Berufsakademie Sachsen

Staatliche Studienakademie Sachsen

Informationstechnik – Medieninformatik

|  |
| --- |
| **Entwicklerdokumentation** |

**Eventmanagementsystem**

**Projektdaten**

|  |  |
| --- | --- |
| Auftraggeber: | Dr. Thomas Nindel |
| Auftragnehmer: | Abdul Hadi AlQawas, Carolin Feurich, Roman Kosovtsev, Jennifer Schön, Yannick Rammelt |
| Modul: | Softwaretechnik 3IM-STECH-00 |
| Beschreibung: | Dieses Projekt ist im Zusammenhang mit dem Softwaretechnik-Projekt der BA Sachsen im Kurs MI22-1 entstanden. Es handelt sich bei dem erstellten System um eine Vertriebsplattform für Eventtickets ähnlich zu Plattformen wie Eventim. |
| Letzte Änderung: | 22.04.2024 |
| Abgabedatum: | 05.05.2024 |

**Gliederung**

|  |  |
| --- | --- |
| 1 Einrichtung der Laufzeitumgebung….………….……………………………………. |  |
| 2 Einrichtung der Entwicklungsumgebung……………………………………………. | x |
| 2.1 Einrichtung der IDE……………………………………………………………………… | x |
| 2.2 Einrichtung der Datenbank…………………………………………………………... | x |
| 2.3 Testen……………………………………………………………………………………….. | x |
| 3 Programmaufbau…………………………..……………………………………………… | x |
| 3.1 Allgemein………………………………………………………………………………….. | x |
| 3.2 Klassendiagramm……………………………………………………………………….. | x |
| 3.3 Beispielhaftes Ablaufdiagramm……………………………………………………. | x |
| 4 Potenzielle Erweiterungsmöglichkeiten….…………………………………………. | x |
| 4.1 Allgemein………………………………………………………………………………….. | x |
| 4.2 Exemplarisches Beispiel zur Implementierung……………………………….. | x |

**1 Einrichtung der Laufzeitumgebung**

DOCKER

**2 Einrichtung der Entwicklungsumgebung**

Im Folgenden wird eine Anleitung mitgegeben, welche die ersten Schritte zur Einrichtung der Entwicklungsumgebung beschreibt, um sich so an dem Projekt beteiligen zu können.

**2.1 Einrichtung der IDE:**

Zunächst gilt es die IDE (Integrierte Entwicklungsumgebung) installieren. Da das Projekt zentral in der Programmiersprache C# entwickelt wird, empfiehlt es sich entweder [**Visual Studio**](https://visualstudio.microsoft.com/de/) (von Microsoft, kostenfrei) oder [**Rider**](https://www.jetbrains.com/rider/) (von JetBrains, kostenpflichtig) zu verwenden. Wir empfehlen in diesem Zusammenhang **Visual Studio** von Microsoft. Die folgende Anleitung wird sich daher auf Visual Studio mit seinen Funktionalitäten fokussieren.

Laden Sie sich zunächst Visual Studio im Browser herunter und beginnen Sie mit der Installation. Im Verlauf der Installation gelangen Sie zu diesem Fenster.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Betätigen Sie den Button „Ändern“. Daraufhin sollte sich folgendes Fenster öffnen:

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Um an dem Webprojekt arbeiten zu können, müssen Sie hier den **Workload ASP.Net und Webentwicklung** auswählen und dann ihre Entscheidung bestätigen, indem Sie den Button „Beim Herunterladen installieren“ betätigen.

Sobald der Installationsprozess abgeschlossen ist, sehen Sie folgende Ansicht:

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Betätigen Sie nun den Button „Repository klonen“. Geben Sie an der entsprechenden Stelle den Pfad zum Repository (das Repository finden Sie [**hier**](https://github.com/ElreadRubusErebean/EventManagementSystem.git)) und einen lokalen Speicherort ihrer Wahl an. Klicken Sie abschließend auf „Klonen“.

Das Projekt wird nun mit dem aktuellen Stand auf Ihr System geklont und Sie können theoretisch mit den ersten Arbeiten beginnen. Jedoch fehlt Ihnen noch die korrekte Datenbankanbindung. Damit beschäftigen wir uns im folgenden Schritt.

