Projekt Dokumentation WS2020/2021

IssuePilot

Niklas Knopp - Medieninformatik, Prüfungsordnung 2017
Tim Dohle - Angewandte Informatik, Prüfungsordnung 2016

Abgabedatum: 14.03.2021

Inhaltsverzeichnis

Einleitung

Der Grundgedanke bei dem Projekt war, eine Anwendung zu entwickeln, mit der jeder etwas anfangen kann, damit es ein gutes Projekt zum Lernen und Vorstellen ist.
Die Webanwendung IssuePilot wurde zur Verwaltung und Dokumentieren von Issues, sowie Fortschritt bei der Umsetzung von Projekten entwickelt.
Die Hauptfunktionen ermöglichen das Erstellen und Verwalten von Benutzern, Projekten und Tickets. Weiterhin können in den Statistiken Projekte ausgewertet werden.

Es werden folgende externe Frameworks und Bibliotheken verwendet:

- Entity Framework Core v3.1.9
- ASP.NET Core MVC 2.2.5
- Bootstrap Select v1.13.18
- xUnit v2.4.1
- FullCalendar v5.5.1
- Chart.js v2.9.4

Konzeption des Designs

Werte

Es wurde schnell klar, dass es nicht im Zeitrahmen des Projektes war, ein Helpdesk-System zu schaffen, die alle erdenklichen Funktionen abdeckt. Deswegen wurde darauf geschaut, eine ausgewählte Anzahl der wirklich gebrauchten Funktionen so einer Anwendung möglichst effektiv und einfach einsetzbar zu machen, damit ein Benutzer gerne damit arbeitet.

Substanzwerte

Kooperation

Verlässlichkeit

Effizienz

Kernwerte

Sicherheit

Elegant

Differentiatoren

Professionell

Minimalistisch

Herausstechend

Entwicklung der Primary Persona und User Stories

Durch die Wertsetzung wurde klar, dass die Anwendung sich an junge und kleine bis mittlere Software-Unternehmen richten soll, die offen sind, auf Features zu verzichten, um Zeit einzusparen. Die Zielgruppe besteht aus Arbeitern bei einem Software-Unternehmen mit Fokus auf Software-Entwickler. Dabei wurde beachtet, verschiedene Entwickler mit einzubinden wie auch verschiedene Rollen bei einem Software-Unternehmen.

Arne Arnsen, 27, Fullstack-Entwickler mit Schwerpunkt Frontend

Als Frontend-Entwickler bei seinem Unternehmen ist Arne sehr daran interessiert, dass seine Probleme an die Backend-Entwickler klar und einfach kommuniziert werden. Für unübersichtliche Software mit überhäuften Features hat er wenig übrig. Eine Software muss für Arne auch Spaß machen, damit man sie wirklich nutzt. Hierzu zählt für ihn ein herausstechendes, modernes Design und eine gute User Experience.

Bernd Berndsen, 39, Backend-Entwickler, Startup-Gründer

„Produktivität ist das A und O hier.“

Mit seinen anspruchsvollen Zielen ist Bernd als Startup-Gründer an das Schaffen einer produktiven Arbeitsumgebung interessiert. Der Effizienz und Kooperation soll nichts im Wege stehen. Der Prozess des Entwickelns soll professionell und sicher ablaufen. Dafür muss er auch unter den Entwicklern für die richtige Arbeitshaltung miteinander sorgen. Damit dies passiert, soll die Issue-Tracking-Software für jeden simpel nutzbar sein.

Brandfilter & USP

Das zu einem Satz reduzierte Unterscheidungsmoment (USP) oder auch Alleinstellungsmerkmal lautet wie folgt:

Auf Hauptfunktionen reduzierte, modern designte Issue-Tracking-Webapp für professionelle Verwendungszwecke.

	Minimalistisch	Herausstechend	Professionell
Look & Feel	Simpel, Sauber, Mindestmenge bei Variation von Schrift, Form und Farbe	Einzigartig, modern	Farbtöne eher im kälteren Spektrum, organisiert, klar erkennbar
Informations-architektur	möglichst wenig Seiten und möglichst wenig	Auf jeder Seite schnell einsehbar	Strukturiert, Abtrennung von

	Verzweigungen		Zweigungen
Nutzerführung & Navigation	Schnell Einsehbar	Elemente der Navigation stilistisch neu betonen	Aufgeräumte Darstellung
Bewegung & Interaktion	Offensichtlich	Animationen bei Elementen als Verdeutlichung	Bekanntes nutzen

Informationsarchitektur

Durch die geringe Anzahl an Funktionen und im Sinne der Konzeption wurde bei der Informationsarchitektur geschaut, die Auswahl simpel, und die Abzweigungen klar und übersichtlich zu halten. Die grundlegenden Funktionen “Ticket erstellen” haben einen besonderen Wert erhalten. Sie ist von überall aus anklickbar. Abgesehen davon teilt sich die Architektur in vier Gruppen auf Dashboard, Projekt, Nutzerverwaltung und Statistiken. Da Tickets in Projekten ist dies die größte Gruppe.

