**Dokumentation**

**Entwicklung einer Webanwendung**

**für ein Ticketsystem**

**in der Fiducia & GAD IT AG**

Dokumentation für das 6. Semester

Abgabe am 31.03.2017

David Feldhoff

Moorstraße 88a

48431 Rheine-Mesum

Leon Stapper

Buchdahlstr. 6

48429 Rheine

Hochschule Weserbergland

Studiengang: Wirtschaftsinformatik

Studiengruppe: WI 44/14

Betreuender Dozent: Herr Malte Wildt

1. Inhaltsverzeichnis

[I Inhaltsverzeichnis I](#_Toc476910452)

[II Abkürzungsverzeichnis III](#_Toc476910453)

[III Abbildungsverzeichnis IV](#_Toc476910454)

[IV Tabellenverzeichnis IV](#_Toc476910455)

[1 Kurzbeschreibung der Anwendung 1](#_Toc476910456)

[2 Architekturbeschreibung 1](#_Toc476910457)

[2.1 Oberfläche 1](#_Toc476910458)

[2.2 Services 1](#_Toc476910459)

[2.3 Persistierung 1](#_Toc476910460)

[3 User Stories 1](#_Toc476910461)

[3.1 Anwender 1](#_Toc476910462)

[3.1.1 Veranstaltung suchen 1](#_Toc476910463)

[3.1.2 Details zur Veranstaltung einsehen 1](#_Toc476910464)

[3.1.3 Ticketreservierung 1](#_Toc476910465)

[3.1.4 Buchungscode zu einer Reservierung einsehen 1](#_Toc476910466)

[3.1.5 Einsicht der noch zur Verfügung stehenden Tickets 1](#_Toc476910467)

[3.2 Manager 1](#_Toc476910468)

[3.2.1 Login 1](#_Toc476910469)

[3.2.2 Veranstaltung anlegen 1](#_Toc476910470)

[3.2.3 Veranstaltung veröffentlichen 1](#_Toc476910471)

[3.2.4 Bearbeitung einer unveröffentlichten Veranstaltung 1](#_Toc476910472)

[3.2.5 Einsicht aller Reservierungen 1](#_Toc476910473)

[4 Sprint Backlog 1](#_Toc476910474)

[5 Darstellung des Datenmodells 1](#_Toc476910475)

[5.1 Klassendiagramm 1](#_Toc476910476)

[V Anhangsverzeichnis V](#_Toc476910417)

[VI Anhang A-1](#_Toc476910418)

1. Abkürzungsverzeichnis

|  |  |
| --- | --- |
| **Abkürzung** | **Bedeutung** |
| JPA | Java Persistence API |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

1. Abbildungsverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

[Abbildung 3: Mockup\_Widgetkatalog A-1](#_Toc476910419)

1. Tabellenverzeichnis

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**

# Kurzbeschreibung der Anwendung

Das Ticketsystem namens "eventon" ist eine Webanwendung zur Verwaltung und Buchung von Tickets. Auf der Startseite von "eventon" können Veranstaltungen gesucht werden. Bei erfolgreicher Suche wird eine Liste der gefundenen Veranstaltungen mit einigen Informationen angezeigt. Für nähere Informationen oder für eine Buchung dieser Veranstaltung kann die Veranstaltung in der Tabelle angeklickt werden. Hierdurch gelangt man zur Info- und Buchungsseite. Nach Eingabe der gewünschten Ticketanzahl wird dem Nutzer, sofern er eingeloggt ist und seine Eingabedaten korrekt waren, der Buchungscode zu dieser Buchung angezeigt. Zudem kann der Nutzer über die Navigationsleiste am Kopf der Seite sein Nutzerprofil auswählen und seine bereits getätigten Buchungen einsehen. Im Nutzerprofil können zudem persönliche Nutzerdaten bearbeitet werden.

Falls der Nutzer der Anwender noch kein Konto bei "eventon" besitzt, kann er zwar alle (nicht nutzerbezogenen) Informationen einsehen, aber keine Buchungen tätigen. Zur Erstellung eines Kontos kann der Nutzer sich über den entsprechenden Link in der Navigationsleiste registrieren. Hierbei kann auch ausgewählt werden, ob der Nutzer ein Manager ist oder nicht.

