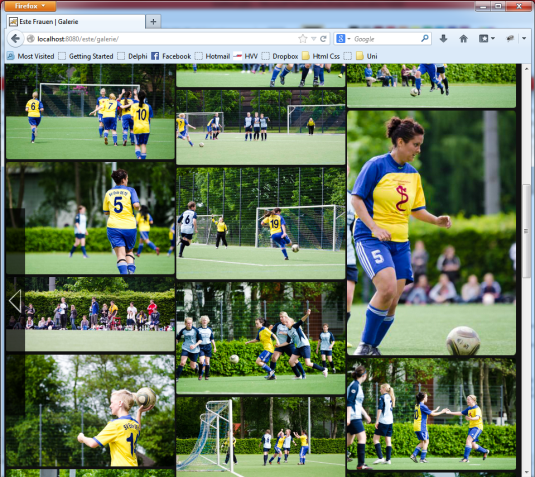
Der AbstractFilesService ist wie folgt aufgebaut:

****

Die abstrakte Methode validate() muss von den erbenden Klassen implementiert werden. Über isFile() kann geprüft werden, ob das angegebene File valide ist. Der Name des Files kann mit Hilfe der validateFileName() überprüft werden, wobei Sonderzeichen entfernt und fehlende Dateiendungen angefügt werden. Zum Speichern einer Datei dient die Methode save(), wobei die FileUtils der apache.commons.io verwendet werden. Zum Auslesen und Anzeigen der Bilder ist es unter Anderem nötig, den relativen Pfad der Resourcen der Form /resources/data/.../<Dateiname> innerhalb der JSPs zu referenzieren. Hierfür liest die getPaths() mit Hilfe der getFilesInDir() die Dateinamen eines Verzeichnisses aus und gibt den entsprechenden Pfad über String-Konkatenation zurück. GetFileNames() liefert die Namen der Dateien in einer ArrayList.

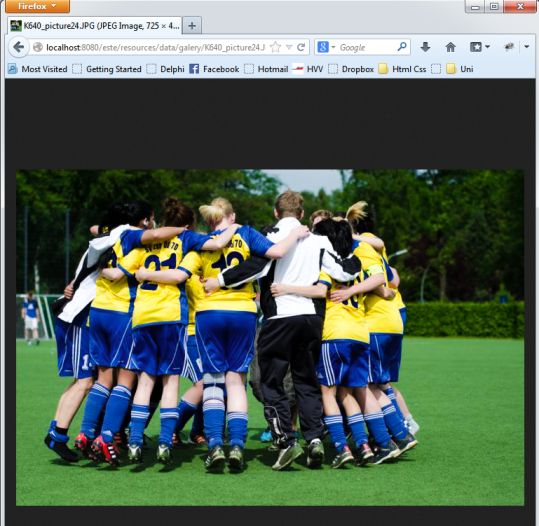
**Galerie**

Als Werkzeug, um den Besuchern der Seite direkte Eindrücke aus dem Alltag des Vereins bieten zu können, dient eine Bildergalerie. Sie fällt als einzige aus dem Grunddesign heraus und bietet die Vorschau der Bilder ausgebreitet auf der kompletten Seite in einem dreispaltigen Design. Bei schmaleren Displays wird auf ein zwei- bzw. einspaltiges Design gewechselt.

Umgesetzt ist das dreispaltige Design mit Hilfe von drei nebeneinanderstehenden <div>-Elementen, die jeweils einen Drittel der Breite des Fensters einnehmen.

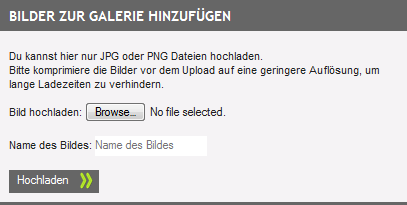
Jede Spalte enthält weiter eine Liste an Bildern, deren URL aus dem Verzeichnis src/main/java/webapp/resources/data/galery/ ausgelesen und als href-Attribut der <img>-Elemente geladen werden.

Die einzige Verbindung zum Rest der Website stellt ein transparenter Home-Button dar, der entweder am linken oder (bei schmaleren Displays) am oberen Rand mitscrollt.

Die Bilder selbst sind über einen schmalen weißen Rand getrennt und haben abgerundete Ecken, um das Design etwas aufzulockern. Die eigentliche Wirkung des Designs entfaltet sich über die verschiedenen Formate der Bilder, welche dafür sorgen, dass die Bilder auf verschiedenen Höhen beginnen und enden.