**2.2 Einrichtung der Datenbank:**

Das zugrundeliegende System erhält als einen zentralen Baustein eine anliegende Datenbank. Diese Datenbank gilt es an das System anzubinden. Da jedoch keine aktiven Daten während der Entwicklung benötigt werden, wird während der Entwicklung nicht auf die Datenbank aus der Laufzeitumgebung zurückgegriffen. Viel mehr hat jeder Entwickler eine eigene lokale Datenbank mit eigenen Daten bei sich auf dem Rechner.

Die Datenbank basiert auf SQL als Abfragesprache. Dankbarerweise bietet Microsoft mit [**SQL Server Management Studio**](https://learn.microsoft.com/de-de/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16) eine zu visuell Studio kompatible Software zur Verwaltung von SQL-infrastruktur an, die den Ansprüchen unseres Projektes genügt.

Laden Sie sich [**SQL Server Management Studio**](https://learn.microsoft.com/de-de/sql/ssms/download-sql-server-management-studio-ssms?view=sql-server-ver16) aus dem Browser herunter und beginnen Sie mit der Installation.

Starten Sie die Software, sobald die Installation abgeschlossen wurde. Nachdem die Software gestartet wurde, sollte sich folgendes Fenster öffnen:

Ein Bild, das Text, Screenshot, Software, Webseite enthält.

Automatisch generierte Beschreibung

Sollten Sie noch nicht registriert sein, müssen Sie dies vorher noch tun. Folgen Sie dazu den Anweisungen der Software!

An dieser Stelle müssen Sie sich den Servernamen kopieren. Speichern Sie sich den Namen irgendwo ab, wo Sie ihn wiederfinden! Sie werden den Servernamen im späteren Verlauf der Einrichtung noch benötigen. Betätigen Sie anschließend den „Verbinden“-Button.

Im Objekt-Explorer unter dem Punkt Datenbanken finden Sie Ihre Datenbanken. Erstellen Sie hier eine leere Datenbank mit einem von Ihnen gewählten Namen. Speichern Sie sich auch den Namen dieser Datenbank!

Ein Bild, das Text, Screenshot, Schrift, Software enthält.

Automatisch generierte Beschreibung

Öffnen Sie nun das Projekt in Visual Studio! In dem Solution-Explorer (befindet sich zumeist an der linken oder rechten Fensterseite) finden Sie die Datei **appsettings.development.json**. Öffnen Sie diese!

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Sobald Sie die Datei geöffnet haben, sollte sich vor Ihnen folgender Anblick eröffnen:

Ein Bild, das Text, Multimedia-Software, Software, Screenshot enthält.

Automatisch generierte Beschreibung

An dieser Stelle müssen Sie nun zwei Werte eintragen. Tragen Sie zunächst den von Ihnen abgespeicherten Servernamen an der entsprechenden Stelle ein! Danach tragen Sie den von Ihnen gespeicherten Datenbanknamen an der entsprechenden Stelle ein! Speichern Sie Ihre Änderungen!

Öffnen Sie nun die Package Manager Console, wie Sie es aus dem unteren Bild entnehmen können!

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

Automatisch generierte Beschreibung

Tragen Sie hier nun folgende Befehle nacheinander ein:

1. add-migration [Migrationname]
2. update-database

Den Migrationnamen dürfen Sie selbst wählen.

Damit haben Sie die Einrichtung der Datenbank abgeschlossen.

**2.3 Testen**

Dankbarerweise stellen sowohl Visual Studio, als auch Rider einen integrierten Test-Server zur Verfügung, so dass ein weitergehendes Setup nicht von Nöten ist. Starten Sie einfach den Debugger aus ihrer IDE heraus, um zu testen.

**3 Programmaufbau**

Dieses Kapitel soll Informationen über den Aufbau und die Funktionsweise des Eventmanagementsystems vermitteln, um so Rückschlüsse auf die Deckweise und die Ideen hinter vereinzelten operativen Entscheidungen zu ermöglichen.

