Technische Hochschule Ingolstadt

Entwicklung eines Bestellsystems für ein Restaurant mit MVC

Seminararbeit

Vinzenz Götz
00107303

Software Engineering für Ingenieure

Prof. Dr.-Ing. Jörn Schlingensiepen

7. Dezember 2021

Abstract

Die nachfolgende Arbeit befasst sich mit der Erstellung eines Bestellsystems für ein Restaurant in MVC. Dort soll der Kunde Gerichte in einer Filiale bestellen können und ein Koch die in der nächsten Viertelstunde zu kochenden Gerichte einsehen können.

Würdigungen

Diese Studienarbeit wurde in LaTeXverfasst. Die Vorlage dafür wurde auf Overleaf gefunden und stammt von Martin Knaebele [Kna20].

Anmerkungen

Das generische Maskulinum im Deutschen existiert und wird in dieser Arbeit verwendet.

In dieser Arbeit werden außerhalb von Tabellen *Beziehungen* und *Rollen* in kursiv dargestellt, **Views** werden fett gedruckt und Attribute unterstrichen.

Inhaltsverzeichnis

Abstract i

Würdigungen ii

Anmerkungen iii

Abbildungsverzeichnis v

Tabellenverzeichnis vi

1 Motivation 1

2 Anforderungsbestimmung 2

2.1 User-Stories . . . 2

2.2 Konkretisieren der Anforderungen . . . 2

2.3 Use-Cases . . . 2

2.3.1 Use-Case 1: Kunde tätigt eine Bestellung . . . 3

2.3.2 Use-Case 2: Gerichte hinzufügen und entfernen . . . 3

2.3.3 Use-Case 3: Koch sieht sich Bestellungen an . . . 4

2.3.4 Weitere Use-Cases . . . 4

3 Datenmodell 5

4 Aufbau der Web-App 7

4.1 Homepage . . . 7

4.2 Layout . . . 7

4.3 Login . . . 8

4.4 Buchungen . . . 9

4.5 Gerichte . . . 10

4.6 Kochen . . . 10

5 Validierung 12

6 Fazit und Ausblick 13

6.1 Fazit . . . 13

6.2 Ausblick . . . 13

Literatur I

Abbildungsverzeichnis

3.1 Datenmodell 5

4.1 Home View mit eingeloggtem Koch “Gusto“ in der Navbar 7
4.2 Login-View der Web-App 8
4.3 Index-View der Buchungen als Koch Paul Gusto welcher die Buchung von Peter Lustig sieht 9
4.4 Detail-View einer Buchung 9
4.5 Edit-View einer Buchung 10
4.6 Gericht zu einer Buchung hinzufügen 10
4.7 Gerichte, die in den nächsten 15 Minuten fertiggestellt werden müssen . . . 11

Tabellenverzeichnis

2.1 Anforderungen an die Web-App 2
2.2 Use-Case 1 3
2.3 Use-Case 2 3
2.4 Use-Case 3 4

3.1 Attribute im Datenmodell 6

5.1 Erfüllung der Anforderungen aus Tabelle 2.1 12

1. Motivation

Im Modul Software Engineering soll eine Web-Anwendung erstellt werden, hinter der eine SQL-Datenbank steht um ein Problem zu lösen. Hier war ein Projekt innerhalb dieses Rahmens selbst zu wählen. Ziel ist es nicht eine voll funktionsfähige Anwendung zu erstellen sondern einen Demonstrator, der die wichtigsten Funktionen der späteren Anwendung aufweist.

Das gewählte Projekt für diese Arbeit war eine Plattform für die Reservierung von Tischen und Bestellung von Gerichten in einem Restaurant da dies in vielen Gaststätten noch nicht oder nur schlecht implementiert existiert.

Um einen Platz in einzelnen Filialen von großen Restaurantketten zu ergattern muss man zur Zeit immernoch anwesend sein und eine Vorbestellung des Tisches oder des Essens funktioniert nicht.

Um die Arbeit der Köche in den Filialen zu vereinfachen wird noch eine Funktion implementiert, welche es erlaubt einem Koch einzusehen welche Gerichte er in der nächsten Viertelstunde zubereiten muss.

2. Anforderungsbestimmung

Die Anforderungen für eine solche Anwendung werden oft durch sogenannte User-Stories bestimmt. Diese User-stories sind Wünsche oder Geschichten von (potenziellen) Benutzern, aus denen der Entwickler die Anforderungen an das Projekt herauslesen anschließend zusammenfassen muss.