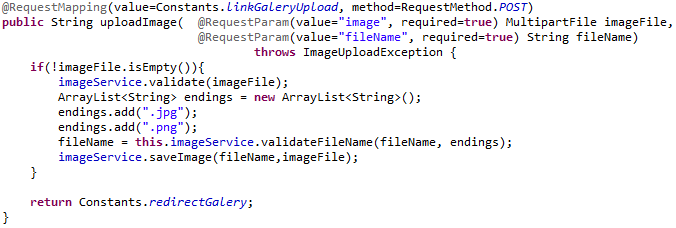
Per Mausklick auf ein Bild lässt sich die Ressource im Browser in groß anzeigen.

Momentan ist es nur möglich, alle Bilder auf einmal zu laden, da noch keine Funktion zur Verfügung steht um einen mehrseitigen Katalog zu erstellen. Aufgrund der geringen Anzahl an Bildern, ist dies zurzeit aber noch ausreichend und kann zu einem späteren Zeitpunkt nachgeholt werden.

Zum Uploaden neuer Bilder steht ein Upload Formular zur Verfügung, über das neue Bilder im JPEG oder PNG Format hochgeladen und ein Name des Bildes angegeben werden können.

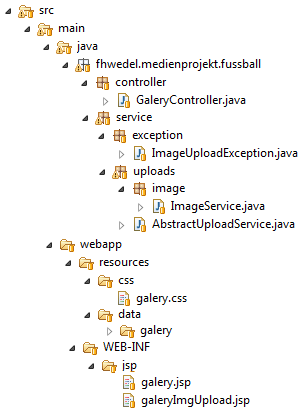
Die momentan auf der Seite zugreifbaren Bilder sind auf eine Breite von 640px komprimiert, bei weiterem Upload von Bildern müssen diese zuvor per Hand komprimiert werden, um Ladezeit der Galerie nicht negativ zu beeinflussen.

Um die angegebenen Bilder speichern zu können, wird ein MultipartResolver (definiert in der mvc-config.xml) verwendet, der Multipart-Daten aus einem Post-Request extrahiert, sodass der Dispatcher Servlet diese an den GaleryController weiterreichen kann.



Im Controller selbst kann das Image anschließend über die @RequestParam-Annotation entgegengenommen werden.

Zur Prüfung des Images und dem anschließenden Speichern steht ein ImageService im Package fhwedel.medienprojekt.fussball.service.files.image zur Verfügung. Geprüft wird zur Validierung des Images mit Hilfe der getContentType()-Funktion in erster Linie das Dateiformat. Erlaubt sind JPG und PNG, so soll verhindert werden dass zum Beispiel EXE-Dateien hochgeladen werden. Auch der im Formular angegebene Name wird auf Richtigkeit geprüft und Sonderzeichen entfernt, sowie eine Endung angehängt, sollte diese fehlen.

Nebenstehend noch einmal die beteiligten Komponenten und Aufgaben:

Der GaleryController übernimmt das Mappen der Requests, die mit /galery/ beginnen.

Die ImageUploadException wird geworfen, sollte beim Upload eines neuen Bildes ein Fehler auftreten.

Der ImageService übernimmt das Validieren und Speichern beim Upload von Bildern und beerbt dabei den AbstractFilesService.

Die Bilder selbst werden im Ordner src/main/webapp/resources/data/galery gespeichert.

Verwendete JSPs sind zum einen die galery.jsp, die über den Link /galery/ abgerufen wird. Um Bilder hochzuladen ist die galeryImgUpload.jsp mit dem Upload-Formular ausgestattet und kann über den Link /galery/upload/ geladen werden.