**3.1 Allgemein**

Das System ist auf dem MVC-Pattern aufgebaut. Dies dient der klaren Trennung von Darstellung (View), Logik (Controller) und Daten (Model), was die Wartbarkeit, Erweiterbarkeit, wie auch die anfängliche Entwicklung in klarer Arbeitsteilung deutlich erleichtert. Eine klare Trennung zwischen Backend, Frontend, sowie innerhalb der Abteilungen wird dadurch möglich und verhindert zugleich, dass ein weitreichender Eingriff in den jeweils anderen Zuständigkeitsbereich nur minimale Komplikationen hervorruft. Dies waren auch die zentralen Gedanken dahinter, auf dieses Design-Pattern zurückzugreifen.

Dankbarerweise stellt Microsoft mit Visual Studio ein entsprechendes Project-Template zur Verfügung. Dieses hat für das vorliegende System als Vorlage gedient. Es hat den großen Vorteil, dass dieses Template auf ASP.Net-Entwicklung und damit auf die Entwicklung von WebApps, optimiert ist. Dieser Umstand war ein weiteres Kriterium für unsere Entscheidung zu diesem Template.

Im Folgenden soll dieser Aufbau etwas genauer aufgeschlüsselt werden. Zusätzlich wird an einem exemplarischen Beispiel die Funktionsweise einer exemplarischen Funktion gezeigt, um so dass Verständnis noch einmal weitereichend zu vertiefen.

**3.2 Klassendiagramm**

Im Folgenden wird das Klassendiagramm für das Eventmanagementsystem gezeigt. Zunächst zur besseren Lesbarkeit. Sämtliche Views sind als gelbe Kästen markiert. Diese wurden nicht im Detail aufgeschlüsselt, da deren Inhalt stetig wechselnd und gleichzeitig keine tiefe Logik beinhalten, so dass uns ein detaillierter Blick darauf unnötig erschien. Jedoch gilt es allgemein dazu einen Punkt zu erwähnen:

Da es sich bei der Anwendung um eine Webanwendung handelt, wird die View als html-File an den Nutzer ausgegeben. Bei der Entwicklung handelt es sich jedoch um Files des cshtml-Formats. Dies ist ein hauseigenes Format, welches speziell zur Kommunikation zwischen html-Files und cs-Klassen entwickelt wurde. Da dies optimal in die Anforderungen an die Technologie passt, empfahl es sich nicht direkt html zu coden, sondern sich auf cshtml zu fokussieren.

Betrachten wir nun das Klassendiagramm:

