Entwicklung eines webbasierten Ressourcenverwaltungssystems

Fachinformatiker – Anwendungsentwicklung

Abschlussprüfung Sommer 2020

e.Consult AG

Technische Dokumentation

Ali Krezan

Inhalt

1. Technische Dokumentation

Dieses Projekt erlaubt eine einfache internetbasierende Verwaltung von Terminen, Veranstaltungen und Ressourcen durch Mitarbeiter der e.Consult.

Der Zugriff, durch registrierte Nutzer, erfolgt dabei ausschließlich über die Active Directory des Firmennetzwerk.

1.1. Versionierung

Version	Datum	Änderungsgrund / Bemerkungen
0.1	03.04.2020	Ersterstellung

Tabelle 1 - Versionierung

1.2. Freigabe

Version	Datum	Änderungsgrund / Bemerkungen
0.1	06.04.2020	Ersteinführung

Tabelle 2 - Freigabe

1.3. Programmüberblick

1.3.1. Programmgröße

Das Programm benötigt 440 MB.

1.3.2. Betriebssysteme

Plattformunabhängig

1.3.3. Programmiersprachen / Compiler

C# - ASP.NET MVC

1.3.4. Sonstige Programme

Browser muss .NET Framework unterstützen

1.3.5. Datenorganisation

Die Daten liegt zentral auf einem ,,Miranda‘‘ Server (Microsoft SQL-Server) der Firma e.Consult.

1.3.6. Programm Ressourcen

Im Firmeneigenen Repository

1.3.7. Bemerkung

Im Kapitel 5 werden exemplarisch anhand von Quellcode Beispielen die Interaktionen zwischen Daten, Model, Controller und View dargestellt.

1.4. Analysen und Design

Um alle Anwendungsfälle deutlich zu machen, wurden sie in einem Use-Case-Diagramm festgehalten

