**Dokumentation**

**Entwicklung einer Webanwendung**

**für ein Ticketsystem**

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](#_Toc478685128)

[II Abkürzungsverzeichnis III](#_Toc478685129)

[III Abbildungsverzeichnis IV](#_Toc478685130)

[1 Kurzbeschreibung der Anwendung 1](#_Toc478685131)

[2 Architekturbeschreibung 2](#_Toc478685132)

[2.1 Allgemeines 2](#_Toc478685133)

[2.2 xhtml-Dateien für die Oberfläche 2](#_Toc478685134)

[2.3 Java-Forms für die Controller 3](#_Toc478685135)

[2.4 Service-Interfaces 3](#_Toc478685136)

[2.5 Services für die Datenhaltung und Persistierung 3](#_Toc478685137)

[2.6 Datenbank (h2) 4](#_Toc478685138)

[2.7 Models 4](#_Toc478685139)

[2.8 Session-Informationen über den SessionContext 4](#_Toc478685140)

[3 User Stories 5](#_Toc478685141)

[3.1 Anwender 5](#_Toc478685142)

[3.1.1 Veranstaltung suchen 5](#_Toc478685143)

[3.1.2 Details zur Veranstaltung einsehen 5](#_Toc478685144)

[3.1.3 Ticketreservierung 6](#_Toc478685145)

[3.1.4 Buchungscode zu einer Reservierung einsehen 7](#_Toc478685146)

[3.1.5 Einsicht der noch zur Verfügung stehenden Tickets 7](#_Toc478685147)

[3.2 Manager 7](#_Toc478685148)

[3.2.1 Login 7](#_Toc478685149)

[3.2.2 Veranstaltung anlegen 8](#_Toc478685150)

[3.2.3 Veranstaltung veröffentlichen 9](#_Toc478685151)

[3.2.4 Bearbeitung einer unveröffentlichten Veranstaltung 10](#_Toc478685152)

[3.2.5 Einsicht aller Reservierungen 11](#_Toc478685153)

[4 Sprint Backlog 12](#_Toc478685154)

[V Anhangsverzeichnis V](#_Toc476910417)

[VI Anhang A-1](#_Toc476910418)

1. Abkürzungsverzeichnis

|  |  |
| --- | --- |
| **Abkürzung** | **Bedeutung** |
| JPA | Java Persistence API |
| JSF | Java Server Faces |
| UUID | Universally Unique Identifier |

1. Abbildungsverzeichnis

[Abbildung 1: Mockup\_Suche\_und\_Anzeige A-1](#_Toc478684967)

[Abbildung 2: Mockup\_Event\_Anzeige\_Buchen A-2](#_Toc478684968)

[Abbildung 3: Mockup\_Event\_gebucht A-3](#_Toc478684969)

[Abbildung 4: Mockup\_Login A-4](#_Toc478684970)

[Abbildung 5: Mockup\_Event\_anlegen A-5](#_Toc478684971)

[Abbildung 6: ER-Modell A-6](#_Toc478684972)

[Abbildung 7: Klassendiagramm A-7](#_Toc478684973)

[Abbildung 8: Schichtenarchitektur A-8](#_Toc478684974)

# 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 Informations- und Buchungsseite. Nach Eingabe der gewünschten Ticketanzahl wird dem Nutzer, sofern er eingeloggt ist, seine Eingabedaten korrekt waren und die Tickets noch zur Verfügung stehen, der Buchungscode zu dieser Buchung angezeigt. Neben der Buchung eines Events 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 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 unter anderem 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 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.

Die Darstellung der Schichten ist im Anhang A3 in der Abbildung 4 grafisch dargestellt.

## xhtml-Dateien für die Oberfläche

Der Aufbau der 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 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 dieses Vorgehen 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, welche einen Wert 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 herangezogen.

## 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 ist eine möglichst lose Kopplung zwischen Controllern und Logik inklusive Datenhaltung gewährleistet.

## 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.

## Datenbank (h2)

Die relationale Datenbank auf die die beschriebenen Services zugreifen ist eine h2-Datenbank, dessen Aufbau im ER-Modell im Anhang in Abbildung 2 dargestellt ist.

## 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, die Adressen der User und der Veranstaltungen sowie die Bankkonten der User. Die dazu notwendigen Klassen, Attribute und Beziehungen werden im Anhang in Abbildung 3 dargestellt.