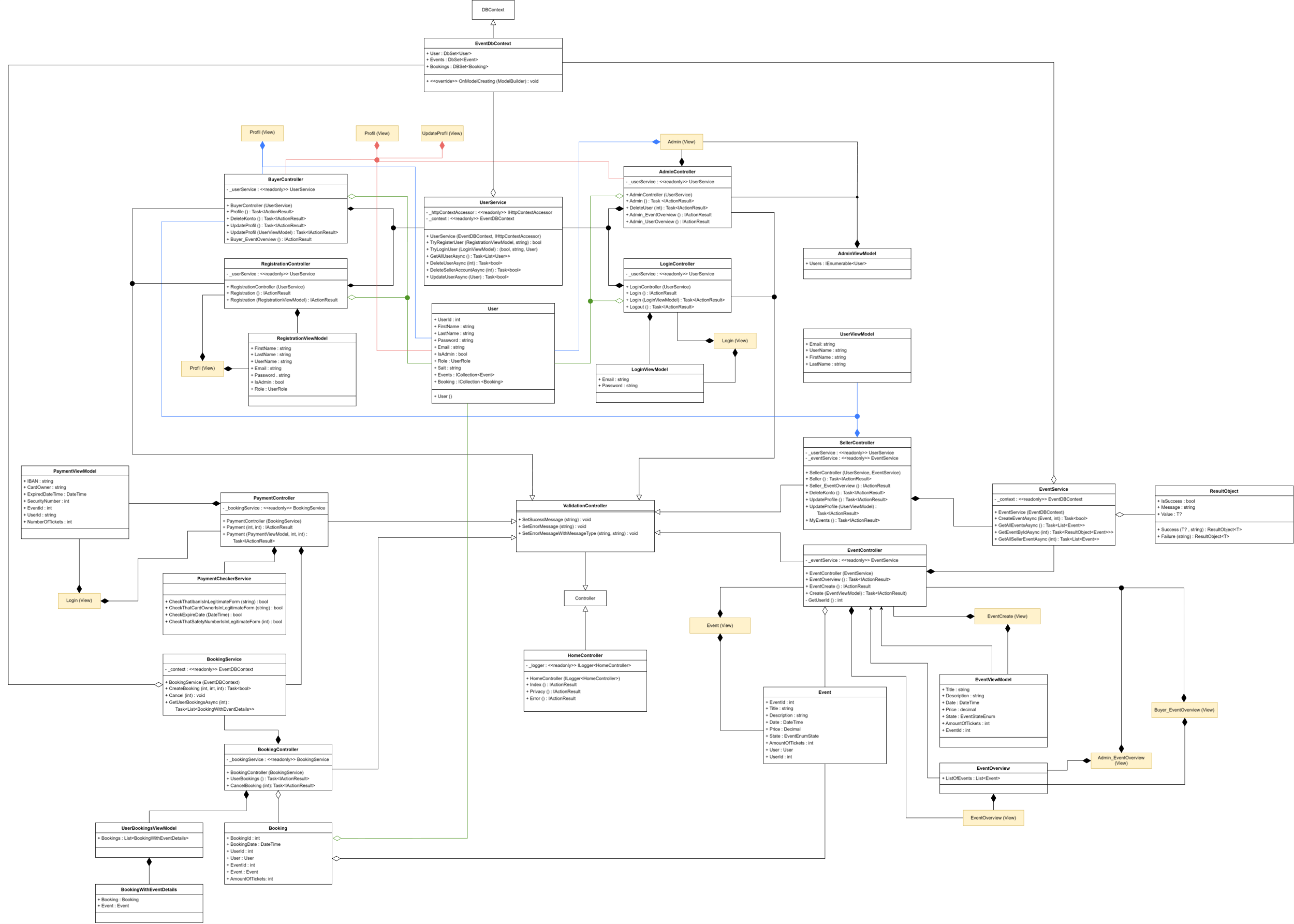


Abbildung 1 Klassendiagramm (**SVG-File befindet sich im Ordner Dokumentation**)

Es lässt sich aus dem Klassendiagramm sowohl eine klare Trennung in Model, View und Controller mit zusätzlichen Anhängen (Services) erkennen. Zusätzlich dazu wurde eine Trennung zwischen User, Buchung und Event vorgenommen, die sich ebenfalls aus dem Klassendiagramm erkennen lässt.

**3.3 Beispielhaftes Ablaufdiagramm**

Im Folgenden betrachten wir einen exemplarischen Prozess aus dem zugrundeliegenden System. Dafür wurde eine Ablaufdiagramm erstellt. Bei dem Prozess handelt es sich um den Kaufprozess durch den Kunden mitsamt der Auswahl des Events, der Eingabe von der Anzahl der gewünschten Tickets bis hin zu der Eingabe und Überprüfung der Zahlungsdaten und der Speicherung der Buchung in der anliegenden Datenbank. Betrachten wir zunächst den vollständigen Ablauf.