1.4.1. Anwendungsfalldiagramm - ec.Reservation General

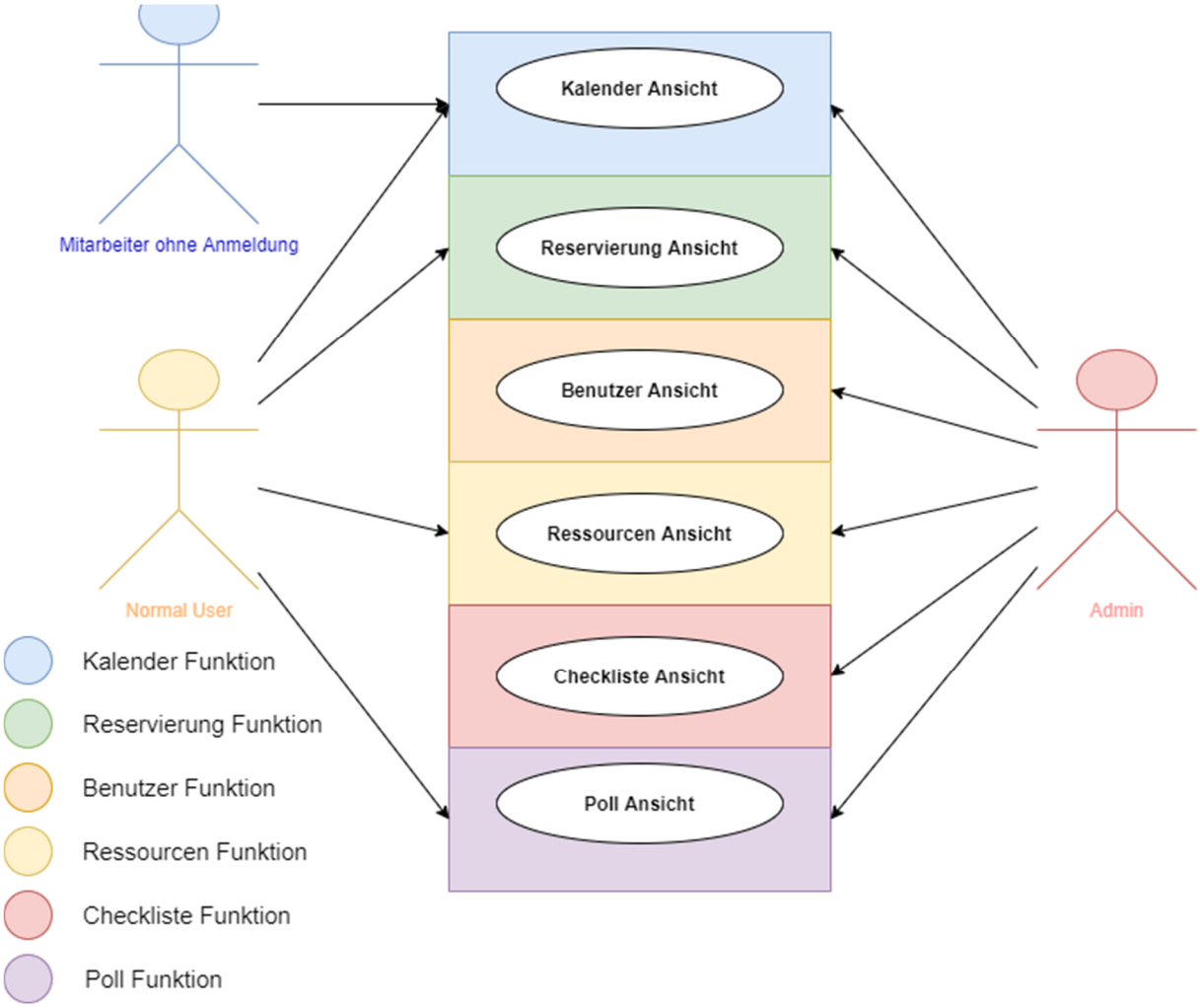


Abbildung 1 - ec.ReservationGeneral

1.4.2. Anwendungsfalldiagramm ,,Reservierungen“

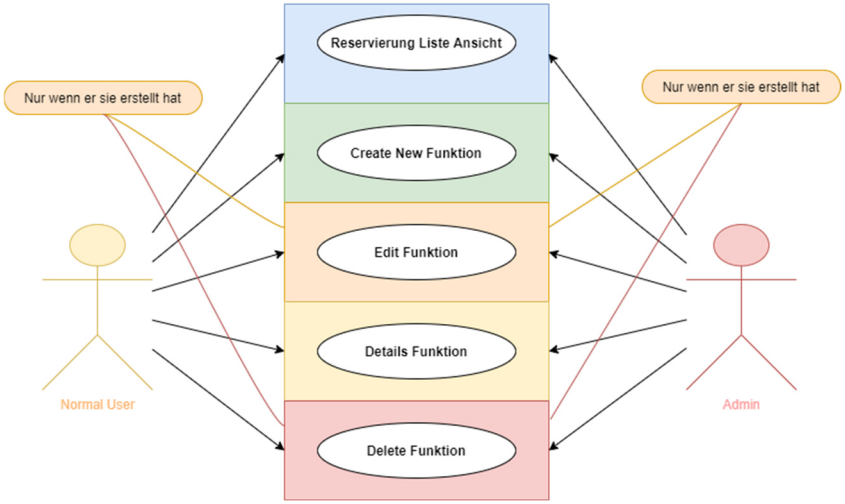


Abbildung 2 - Reservierungen

1.4.3. Anwendungsfalldiagramm „Users“

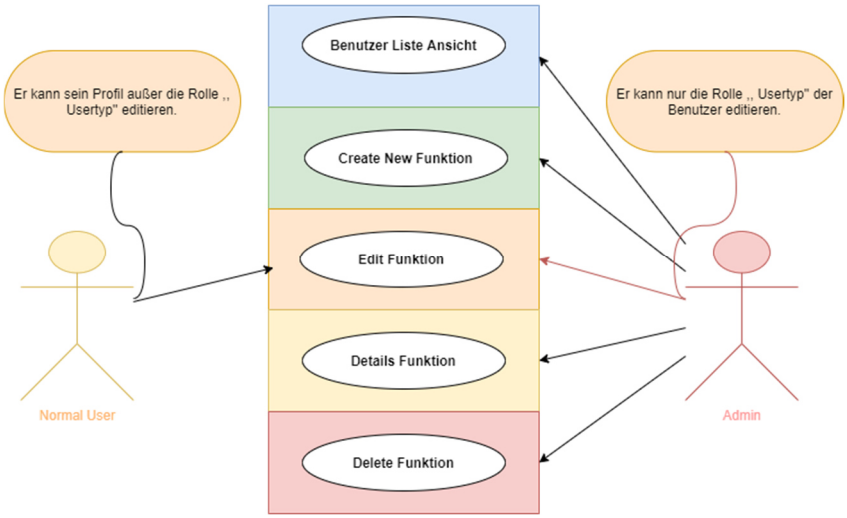


Abbildung 3 - Users

1.4.4. Anwendungsfalldiagramm „Polls“

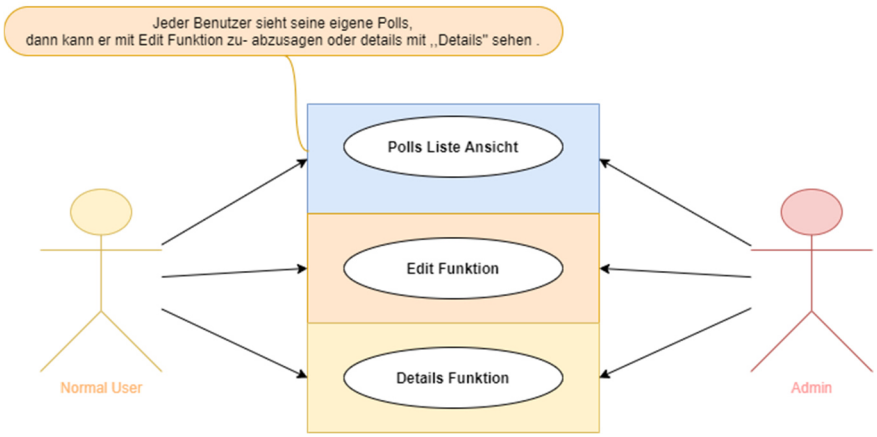


Abbildung 4 - Polls

1.4.5. Anwendungsfalldiagramm „Ressourcen“

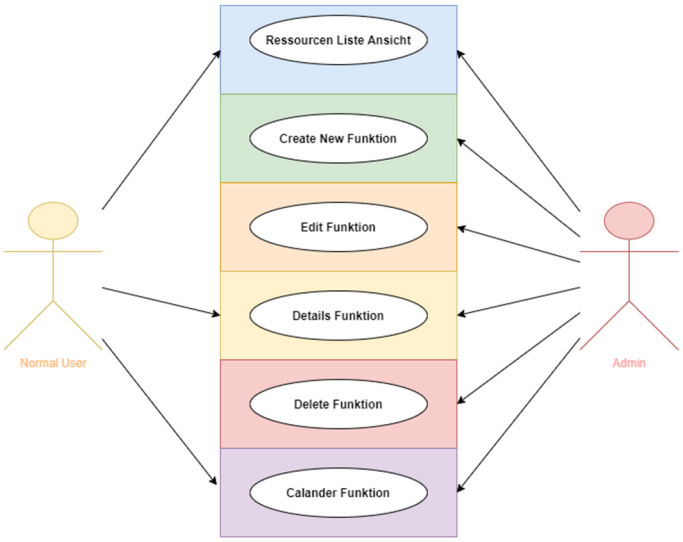


Abbildung 5 - Ressourcen

1.4.6. Aktivitäten Diagramm

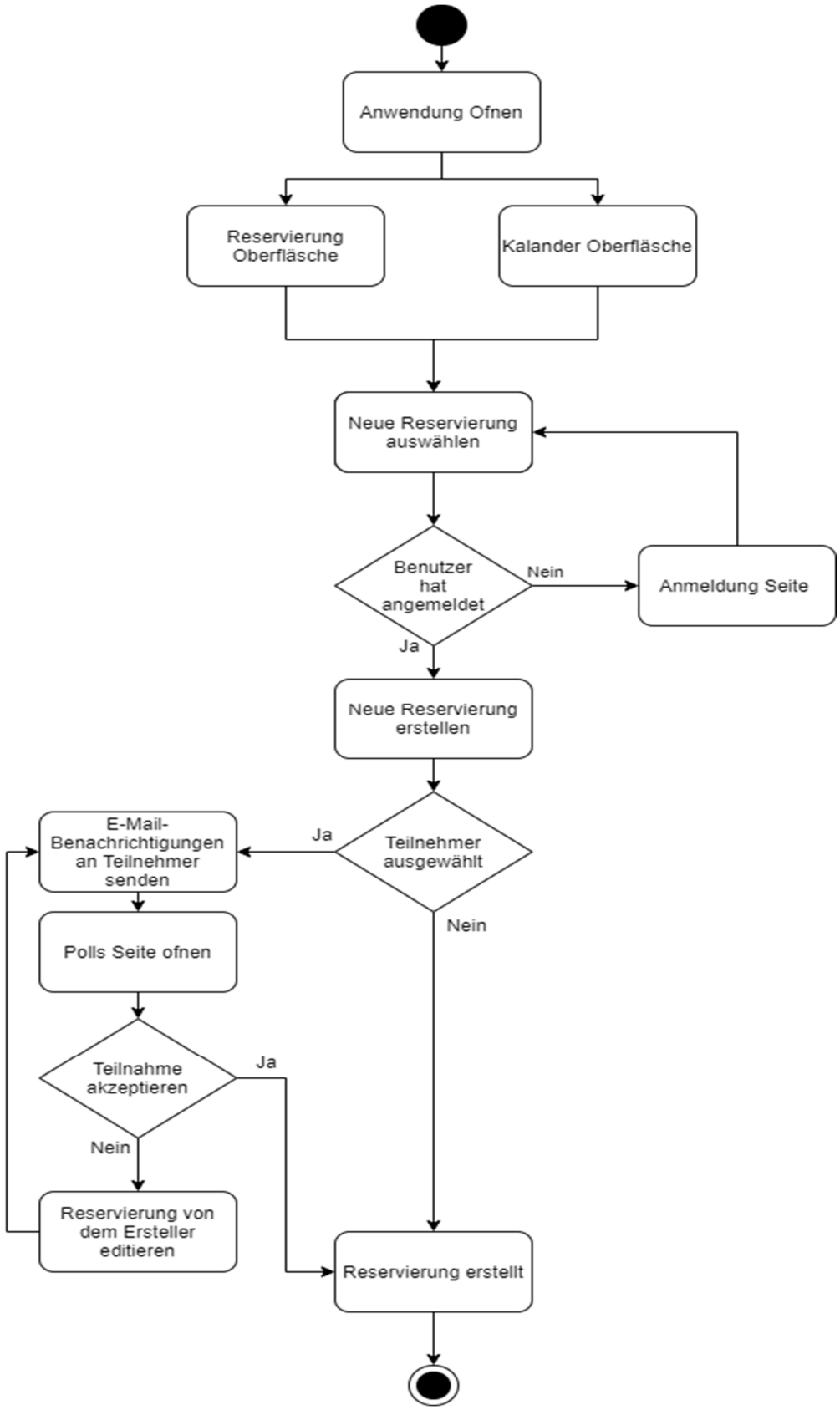
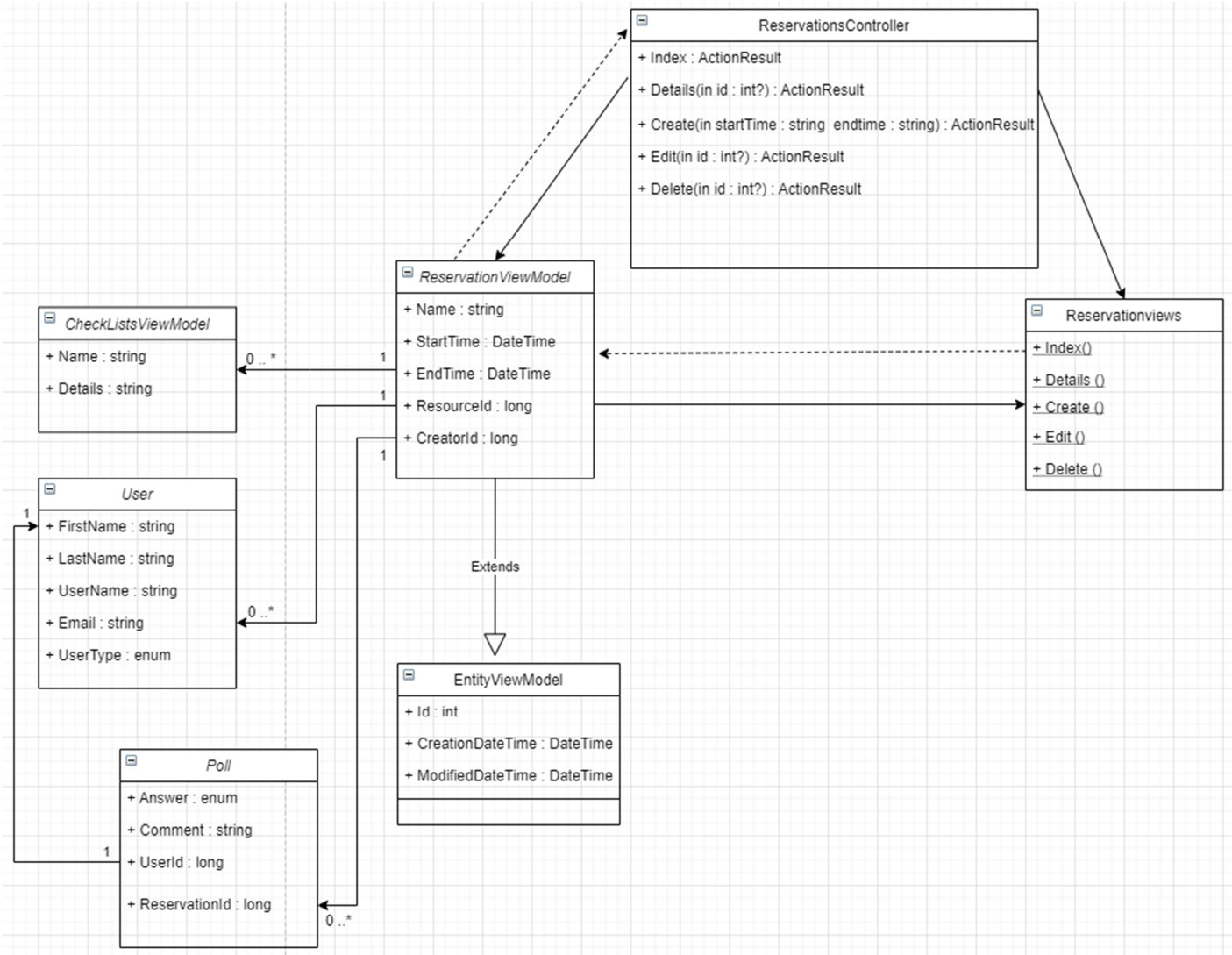


Abbildung 6 - Aktivitätendiagramm

1.4.7. Exemplarisches Klassen Diagramm - Reservation



2. Implementierungsbeschreibung

In Asp.net wurde eine Visual Studio Solution ,,ec.Reservation'' erstellt diese Solution ist mit .Net Framework 4.5.2 Version in vier Projekte verteilt :

ec.Reservation.Clients.Web	
Projekt-Typ	Asp.Net MVC
Dieses Projekt beinhaltet die Controller, Views sowie die View-Models und mit dem kann der Benutzer mit dem System kommunizieren.	

Tabelle 3 - ec.Reservation.Clients.Web

ec.Reservation.Entities	
Projekt-Typ	Net Class Library