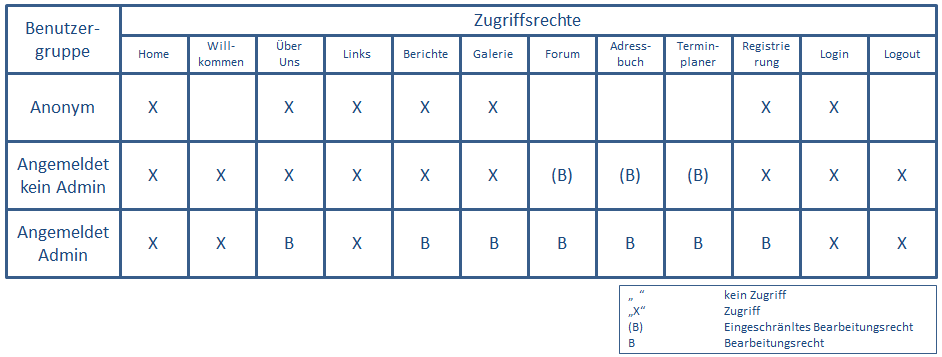
**Zugriffsrechte und Editieren von Content**

Die Website gliedert sich in einen öffentlichen und einen authorisierten Bereich, der je nach Login-Status der Besuchers zugreifbar ist. Die öffentlichen Seiten (siehe Anforderungen) sind für alle sichtbar, wohingegen nur angemeldete Nutzer den authorisierten Bereich der Website einsehen dürfen. Dazu zählen zum Beispiel das Forum oder das Adressbuch.

Zusätzlich besitzen die User der Seite verschiedene Rechte – sie lassen sich aufteilen in Administratoren und normale User.

Administratoren (gehören zur UserGroup USER\_GROUP\_ADMIN) besitzen gegenüber normalen Usern die Rechte, Content zu verfassen, ihn zu editieren oder zu löschen und neue Registrierungszulassungen zu vergeben.

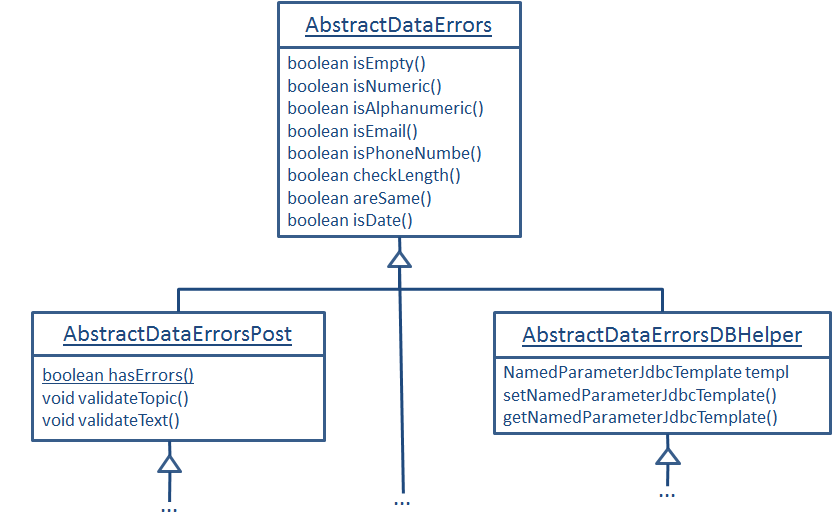
User ohne Administratorrechte (sie gehören zu UserGroup USER\_GROUP\_NO\_ADMIN) können ausschließlich ihre eigenen Daten (wie z.B. ihre Adresse) und selbst verfasste Kommentare zu Foreneinträgen löschen.

Eine spezifischere Einteilung der User sahen wir als nicht nötig an, da die Userzahl der Fussballseite relativ überschaubar bleibt.

Genaueres zur Umsetzung der Zugriffsrechte ist unter dem Punkt „Sicherheit“ erläutert.

**Inputvalidierung und Fehlermeldungen**

Zur Überprüfung der Eingaben, die ein Nutzer auf der Website machen kann, stehen im Package src/main/java/fhwedel/medienprojekt/fussball/service/dataErrors DataError-Klassen zur Verfügung. Alle Services zum Erkennen von Eingabefehlern beerben die AbstractDataErrors, welche allgemeine Prüfmethoden (wie isEmtpy() oder isEmail()) enthält. Als weitere Abstrakte Klasse kann auch die AbstractDataErrorsDBHelper.java beerbt werden, die ein JdbcTemplate und die Funktoin inDB() zur Verfügung stellt, um zu prüfen, ob ein bestimmter Datensatz bereits in eienr Tabelle vorhanden ist. Die Klassen zur Validierung der Foreneinträge, des Spielberichte und der Kommentare basieren auf der AbstractDataErrorsPost.