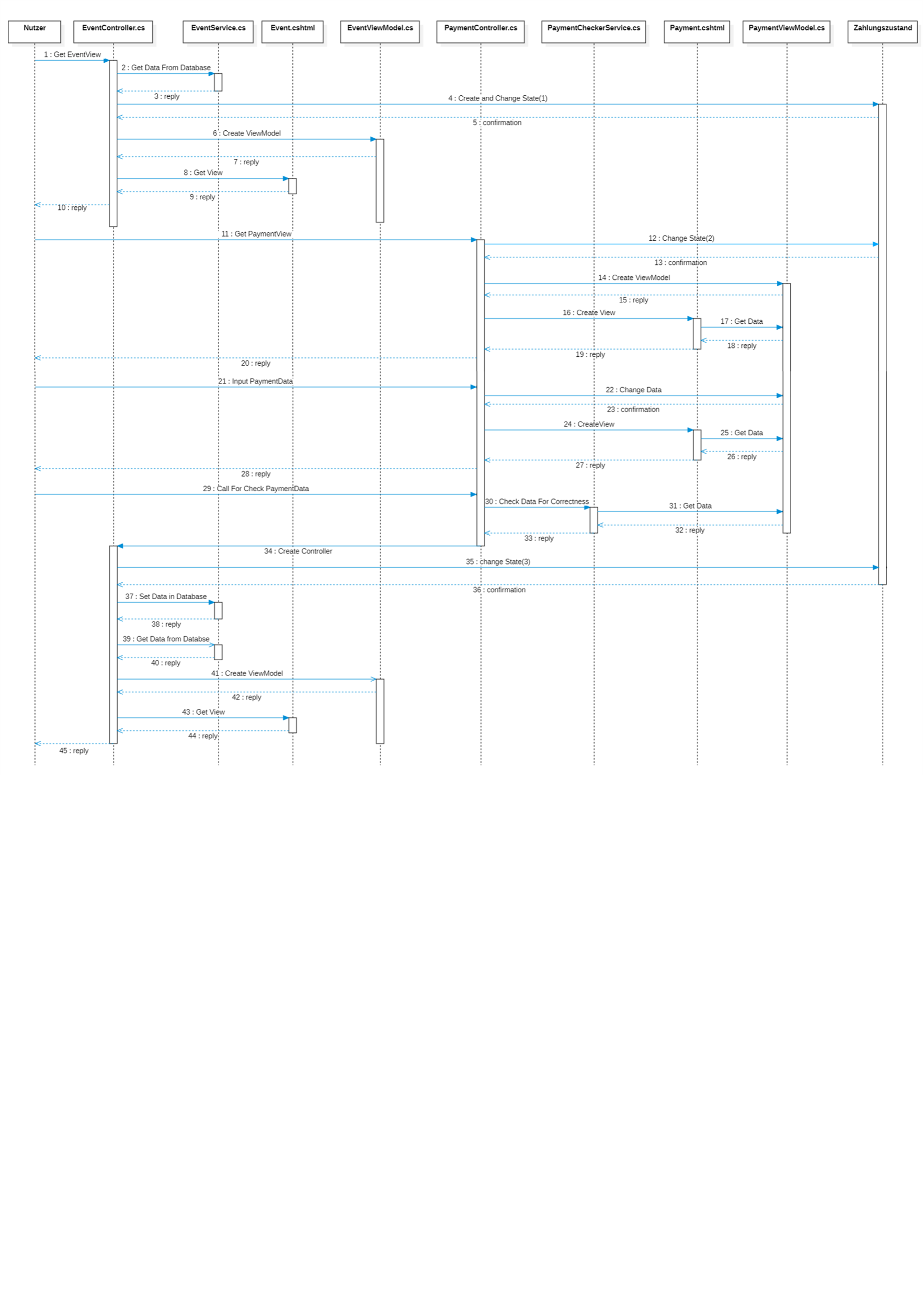


Abbildung 2 Ablaufdiagramm (**SVG-Version findet sich zusätzlich im Ordner Dokumentation**)

Der Prozess selbst umfasst 45 Teilschritte und 8 Klassen. Wie Sie feststellen werden, sind jedoch zwei weitere Instanzen in dem Ablaufdiagramm eingezeichnet, den Nutzer und den Zahlungszustand. Um den Ablauf besser verstehen zu können, empfiehlt es sich diese Instanzen kurz zu beleuchten.

Die Nutzer-Instanz dient als Repräsentant des Anwenders. Seine Eingaben werden hier als Ausgangspunkt des Prozesses, sowie seinen Teilschritten verstanden. Die Eingabe des Nutzers im Browser wird hier durch seine Lebenslinie widergespiegelt.

Der Zahlungszustand ist eine abstrahierte Darstellung integrierter Funktionen des MVC-Templates von C#. Durch Mitgeben von Eigenschaften in der URL (hier die EventId, die Anzahl der Tickets, etc.) lässt sich der Zustand der Zahlung im Prozess widerspiegeln, wodurch das System ermitteln kann, in welchem Schritt der Buchung es sich aktuell befindet. Folgende Zustände sind dabei im System vorhanden:

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Abbildung 3 Zustandsdiagramm (**PNG-File befindet sich im Ordner Dokumentation**)

Aus diesen Informationen lässt sich ein klares Bild für den Prozess zeichnen. Anhand der Zustände lassen werden unterschiedliche Klassen angesprochen und in den Programmablauf einbezogen. Man den Prozess grob in drei Abschnitte einteilen: Alle Messages von 1 bis 10, von 11 bis 34 und von 35 bis 45. In jedem wird zunächst der Grundaufbau aus Model, View und Controller durch den Controller hergestellt.