Manager haben dabei einige zusätzliche Funktionen. Sie können ein Event erstellen. Bei der Erstellung einer Veranstaltung müssen Daten zur Beschreibung, zur Anzahl der verfügbaren Plätze, zu Preisen sowie zur Adresse der Veranstaltung angegeben werden. Darüber hinaus kann ein Bild hochgeladen werden. Dieses Event kann entweder nur gespeichert oder auch direkt veröffentlicht werden. Noch nicht veröffentlichte Events können über einen entsprechenden Link in der Navigationsleiste bearbeitet werden. Alle veröffentlichten Events können ebenfalls über die Navigationsleiste eingesehen werden. Außerdem können hierzu Informationen zur den Ticketverkäufen aufgerufen werden.

Sämtliche dem Manager angezeigten Informationen zu Veranstaltungen beziehen sich dabei ausschließlich auf die persönlich erstellten und verwalteten Veranstaltungen. Veranstaltungen anderer Manager können zwar gebucht, aber nicht verwaltet werden.

# Architekturbeschreibung

## Allgemeines

Die Anwendung "eventon" wurde als Model-2-Architektur realisiert. Eine Model-2-Architektur besteht aus grafischer Darstellung, Logik und Datenhaltung. Daher unterteilt sich die Anwendung in mehrere Schichten. Die erste Schicht, bestehend aus xhtml-Dateien, realisiert die grafische Darstellung. Die zweite Schicht besteht aus .java-Klassen und bildet die Steuerung der Anwendung ab. Die Klassen dieser Schicht werden im Folgenden Controller genannt, da diese Anfragen entgegennehmen, diese validieren und und an die Logik- und Persistierungsschicht weiterleiten. Diese besteht ebenfalls aus .java-Klassen, die den Zugriff auf die Datenbank realisieren und die Anwendungslogik abbilden.

Übergreifend existiert darüber hinaus noch ein SessionContext, der Informationen zur derzeitigen Session beinhaltet.

Bei der Implementierung der Schichten wurde strikt darauf geachtet, dass tiefer gelegene Schichten keine Aufrufe auf höher gelegenen Schichten tätigen können.

## xhtml-Dateien für die Oberfläche

Die Oberfläche wurde ausschließlich in den xhtml-Dateien realisiert. Dabei basieren alle Seiten auf der Datei base.xhtml. Diese beinhaltet den grundlegenden Aufbau der Seite, bestehend aus einer Navigationsbar und dem Seiteninhalt. Dementsprechend wurde dient die base-Seite als grundlegende Struktur für das Templating. Zur Gestaltung der Oberfläche wurde das CSS-Framework "bootstrap" eingesetzt. Dieses wurde ebenfalls über die Templating-Seite eingebunden, um eine Verfügbarkeit über die gesamte Anwendung hinweg zu gewährleisten ohne auf jeder Seite "bootstrap" erneut einbinden zu müssen.

Bei der Entwicklung der Oberflächen-Dateien wurde strikt darauf geachtet, dass die xhtml-Dateien ausschließlich mit den vorgesehenen Logik-Klassen kommunizieren. Eine Kommunikation mit den Service-Klassen der Persistierungsschicht wurde bewusst vermieden, um die Verantwortlichkeiten der einzelnen Klassen strikt zu trennen.

Das Databinding zwischen xhtml-Dateien und den javaseitigen Controllern erfolgt über die Namensgebung der abgebildeten Attribute.

## Java-Forms für die Controller

Die Steuerung der Anwendung besteht aus javaseitigen Klassen, welche als Controller bezeichnet werden. Der Name dieser Klassen endet immer mit "Form".

Für das beschriebene Databinding zwischen den Form-Klassen und den xhtml-Dateien wurden ausschließlich einzelne Attribute genutzt. Für die Bearbeitung von Attributen eines Objektes wurden zunächst alle Objektattribute in die Form-Attribute übertragen. Bei Abschluss der Bearbeitung wurden die Form-Attribute ins Objekt übernommen. Durch diesen Umweg sollte gewährleistet werden, dass Objekte ausschließlich aus der Form bearbeitet werden können.

Falls eine Konvertierung oder Validierung des einzelnen Eingabewertes notwendig war, erfolgte dies mit Hilfe von Convertern und Validatoren, die über entsprechende jsf-Attribute bzw. jsf-Tags an die Eingabewidgets angebunden wurden. Validierungen des gesamten erstellten Objektes erfolgen wiederum in der Logik, da die Validatoren lediglich die Eingabewerte einzelner Felder übergeben bekommen. Eine Ermittlung sämtlicher Eingabewerte wurde als zu umständlich angesehen.

Bei Abschluss einer Aufgabe, die Werte verändert, gelöscht oder neu erstellt hat, wurden aus den Forms entsprechende Methoden aus der Logik- und Persistierungsschicht aufgerufen. Dessen Services sind dabei mit Interfaces in den Form-Klassen hinterlegt und werden mittels Dependency-Injection angezogen.