- beinhaltet alle Domain Modelle als Entity. - verantwortlich für die Verbindung mit der Datenbank durch Microsoft Entity Framework. - dient als data access layer - Datenänderungen erfolgen nur hier	

Tabelle 4 - ec.Reservation.Entities

ec.Reservation.Services	
Projekt-Typ	Net Class Library
- dies ist der Business-Logic layer - beinhaltet alle Service Klassen - Service steuert die Business-Logik und kommuniziert mit ec.Reservtion.Entities	

Tabelle 5 - ec.Reservation.Services

ec.ReservationClientsWebTest	
Projekt-Typ	Net Unittest Projekt
- beinhaltet Test Methoden, um die Controller in ec.Reservation.Clients.Web zu testen.	

Tabelle 6 - ec.ReservationClientsWebTest

2.1. Beschreibung der Grundfunktionen

2.1.1. ec.Reservation.Clients.Web

Dieser Teil der Applikation enthält die Ansichten, Modelle und Kontrollelemente zur Verwaltung der Anwendung e.Reservation.

Tabelle 7 - ec.Reservation.Clients.Web

Controller	**Model**	**View**
AccountController	AccountViewModels EntityViewModelBase	Account
CheckListsController	CheckListsViewModel	CheckLists
HomeController	IdentityModels	Home
PollsController	PollsViewModel	Polls
ReservationController	ReservationViewModel	Reservations
RessourcesController	ResourcesViewModel	Ressources
UsersController	UsersViewModels	Users

Sie beinhaltet die Attribute über die Berichtigungen, die Anwendungssicherheit und blockiert den anonymen Zugriff.