## 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 aus nahezu allen Schichten, wie in Abbildung 4 zu sehen.

# Installationsanleitung

1. Eventon.war auf den Glassfish-Server deployen
2. Erstellung eines Ordners zur Ablage der Event-Fotos unter "/var/webapp/images" bzw. "C:/var/webapp/images"
3. Den abgelegten Pfad ohne "images" in der Datei "glassfish-web.xml" im "Eventon.war" eintragen
4. h2-Datenbank starten
5. Aufruf der URL "localhost:8080/Eventon/initData.xhtml" zur Initialisierung der Testdaten

# 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

**User-Story:** Als Manager möchte ich nach Login eine neue Veranstaltung anlegen können.

**Anwendungsschritte:**

1. Voraussetzung: Ein Nutzer ist als Manager eingeloggt (3.2.1)
2. Klick auf "Meine Events" 🡪 “Event erstellen“ in der Navigationsleiste
3. Eingabe der Eventdaten
4. Klick auf „Veranstaltung erstellen“

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| createEvent.xhtml | Darstellung der Eventerstellungsseite; Triggern der Anlagefunktion | Oberfläche |
| Validatoren für Address, Event, Datetime | Validierungen auf Basis von Regex, ob der Wert gültig ist bzw. Überprüfung, ob ein Integer-, Double oder Datetime-Wert angegeben wurde | Zwischen Oberfläche und Controller |
| FileUploadValidator.java | Validierung, ob die Datei kleiner 2 MB ist und ob die Datei eine JPEG Datei ist. | Zwischen Oberfläche und Controller |
| Konverter für Event | Konvertierung in den entsprechenden Datentypen (String, Double oder DateTime) | Zwischen Oberfläche und Controller |
| CreateEventForm.java | Letzte Validierungen und anschließender Aufruf der CreateEvent-Methode im EventService bzw. Erzeugung von sprechenden Fehlermeldungen | Controller |
| IsEventService.java | Interface für den EventService | Service-Interface |
| EventService.java | Persistieren des Events | Service für Logik und Datenhaltung |
| NavigationService.java | Navigation nach dem Erstellen des Events ist abhängig davon, ob das Event direkt veröffentlicht wurde: Entweder auf Manageransicht der veröffentlichten oder noch zu bearbeitenden Events | Service für Logik und Datenhaltung |
| Event.java | Objekt zur Speicherung von Eventdaten | Model |

### Veranstaltung veröffentlichen

**User-Story:** Als Manager möchte ich nach Login eine Veranstaltung veröffentlichen können.

**Anwendungsschritte:**

1. Voraussetzung: Ein Nutzer ist als Manager eingeloggt (3.2.1) und ein Event wurde erstellt, welches nicht direkt veröffentlicht wurde (3.2.2).
2. Klick auf "Meine Events" 🡪 “In Bearbeitung“ in der Navigationsleiste.
3. Setzen der Checkboxes in der Spalte „Veröffentlichen“
4. Klick auf den Button „Veröffentlichen“

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| managerOverviewEventsInProcess.xhtml | Darstellung der noch nicht veröffentlichten Events; Triggern der „Veröffentlichen“-Funktion | Oberfläche |
| Konverter für Event | Konvertierung in das entsprechende Anzeigeformat (Währung oder korrekte Datumsanzeige) | Zwischen Oberfläche und Controller |
| ManagerOverviewEventsForm.java | Aufruf der UpdateEvent-Methode im EventService mit gesetztem Publish-Flag | Controller |
| IsEventService.java | Interface für den EventService | Service-Interface |
| EventService.java | Updaten des Events | Service für Logik und Datenhaltung |
| NavigationService.java | Navigation nach dem Veröffentlichen des Events zur „In Bearbeitung“-Seite | Service für Logik und Datenhaltung |
| Event.java | Objekt zur Speicherung von Eventdaten | Model |

### Bearbeitung einer unveröffentlichten Veranstaltung

**User-Story:** Als Manager möchte ich nach Login eine noch nicht veröffentlichte Veranstaltung bearbeiten können.

**Anwendungsschritte:**

1. Voraussetzung: Ein Nutzer ist als Manager eingeloggt (3.2.1) und ein Event wurde erstellt, welches nicht direkt veröffentlicht wurde (3.2.2).
2. Klick auf "Meine Events" 🡪 “In Bearbeitung“ in der Navigationsleiste.
3. Klick auf den Link des Events (Name des Events), um in die Bearbeitungsseite zu gelangen.
4. Verändern der Event-Daten
5. Klick auf den Button „Veranstaltung bearbeiten“

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| managerOverviewEventsInProcess.xhtml | Darstellung der noch nicht veröffentlichten Events; Bietet über Verlinkung Zugang zu der Bearbeitungsseite | Oberfläche |
| Konverter für Event | Konvertierung in das entsprechende Anzeigeformat (Währung oder korrekte Datumsanzeige) | Zwischen Oberfläche und Controller |
| Validatoren für Address, Event, Datetime | Validierungen auf Basis von Regex, ob der Wert gültig ist bzw. Überprüfung, ob ein Integer-, Double oder Datetime-Wert angegeben wurde | Zwischen Oberfläche und Controller |
| FileUploadValidator.java | Validierung, ob die Datei kleiner als 2 MB ist und ob die Datei eine JPEG Datei ist. | Zwischen Oberfläche und Controller |
| ManagerOverviewEventsForm.java | Aufruf der UpdateEvent-Methode im EventService mit geänderten Werten | Controller |
| IsEventService.java | Interface für den EventService | Service-Interface |
| EventService.java | Updaten des Events | Service für Logik und Datenhaltung |
| NavigationService.java | Navigation nach dem Bearbeiten des Events zur „In Bearbeitung“-Seite | Service für Logik und Datenhaltung |
| Event.java | Objekt zur Speicherung von Eventdaten | Model |