## Service-Interfaces

Wie bereits im vorherigen Kapitel beschrieben, ist den Service-Klassen der Logik- und Persistierungsschicht ein Interface vorgeschaltet, sodass nie direkt mit den Service-Klassen kommuniziert wird, sondern lediglich deren Schnittstelle verwendet wird. Durch die dadurch entstehende Abstraktion sollte eine möglichst lose Kopplung zwischen Controllern und Logik inklusive Datenhaltung gewährleistet werden.

## Services für die Datenhaltung und Persistierung

Die Services zur Realisierung der Anwendungslogik inklusive der Datenhaltung und Persistierung implementieren die zuvor beschriebenen Interfaces. Die Services beziehen über Dependency-Injection eine Instanz des JPA-EntityManagers zur Kommunikation mit der Datenbank. Darüber können Objekte in der Datenbank erstellt, bearbeitet und gelöscht werden. Außerdem können hierüber Objekte ausgelesen werden. Dafür wurden unter anderem auch kriterienbasierte Suchanfragen mit Hilfe des "CriteriaBuilders" erstellt.

Nicht alle Services haben eine Anbindung zur Datenbank. Einige sind auf die Anwendungslogik beschränkt und liefern Informationen, für die keine Datenbankzugriffe erforderlich sind. Ein Beispiel hierfür ist der NavigationService, der die Navigation innerhalb der gesamten Webanwendung von zentraler Stelle steuert.

## Session-Informationen über den SessionContext

Zusätzlich zu den Informationen, die in der Datenbank persistiert werden, wurde ein SessionContext eingerichtet. Dieser ist über die gesamte Anwendung hinweg zugänglich und dementsprechend nicht Teil der Datenhaltung. Im SessionContext werden Informationen zur derzeitigen Session gespeichert. Dies umfasst ausschließlich den derzeitigen User. Über den SessionContext kann dieser abgefragt werden. Der Zugriff auf den SessionContext erfolgt unter anderem auch aus den xhtml-Dateien.

## Models

Die Speicherung der Daten zu den Attributen erfolgt in den Models. Diese liegen im Package "de.eventon.core". Die Models umfassen die User, die Veranstaltungen, die Buchungen, Adressen sowie Bankkonten.

# User Stories

## Anwender

### Veranstaltung suchen

**User-Story**: Als Anwender möchte ich nach einer Veranstaltung suchen können.

**Anwendungsschritte**:

1. Aufruf der Webanwendung über "localhost:8080/Eventon"
2. Eingabe des Suchbegriffes auf der Seitenmitte (bspw. "Kraftklub")
3. Klick auf den Button "Suche"

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| Index.xhtml | Darstellung der Startseite, Anzeige der Suchergebnisse; Triggern der Suchfunktion; Verlinkung zur Event-Detail-Seite | Oberfläche |
| EventSearchForm.java | Aufruf der Suche im Service; Bereitstellung der Suchergebnisse für die index.xhtml | Controller |
| IsEventService.java | Interface für den EventService | Service-Interface |
| EventService.java | Kriterienbasierte Suche der Events mit dem übergebenen Suchparameter in der Datenbank (nur veröffentlichte und noch nicht in der Vergangenheit liegende Events) | Service für Logik und Datenhaltung |
| NavigationService.java | Navigation nach der Suche (in diesem Fall: auf der Seite verweilen) | Service für Logik und Datenhaltung |
| Event.java | Objekt zur Speicherung von Veranstaltungsdaten | Model |

### Details zur Veranstaltung einsehen

**User-Story**: Als Anwender möchte ich die Details (Name, Beschreibung, Datum & Uhrzeit) ansehen können.

**Anwendungsschritte**:

1. Durchführung einer Suche (analog zu 3.1.1)
2. Klick auf den Namen der gewünschten Veranstaltung

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| event.xhtml | Darstellung der Veranstaltungsdetails; Verknüpfung der Converter und Validatoren | Oberfläche |
| EventBookingForm.java | Initialisierung der Darstellung über PostConstruct-Methode; Bezug des Events über EventService; Weiterleitung auf ErrorPage (sofern das Event nicht existiert), | Controller |
| IsEventService.java | Interface für den EventService | Service-Interface |
| EventService.java | Bezug des Events aus der Datenbank über die ID des Events | Service für Logik und Datenhaltung |
| NavigationService.java | Weiterleitung auf ErrorPage, falls das Event nicht existiert | Service für Logik und Datenhaltung |
| Event.java | Objekt zur Speicherung von Veranstaltungsdaten | Model |

### Ticketreservierung

**User-Story**: Als Anwender möchte ich mit meinem Namen zu einer Veranstaltung eine beliebige Anzahl Tickets reservieren können.