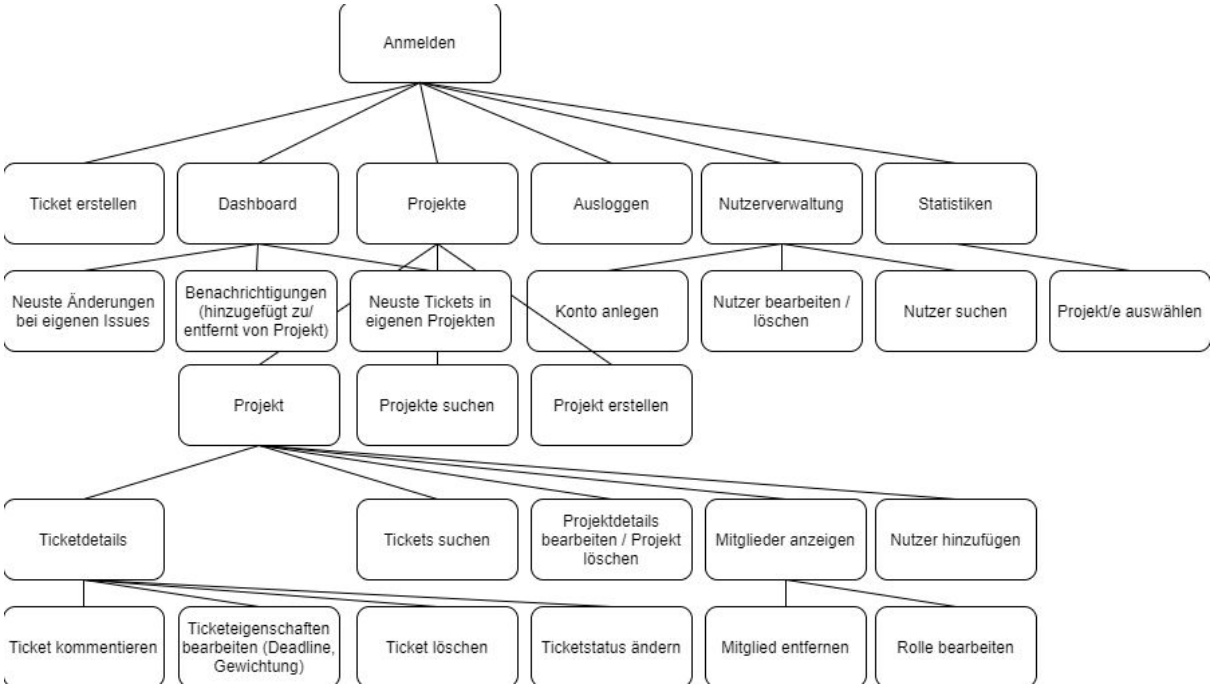

Abbildung 1: Informationsarchitektur

Farben

Der Konzeption entsprechend wurde sich für ein professionelles und modernes Farbschema entschieden. Weiß, Blau und Schwarz verraten, dass die Anwendung in einer Arbeitsumgebung genutzt werden soll. Diese Ordnung wird bei Highlights gebrochen, um das Auge gespannt zu halten. Dies passiert durch Farbverläufe, herausstechende Buttons und herausstechende Icons.

Logo

Das Logo stellt mit dem Namen die Werte klar dar: Das englische Wort “pilot” was mit “Pilot”, “steuern”, oder “lotsen” übersetzt werden kann. Dies verdeutlicht die Kooperation und Professionalität. Das Logo bleibt mit seinen Farben professionell, ist durch seinen modernen Farbverlauf herausstechend und elegant und stellt ein Flugzeug minimalistisch dar.

Abbildung 2: Logo

Design-Prototyp (Mock-Ups)

Für das Prototyping wurde die Software Figma verwendet. Hier wurde sich über Schriften, Buttons und Layout Gedanken gemacht.
Die Prototypen sind anschaubar über dem folgenden Link:
https://www.figma.com/file/c9zyg8ASpA7HVKbgPHiWUj/Untitled?node-id=0%3A1

Entity Framework (EF) Core

Datenbank

Die Datenbank wird mithilfe von Modell-Klassen (Code-First) erzeugt, welche sich im Projekt unter “/Models/DBModels” befinden. In diesen Klassen werden Primärschlüssel, Attribute und weitere Spezifikationen wie z.B. Datentyp des Datensatzes festgelegt.