Attribut:
ApiKeyAuthorize
CustomActionFilter
CustomAuthorizeAttribute

Tabelle 8 - Attribut

2.1.2. ec.Reservation.Entities

Diese Funktion enthält die Klassen zur Erstellung der Datenbank ReservationDB.

Zur Erstellung der ReservationDB-Datenbank wurde Code First-Migration mit Microsoft Entity Framework genutzt um die Datenbank leicht ändern oder erweitern zu können.

Die verwendeten Klassen:

- ✓ Entity
- ✓ Reservation
- ✓ User
- ✓ Resource
- ✓ CheckList
- ✓ Poll

Dann wurde die Klasse EntitiesContext erstellt, vererbt von DBContext. In dieser Klasse ist das "properties DbSet" für alle open genannten Entitys hinterlegt.Das Entity framework erstellt die Datenbanktabelle, Collum, Primary key und Foren key in der DbReservation Datenbank.

2.1.3. ec.Reservation.Services

Die EmailService Klasse

Sie enthält zwei Methoden:

- ✓ SendReservationNotificationEmail ist verantwortlich Emails an die Teilnehmer einer Reservation zu senden.
- ✓ SendNewUserNotificationEmail ist verantwortlich, eine Email an den Administrator zu senden, wenn ein neuer Benutzer sich erstmals in der Anwendung anmeldet.

Die ReservationService Klasse

Diese Klasse enthält verschiedene Methoden, welche Reservierungen in der Datenbank durch ec.Reservation.Entities zu steuern (CreateReservation, UpdateReservation).

Die UserService Klasse

Die Klasse enthält verschiedene Methoden Benutzer in der Datenbank durch ec.Reservation.Entities Projekt zu steuern.

2.1.4. ec.ReservationClientsWebTest

Dieser Projektteil enthält die Test-Klassen zur Überprüfung des Verhaltens zur Laufzeit.

3. Daten

Die Daten, die in diesem Projekt verarbeitet werden, befinden sich in der Datenbank ReservationDB, welche auf einem Miranda Server der Firma e.Consult zur Verfügung gestellt werden. Die Primärschlüssel sind mit einem Stern gekennzeichnet: (*).

3.1. Reservations

Spaltenname	Datentyp
Id(*)	Int
Name	nvarchar(MAX)
StartTime	Datetime
EndTime	Datetime
ResourceId	Int
CreatorId	Int
CreationDateTime	Datetime
CreationDateTime	Datetime
ModifiedDateTime	Datetime

Tabelle 9 – dbo.Reservations

3.2. Users

Spaltenname	Datentyp
Id(*)	Int
FirstName	nvarchar(MAX)
LastName	nvarchar(MAX)
UserName	nvarchar(MAX)
Email	nvarchar(MAX)
UserType	Int
CreationDateTime	Datetime
ModifiedDateTime	Datetime

Tabelle 10 – dbo.Users

3.3. Resources

Spaltenname	Datentyp
Id(*)	int
Name	nvarchar(MAX)
Code	nvarchar(MAX)
CreationDateTime	datetime
ModifiedDateTime	datetime

Tabelle 11 – dbo.Resources

3.4. CheckLists

Spaltenname	Datentyp
Id(*)	int
Name	nvarchar(MAX)
Details	nvarchar(MAX)
CreationDateTime	datetime
ModifiedDateTime	datetime

Tabelle 12 – dbo.CheckLists

3.5. Polls

Spaltenname	Datentyp
Id(*)	int
Answer	Int
Comment	nvarchar(MAX)
UserId	Int
ReservationId	Int
CreationDateTime	datetime
ModifiedDateTime	datetime

Tabelle 13 – dbo.Polls

4. Verknüpfungen

Zusätzlich zu den in Punkt 2. beschriebenen Daten gibt es auch Verknüpfungen zwischen den Datensätzen. Diese werden ebenfalls in Tabellen gespeichert.

4.1. UserReservations

Spaltenname	Datentyp
Reservation_Id (*)	int
UserId (*)	Int

Tabelle 14 – dbo.UserReservations

4.2. ReservationCheckLists

Spaltenname	Datentyp
Reservation_Id (*)	int
CheckList_Id (*)	Int

Tabelle 15– dbo.ReservationCheckLists

5. Exemplarischer Programmcode

Anhand von Code-Beispielen soll hier erläutert werden, wie die Daten in die Anwendung übertragen und dort verarbeitet werden.

5.1. ec.Reservation.Entities

5.1.1. ec.Reservation.Entities\EntitiesContext.cs

Datenbank Kontext

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ec.Reservation.Entities
{
   public class EntitiesContext : DbContext
    {
        public EntitiesContext() : base("ReservationDB")
        {

        }
        public DbSet<User> Users { get; set; }
        public DbSet<Resource> Resources { get; set; }
        public DbSet<CheckList> CheckLists { get; set; }
        public DbSet<Poll> Polls { get; set; }
        public DbSet<Reservation> Reservations { get; set; }

       protected override void OnModelCreating(DbModelBuilder modelBuilder)
        {
            modelBuilder.Conventions.Remove<OneToManyCascadeDeleteConvention>();

            modelBuilder.Entity<Reservation>().
                HasMany(c => c.Attendees).
                WithMany(p => p.Reservations).
                Map(
                    m =>
                    {
                        m.MapLeftKey("ReservationId");
                        m.MapRightKey("UserId");
                        m.ToTable("UserReservations");
                    });
        }
    }
}
```

Quellcodeauszug 1 Code 5.1.1 Datenbank Kontext

5.1.2. ec.Reservation.Entities\Entity.cs

Base Entity -beibehaltet ID und Datum, für alle geerbten Entitäten.

```
using System;

namespace ec.Reservation.Entities
{
    public class Entity
    {
        public long Id { get; set; }
        public DateTime CreationDateTime { get; set; }
        public DateTime? ModifiedDateTime { get; set; }
    }
}
```

Quellcodeauszug 2 -Code 5.1.2 Base Entity

5.1.3. ec.Reservation.Entities\Reservation.cs

Erstellen und ändern von Reservierungen

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Security.Cryptography.X509Certificates;

namespace ec.Reservation.Entities
{
    public class Reservation : Entity
    {
        public string Name { get; set; }
        public DateTime StartTime { get; set; }
        public DateTime EndTime { get; set; }
        public long ResourceId { get; set; }
        public long CreatorId { get; set; }
        public ICollection<long> ChecklistIds { get; set; }
        public ICollection<long> AttendeeIds { get; set; }
        public virtual Resource Resource { get; set; }
        public virtual User Creator { get; set; }
        public virtual ICollection<CheckList> CheckLists { get; set; }

        public virtual ICollection<User> Attendees { get; set; }
        public virtual ICollection<Poll> Polls { get; set; }

    }
}
```

Quellcodeauszug 3 - Code 5.1.3 Erstellen und ändern von Reservierungen

5.1.4. ec.Reservation.Entities\User.cs

Erstellen und ändern der Benutzereinträge

```
using System.Collections.Generic;

namespace ec.Reservation.Entities
{
    public class User: Entity
    {
        public string FirstName { get; set; }
        public string LastName{ get; set; }
        public string UserName { get; set; }
        public string Email { get; set; }
        public UserType UserType { get; set; }

        public virtual ICollection<Reservation> Reservations { get; set; }
        public virtual ICollection<Poll> Polls { get; set; }
    }
    public enum UserType
    {
        Administrator,
        NormalUser
    }
}
```