Abstrakte

Klassen

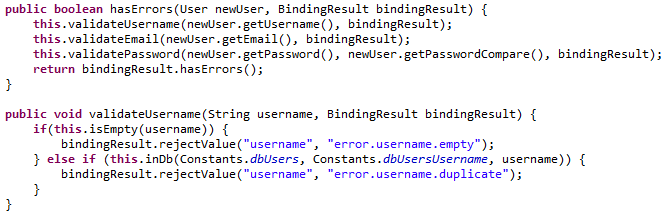
der

Fehler-

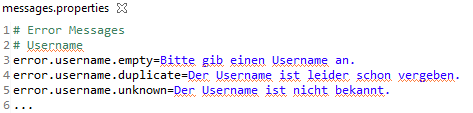
behandlung

Die hasErrors()-Methoden sind allgemein so aufgebaut, dass sie das neue oder bearbeitete jeweilige Objekt (z.B. User oder ForumEntry) und ein BindingResult als Parameter erhalten. Das BindingResult dient hier dazu, Fehlermeldungen bekannt zu machen und diese später in den entsprechenden JSPs anzuzeigen.

Zur Verdeutlichung ein Auszug aus der DataErrorsUsers:



Die Funktion rejectValue() teilt den Parametern des Objektes Fehlermeldungen zu, die über einen Key zu Strings aufgelöst werden. Diese Keys sind in einer Properties-File im Ordner src/main/resources/messages/ zu finden und werden über die org.springframework.context.support.ResourceBundleMessageSource, die in der mvc-config.xml als Bean definiert und so in den ApplicationContext geladen wird, aufgelöst.



So wir beispielsweise rejectValue("username", "error.username.empty") zur Fehlermeldung "Bitte gib einen Usernamen an." ausgewertet und in der JSP angezeigt.

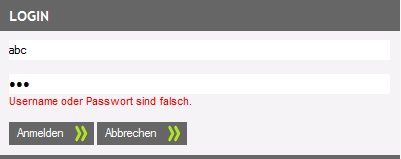
Zu welchem Feld die Fehlermeldung gehört, wird über den ersten Parameter der rejectValue()-Methode angegeben, der den Pfad des Objektes bezieht.



Liegt ein Fehler für einen Pfad vor, wird das <sf:error>-Element, das ebenfalls den Pfad referenziert (angegeben über das path-Attribute), als <span>-Element mit der Fehlermeldung als Inhalt dargestellt und über CSS rot eingefärbt, ansonsten fällt dieses Element beim Rendern der Seite weg.

Im rechten Beispiel ist ein Fehlerfeld für beide Inputfelder vorhanden (referenziert über „\*“), um keine näheren Informationen zu den Logindaten zu geben.

Das Absenden des Login-Formulars mit leere Eingaben resultiert beispielsweise in folgender Fehlermeldung:



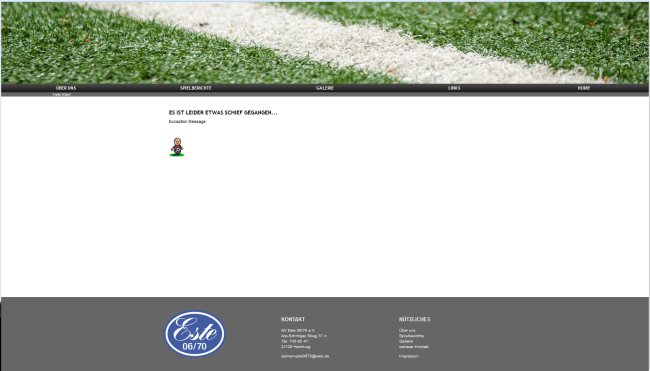
Um die Anzeige der Fehler zu garantieren, muss nun noch dafür gesorgt werden, dass das Objekt, das über ein Eingabeformular gefüllt wurde, und die darauf zugeordneten Fehler in der JSP auch wieder zugreifbar sind. Hierfür kann die @ModelAttribute(<name>) Annotation verwendet werden:



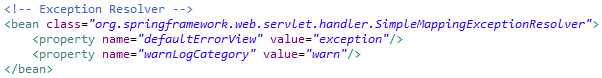
Neben den Fehlermeldungen sind dann auch die vorher eingetragenen Werte zugreifbar, was zum Beispiel beim Eintragen eines neuen Spielberichts absichert, dass die vorher getätigten Eingaben nicht verloren gehen.