2.1 User-Stories

"Um einen Tisch in meiner Lieblingsfiliale zu erhalten muss ich immer vor Ort sein und habe nicht schon vorher die Gewissheit einen Platz zu bekommen."

"Wenn ich Essen bestellen will muss ich immer in der Gaststätte anrufen obwohl ich Telefonieren wirklich verabscheue."

"Wenn ich in der Küche stehe und überall Zettel mit Bestellungen herumliegen komme ich ganz durcheinander. Manchmal geht ein Zettel unter und ich vergesse ein Gericht zu machen."

Aus diesen User-Stories kann man die beiden Rollen von Usern erkennen. Auf der einen Seite den *Koch*, auf der anderen Seite den *Kunden*. Beide sollten eine individualisierte Plattform erhalten auf der ihre Bedürfnisse gedeckt werden.

2.2 Konkretisieren der Anforderungen

Die aus den User-Stories gewonnenen Erkenntnisse müssen nun noch konkretisiert werden.

Tabelle 2.1: Anforderungen an die Web-App

A	Reservierung muss schon vor dem Termin möglich sein
B	Reservierung muss ohne zwischenmenschlichen Kontakt erfolgen können
C	Alle in den nächsten 15 Minuten bestellten Gerichte müssen von einem *Koch* eingesehen werden können
D	Gerichte müssen ohne viel Aufwand zu einer Bestellung hinzugefügt und entfernt werden können
E	Eine Login- und Registrierungs-Funktion muss existieren um Kunde von Koch zu unterscheiden

2.3 Use-Cases

In der Anwendung existieren drei Haupt-Use-Cases. Einmal für den *Koch*, der die Gerichte letztendlich kochen soll, und für den *Kunden*, welcher seine Bestellung aufgeben und bearbeiten will.

2.3.1 Use-Case 1: Kunde tätigt eine Bestellung

Tabelle 2.2: Use-Case 1

Ziel	Bestellung in die Datenbank eintragen
Status	Demonstrator
Akteure	Kunde
Auslöser	“Bestellung Hinzufügen“ Button im GUI
Bedingung	Eingeloggter Kunde
Priorität	Hoch
Erfolg	Bestellung in der Datenbank angelegt
Miserfolg	Bestellung wurde nicht in der Datenbank angelegt
Normal Flow	Der Kunde wählt die “Bestellung hinzufügen“ Option im GUI. Anschließend trägt er die Personenzahl, die Filiale und die Zeit ein. Abschließend wird die Bestellung mit “Submit“ zur Datenbank hinzugefügt.
Alternate Flow	Der Kunde vergisst ein Feld der Bestellung auszufüllen Er wird dann darauf hingewiesen und kann nicht voranschreiten. Der Kunde entscheidet sich doch gegen eine Bestellung und bricht den Vorgang ab. Es erfolgt eine Weiterleitung auf die Übersicht der Bestellungen.

2.3.2 Use-Case 2: Gerichte hinzufügen und entfernen

Tabelle 2.3: Use-Case 2

Ziel	Gericht hinzufügen oder entfernen
Status	Demonstrator
Akteure	Kunde
Auslöser	Button “Gericht hinzufügen“ oder “Gericht entfernen“
Bedingung	Eingeloggter Kunde
Priorität	Hoch
Erfolg	Gericht wird der Bestellung hinzugefügt oder entfernt
Miserfolg	Die Bestellung wird nicht Verändert
Normal Flow	Der Button “Gericht Hinzufügen [Entfernen]“ im GUI führt den Kunden zu einer Liste in der er die Gerichte sieht, die er der Bestellung hinzufügen [entfernen] kann.
Alternate Flow	Der Kunde entscheidet sich dagegen ein Gericht hinzuzufügen [zu entfernen]. Ein Button im GUI bringt ihn zurück zu seinen Bestellungen.

2.3.3 Use-Case 3: Koch sieht sich Bestellungen an

Tabelle 2.4: Use-Case 3

Ziel	Koch sieht alle Gerichte, die in der nächsten Viertelstunde zu kochen sind
Status	Demonstrator
Akteure	Koch
Auslöser	Koch greift in der Navbar auf den Button “Kochen“ zu
Bedingung	Eingeloggter Koch
Priorität	Hoch
Erfolg	Dem Koch werden alle Gerichte angezeigt, welche sich in einer Bestellung befinden, die weniger als 15 Minuten in der Zukunft liegt und die Filiale des Kochs angegeben hat.
Miserfolg	Nichts passiert
Normal Flow	Der Koch greift auf den Button “Kochen“ zu
Alternate Flow	Nicht vorhanden

2.3.4 Weitere Use-Cases

Weitere Funktionen, welche die vorhergehenden Use-Cases möglich machen sind ein Login, womit zwischen *Kunde* und *Koch* unterschieden werden kann. Außerdem eine Registrierung mithilfe derer sich neue Kunden einen Account anlegen können.