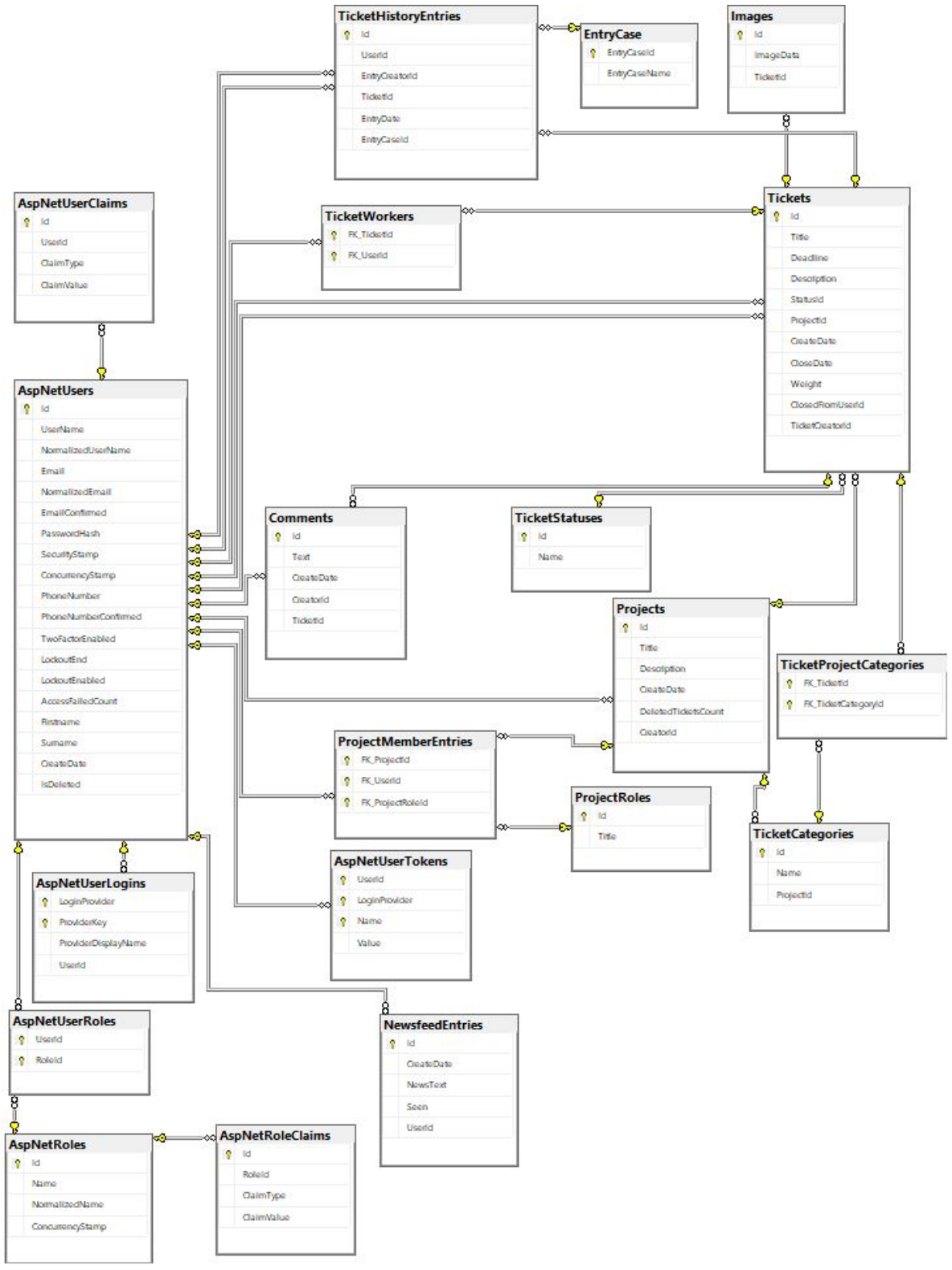

Abbildung 3: Datenbankdiagramm

Migration

Da Entity Framework Core automatisierte Migration nicht unterstützt, muss bei einer Änderung am Datenbankmodell oder der Seed-Daten manuell über die Konsole mit dem

Befehl `PM> dotnet ef migrations add eigenerNameFürDieMigration` eine aktualisierte Migrations- / Snapshot-Datei erzeugt werden.
Um die Datenbank nicht manuell mit der Konsole erzeugen zu müssen, wird dies in der Startup.cs durch `applicationDbContext.Database.Migrate();` übernommen. Die Datenbank muss zuvor jedoch manuell gelöscht werden, sollte bereits eine existieren.

Repository

Die Repositorys beinhalten alle Datenbankoperationen des Projekts, und soll als Abstraktionsebene zwischen Controller und der Datenzugriffsebene fungieren. Somit wird der Datenbandkontext ausschließlich in den Repositorys aufgerufen. Dazu setzten diese die jeweils zugehörigen Interfaces um.

Globaler zugriff

Um auf die Repositorys zugreifen zu können, müssen diese mit dem zugehörigen Interface in der Startup.cs durch die Zeile `services.AddScoped<IUserRepository, UserRepository>();` bekannt gemacht werden.

Demo-Daten

Die Demodaten werden bei der Migration der Datenbank aus den Konfigurationsdateien (/Data/Configuration), welche die Demo-Daten enthalten, generiert. Dazu muss in der OnModelCreating-Methode der ApplicationDbContext.cs jede Konfigurationsdatei dem Builder des Models mit der ApplyConfiguration-Methode hinzugefügt werden.

Demo-Konten

Die Demokonten werden bei den Demo-Daten mit generiert. Der Login per Button auf der Startseite ermöglicht das Einloggen mit einem Klick darauf, indem die Zugangsdaten für die Konten in versteckten Eingabefelder gespeichert wurden, welche beim Betätigen des Buttons in einer eigenen Form abgesendet werden.

```
<form method="post">
  <div class="form-group">
    <input type="hidden" asp-for="Input.UserName" class="form-control"
value="DemoAdmin" />
  </div>
  <div class="form-group">
    <input type="hidden" asp-for="Input.Password" class="form-control"
value="password" />
  </div>
  <div class="form-group">
    <button type="submit" class="btn btn-primary">Admin-Demokonto</button>
  </div>
</form>
```

[Siehe: \Areas\Identity\Pages\Account\Login.cshtml]

ASP.NET Core

ASP.NET Core ist ein Framework, welches zur Entwicklung von Webanwendungen benutzt wird und hier das MVC-Pattern verwendet. [9]