### Einsicht aller Reservierungen

**User-Story:** Als Manager möchte ich nach Login eine Liste aller Reservierungen sehen.

**Anwendungsschritte:**

1. Voraussetzung:
   * Ein Nutzer ist als Manager eingeloggt (3.2.1)
   * Event wurde erstellt, welches bereits veröffentlicht ist (3.2.2 oder 3.2.3)
   * Event wurde bereits einmal gebucht
2. Klick auf "Meine Events" 🡪 “Veröffentlicht“ in der Navigationsleiste.
3. Klick auf den Link im Event (auf die Zahl der Spalte „Anzahl Reservierungen“), um die Reservierungen pro Event sich anzusehen

|  |  |  |
| --- | --- | --- |
| Komponente | Aufgabe | Schicht |
| managerOverviewEventsReleased.xhtml | Darstellung der bereits veröffentlichten Events; Bietet über Verlinkung Zugang zu der Ansicht der Reservierungen | Oberfläche |
| managerEventReservations.xhtml | Ansicht der Reservierungen eines Events | Oberfläche |
| Konverter für Event | Konvertierung in das entsprechende Anzeigeformat (Währung oder korrekte Datumsanzeige) | Zwischen Oberfläche und Controller |
| ManagerOverviewEventsForm.java | Aufruf der getEventById Funktion des EventServices und Weitergabe der Reservierungen an die managerEventReservations.xhtml | Controller |
| IsEventService.java | Interface für den EventService | Service-Interface |
| EventService.java | Holt das Event, welches in der URL über die ID übergeben wurde | Service für Logik und Datenhaltung |
| Event.java | Objekt zur Speicherung von Eventdaten | Model |
| Booking.java | Objekt zur Speicherung von Buchungsdaten | Model |

# 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-Profil anzeigen und bearbeiten | David Feldhoff |
| Regex-Entwicklung zur Validierung | David Feldhoff |
| ER-Modell erstellen | David Feldhoff |
| Klassenmodell erstellen | David Feldhoff |
| Schichtenarchitektur erstellen | David Feldhoff |

## 

Anhangsverzeichnis

[A1 Mockups A-1](#_Toc478684882)

[A2 Darstellung des Datenmodells A-6](#_Toc478684883)

[A2.1 ER-Modell A-6](#_Toc478684884)

[A2.2 Klassendiagramm A-7](#_Toc478684885)

[A3 Schichtenarchitektur A-8](#_Toc478684886)

A

Anhang

1. Mockups

Im Folgenden finden sich die ursprünglichen Oberflächenentwürfe bzw. Mockups zum Ticketsystem wieder. An diesen wurde sich bei der Entwicklung orientiert. Allerdings wurden gegebenenfalls noch Anpassungen während der Implementierungsphase vollzogen, die in den Mockups nicht abgebildet sind.