Der **Login**-View findet sich, als Demonstrator implementiert, in der Web-App wieder. Dort kann ein Benutzer ausgewählt und so die User-Rolle gewählt werden.

Die Registrierung ist nicht implementiert. Stattdessen wird ein vorgegebenes Set an Benutzern, Filialen und Gerichten per **Import** in die Datenbank geladen. Anschließend bringt der View den User wieder auf die Homepage.

Die Verwaltung aller Gerichte existiert, auch als Demonstrator, unter **Gerichte**. Auf diesen View kann allerdings nur zugegriffen werden wenn der eingeloggte User die Rolle *Koch* besitzt.

3. Datenmodell

Das Datenmodell beinhaltet wie in Abbildung 3.1 zu sehen die Objekttypen Filiale, Buchung [1], Kunde [2] und Gericht.

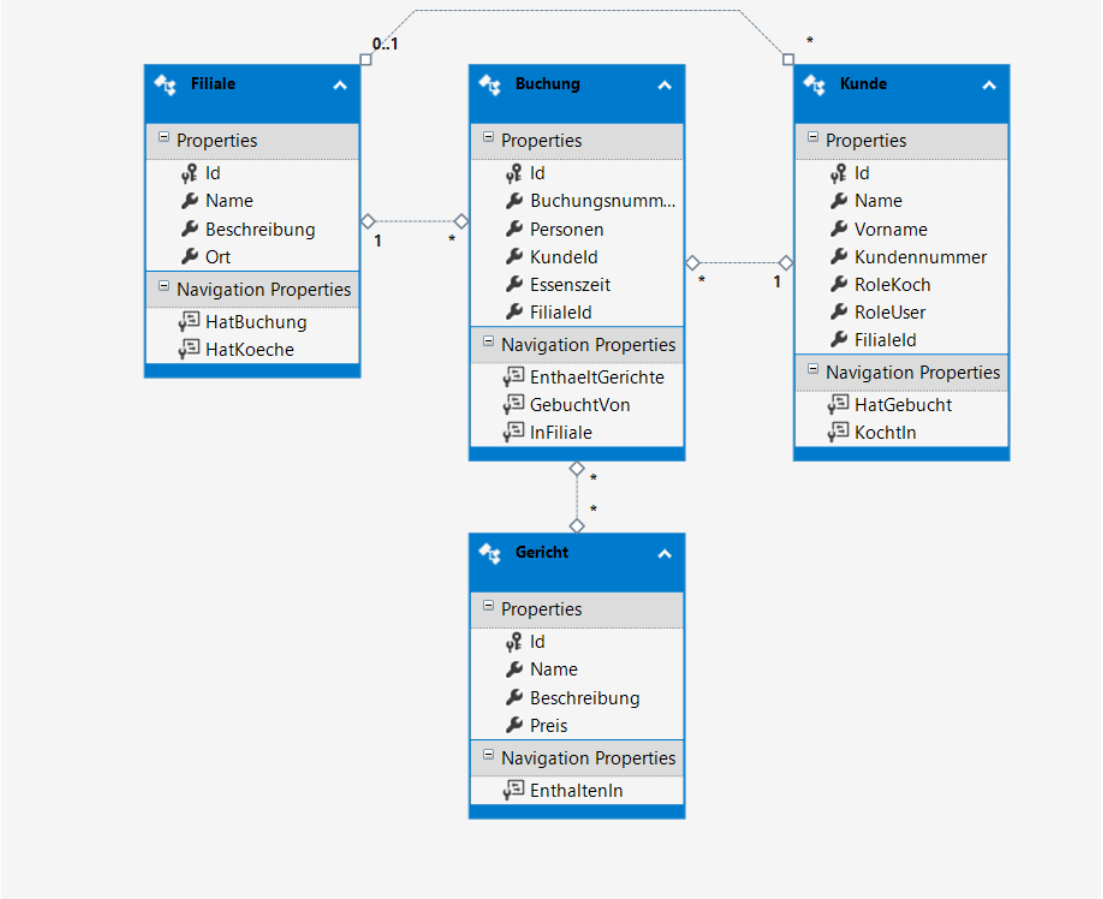

Abbildung 3.1: Datenmodell

Die Attribute der Objekttypen im Datenmodell sind in Abbildung 3.1 und ihre Typen in Tabelle 3.1 aufgelistet.