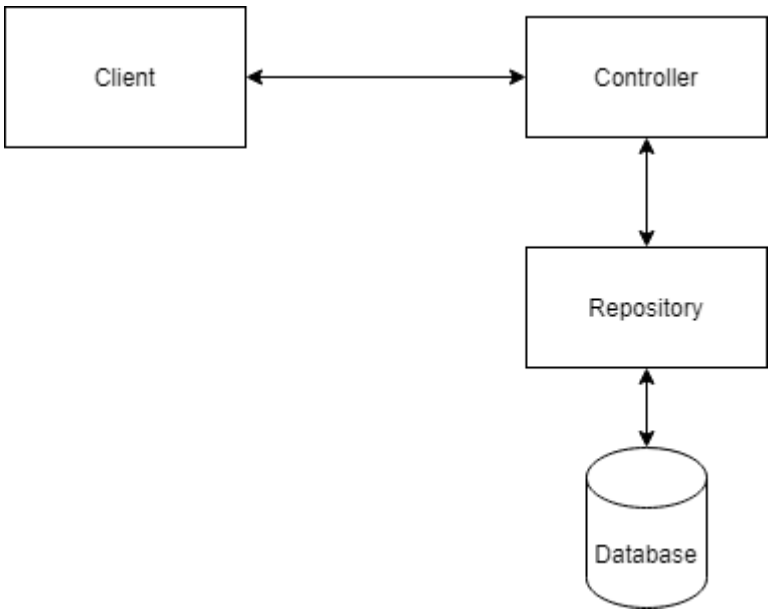

Abbildung 4: Kommunikation zwischen den Komponenten

Kommunikation zwischen den Komponenten

Client

Der Client nutzt .cshtml Dateien zum Anzeigen von Daten. Der Dateityp erlaubt die Nutzung von HTML und C#.
Zum Daten senden werden GET / POST-Requests an den jeweiligen Controller und die dazugehörige Methode gesendet. Eine cshtml-Datei muss dabei den gleichen Namen haben, wie die in dem Controller zugehörigen Funktion.
Zum Empfangen und Darstellen der Daten vom Controller müssen diese mithilfe eines ViewModel übergeben werden, welches über eine using-Direktive in der .cshtml bekannt gemacht werden muss.

Controller

Zum Verarbeiten der eingehenden Anfragen der View (u.a. CRUD-Operationen), verwendet der Controller die Funktionen der Repositorys. Bei Rückgabewerten von den Repository-Methoden verarbeitet der Controller diese und gibt diese Daten dann an die jeweilige View mithilfe eines ViewModels weiter.

Repository

Innerhalb der Repositorys werden mit Entity Framework Core, welches LINQ benutzt, Methoden zur Kommunikation mit der Datenbank zur Verfügung gestellt. Diese bauen auf der Funktionalität der CRUD-Operationen (Create, Read, Update, Delete) auf.

Benutzer und Rollen

System Zugriff- und Rechteverwaltung

Zur Zugriff- und Rechteverwaltung wird die von ASP.NET Core mitgelieferte rollenbasierte Autorisierung verwendet.[6]
Die Rollenbezeichnungen im Projekt lauten: “Admin”, “Projektmanager” und “Benutzer”.

Projekt Zugriff- und Rechteverwaltung

Die Projektrollen “Eigentümer/in” und “Teilnehmer/in” werden über eine Assoziationstabelle in der Datenbank an Benutzer vergeben. Basierend auf diesen Einträgen wird dann geprüft, ob der Benutzer ein Mitglied des Projektes ist und welche Rechte er innerhalb dieses Besitzt. Dabei wird ebenfalls überprüft, ob der Benutzer ein Administrator ist, da diese Zugriff auf alle Projekte und Funktionen besitzen.
[Siehe \Helper\ProjectAccessController.cs]

Benutzerverwaltung

ASP.NET Core Identity

Das von ASP.NET Core mitgelieferte Identity dient zum Verwalten und Speichern von Benutzerkonten. Es wird weiterhin zur Authentifizierung und Rechteverwaltung bezogen auf Benutzerkonten und deren zugeordneten Rollen verwendet. [2]

User-DB-Model

Das von Identity verwendete Modell für den Benutzer wurde um Attribute und Beziehungen zur Benutzernamen-Generierung, “Löschung” des Benutzeraccounts, Zuordnung von Projekten und Tickets erweitert.
[Siehe \Models\DBModels\User.cs]

Login

Der Login wurde so modifiziert, dass sich der Benutzer mit Benutzernamen und Passwort und nicht länger mit der E-Mail und Passwort einloggen soll.
[Siehe \Areas\Identity\Pages\Account\Login.cshtml.cs]

Benutzerkonto bestätigen

Die standardmäßig aktive Benutzerkonten-Bestätigung von Identity wurde in der Startup.cs mit der Zeile `options.SignIn.RequireConfirmedAccount = false` deaktiviert.

Benutzernamen Generierung

Der Benutzername wird nach Erstellung eines neuen Benutzerkontos aus mit `generatedUserName = firstname + surname + randomNumber;` generiert.

Benutzer löschen

Die Benutzerkonten werden nicht aus dem System gelöscht, sondern werden alle personenbezogenen Daten des Kontos entfernt. Dies dient dazu, um Beiträge und Kommentare noch nach Löschung auseinanderhalten zu können. Beim Löschen wird ein zufälliger Benutzername (“gelöschterNutzer” + Zufallszahl) generiert.

Automatischer Logout

Der Benutzer wird automatisch ausgeloggt, sofern er die letzten 30 Minuten nicht aktiv war. Dies wird über Cookies, in der Startup.cs mit dem Service `ConfigureApplicationCookie`, realisiert. Nach dem Logout wird der Benutzer wieder auf die Login-Seite geleitet.

Projekt

Die Komponente Projekt wird mit ASP.NET Core und den zuvor beschriebenen CRUD-Operationen umgesetzt.

Tickets

Zuerst wurden auch hier die CRUD-Operationen im Ticket-Controller angepasst und mit in das Repository ausgelagert. Für die Methoden bei Details können mehrere, verschiedene Funktionen ausgeführt werden. Dafür musste ein Switch für die verschiedenen Werte der Inputs aus der View erstellt werden, der diese bearbeitet.