Quellcodeauszug 4 - Code 5.1.4 – Erstellen und ändern der Benutzereinträge

5.1.5. ec.Reservation.Entities\Resource.cs

Erstellen und ändern der Ressourcen

```
using System.Collections.Generic;

namespace ec.Reservation.Entities
{
    public class Resource: Entity
    {
      public string Name { get; set; }
      public string Code { get; set; }
    }
}
```

Quellcodeauszug 5 - Code 5.1.5 – Erstellen und ändern der Ressourcen

5.1.6. ec.Reservation.Entities\CheckList.cs

Erstellen und ändern der Checkliste

```
using System.Collections.Generic;

namespace ec.Reservation.Entities
{
   public class CheckList: Entity
    {
        public string Name { get; set; }
        public string Details { get; set; }
        public virtual ICollection<Reservation> Reservations { get; set; }

    }
}
```

Quellcodeauszug 6 - Code 5.1.6 – Erstellen und ändern der Checkliste

5.2. ec.Reservation.Clients.Web

Dieses Beispiele Zeigt die Zusammenarbeit der Api, Controllers, Views und den ViewModels:

5.2.1. Kalenderoberfläche

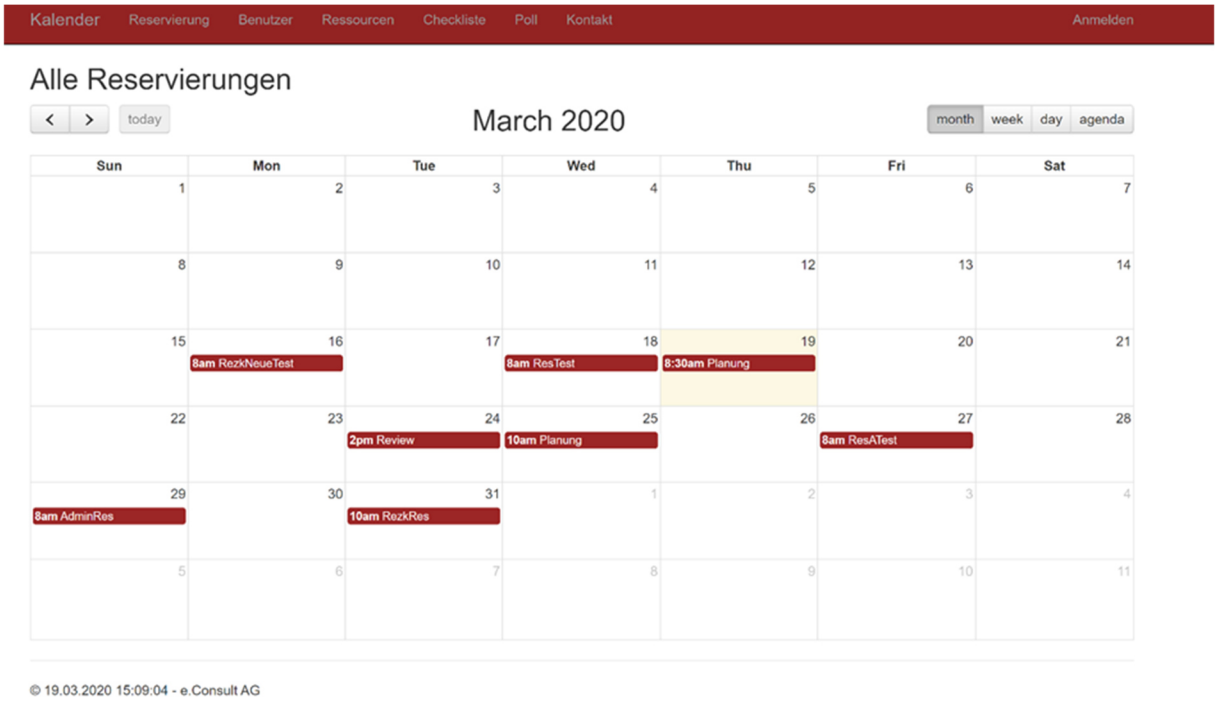


Abbildung 7 - Kalenderoberfläche

die Oberfläche nutzt die Klasse Homecontroller.cs um die Kalander durch die Datei Index.cshtml zu laden. Die Reservierungsdaten kommen in eine Liste von Event.cs, durch Ajax über die Rest Api ,,ReservationApiController.cs‘‘, der Rest Api Zugriff wird mit ApiKeyAuthorize.cs geprüft.

Homecontroller.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;
using Antlr.Runtime;
using ec.Reservation.Clients.Web.Models;
using ec.Reservation.Entities;

namespace ec.Reservation.Clients.Web.Controllers
{  public class HomeController : Controller
    {
        public ActionResult Index(int? id)
        {
            if (id == null || id == -1)
            {
                TempData["RessourceId"] = -1;
            }
            else
            {
                using (EntitiesContext db = new EntitiesContext())
                {
```

```
                    string ressourceName = (from r in db.Resources
                        where r.Id == id
                        select r.Name).Single();

                    ViewBag.RessourceName = ressourceName;
                }
                TempData["RessourceId"] = id;
            }
            return View();
        }

        public ActionResult About()
                {
                    ViewBag.Message = "Your application description page.";

                        return View();
                    }

        public ActionResult Contact()
                    {
                        ViewBag.Message = "Your contact page.";

                        return View();
             }
    }
}
```

Quellcodeauszug 7 - Code 5.2.2. – HomeController

Index.cshtml

```
@using System.Configuration;

@{
    ViewBag.Title = "Home Page";
    var ressourceId = TempData["RessourceId"];
    var ressourceName = ViewBag.RessourceName;
}
@{
    if (TempData["FlashMessage"] != null)
    {
        <div class="alert alert-danger alert-dismissible" role="alert">
            <button type="button" class="close" data-dismiss="alert" aria-
label="Close"><span aria-hidden="true">&times;</span></button>
            <strong>Entschuldigung!!</strong> @TempData["FlashMessage"]
        </div>
    }
}
@{
    int Id = Convert.ToInt32(ressourceId);
    if (Id == -1) {
        string reservationname = "Alle Reservierungen";

        <h2>
             @reservationname
        </h2>
    }
    else
    {
        <h1>
            @ressourceName
        </h1>
    }
}
<div id="calender"></div>
```

```
<div id="myModal" class="modal fade" role="dialog">
    <div class="modal-dialog">
        <div class="modal-content">
            <div class="modal-header">
                <button type="button" class="close" data-
dismiss="modal">&times;</button>
                <h4 class="modal-title"><span id="eventTitle" style="font-size: 22px;
font-style: italic; color: brown"></span></h4>
            </div>
            <div class="modal-body">
                <a id="btnDelete" href="" class="btn btn-default btn-sm pull-right">
                    <span class="glyphicon glyphicon-remove"></span> Remove
                </a>
                <a id="btnDetails" href="" class="btn btn-default btn-sm pull-right">
                    <span class="glyphicon glyphicon-list-alt"></span> Details
                </a>
                <a id="btnEdit" href="" class="btn btn-default btn-sm pull-right">
                    <span class="glyphicon glyphicon-pencil"></span>Edit
                </a>
                <p id="pDetails"></p>
            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-default" data-
dismiss="modal">Close</button>
            </div>
        </div>
    </div>
</div>