Es existiert zwischen Kunde und Filiale die Beziehung *KochtIn*, welche dem Kunden null oder eine Filiale zuweist und einer Filiale eine Anzahl an Kunden. Diese Beziehung ist nur für den *Koch* wichtig, da nur dieser eine gewisse Filiale bekocht. Ein *Kunde* benötigt diese Beziehung nicht.

Zwischen Kunde und Buchung gibt es die Beziehung *GebuchtVon*, welche einer Buchung genau einen Kunden zuweist und einem Kunden ein Anzahl von Buchungen.

Ein Gericht und eine Buchung hängen mit der Beziehung *EnthaeltGerichte* zusammen. Dort kann ein Gericht in vielen Buchungen enthalten sein und eine Buchung viele Gerichte enthalten.

[1] In dieser Arbeit eine äquivalente Bezeichnung zu Bestellung.

[2] Die Nomenklatur Kunde im Datenmodell ist unglücklich gewählt da sich der Begriff Kunde auf eine Rolle und den Objekttyp bezieht. Fortan wird auf die Rolle *Kunde* in kursiv verwiesen.

Eine Buchung und eine Filiale besitzen die Beziehung *InFiliale*. Hier wird einer Buchung genau eine Filiale zugewiesen und einer Filiale eine beliebige Anzahl an Buchungen.

Tabelle 3.1: Attribute im Datenmodell

Obejkt	**Attributname**	**Attributtyp**
Kunde	Id	Int
	Name	Str
	Vorname	Str
	Nachname	Str
	Kundennummer	Str
	RoleKoch	Bool
	RoleUser	Bool
Filiale	Id	Int
	Name	Str
	Beschreibung	Str
	Ort	Str
Gericht	Id	Int
	Name	Str
	Beschreibung	Str
	Preis	Double
Buchung	Id	Int
	Buchungsnummer	Str
	Personen	Int
	Essenszeit	DateTime

4. Aufbau der Web-App

Der Aufbau der Web-App wird anhand der wichtigsten Funktionen erläutert und nachfolgend bebildert dargestellt.

4.1 Homepage

Auf der Homepage findet sich wenig mehr als eine Wilkommensbotschaft.

4.2 Layout

Das Layout findet sich als Framework in jedem View. Seine Navbar befindet sich am oberen Rand der Seite. Nach dem Update der Packages unter Visual Studio war diese nicht mehr Funktionsfähig und musste mit Fixes [Yij21] und [Ars18] repariert werden.

Ins Layout werden die Informationen wie Rollen und Name mittels “ViewBag“ übergeben. Dies muss jedoch einzeln geschehen und kann nicht direkt als ganzer Kunde übergeben werden. Im Layout selbst geschieht die Anpassung der Navbar mittels if-Abfragen wie in Code-Snippet 4.1 zu sehen. Es zeigt auch die Implementierung eines Buttons mit dem Namen des Kunden um direkt zu erkennen wer eingeloggt ist. Dieser Button führt den Kunden wieder auf die Homepage.

Code 4.1: if-Abfrage im Layout-View

```
@if (ViewBag.UserName != null)
{
    string name = (string)ViewBag.UserName;
    <li class="nav-item">@Html.ActionLink("Buchungen","Index","
        Buchungs") "␣"</li>
    <li class="nav-item">@Html.ActionLink(name,"Index","Home") "␣"
        </li>
}
```

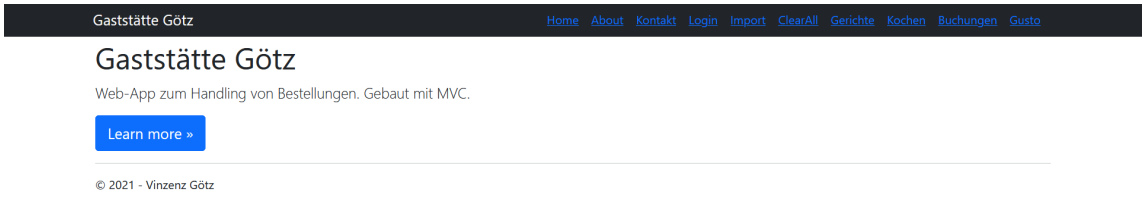

Abbildung 4.1: Home View mit eingeloggtem Koch “Gusto“ in der Navbar

Auf der **Kontakt** und der **About** Seite finden sich jeweils Kontaktdaten und eine Kurze Beschreibung der Web-App.