Statistiken

Für Statistiken wird nur die Read-Operaton der CRUD-Operationen verwendet. Die dadurch erhaltenen Daten werden neben tabellarischer Darstellung noch als Kalender und Kuchendiagramm angezeigt.
Zur Darstellung als Kreis-/ Kuchendiagramme wurde mit Chart.js verwendet.[8]
Die kalendarische Darstellung wurde mit dem JavaScript-Kalender FullCalendar umgesetzt. [7]
Um die Daten in JavaScript verwenden zu können müssen diese in der .cshtml als Raw-Html mit JSON formatiert werden.

```
var XLabelsStatus = 
@Html.Raw(Newtonsoft.Json.JsonConvert.SerializeObject(Model.ListNumbersOfTicketS
tatus.Select(x => x.StatusName).ToList()));
```

[\Views\Statistics\Details.cshtml]

Paginated List

Auf Seiten, wo eine unbegrenzte Anzahl von Elementen angezeigt werden soll, wurde eine Pagination mit Hilfe Microsofts Vorgaben [10] eingebaut. Dies betrifft die folgenden Seiten:
Views/Administration/Index
Views/Dashboard/Index
Views/Project/Index
Views/Project/Members

Views/Project/Other
Views/Ticket/Tickets
Views/Ticket/TicketHistory
Views/Ticket/SelectProject

Für die Pagination wird intern die Klasse PaginatedList verwendet. Diese teilt eine Liste von Elementen in Seiten auf, die in den Views über ViewModel dargestellt werden. Über Asp-Routing werden in der View dann die Seiten gewechselt.

Sortierung, Filterung und Suche

Für die Sortierung wurden in den Get-Methoden ViewDatas gesetzt, die für die View die Sortierungs-Option bereitstellt. Z. B. wird in der Methode für das Anzeigen der Projekte ein ViewData für das Datum erstellt, der den Parameter “sortOrder” festlegt.

```
ViewData["DateSortParm"] = String.IsNullOrEmpty(sortOrder) ? "Date" : "";
```

Der String “sortOrder” wird danach zum Sortieren der aus der Datenbank geholten Projektliste genutzt.

```
switch (sortOrder)
            {
                case "title_desc":
                    projects = projects.OrderByDescending(s =>
s.Title).ToList();
                    break;
                case "Date":
                    projects = projects.OrderBy(s => s.CreateDate).ToList();
                    break;
                case "Title":
                    projects = projects.OrderBy(s => s.Title).ToList();
                    break;
                default:
                    projects = projects.OrderByDescending(s =>
s.CreateDate).ToList();
                    break;
            }
```

Bei der Suche wird über Asp-Routing in der View der eingegebene String als Parameter der Controller-Methode übergeben. Mit diesem wird dann die zu durchsuchende Liste mit der Linq-Methode “Contains” aus der Datenbank abgefragt.

Bei Tickets kann auch nach Status gefiltert werden, indem bool-ViewDatas angeben, welche Status mit der Linq-Methode “Except” aussortiert werden.

Umsetzung des Designs mit Bootstrap

Die Views wurden mit Hilfe von Bootstrap umgesetzt. Eigenes CSS wurde auch geschrieben in Site.css. Als Ressource diente hier hauptsächlich die Bootstrap-Dokumentation. Die Bootstrap-Erweiterung “bootstrap-select” wurde für das multiple Auswählen bei Views

verwendet. Viele der Ideen aus dem Prototyping erwiesen sich als schlechte Ideen oder als zu aufwendig. Die Idee der linksliegenden, ausfahrbaren Navigationsleiste war für die User Experience wegen der bereits überschaubaren Verzweigung und Navigation kein Vorteil.

Tests

Zur Umsetzung der kontinuierlichen Integration werden mit dem Framework xUnit und der von EF Core mitgelieferten In-Memory-Datenbank Unit-Tests durchgeführt.

Zur Verwendung der In-Memory-Datenbank wird der Kontext mit Seed-Daten über die Klasse InitDbWithData.cs erstellt und an die Testklassen vererbt. In der Basisklasse sind Listen mit Daten angelegt, welche zum einen als Seed-Daten dienen und zum anderen zu vergleichen mit den Tests nach Anwendung der Repository-Methoden.

Es werden alle Repository-Methoden getestet, welche nicht aus reinen LINQ (Language Integrated Query) Operationen bestehen. Somit ist es nicht das Ziel eine 100 % Code-Abdeckung zu erreichen.
Aufgrund der Unzugänglichkeit bei der Testung, werden alle Methoden, welche teilweise oder komplett auf ASP.NET Core Identity basieren nicht getestet.
Die Tests prüfen auf erwartete Ergebnisse der Repository-Methoden und händisch eingefügte Exceptions. Auf Exceptions von EF Core wird nicht getestet.

Die Tests sind auf dem AAA-Muster (Arrange, Act, Assert) aufgebaut, wie hier beispielsweise zu sehen:

```
[Fact]
public async System.Threading.Tasks.Task
UpdateProjectAsyncTest_SucceedDescriptionUpdate()
{
   using var context = InitWithDataAndContext();

   // Arrange
   ProjectRepository projectRepository = new ProjectRepository(context);
   Project project = await context.Projects.FirstOrDefaultAsync();

   string description = "updated2";

   ProjectUpdateViewModel modelSucceedDescription = new
ProjectUpdateViewModel() { Id = project.Id, Title = project.Title,
Description = description };

   // Act
   var updateSucceedDescription = await
projectRepository.UpdateProjectAsync(modelSucceedDescription);

   // Assert
   // Is description updated if title is unchanged?
```

```
    Assert.Equal(description, updateSucceedDescription.Description);
}
```

[\IssuePilot.Test\ProjectRepositoryTests.cs]

Fehlerbehandlung

Die globale Fehlerbehandlung durch die Middleware von ASP.NET Core reguliert, wodurch alle nicht behandelten Fehler abgefangen werden.
Dabei wird unterschieden, ob das Programm im “Development”- oder “Release”-Modus ausgeführt wird. Dies wird in der launchSettings.json festgelegt.
Beim Development-Modus wird dem Anwender eine Seite mit detaillierten Informationen angezeigt, wenn eine Exception ausgelöst wird. Im Production-Modus wird dem Anwender nur eine allgemeine Fehlerseite angezeigt.