Im ersten und dritten Abschnitt wird zusätzlich die EventService-Klasse zugegriffen, welche als Schnittstelle an die Datenbank fungiert. Sie greift sowohl die Daten für die Events ab und sendet am Ende die Buchungsinformationen wieder dorthin zurück.

Im zweiten Abschnitt wird statt dem EventService auf die PaymentService-Klasse zugegriffen. Diese überprüft die Eingaben der Zahlungsdaten hinsichtlich der Validität. Was wir an daran sehen ist, dass zusätzlich zum klassischen MVC-Pattern Services verwendet werden, um zusätzlich Funktionalitäten auf Server-Seite, die nicht klar einem Controller zuordbar sind beizufügen.

Darüber hinaus findet sich im zweiten Abschnitt die Verwendung von einem ViewModel. Diese dienen dazu Eingabedaten aus der View serverseitig ablegen zu können, um mit den eingegebenen Daten arbeiten zu können. Wir machen hier eine Unterscheidung zu den klassischen Models auf, da ein Model dazu dient eine Datenbankstruktur zu repräsentieren, während ein ViewModel nur als Speicherklasse für die Eingabe der View dient.

**4 Potenzielle Erweiterungsmöglichkeiten**

Dieses Kapitel wird die aktuellen Pläne für mögliche zukünftige Erweiterungen aufschlüsseln. Hier wird also gezeigt, was in der Zukunft noch kommen könnte. Außerdem zeigt dieses Kapitel für zwei dieser Funktionalität, wie eine mögliche Umsetzung aussehen kann.

**4.1 Allgemein**

Im Zusammenhang des anfänglichen Brainstormings sind folgende Features entstanden. Sämtliche Features wurden von als Sinnvoll bestimmt möglicherweise weiter verfolgt zu werden, wenn Kapazitäten frei werden sollten. Da sich dies jedoch nicht in der Zeit ergeben hat, bietet es sich an, diese als sinnvolle Features für die Zukunft heranzuziehen. Folgende Features haben sich herausentwickelt:

* Kommentarfunktion für Events
* Bilder für Events
  + Zum Zeitpunkt der Erstellung dieser Dokumentation erhält jedes Event ein generisches Bild. Der Kunde sollte jedoch die Möglichkeit erhalten, selbst zu entscheiden, wie er sein Event darstellen will.
* Such- und Filterfunktionen für Events
* Zuordnung von Kunden zu bestimmten Kundengruppen (VIP, etc.)
* Zugriffsrechte auf Tickets für bestimmte Gruppen
  + Bsp.: Nur VIP-Kunden erhalten Zugriff auf bestimmte Events.
* Erstellen von Rabatt-Codes
  + Diese Codes kann der Verkäufer persönlich an seine potenziellen Kunden rausgeben und ist somit nur für die Kunden bekannt, an die der Verkäufer den Rabatt-Code rausgegeben hat.
* Einbau von Rabatten und Angeboten
  + Dieser Punkt unterscheidet sich vom vorherigen. Der Unterschied liegt darin, dass diese Rabatte jedem Kunden automatisch angerechnet werden.
* Das System errechnet automatisch eine Summe, die zusätzlich auf den Preis des Verkäufers als Anbieter-Gebühr aufgeschlagen wird.
* Unterscheidung der Karten nach Kategorien
  + Bsp.: Sitzplätze unterscheiden sich von Stehplätzen.

Jede der aufgelisteten Funktionalitäten verbessert das Kundenerlebnis signifikant und baut den Plattform-Charakter des Systems weiter aus. Daher sind sie besonders geeignet als weitere Features um so den Kundenkreis schnellstmöglich ausbauen zu können.

**4.2 Exemplarisches Beispiel zur Implementierung**

Betrachten wir exemplarisch das Beispiel der Unterscheidung der Karten nach Kategorien. Es soll daran eine mögliche Version der Implementierung gezeigt werden.