Im **Login** View sieht man die User, mit welchen man sich anmelden kann.

Mit dem Button **Import** werden vordefinierte Kunden, Filialen und Gerichte in die Datenbank geladen. Diesen Button bekommen nur User mit der Rolle *Koch* angezeigt.

Mit **ClearAll** werden alle Einträge aus der Datenbank gelöscht. Diesen Button bekommt ebenfalls nur ein *Koch* zu Gesicht.

In den **Buchungen**-View gelangt ein User nur wenn er eingeloggt ist. Ein *Koch* sieht dann alle Buchungen von allen Kunden. Ein *Kunde* sieht nur seine eigenen Buchungen. Wenn der User nicht eingeloggt ist sieht er den Button nicht.

Im **Gerichte**-View werden alle Gerichte aufgelistet und die Möglichkeit zur Erstellung, Bearbeitung und Löschung von Gerichten angeboten. Ein User ohne die Rolle *Koch* sieht diesen Button nicht.

Falls der User eingeloggt ist und die Rolle *Koch* besitzt werden im **Kochen**-View die Gerichte angezeigt, welche in seiner Filiale in den nächsten 15 Minuten fertig sein müssen.

4.3 Login

Der Login-View enthält Einträge aller User die in der Datenbank angelegt sind und zeit für diese ihre Rollen, Namen, Vornamen und Kundennummern an wie in Abbildung 4.2 zu sehen. Ist ein User ausgewählt erfolgt eine Weiterleitung auf die Homepage. Große Teile des **Login**-Views, des Setups und des BaseControllers von dem sich alle anderen ViewController dieser Arbeit ableiten sind aus den Präsentationen von Prof. Schlingensiepen [Sch21] entnommen und für die Zwecke dieser Arbeit angepasst worden.

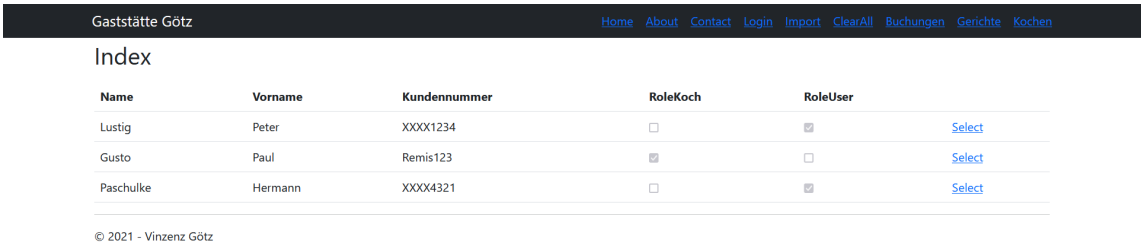

Abbildung 4.2: Login-View der Web-App

Code 4.2: Select Methode mit Weiterleitung auf die Homepage

```
public ActionResult Select(int? id)
{
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest
            );
    }
    Kunde user = db.Kunden.Find(id);
    if (user == null)
    {
        return HttpNotFound();
    }
    setUserId(id.Value);
```

```
    return RedirectToAction("Index", "Home");
}
```

4.4 Buchungen

Der **Index**-View der Buchungen listet alle Buchungen des eingeloggten Kunden auf. Ist ein *Koch* eingeloggt sieht er alle Buchungen aller Kunden. Ein Beispiel ist in Abbildung 4.3 zu sehen.

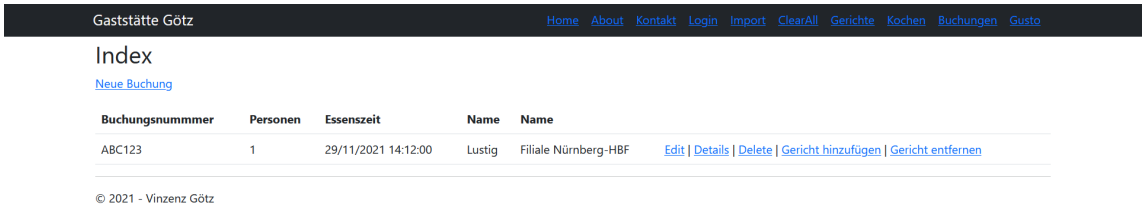

Abbildung 4.3: Index-View der Buchungen als Koch Paul Gusto welcher die Buchung von Peter Lustig sieht

Der **Detail**-View Enthält alle Details zur Buchung wie Kunde, Gerichte oder Gesamtpreis. In Abbildung 4.4 ist ein solcher Detail-View zu sehen.