Nicht implementierte Anforderungen

2.18 “Wenn ein Benutzer gelöscht wurde, soll er aus allen Projekten als Mitglied ausgetragen werden.” [Priorität: kann]
Begründung: Diese Anforderung wurde bei der Umsetzung übersehen. Dies fiel zu spät auf und wurde daher aus Zeitgründen nicht umgesetzt.

5.5 “Es soll möglich sein, nach ID oder Projekttitel (alphabetisch) zu sortieren.” [Priorität: kann]
Begründung: ID wurde mit Erstelldatum ausgetauscht, da dies die gleiche Sortierreihenfolge ergibt.

5.20 “Die Tickets sollen nach einzelnen Kategorien gefiltert werden können. [Priorität: kann]
Begründung:
Da es unbegrenzt viele Kategorien geben kann, ergab sich als schwer, diese in der Suche mit anklickbaren Filtern skalierbar einzubauen.

6.30 “Die Nutzernamen, bei Kommentaren, sollen in unterschiedlichen Farben dargestellt werden.”
Begründung: Umsetzung war nicht mehr im Zeitrahmen.

Probleme und Fehlermeldungen

Probleme Visual Studio 2019

Es wird kein Browserfenster der Anwendung geöffnet

Problem: Beim Starten des Programms (Debug/ Release) wird kein Browserfenster mit der Anwendung geöffnet.
Lösung: Entweder werden alle Prozesse, welche involviert sein können, beendet oder der Computer muss neu gestartet werden.

Projekte können nicht erstellt werden

Problem: Beide Projekte (Programm und Tests) enthalten angeblich je einen Fehler, wodurch die Projekte nicht erstellt bzw. neu erstellt werden können. Diese Fehler werden jedoch nicht in der Fehlerliste von Visual Studio angezeigt. Dies scheint ein Bug von VS 2019 zu sein.
Lösung: Die Anwendung Visual Studio schließen und neu starten.

EF Core

Sql Migrationsfehler

Problem: Beim Generieren der Migration / Snapshot-Datei wird ein Fehler ausgeworfen.
Lösung: Ordner mit den Migrations-Versionen (Snapshot) löschen, (wenn existiert) die Datenbank löschen, Visual Studio neu starten und den Konsolenbefehl zur Migration erneut ausführen lassen.

Änderungen am Model werden ignoriert

Problem: Änderungen am Model werden nach der Migration bei der Erstellung der Datenbank nicht berücksichtigt.
Lösung: Der Grund konnte nicht ermittelt werden, jedoch ist dieses Problem behoben, sobald der Ordner mit den Migrations-Versionen (Snapshot) gelöscht und neu generiert wird.

IQueryable

Ein wichtiger Teil des Projektes war das Einfügen der PaginatedList, die für das Wechseln der Seiten bei Index-Seiten mit vielen Einträgen verantwortlich war. Dafür wurde die Vorgabe aus den Microsoft Docs für Asp.net Core 3.1 verwendet. [10]
Es erwies sich als schwer, diese umzusetzen, da die Funktion zum Erstellen der Liste ein IQueryable statt einer Liste erwartet. Dieser Datentyp stellte einige Probleme dar: Casten einer List in ein IQueryable oder Vergleich der Typen sowie Datenbankfunktionen mit Parametern, die nicht IQueryable waren, war durch einen Bug in EF-Core in Asp.net Core 3.1 nicht möglich. Da aber alle Repository-Methoden eine Liste zurückgegeben haben, wurde versucht, diese anzupassen.
Nach Engpässen bei komplizierten Datenbankoperationen wurde die Funktion zum Erstellen einer PaginatedList neu geschrieben, um mit einer Liste zu funktionieren. Das Sortieren und Filtern wurde angepasst bei Seiten wie der Ticketliste, wurde dann auch angepasst, um mit Listen zu funktionieren.

ASP.NET Core

Refactoring

Problem: Es treten Fehler beim Refactoring von Viewmodels in cshtml.g.cs Dateien auf. Dies geschieht, da beim Refactoring die Using-Direktive auf das Viewmodel nicht angepasst wird.

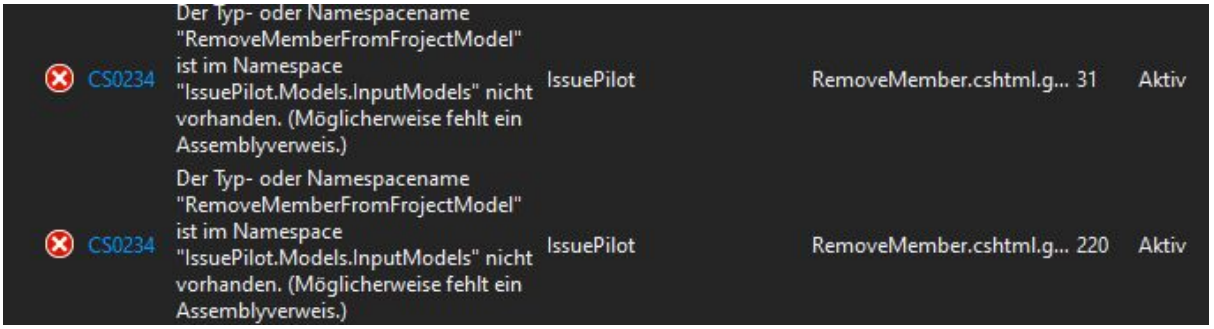

Abbildung 5: Refactoring Fehlermeldungen
Lösung: Die Using-Direktive muss manuell in den cshtml-Dateien angepasst werden.