Es gilt zu unterscheiden zwischen unterschiedlichen Arten der Unterscheidung der Kategorien. Es ist zum einen möglich nur zwischen Sitz- und Stehplätzen zu unterscheiden. Dies betrachten wir als Fall 1. Fall 2 hingegen ermöglicht dem Nutzer eine beliebige Anzahl unterschiedlicher Kategorien für ein Event zu erstellen.

*Fall 1*: Dieser Fall lässt sehr simpel zu implementieren. Dies ist insofern von Relevanz, da es sich aus Perspektive eines agilen Projektmanagementansatzes empfehlen würde zunächst diese Weiterentwicklung zu implementieren und User-Feedback zu erwarten.

Die Umsetzung selbst, würde zwei weitere Parameter in der Model (und damit auch in der Datenbank) benötigen, welchen die Anzahl der Sitzplätze, als auch deren Preis beschreiben würde. Diese würden die selbe Form, wie die bisherigen Parameter für Plätze allgemein.

*Fall 2*: Diese Fall fordert tiefgreifendere Änderungen am System. Eine erste Analyse hat gezeigt, dass wir zwei weitere Tabellen in der relationalen Datenbank empfehlen würden. Die eine soll die erstellten Sitzplatzkategorien mitsamt Anzahl und Preis pro Karte umfassen. Jeder Eintrag verweist auf ein Element eines Events um so eine Verbindung herzustellen. Dies kann möglicherweise wie folgt aussehen:

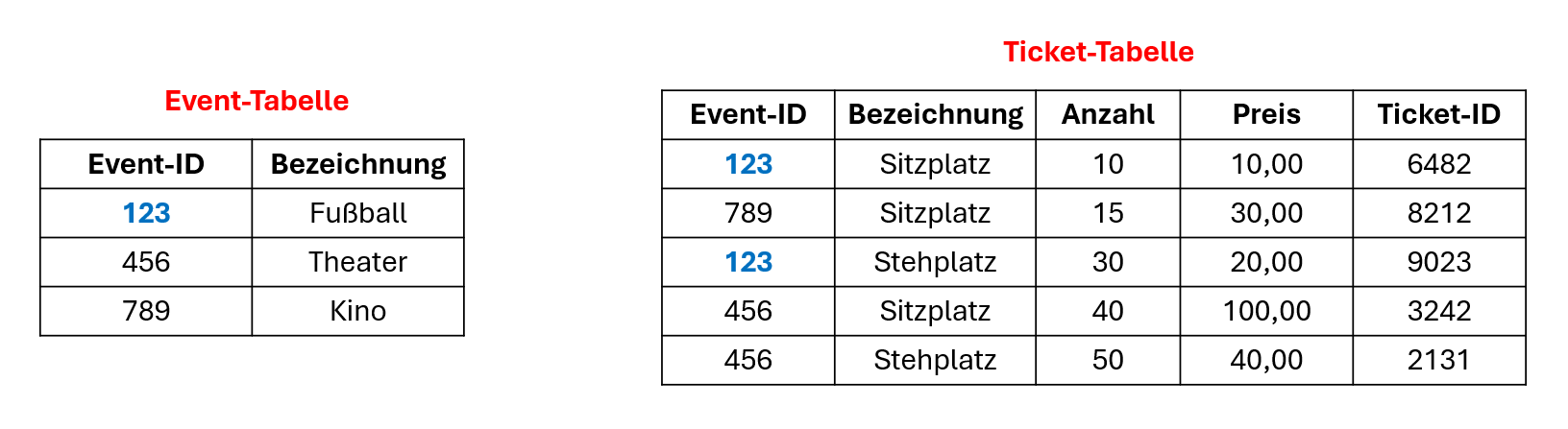


Abbildung 4 Exemplarische Darstellung der Datenbank

Dementsprechend muss das EventModel angepasst und ein weiteres TicketModel angelegt werden um die DB abzubilden. Das EventModel beinhaltet nun eine Liste von TicketModels.

Zusätzlich muss eine Entsprechende neue View erstellt werden zum Erstellen der Ticket-Kategorien und der Controller muss entsprechend um Funktionalität erweitert werden.