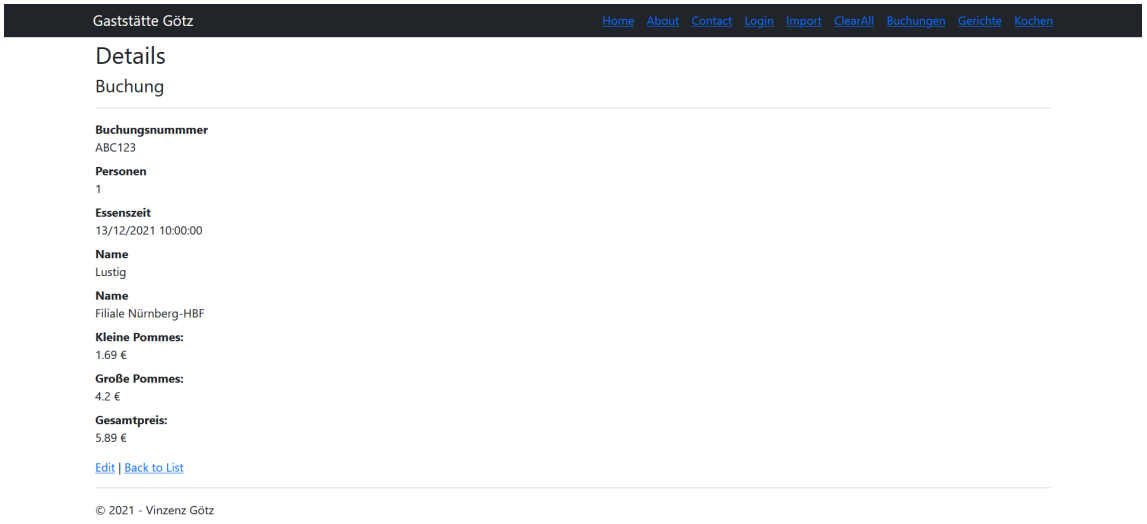

Abbildung 4.4: Detail-View einer Buchung

Der **Edit**-View kann vom Index- oder Detail-View der Buchungen erreicht werden. Dort können Änderungen an der Buchung vorgenommen werden. Dieser View ähnelt dem **Create**-View und wird in Abbildung 4.5 dargestellt. Der Unterschied zwischen beiden Views ist die Überschift des Views.

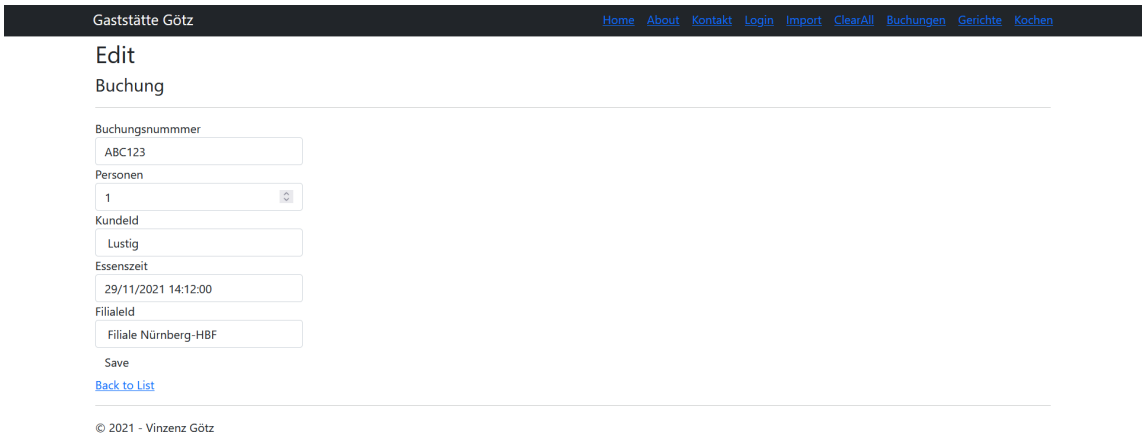

Abbildung 4.5: Edit-View einer Buchung

Mittels **Gericht Hinzufügen** kommt man in einen View, der alle Gerichte enthält. Mit **Gericht entfernen** sieht man eine Liste mit allen Gerichten, welche in der Buchung enthalten sind. In diesen Views kann man dann Gerichte hinzufügen, respektive entfernen. Der **Gericht Hinzufügen** View findet sich in Abbildung 4.6. Sollte der User doch kein Gericht entfernen oder hinzufügen wollen, gelangt er mit dem Button "Zurück" wieder auf den **Index**-View der Buchung.