**Anwendungsschritte**:

1. Registrierung eines Nutzers über die Navigationsleiste
2. Login mit den Nutzercredentials über die Navigationsleiste
3. Durchführung einer Suche mit anschließender Detailansicht (analog zu 3.1.2)
4. Eingabe der gewünschten Ticketanzahl
5. Klick auf "Buchen"

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| event.xhtml | Darstellung der Veranstaltungsdetails; Verknüpfung der Converter und Validatoren; Triggern der Buchungsmethode; Anzeige der UUID nach erfolgreicher Buchung | Oberfläche |
| AmountTicketValidator | Überprüfung der Anzahl der gewünschten Tickets (nicht negativ) | Zwischen Oberfläche und Controller |
| EventBookingForm.java | Initialisierung der Darstellung über PostConstruct-Methode; Bezug des Events über EventService; Weiterleitung auf ErrorPage (sofern das Event nicht existiert); Validierung der Buchung (weiterhin genügend Tickets vorhanden?, Event noch in der Zukunft); Aufruf des EventBookingServices | Controller |
| SessionContext | Dient in diesem Fall zur Überprüfung, ob der Nutzer zur Buchung eingeloggt ist | Session (übergreifend) |
| EventService.java | Bezug des Events aus der Datenbank über die ID des Events | Service für Logik und Datenhaltung |
| EventBookingService.java | Erstellung der Buchung; Persistierung der Buchung in der Datenbank; Rückgabe der UUID | Service für Logik und Datenhaltung |
| NavigationService.java | Weiterleitung auf ErrorPage, falls das Event nicht existiert; Weiterleitung auf Login, falls der Nutzer noch nicht angemeldet ist; Rücknavigation auf Buchungsseite nach der Anmeldung | Service für Logik und Datenhaltung |
| Event.java | Objekt zur Speicherung von Veranstaltungsdaten | Model |
| User.java | Objekt zur Speicherung von Benutzerdaten | Model |

### Buchungscode zu einer Reservierung einsehen

**User-Story**: Als Anwender möchte ich bei der Reservierung einen eindeutigen Reservierungscode erhalten.

**Anwendungsschritte:**

1. Durchführung einer Buchung (analog zu )
2. Nach Abschluss der Buchung wird der Buchungscode eingeblendet

**Komponenten analog zu 3.1.3**

### Einsicht der noch zur Verfügung stehenden Tickets

**User-Story**: Als Anwender möchte ich zu einer Veranstaltung sehen, wie viele Tickets noch zu reservieren sind.

**Anwendungsschritte:**

1. Durchführung einer Suche (analog zu 3.1.1)
2. Nach Abschluss der Suche wird die Anzahl der Tickets in der Tabelle angezeigt

**Komponenten analog zu 3.1.1**

## Manager

### Login

**User-Story:** Übergreifend über mehrere User-Stories, welche einen Login erfordern

**Anwendungsschritte:**

1. Voraussetzung: Ein Nutzer ist registriert
2. Klick auf "Login" in der Navigationsleiste
3. Eingabe der Nutzercredentials
4. Klick auf Login

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| login.xhtml | Darstellung der Loginseite; Triggern der Loginfunktion | Oberfläche |
| UserConverter.java | Konvertierung der Email-Adresse in einen Nutzer, falls dieser vorhanden ist | Zwischen Oberfläche und Controller |
| PasswordConverter.java | Verschlüsselung des Passwortes mit SHA-256 | Zwischen Oberfläche und Controller |
| LoginForm.java | Aufruf des Logins im LoginService | Controller |
| IsLoginService.java | Interface für den LoginService | Service-Interface |
| LoginService.java | Überprüfung der Usercredentials; ggf. LoginException werfen; Setzen des Nutzers im SessionContext | Service für Logik und Datenhaltung |
| NavigationService.java | Navigation nach dem Login auf Home oder auf die letzte signifikante Seite (Buchung) | Service für Logik und Datenhaltung |
| User.java | Objekt zur Speicherung von Nutzerdaten | Model |