<link href="//cdnjs.cloudflare.com/ajax/libs/fullcalendar/3.4.0/fullcalendar.min.css"
rel="stylesheet" />
<link
href="//cdnjs.cloudflare.com/ajax/libs/fullcalendar/3.4.0/fullcalendar.print.css"
rel="stylesheet" media="print" />

@section scripts
{
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/moment.js/2.18.1/moment.min.js"></script>
    <script
src="//cdnjs.cloudflare.com/ajax/libs/fullcalendar/3.4.0/fullcalendar.min.js"></script
>
    <script>

        $(document).ready(function () {
            var events = [];
            var selectedEvent = null;
            var id = @ressourceId;

            $.ajax({
                beforeSend: function(request) {
request.setRequestHeader("ApiKey",'@ConfigurationManager.AppSettings["ValidApiKey"]');
                },
                type: "GET",
                url: '@ConfigurationManager.AppSettings["host"]'
                          +"/api/ReservationApi/" + id,
                success: function (data) {
                    $.each(data.Data, function (i, v) {
                        events.push({
                            eventID: v.EventID,
                            title: v.Subject,
                            description: v.Description,
                            creator: v.Creator,
```

```
                        start: moment(v.Start),
                        end: v.End != null ? moment(v.End) : null,
                        color: v.ThemeColor,
                        allDay: v.IsFullDay
                    })
                })

                GenerateCalender(events);
            },
            error: function (error) {
                console.info(error);
            }
        })


        function GenerateCalender(events) {
            $('#calender').fullCalendar('destroy');
            $('#calender').fullCalendar({
                contentHeight: 500,
                defaultDate: new Date(),
                timeFormat: 'h(:mm)a',
                header: {
                    left: 'prev,next today',
                    center: 'title',
                    right: 'month,basicWeek,basicDay,agenda'
                },
                eventLimit: true,
                eventColor: '#378006',
                events: events,
                eventClick: function (calEvent, jsEvent, view) {
                    selectedEvent = calEvent;
                    $('#myModal #eventTitle').text(calEvent.title);
                    var $description = $('<div/>');
                    $description.append($('<p/>').html('<b>Ersteller:</b>' +
                                    calEvent.creator));
                    $description.append($('<p/>').html('<b>Beschreibung:</b>' +
                                    calEvent.description));
                    $description.append($('<p/>').html('<b>Anfang:</b>' +
                              calEvent.start.format("DD-MMM-YYYY HH:mm a")));
                    if (calEvent.end != null) {
                        $description.append($('<p/>').html('<b>Ende:</b>' +
                              calEvent.end.format("DD-MMM-YYYY HH:mm a")));
                    }
                    $('#btnDelete').attr('href',
                       '@ConfigurationManager.AppSettings["host"]'
                       +"/Reservations/Delete/" + calEvent.eventID);
                    $('#btnDetails').attr('href',
                       '@ConfigurationManager.AppSettings["host"]'
                       +"/Reservations/Details/" + calEvent.eventID);
                    $('#btnEdit').attr('href',
                       '@ConfigurationManager.AppSettings["host"]'
                       +"/Reservations/Edit/" + calEvent.eventID);
                    $('#myModal #pDetails').empty().html($description);

                    $('#myModal').modal();
                },
                selectable: true,
                select: function (data) {
                    newDateStart = data.format("DD.MM.YYYY 08:00").toString();
                    newDateEnd = data.format("DD.MM.YYYY 09:00").toString();
                    var url = '@Url.Action("Create", "Reservations")';
                    window.location.href = url + "?startTime=" + newDateStart +
                                           "&endTime=" + newDateEnd;
                },
```

```
                    touch: addEventListener('touchstart', function (data) {

                        var newDateToCreate = data.srcElement.dataset.date.split('-');

                        var url = '@Url.Action("Create", "Reservations")';
                        window.location.href = url + "?startTime=" +
                           newDateToCreate[2] + "." + newDateToCreate[1] + "." +
                           newDateToCreate[0] + " 08:00" + "&endTime=" +
                           newDateToCreate[2] + "." + newDateToCreate[1] + "." +
                           newDateToCreate[0] + " 09:00";

                    }),

                    editable: true,
                    eventDrop: function (event) {
                        var data = {
                            EventID: event.eventID,
                            Subject: event.title,
                            Start: event.start.format('DD/MM/YYYY HH:mm A'),
                            End: event.end != null ?
                            event.end.format('DD/MM/YYYY HH:mm A') : null,
                            Description: event.description,
                            ThemeColor: event.color,
                            IsFullDay: event.allDay
                        };
                        SaveEvent(data);
                    }
                })

            }
        })

    </script>
}
```

Quellcodeauszug 8 - Code 5.2.3. –Kalander Oberfläche Seite mit Ajax

ApiKeyAuthorize.cs

```
using System.Web.Http;
using AutoMapper;
using System.Web.Http.Controllers;
using System.Collections.Generic;
using System.Linq;
using System;
using System.Configuration;

namespace ec.Reservation.Clients.Web.Helpers
{
    public class ApiKeyAuthorize : AuthorizeAttribute
    {
        protected override bool IsAuthorized(HttpActionContext httpActionContext)
        {
            var httpContext = httpActionContext;
            IEnumerable<string> values;
            var allKeys = httpActionContext.Request.Headers.TryGetValues("ApiKey", out
                          values);

            if(values!=null && !string.IsNullOrEmpty( values.FirstOrDefault()))
            {
               if(values.FirstOrDefault()==
                    ConfigurationManager.AppSettings["ValidApiKey"])
                {
                    return true;
```

```
                }
            }

            throw new Exception("Api Key not valid");

            return false;
        }
    }
}
```

Quellcodeauszug 9 - Code 5.2.4 – Rest API-Zugriff Validierung

ReservationApiController.cs

```
using ec.Reservation.Clients.Web.Controllers;
using ec.Reservation.Clients.Web.Helpers;
using ec.Reservation.Entities;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using System.Web.Mvc;

namespace ec.Reservation.Clients.Web
{
    [ApiKeyAuthorize]
    public class ReservationApiController : ApiController
    {

        // GET api/<controller>/5
        public JsonResult Get(int id)
        {
            using (EntitiesContext dc = new EntitiesContext())
            {
                var reservations = (from r in dc.Reservations
                                    where r.ResourceId == id
                                    join u in dc.Users
                                        on r.CreatorId equals u.Id
                                    join rec in dc.Resources
                                        on r.ResourceId equals rec.Id
                                    select new
                                    {
                                        ID = r.Id,
                                        Name = r.Name,
                                        StartTime = r.StartTime,
                                        EndTime = r.EndTime,
                                        Creator = u.FirstName,
                                        Resource = rec.Name
                                     }).ToList();

                if (id < 0)
                {
                    reservations = (from r in dc.Reservations
                                    join u in dc.Users
                                        on r.CreatorId equals u.Id
                                    join rec in dc.Resources
                                        on r.ResourceId equals rec.Id
                                    select new
                                    {
                                        ID = r.Id,
                                        Name = r.Name,
```