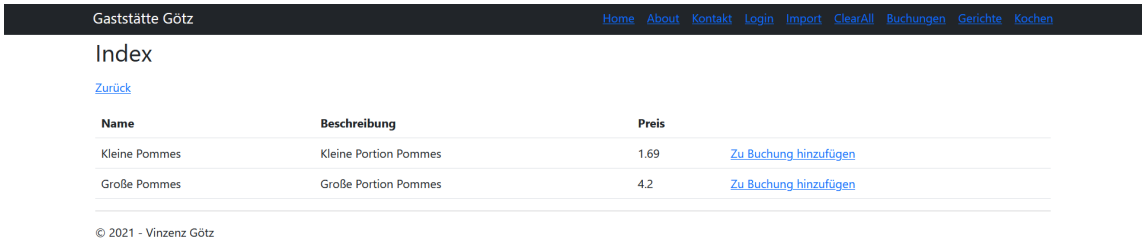

Abbildung 4.6: Gericht zu einer Buchung hinzufügen

4.5 Gerichte

Falls der eingeloggte User die Rolle *Koch* besitzt kommt dieser mit dem Button Gerichte auf eine Auflistung der Gerichte, ähnlich zu Abblíldung 4.3. Dort Werden alle Gerichte und ihre Eigenschaften aufgelistet. Es existieren jedoch keine **Gericht Hinzufügen** und **Gericht Entfernen** Buttons.

Der **Detail**-View sieht ebenso ähnlich aus wie Abbildung 4.4.

Im **Edit**- und **Create**-View der Gerichte werden, wie bei den Buchungen, alle Attribute wie Preis, Name und Beschreibung für das Gericht vom User angegeben (vgl. Abbildung 4.5).

4.6 Kochen

Ein eingeloggter *Koch* kann die in den nächsten 15 Minuten fertigzustellenden Gerichte unter *Kochen* einsehen. Diese Liste ist in Abbildung 4.7 dargestellt. Ein User, der nicht

die Rolle *Koch* zugewiesen hat wird auf die Homepage weitergeleitet. Die Auswahl der Einträge aus der SQL-Datenbank ist in Code-Snippet 4.3 zu sehen.

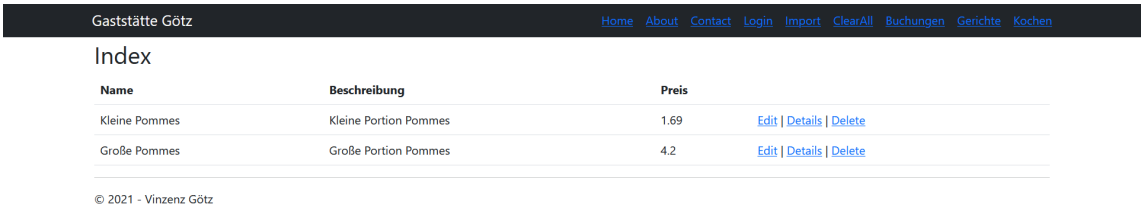

Abbildung 4.7: Gerichte, die in den nächsten 15 Minuten fertiggestellt werden müssen

Code 4.3: Auswahl der Gerichte, die weniger als 15 Minuten in der Zukunft liegen

```
var liste = db.Bestellungen
            .Where(b => b.Essenszeit.Year == now.Year &&
            b.Essenszeit.Month == now.Month &&
            b.Essenszeit.Day == now.Day &&
            -b.Essenszeit.Hour * 60 - b.Essenszeit.Minute + now.
                Hour * 60 + now.Minute + 15 >= 0 &&
            -b.Essenszeit.Hour * 60 - b.Essenszeit.Minute + now.
                Hour * 60 + now.Minute <= 0 &&
            b.InFiliale.Id == LoggedInUser.KochtIn.Id)
            .SelectMany(b => b.EnthaeltGerichte).ToList();
```

5. Validierung

Tabelle 5.1: Erfüllung der Anforderungen aus Tabelle 2.1

Anforderung	**Erfüllungsgrad**
A	Erfüllt zu 90% durch Datenmodell (Kap. 3) und Funktion in Abb. 4.5
B	Erfüllt zu 100% durch Web-App ohne Telefon oder Mail
C	Erfüllt zu 100% durch Funktion in Kap. 4.6
D	Erfüllt zu 50% durch Funktion in Abb. 4.6
E	Erfüllt zu 30% durch Funktion in Kap. 4.3

Die Meisten der Anforderungen sind also durh die Web-App erfüllt. Eine Registrierungs- und Login-Funktion sicher zu implementieren um gegen Angriffe sicher zu sein liegt außerhalb dieser Arbeit und es wird sich somit mit einem Pseudo-Login zufriedengegeben.