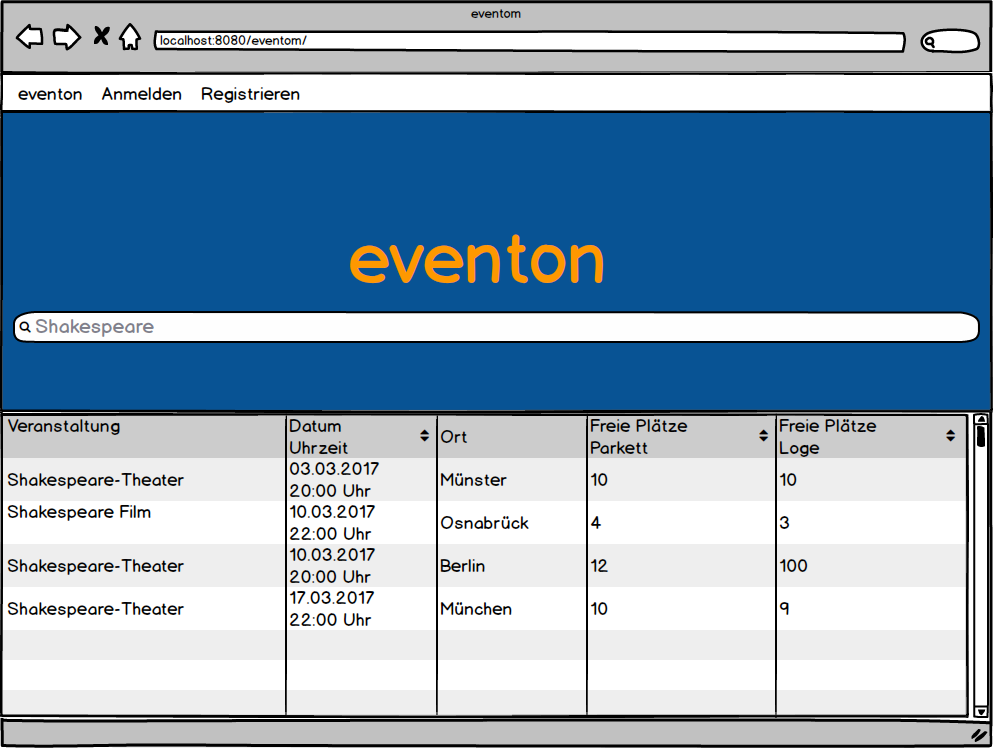


Abbildung : Mockup\_Suche\_und\_Anzeige

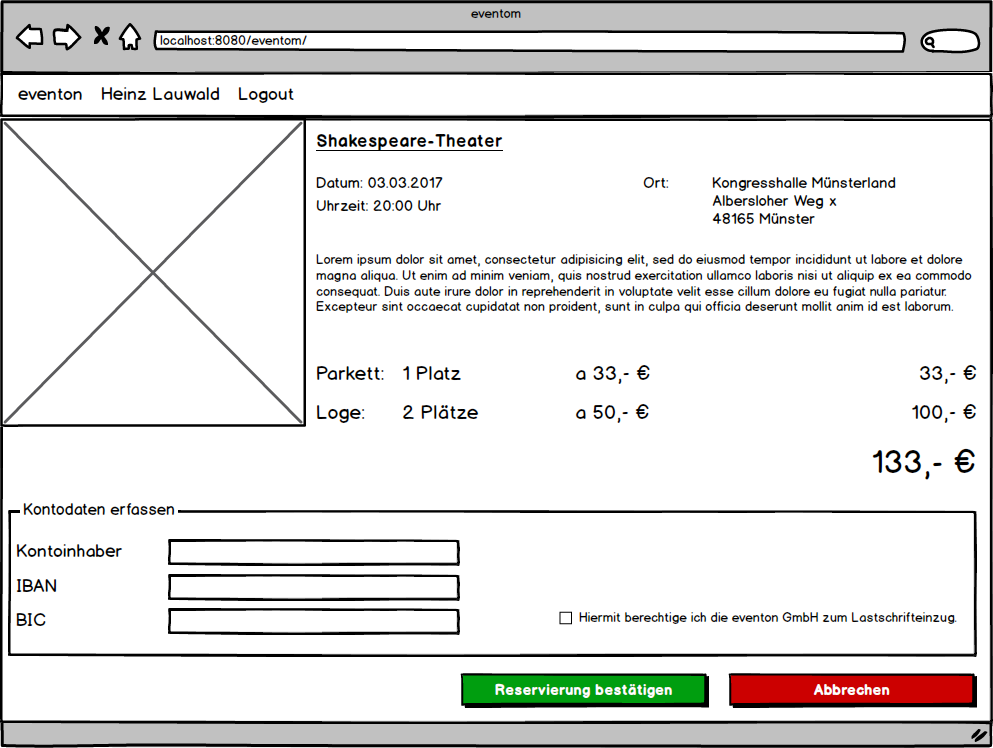


Abbildung : Mockup\_Event\_Anzeige\_Buchen