```
                                        StartTime = r.StartTime,
                                        EndTime = r.EndTime,
                                        Creator = u.FirstName,
                                        Resource = rec.Name
                                    }).ToList();
                }
                var events = new List<Event>();
                foreach (var reservation in reservations)
                {
                    var eventViewModel = new Event()
                    {
                        EventID = reservation.ID,
                        Subject = reservation.Name,
                        Description = reservation.Resource,
                        Creator = reservation.Creator,
                        IsFullDay = false,
                        ThemeColor = "#9c2525",
                        End = reservation.EndTime,
                        Start = reservation.StartTime
                    };
                    events.Add(eventViewModel);
                }
                return new JsonResult { Data = events, JsonRequestBehavior =
                                        JsonRequestBehavior.AllowGet };
            }
        }

        // POST api/<controller>
        public void Post([FromBody]string value)
        {
        }
        // PUT api/<controller>/5
        public void Put(int id, [FromBody]string value)
        {
        }
        // DELETE api/<controller>/5
        public void Delete(int id)
        {
        }
    }
}
```

Quellcodeauszug 10 - Code 5.2.5 – Alle Reservierungen von der Datenbank abrufen

Event.cs

```
public partial class Event
{
        public long EventID { get; set; }
        public string Subject { get; set; }
        public string Description { get; set; }
        public string Creator { get; set; }
        public System.DateTime Start { get; set; }
        public Nullable<System.DateTime> End { get; set; }
        public string ThemeColor { get; set; }
        public bool IsFullDay { get; set; }
}
```

Quellcodeauszug 11 - Code 5.2.6 – Event Klasse beinhaltet Reservierung-Details

5.3. ReservationClientsWebTest

Das Projekt hat Unittest Methoden in HomeControllerTest.cs definiert. Diese überprüfen, ob der Code wie erwartet funktioniert, indem Komponententests ausgeführt werden. Diese Tests werden als „Unittests“ bezeichnet, da Sie die Funktionalität des Programms, in einzelnes testfähiges Verhalten gliedert, die man als einzelne Komponenten testen kann. Mit dem Test-Explorer von Visual Studio kann man Komponententests flexibel und effizient ausführen und die Ergebnisse in Visual Studio anzeigen.

Komponententests dienen der Qualitätssicherung des Codes, da sie ein integraler Bestandteil des Softwareentwicklungsworkflows sind. So wird die Eingabe von Standarddaten, falschen Daten und Daten an der Grenze des Gültigkeitsbereichs überprüft. Zudem bieten die Tests die Möglichkeit, alle im Code enthaltenen expliziten oder impliziten Annahmen zu überprüfen.

5.3.1. HomeControllerTest.cs