ASP.Net Core Identity

Test mit SQL-InMemory

Problem: Ursprünglich war geplant die Tests mit einem relationalen Provider (SQL) und einem nicht-relationalen Provider (InMemory DB von EF Core) zu testen.[4] Dies war jedoch nicht möglich, da aufgrund von internen Abhängigkeiten von ASP.NET Core der Context nicht generiert werden konnte. Dieses Problem konnte nach 20 Stunden nicht gelöst werden. Daher wurden die Tests nur mit EF Core InMemory getestet.

Agiles Vorgehensmodell

Dies ist ein selbstdefiniertes agiles Vorgehensmodell, welches sich aus verschiedenen Modellen zusammensetzt. (z. B. EP und SCRUM)
Bei Abweichungen vom Modell sollen diese mit Begründung dokumentiert werden.

Sprints

- Features werden nach Priorität eingebaut.
- Sprintlänge variiert von Feature zu Feature (bis 4 Wochen).
- Ein Sprint soll ein fehlerfreies System hervorbringen. (Continuous Integration)
- Geplante Features, an denen noch nicht gearbeitet wurde, können in Sprints durch neue Features ausgetauscht werden.
- Sprints sollten schriftlich dokumentiert werden. (Protokolle)
- Sprints sollen mit dem Board-System von GitLab umgesetzt werden.

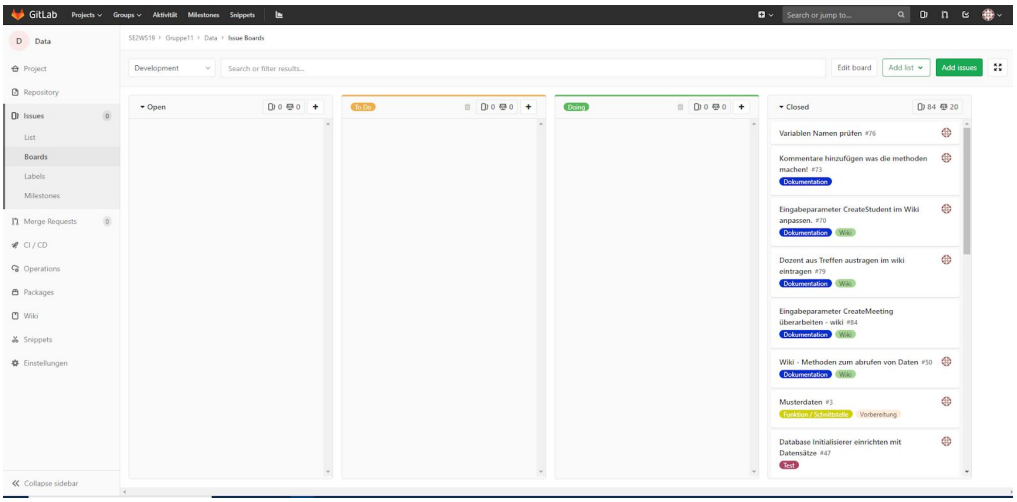

Abbildung 6: Board-System von GitLab

Review-Runde

- Während Sprints sind Veränderungen nach einer Review-Runde erlaubt.
- Dienstag und Donnerstag werden in Review-Runden Produktziele festgelegt, Feedback für Anpassungen besprochen und Features priorisiert.
- In Review-Runden wird sich in einer Phase abwechselnd in den Kunden und den Manager versetzt und Features priorisiert bestehend der entwickelten Software.

Feedback-Runde

- Am Anfang jedes Arbeitstages wird eine Feedback-Runde gehalten, um Änderungen an Sprints zu ermöglichen.
- Feedback und Tracking des Fortschrittes wird notiert für die Review-Runde
- Änderungen an der Software, die nicht die Funktionen und Produktziele der Webanwendung ändern, können auch hier entschieden werden.

Arbeitstechniken

- Pair Programming wird genutzt, um die Softwarequalität zu steigern und das Arbeitsrisiko zu verringern.
- Coding-Standards festlegen.
- Continuous Integration / test-driven development
- You ain’t gonna need it. (YAGNI)
- Don’t repeat yourself. (DRY)
- Refactoring (funktionierenden Code aufbessern, um nach dem Sprint einen fehlerfreien Mini-Release zu ermöglichen.)
- Arbeitszeit pro Woche auf vorläufig 15h begrenzen. Überstunden sind zu vermeiden.

Rollen

Kunde

Der Kunde wird als Teammitglied behandelt, welcher regelmäßig an den Treffen teilnimmt. Dies geschieht, um ggf. Anforderungen zu stellen und Vorgaben für die Priorisierung der umzusetzenden Bereiche zu machen. Dabei wird den Entwicklern überlassen, wie sie die Anforderungen erreichen.

Entwickler

Jeder der aktiv am Projekt arbeitet (Programmierer, ...), wird als Entwickler bezeichnet. Neben der Entwicklung des vorgegebenen Projektes soll der Entwickler auch auf die neuen Wünsche / Ansprüchen des Kunden reagieren. Das Team von Entwickler legt ihren Zeitplan für Planung, Umsetzung usw. eigenständig fest.