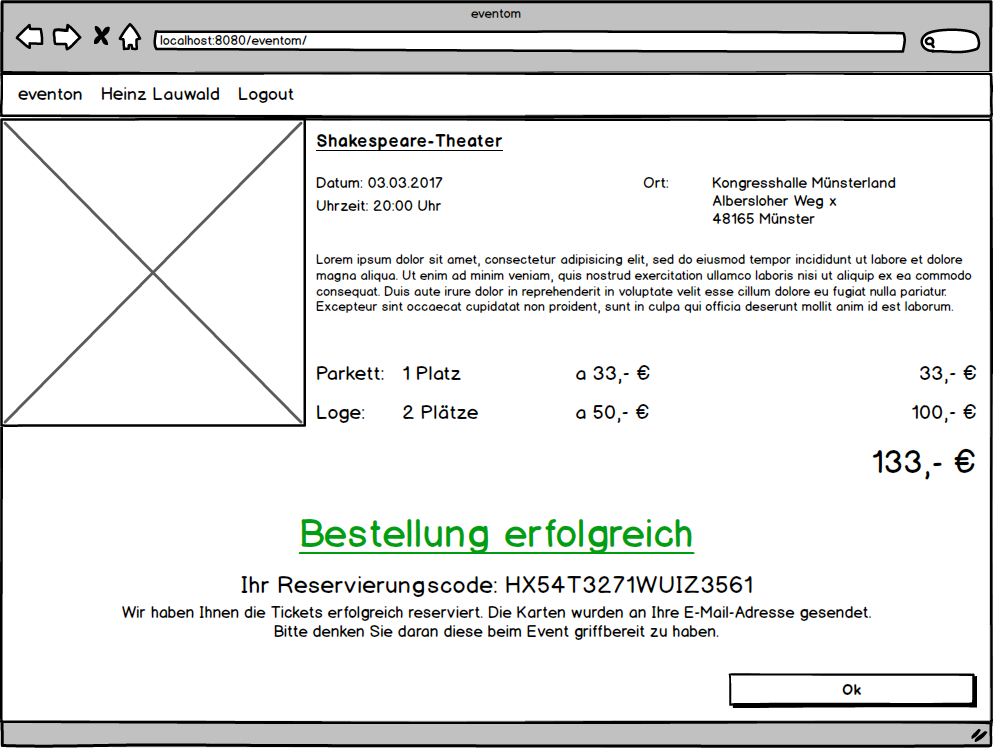


Abbildung : Mockup\_Event\_gebucht

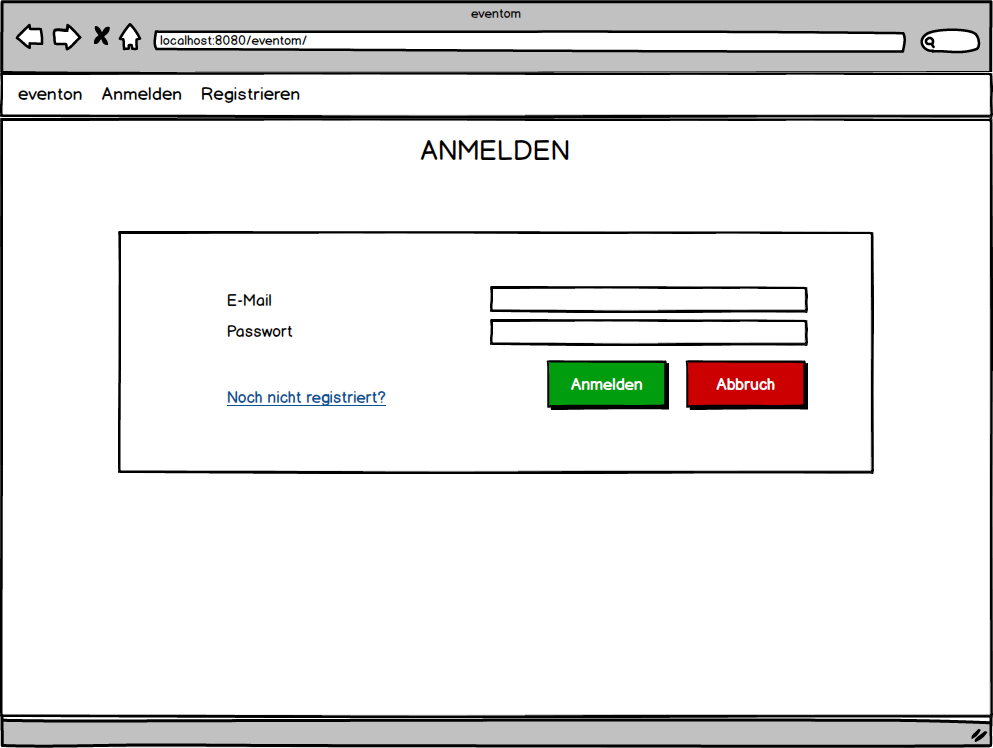


Abbildung : Mockup\_Login

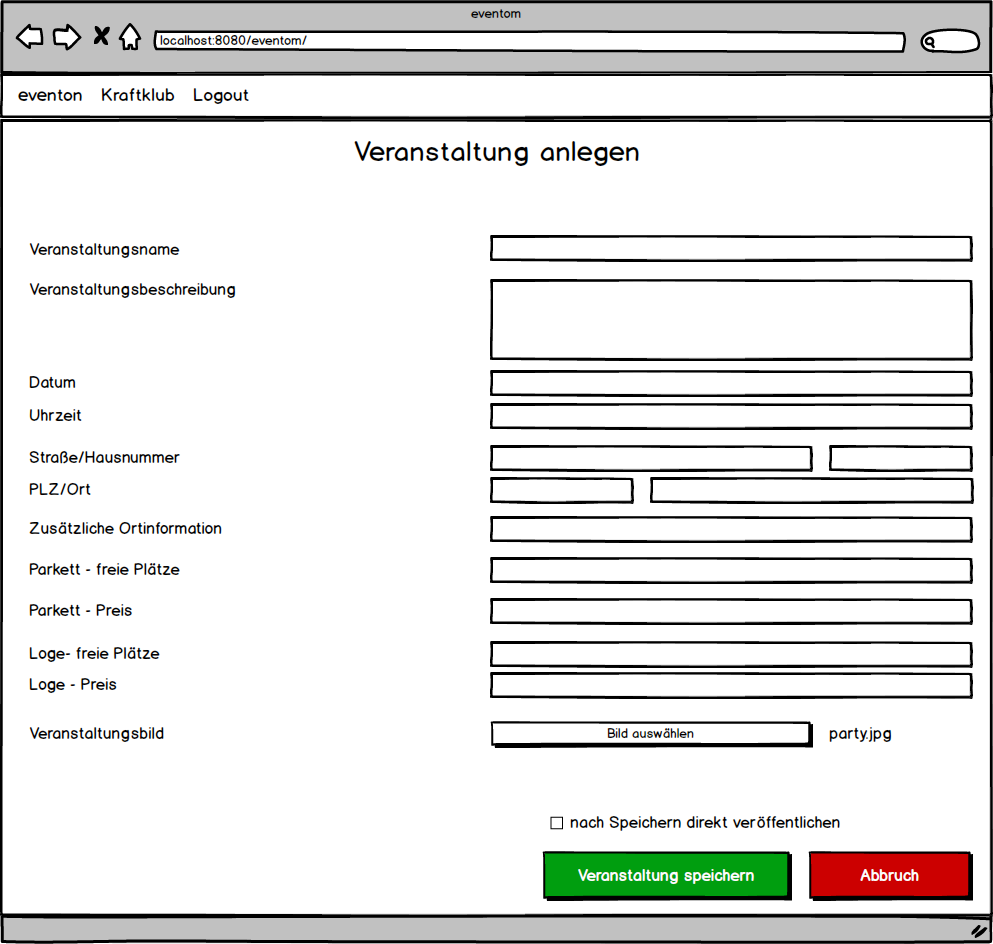