```
using System;
using System.Web.Mvc;
using ec.Reservation.Clients.Web.Controllers;
using Microsoft.VisualStudio.TestTools.UnitTesting;
namespace ecReservationClientsWebTest
{
    [TestClass]
    public class HomeControllerTest
    {
        [TestMethod]
        public void Index_Controller_Erfolgreich_Erreichbar()
        {
            // Arrange
            HomeController controller = new HomeController();
            // Act
            ViewResult result = controller.Index(-1) as ViewResult;
            // Assert
            Assert.IsNotNull(result);
     }
        [TestMethod]
        public void About_Controller_Erfolgreich_Erreichbar()
        {
            // Arrange
            HomeController controller = new HomeController();
            // Act
            ViewResult result = controller.About() as ViewResult;
            // Assert
            Assert.AreEqual("Your application description page.",
              result.ViewBag.Message);
        }
        [TestMethod]
        public void Contact_Controller_Erfolgreich_Erreichbar()
        {
            // Arrange
            HomeController controller = new HomeController();
            // Act
            ViewResult result = controller.Contact() as ViewResult;
            // Assert
            Assert.IsNotNull(result);
        }
    }
}
```

Quellcodeauszug 12 - Code 4.4.1 – HomeControllerTest

5.3.2. .NetTest Explorer

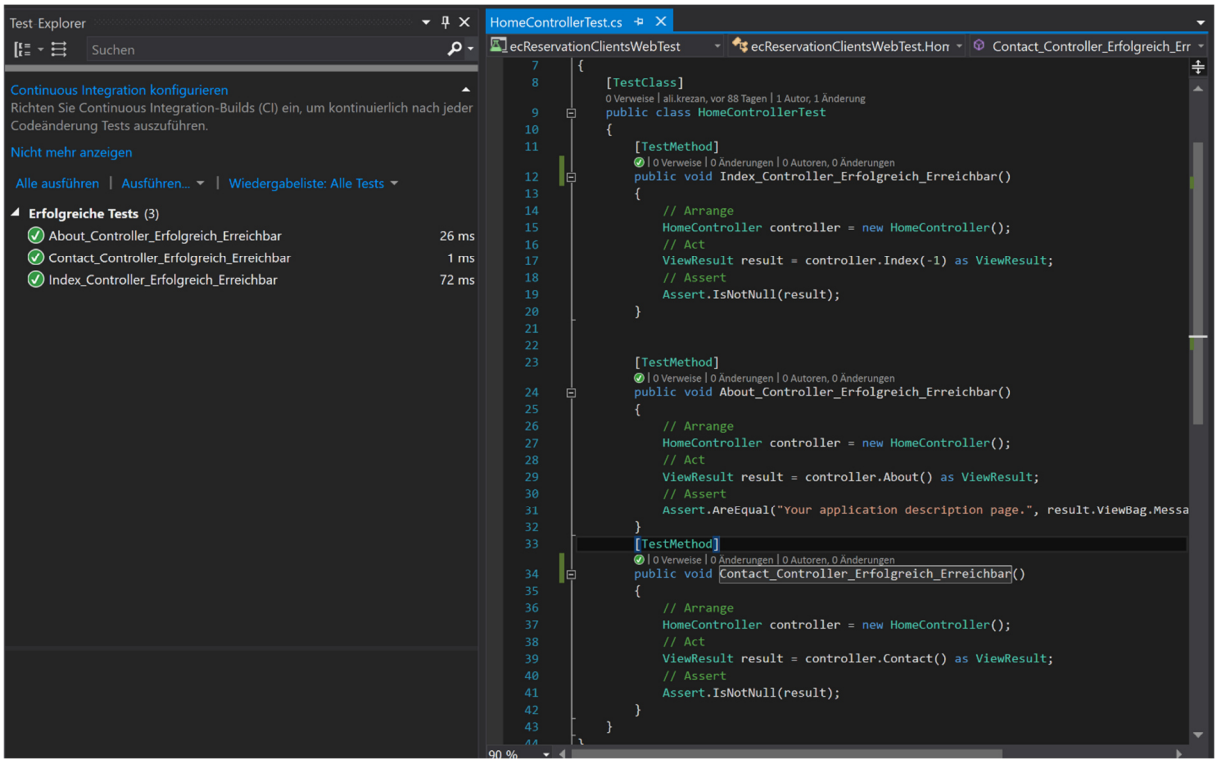


Abbildung 8 - HomeController Testergebnisse

5.4. Testprotokoll

Nr.	**Welche Funktionalität wird getestet? Erwartetes Ergebnis - Bemerkungen**	OK	Nicht OK
A	**Kalender**		
1	Alle Mitarbeiter können alle Reservierungen ohne Anmeldung sehen	✓	
2	Durch Klicken der „Vor“- und „Zurück“-Buttons (< >) kann zu einem anderen Monat gewechselt werden.	✓	
3	Durch Klicken des „Today“-Buttons kann man zum aktuellen Monat wechseln	✓	
4	Durch Klicken des „Month“-Buttons kann man die monatliche Kalender Ansicht sehen.	✓	
5	Durch Klicken des „Week“-Buttons kann man die wöchentliche Kalender Ansicht sehen.	✓	
6	Durch Klicken des „Day“-Buttons kann man die tägliche Kalender Ansicht sehen.	✓	
7	Durch Klicken des „Agenda“-Buttons kann man die stündliche Kalender Ansicht sehen.	✓	
8	Durch Klicken auf eine Veranstaltung öffnet sich ein Dialog, in welchem der Name der Veranstaltung, der Ersteller, die Ressource, die Terminierung, sowie die Buttons „Edit“, „Details“, und „Remove“ zu finden sind.	✓	
9	Durch Klicken auf ein Datumsfeld öffnet sich ein Dialog zum Erstellen eines neuen Eintrages.	✓	
B	**Authentifizierung**		
1	Die Anmeldung muss durch das Active-Directory des Unternehmens authentifiziert werden.	✓	

C	**Reservierung**		
1	Mit klicken des ,,Reservierung“ Button kann man alle Reservierung sortiert nach Erstellungsdatum sehen, in welchem der Name der Veranstaltung, der Ersteller, die Ressource, die Terminierung zu finden sind.	✓	
2	Mit klicken des ,,Edit“ Button von - Ersteller der Reservierung - öffnet die Seite ,,Edit Reservation“ und die Reservierung kann bearbeitet werden.	✓	
3	Mit klicken des ,,Details“ Button öffnet eine Seite, in welchem der Name der Veranstaltung, der Ersteller, die Ressource, die Terminierung, Datum und Uhrzeit der erstellen, Datum und Uhrzeit der Änderung, sowie Checkliste und die Teilnehmer mit ihrer Antwort zu finden sind.	✓	
4	Mit klicken des ,,Delete“ Button - von Ersteller der Reservierung - öffnet die Seite ,,Delete Reservation“ und die Reservierung kann gelöscht werden.	✓	
D	**Benutzer**		
1	Mit klicken des ,,Benutzer“ Button kann der Benutzer „nur mit Admin Rolle“ die Infos der Benutzer sortiert nach alphabetisch in Benutzer Seite sehen, in welchem der First Name, der Last Name, die User Name, die E-Mail und Benutzer Rolle zu finden sind.	✓	
2	Mit klicken des ,,Edit“ Button in der Benutzer Seite öffnet eine Seite, die nur die Rolle dem Benutzer enthält, und die Rolle kann bearbeitet werden. Wenn Sie jedoch auf den Namen des aktuellen Benutzers klicken, enthält die Seite alle zugehörigen Informationen, und die Info kann bearbeitet werden.	✓	
3	Mit klicken des ``Details`` Button öffnet eine Seite, in welchem der First Name dem Benutzer, die Last Name, die User Name, die E-Mail, die Rolle und Uhrzeit der erstellen, Datum und Uhrzeit der Änderung zu finden sind.	✓	
4	Mit klicken des ,,Delete“ Button öffnet die Seite ,,Delete User“ und den Benutzer kann gelöscht werden.	✓	
E	**Ressourcen**		
1	Mit klicken des ,,Ressourcen“ Button öffnet eine Seite, in der kann man alle Ressourcen sehen, diese Seite enthält der Name und dem Code der Ressource, sowie die Buttons „Edit“, „Details“, „Delete“ und „Calander“.	✓	
2	Mit klicken des ,,Edit“ Button öffnet die Seite ,,Edit Ressource“ und die Ressource kann bearbeitet werden.	✓	
3	Mit klicken des ,,Details“ Button öffnet eine Seite, in welchem der Name der Ressource, Datum und Uhrzeit der erstellen, Datum und Uhrzeit der Änderung zu finden sind.	✓	
4	Mit klicken des ,,Delete“ Button öffnet die Seite „Delete Ressource“ und die Ressource kann gelöscht werden.	✓	
5	Mit klicken des „Calander“ Button öffnet die Hauptseite „Vollständig Calander“, in welchem kann man sehen nur die Reservierung für zu dieser Ressource gehören sind.	✓	
6	Mit klicken des „Create New“ Button öffnet die Seite „Create Ressource“ und die neue Ressource kann hier erstellt werden.	✓	
F	**Checkliste**		
1	Mit klicken des „Checkliste**“** Button öffnet eine Seite, in der kann man alle Checklisten sehen, diese Seite enthält der Name der Checkliste und Einzelheiten über dieser Checkliste, sowie die Buttons „Edit“, „Details“, „Delete“.	✓	
2	Mit klicken des „Edit“ Button öffnet die Seite „Edit Checkliste“ und die Checkliste kann bearbeitet werden.	✓	
3	Mit klicken des „Details“ Button öffnet eine Seite, in welchem der Name der Checkliste, die Einzelheiten der Checkliste, Datum und Uhrzeit der erstellen, Datum und Uhrzeit der Änderung zu finden sind.	✓	
4	Mit klicken des „Delete“ Button öffnet die Seite „Delete Checkliste“ und die Checkliste kann gelöscht werden.	✓	
5	Mit klicken des „Create New“ Button öffnet die Seite „Create Checkliste“ und die neue Checkliste kann hier erstellt werden.	✓	

G	Poll		
1	Mit klicken des „Poll“ Button öffnet eine Seite, in dieser Seite kann man sehen alle Polls gehören zu ihm sortiert nach Erstellungsdatum, diese Seite enthält der Name der Reservierung , der Ersteller der Reservierung, Antwort dem Benutzer, Kommentar dem Benutzer, Datum und Uhrzeit der erstellen, Datum und Uhrzeit der Änderung, sowie die Buttons „Edit“, „Details“. Wenn der Benutzer noch nicht Teilnehmer in einer Reservation ist, findet den Benutzer keine Poll in der Seite aber mit Alert, dass er noch keine Poll hat.	✓	
2	Mit klicken des „Edit“ Button öffnet die Seite „Edit Poll“ und die Antwort und Kommentar über dieser Reservierung kann bearbeitet werden.	✓	
3	Mit klicken des „Details“ Button öffnet eine Seite, in welchem der Name der Reservierung, der Ersteller der Reservierung, Antwort der Benutzer (Teilnehmer), der Kommentar der Benutzer, Datum und Uhrzeit der erstellen, Datum und Uhrzeit der Änderung zu finden sind.	✓	
H	**Mobile Ansicht**		
	A -G wird mit dem Mobilgerät wiederholt	✓	

Tabelle 16 - Testprotokoll

6. Quellenverzeichnis

6.1. Abbildungen

6.2. Tabellen

6.3. Quellcodeauszug