Manager

Der Manager ist der Vermittler zwischen den Entwicklern und den Kunden. Dies tut er, indem er unter anderem die Treffen moderiert und achtet darauf, dass zuvor festgelegte Regelungen (z.B. vorgesehener Zeitaufwand für das Projekt) eingehalten werden. [5]

Erfahrungsberichte

Niklas

Die Einteilung in Sprints war in dem Projekt grundsätzlich sinnvoll. Durch das wöchentliche Anpassen und Reflektieren konnten der Fortschritt gut beobachtet und Anpassungen umgesetzt werden. Gerade das Festhalten der genauen, einzelnen Aufgaben mit Hilfe des Lasten- und Pflichtenheftes erwies sich als hilfreich, um im Nachhinein nichts zu übersehen. Somit waren die Feedback-Runden essenziell für das Projekt.

Das Reflektieren über die Fortschritte aus unterschiedlichen Perspektiven war für die Anpassung meist schwer umsetzbar. Oft kam hier wenig Hilfreiches zustande. Ohne die in der Konzeption des Designs festgelegten Zielgruppe und Werte wären die Review-Runden fast gar nicht hilfreich gewesen, da man sonst keine klaren Gedanken hätte, was für Ziele die Rollen haben.

Im späteren Verlauf des Projektes und der dadurch hin zu stoßenden Last des Semesters kam mehr Unregelmäßigkeit in die Sprints zustande, auf die wir schwer mit unseren festgelegten Regeln reagieren konnten. Hier wäre eine Anpassung der Sprints auf kleine, intensive Zeiten und darauf folgende Pausen hilfreich gewesen, denn es entstand Leerlauf bei den Feedback- und Review-Runden.

Tim

Das Konzept des von uns verwendeten agilen Vorgehensmodells ist für die Entwicklung von Programmen, wo die Rollen von unterschiedlichen Personen tatsächlich bekleidet werden, sinnvoll.

Für unser Projekt ist es jedoch relativ schwierig, dies umzusetzen, auch wenn wir uns in die jeweiligen Rollen hineinversetzen. Durch unsere Position als Entwickler besitzt unsere Meinung und Ansicht schon voreingenommen sind, was die Rolle als Kunden deutlich erschwert.

Es war jedoch ein deutlicher Vorteil, dass Konzept für unsere Umsetzung zu verwenden, da dadurch eine durchgehende Kommunikation durch die nahezu täglichen Feedback-Runden vorhanden war. Die Review-Runden waren ebenfalls hilfreich, wenn es um Änderungen/ Änderungswünsche ging. Diese konnten zeitnah und auch mehreren Perspektiven auf Sinnhaftigkeit geprüft werden.

Durch die Review-Runden war es interessant und lehrreich zu sehen, wie man ein Anliegen z. B. aus der Sicht des Kunden wahrnimmt und behandeln würde.

Als Nachteil ist jedoch aufgefallen, dass nach einiger Zeit die Review-Runden mit 2-mal pro Woche zu viel sind. Dies kommt davon, dass bei der Umsetzung in Quellcode weniger zu besprechen war und andere Fächer im Studium mehr Zeit beanspruchten. Daher haben wir diese auf 1-mal wöchentlich (Donnerstag) gekürzt.

Quellen

1. TekTurorialsHub: >>Passing data from Controller to View in ASP.NET Core MVC<<
   URL: https://www.tektutorialshub.com/asp-net-core/asp-net-core-passing-data-from-controller-to-view/ [Stand: 10.03.2021]
2. Rick Anderson / olprod (2020): >>Einführung in Identity ASP.net Core<<
   URL: https://docs.microsoft.com/de-de/aspnet/core/security/authentication/identity?view=aspnetcore-5.0&tabs=visual-studio [Stand: 10.03.2021]
3. Arthur Vickers/ olprod (2020): >>Entity Framework Core<<
   URL: https://docs.microsoft.com/de-de/ef/core/ [Stand: 10.03.2021]
4. Jernej Kavka (2018): >>Making unit tests simple again with .Net Core and EF Core | Jernej Kavka (JK) at DDD Sydney 2018<<
   URL: https://www.youtube.com/watch?v=6nYefHkKby8 [Stand: 11.03.2021]
5. 1&1 IONOS SE (2020): >>Extreme Programming: Softwareentwicklung extrem gedacht<<
   URL: https://www.ionos.de/digitalguide/websites/web-entwicklung/extreme-programming/ [Stand: 10.01.2021]
6. Rick Anderson / olprod (2016): >>Rollenbasierte Autorisierung in ASP.net Core<<
   URL: https://docs.microsoft.com/de-de/aspnet/core/security/authorization/roles?view=aspnetcore-5.0 [Stand: 25.01.2021]
7. Adam Shaw: >>Documentation<<
   URL: https://fullcalendar.io/docs [Stand: 02.03.2021]
8. Jukka Kurkela (2020): >>Simple HTML5 Charts using the <canvas> tag<<
   URL: https://github.com/chartjs/Chart.js [Stand 01.03.2021]
9. olprod, Steve Smith (2020): >>Übersicht über ASP.NET Core MVC<<
   URL: https://docs.microsoft.com/de-de/aspnet/core/mvc/overview?view=aspnetcore-5.0 [Stand: 14.03.2021]
10. Rick Anderson / olprod (2019): >>Tutorial: Hinzufügen von Sortieren, Filtern und Paging: ASP.NET MVC mit EF Core<<
    URL: https://docs.microsoft.com/de-de/aspnet/core/data/ef-mvc/sort-filter-page?view=aspnetcore-3.1 [Stand: 14.03.2021]

Abbildungsverzeichnis