Abbildung : Mockup\_Event\_anlegen

1. Darstellung des Datenmodells
   1. ER-Modell

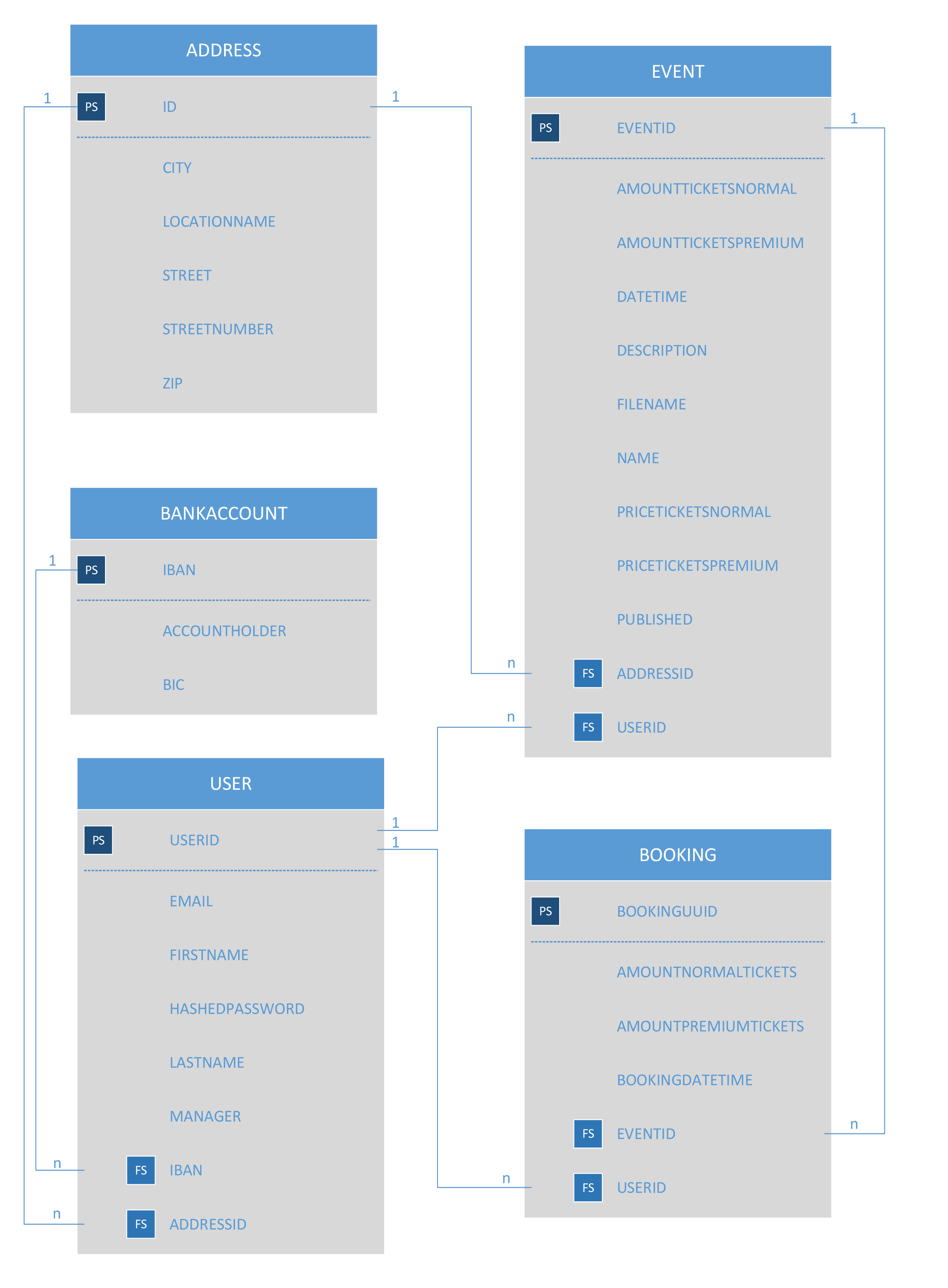


Abbildung : ER-Modell

* 1. Klassendiagramm