### Veranstaltung anlegen

### Veranstaltung veröffentlichen

### Bearbeitung einer unveröffentlichten Veranstaltung

### Einsicht aller Reservierungen

# Sprint Backlog

|  |  |
| --- | --- |
| Aufgabe | Bearbeiter |
| Projekt aufsetzen (gradle, bootstrap) | Leon Stapper |
| Mockups erstellen | Leon Stapper |
| Templating | Leon Stapper |
| Navigation | Leon Stapper |
| EventService zur Event-Verwaltung | Leon Stapper |
| UserService zur User-Verwaltung | Leon Stapper |
| Registrierung | David Feldhoff |
| Login | Leon Stapper |
| Events erstellen und bearbeiten | Leon Stapper |
| Bilderupload | David Feldhoff |
| Verwaltung unveröffentlichte Events | David Feldhoff |
| Verwaltung veröffentlichte Events | David Feldhoff |
| Event-Suche | Leon Stapper |
| Event anzeigen buchen | Leon Stapper |
| User-Übersicht | David Feldhoff |
| Regex-Entwicklung zur Validierung | David Feldhoff |
| ER-Modell erstellen | David Feldhoff |
|  |  |
|  |  |
|  |  |

# Darstellung des Datenmodells

## Klassendiagramm

Anhangsverzeichnis

[A1 Mockups A-1](#_Toc476910383)

A

Anhang

1. Mockups

Im Folgenden finden sich die Oberflächenentwürfe bzw. Mockups zum Ticketsystem wieder.

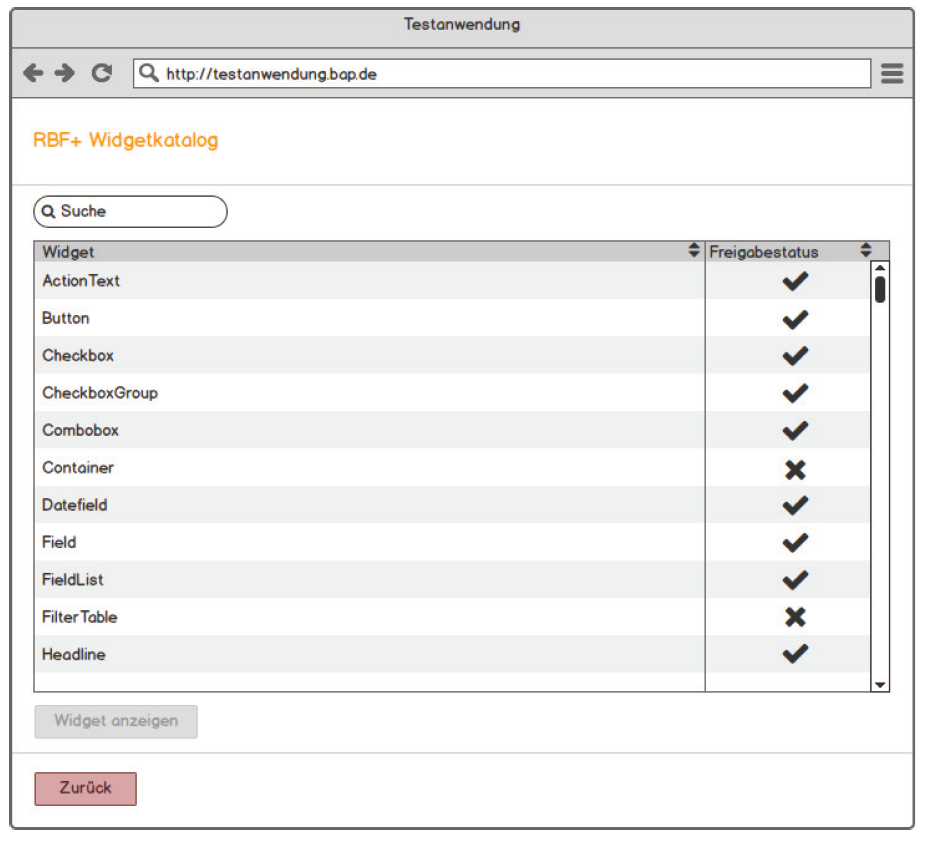


Abbildung : Mockup\_Widgetkatalog

**Eigenständigkeitserklärung**

"Wir versichern hiermit, dass wir die Arbeit selbstständig verfasst, keine anderen Quellen und Hilfsmittel als die angegebenen benutzt und die Stellen der Arbeit, die anderen Werken dem Wortlaut oder dem Sinn nach entnommen sind, in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht haben. Das Gleiche gilt auch für eingefügte Zeichnungen, Kartenskizzen und Darstellungen.”

|  |  |
| --- | --- |
| Hameln, den 29.03.2017 |  |
| Ort, Datum | Unterschrift |

|  |  |
| --- | --- |
| Hameln, den 29.03.2017 |  |
| Ort, Datum | Unterschrift |