**Auftreten von Exceptions**

Sollten während des Besuchs der Seite Exceptions auftreten, wird die exception.jsp mit der entsprechenden Exception Message angezeigt. Die Seite selbst ist ebenfalls im vorherrschenden Design erstellt und enthält anstelle des Contents die Meldung, etwas sei schief gegangen, gefolgt von der Exception Message sowie einem animierten GIF, bei dem ein Fussballer „den Kopf verliert“.



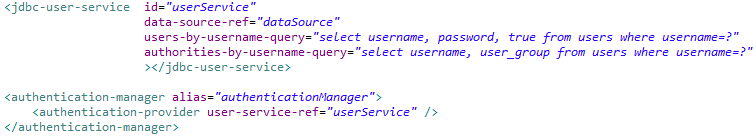


Um dafür zu sorgen, dass diese Excpetion-Seite angezeigt wird, sind folgende Konfigurationen in der mvc-config.xml notwendig:

Der SimpleMappingExceptionResolver greift anschließend bei Exceptions, löst den Wert „exception“ über den InternalViewResolver (siehe „Framework und wichtige Komponenten“) zu /WEB-INF/jsp/exception.jsp auf und bringt diese zur Anzeige.

Sicherheit

Das Spring Framework bringt als weiteres Feature die Spring Security (siehe <http://projects.spring.io/spring-security/>) mit. Auch in unserer Webanwendung findet die Spring Security Einsatz, um zum Beispiel die Zugriffsrechte auf Seiten oder Inhalte auf den Seiten umzusetzen.

Grundlage für den Einsatz der Sicherheits-Features sind die Konfigurationen der security.xml:

Wichtigster Bestandteil ist hier der AuthenticationManager, der die Authentifikation von Usern ermöglicht und diese organisiert. Als UserService verwenden wir in unserer Applikation einen JdbcUserService, der dazu dient Nutzer gegen eine Datenbank zu Authentifizieren. Zur Konfiguration für den User Service sind neben der dataSource, die die Datenbankverbindung repräsentiert, die SQL-Abfrage über die Attribute user-by-username-query und authorities-by-username-query angegeben, mit denen die Login-Daten und zugesprochenen Rollen der User abgefragt werden.

Um nun dafür zu sorgen, dass Authentifikationen geprüft werden sind noch weitere Einstellungen nötig (im Folgenden ein verkürzter, beispielhafter Auszug):

Im <http>-Element sind weitere Angaben möglich. Das auto-config-Attribut ermöglicht Konfigurationen zur Automatisierung von Login und Logout, wohingegen das use-expressions=“true“ dafür sorgt, dass beim Abfragen der Rollen oder Authentifikationen Ausdrücke verwendet werden können (s.u.). Die <form-login> und <logout> Elemente definieren auf welche URLs reagiert werden soll. Über das <intercept-url>-Element kann festgelegt werden, welche Seiten über http abgerufen werden können, und welche hingegen zwingend über https laufen müssen.

Um die Zugriffsrechte einzugrenzen kann auf zwei Arten vorgegangen werden: zum einen können in der XML selbst Zugriffsrechte für gesamte Seiten über das intercept-url-Attribut angegeben, oder in der JSP selbst mit Hilfe des <security:authorize>-Elements Teile der Seite eingegrenzt werden.

Um den Zugriff auf eine gesamte Seite einzugrenzen können im access-Attribut des <intercept-url>-Elements Ausdrücke verwendet werden, die z.B. mit isAuthenticated() abfragen ob ein User angemeldet ist, oder über die hasRole()-Funktion bestimmte Benutzergruppen vorausgesetzt werden:

Um einzelne Bestandteile einer JSP einzugrenzen können im <security:authorize>-Element die gleichen Abfragen über das acces-Attribut getätigt werden. Die so gesetzten Zugriffsrechte beziehen sich dann auf den Inhalt des <security:authorize>-Elements:

So wird beispielsweise auf der Login-Seite für bereits angemeldete User die Meldung „Du bist bereits angemeldet“, anstelle des Login-Formulars, angezeigt.

Um anzugeben, welche Seiten über „http“ und welche über „https“ abgerufen werden sollen, kann das <intercept-url>-Element außerdem wie folgt verwendet werden:



Das pattern-Attribut gibt an, zu welcher URL Einstellungen vorgenommen werden. Ergänzend kann das require-channel-Attribut verwendet werden, um zu erzwingen, dass „http“ oder „https“ verwendet werden soll.

Weiterführende Aufgaben

Fazit