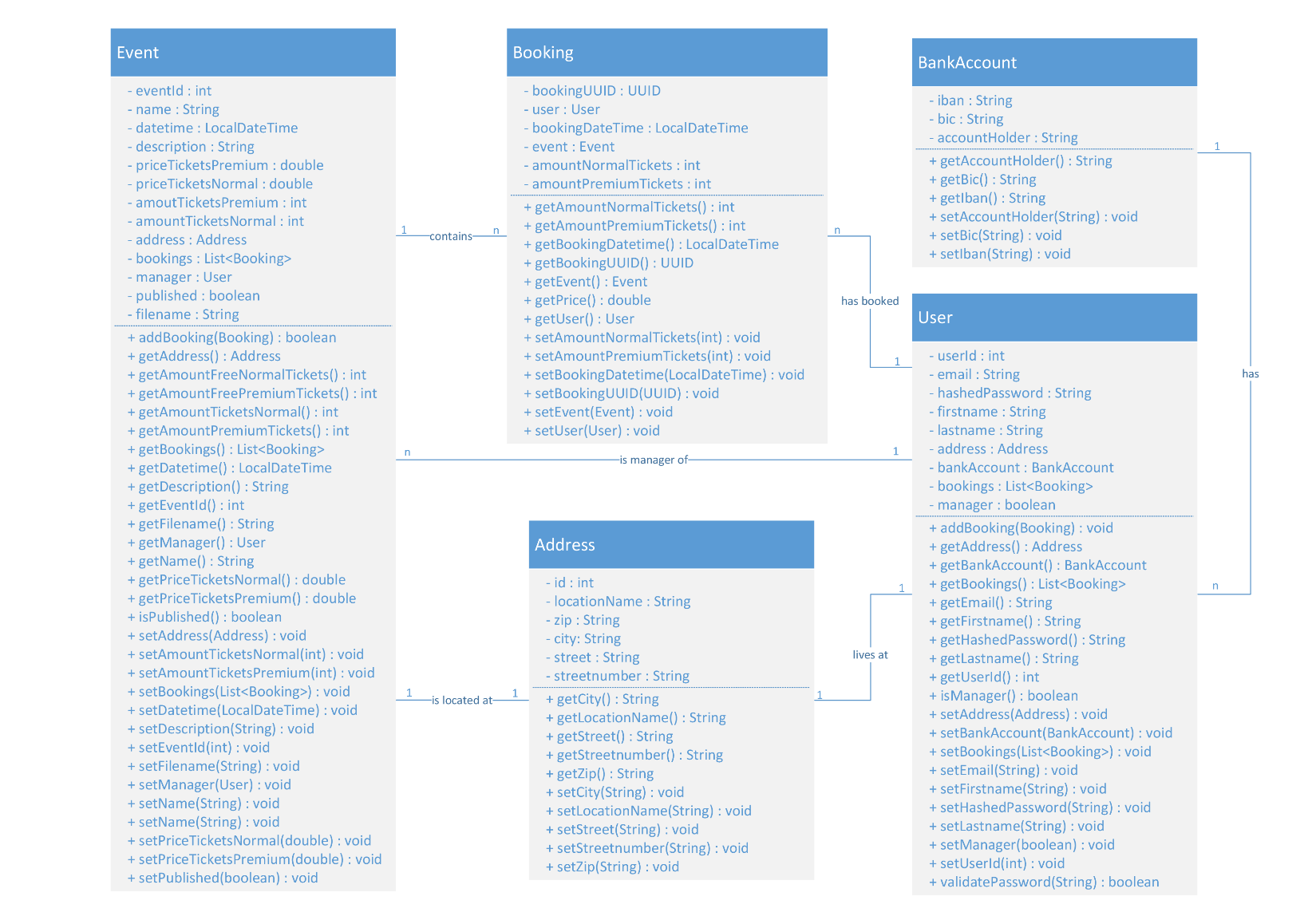


Abbildung : Klassendiagramm

1. Schichtenarchitektur

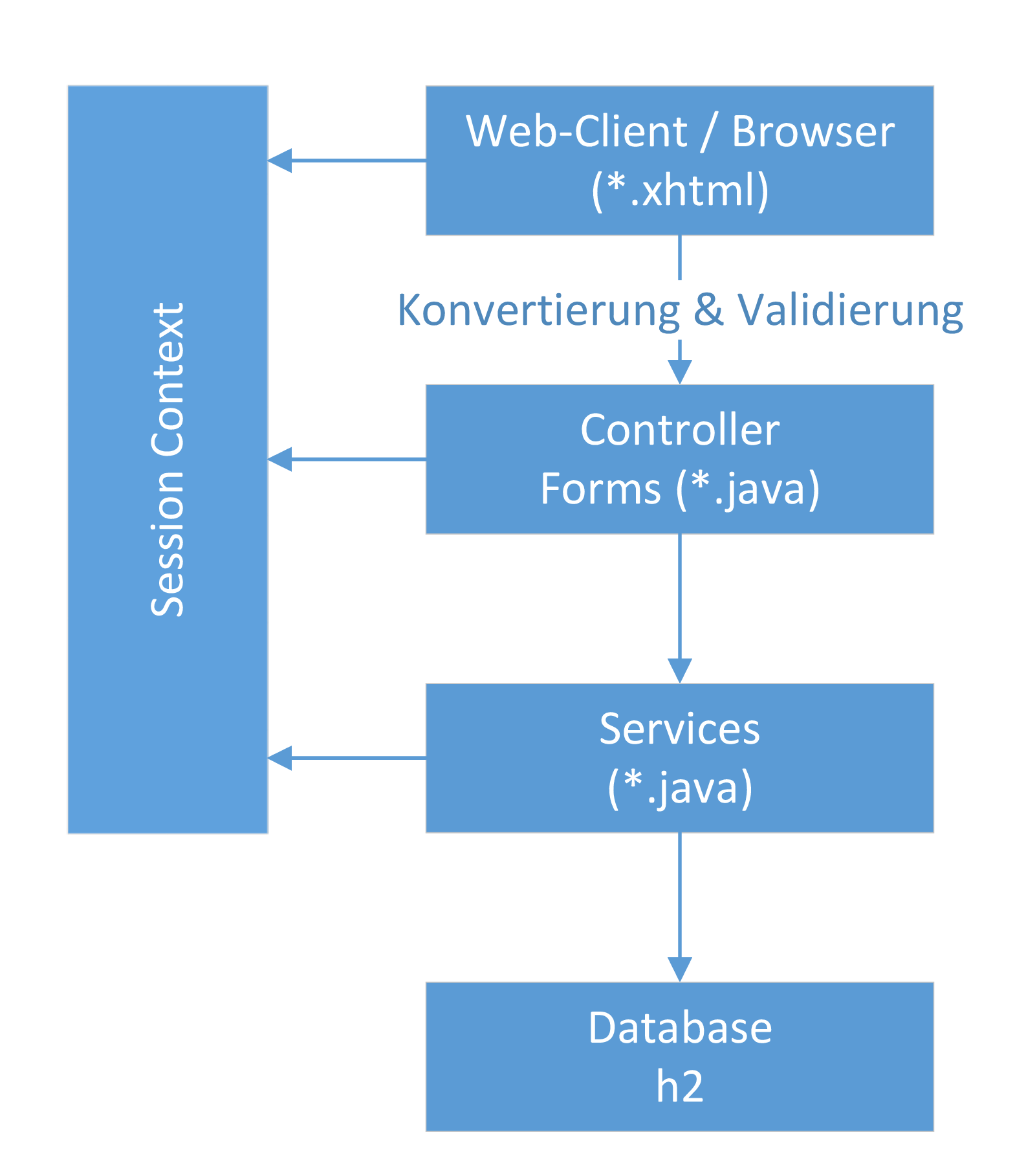


Abbildung : Schichtenarchitektur

**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 31.03.2017 |  |
| Ort, Datum | Unterschrift |

|  |  |
| --- | --- |
| Hameln, den 31.03.2017 |  |
| Ort, Datum | Unterschrift |