Gerichte zu einer Buchung hinzuzufügen funktioniert zwar, jedoch kann eine Instanz eines Gerichts nur genau ein mal zu einer Buchung hinzugefügt werden. Dies sollte zwar möglich sein, wird jedoch durch ein "***feature***" von SQL verhindert. Einen Workaround um das Problem zu finden und implementieren ist sicher möglich, nähme jedoch deutlich mehr Zeit in Anspruch als für diese Arbeit eingeplant ist.

Durch die Eigenschaft der Web-App ein Interface zum Kunden zu bilden, wären Ästhetische Mängel wie das Design der Buttons oder der Navbar noch deutlich zu verschönern um die User Experience zu verbessern.

6. Fazit und Ausblick

6.1 Fazit

Abschließend kann man die im Rahmen dieser Seminararbeit entwickelte Web-App als hinreichend funktionsfähig bezeichnen. Alle Anforderungen sind, zumindest teilweise, erfüllt. So wie sie ist bietet die Web-App also ein Grundgerüst für den Aufbau einer funktionsfähigen Webseite für eine Restaurantkette oder auch nur eine einzelne Gaststätte.

Aktuell kann ein *Kunde* auch noch eine Bestellung für einen anderen Kunden aufgeben, was nur einem *Koch* erlaubt sein sollte. Allerdings wird beim Kunden so kein finanzieller Schaden angerichtet, da die erstellte Buchung anschließend nicht mit Gerichten gefüllt werden kann. So ist die Funktion also nicht direkt schädlich, kann aber zu Verwirrung beim Kunden führen.

Außerdem ist das Login-System derzeit nur ein Pseudologin ohne Eingabe eines Passworts oder Benutzernamens. Ein Registrierungssystem um neue User zu Registrieren existiert noch gar nicht.

Um volle Funktionsfähigkeit zu erlangen müssen also noch eine adäquate Login- und Registrierungs-Funktion, ein Workaround um in Kapitel 5 erwähntes SQL-Feature, ein schöneres User Interface implementiert und das Bestellsystem erweitert werden.

6.2 Ausblick

Ein Workaround um besagtes SQL-Feature wäre beispielsweise, für jede neue Buchung eine neue Entität des Gerichts anzulegen, dessen Implementierung aber den Rahmen dieser Arbeit übersetiegen würde.

Weitere Funktionen welche für ein Restaurant und den Kunden “nice to have“ wären, wären beispielsweise eine Option zum Reservieren von bestimmten Tischkategorien wie “gemütlich“, “bar“ oder “abholen“, wobei “abholen“ einen Kunden ansprechen soll, der sich sein Essen nach hause mitnehmen wollen. Weiter wäre eine Kapazitätsprüfung für Filialen wichtig um zu hochfrequentierten Zeiten eine Überreservierung zu verhindern.

Um eine Flexiblere Nutzung der Web-App zu ermöglichen könnte eine Rolle für das Management hilfreich sein. Unter dieser Rolle könnte beispielsweise ein neuer View für Filialen erreicht werden, unter welchem man auch neue Filialen hinzufügen oder geschlossene Filialen entfernen kann.

Um dem Koch das Leben noch weiter zu vereinfachen könnte dieser die Gerichte von den Bestellungen nehmen die er schon gekocht hat wenn er sich im **Kochen**-View befindet.

Literatur

[Ars18] Arseniy. *Layout Fix*. englisch. 2018. URL: https://stackoverflow.com/questions/48550955/mvc-bootstrap-navbar-not-working-after-running-nuget-updates.

[Kna20] Merlin Knaeble. *Vorlage für LaTeXauf overleaf*. deutsch. 2020. URL: https://www.overleaf.com/latex/templates/issd-thesis-template-iism-kit/jbjnwktscfdj.

[Sch21] Prof. Dr. Jörn Schlingensiepen. *Erweiterte Controller*. deutsch. 2021. URL: https://docs.google.com/presentation/d/1Sgxrqt4RBaGHiJ35LPKq8ak_CS64zpjPf_qfRPExips/edit#slide=id.g221d318674_0_18.

[Yij21] YijingSun-MSFT. *Update der Packages*. englisch. 2021. URL: https://docs.microsoft.com/en-us/answers/questions/502208/-42.html.