Programmier-Projekt „Thesis Datenbank“

Das im Folgenden beschriebene Programmierprojekt soll bis zum Ende der Vorlesungszeit in Kleingruppen bearbeitet werden. Dabei werden Sie eine Webanwendung entwickeln, um Lehrstühle der Universität bei der Verwaltung von Thesen und Thesisthemen zu unterstützten. Die dabei zu erstellende Webanwendung ist Grundlage für die abschließende mündliche Prüfung, in der Sie Ihr Projekt präsentieren.

Hintergrund & Zielsetzung

Am Lehrstuhl für Wirtschaftsinformatik werden Abschlussarbeiten (d.h., Bachelor- und Master-Thesen) in verschiedenen Studiengängen betreut. Dabei durchläuft jedes Thema einer Abschlussarbeit über Monate hinweg einen mehrstufigen Lebenszyklus, angefangen von der anfänglichen Themendefinition bis zur Erstellung des Gutachtens durch den Betreuer. Zur Unterstützung dieses Prozesses existiert bereits eine Webanwendung, welche auf einer relationalen Datenbank aufbaut.

Da sich die Anwendung bewährt hat, wollen weitere Lehrstühle die Webanwendung nutzen. Ihre Aufgabe wird es sein die bestehende Anwendung, welche die oben genannten Funktionen bereits erfüllt, entsprechend zu erweitern. Dazu bedarf es einiger Änderungen, um mehrere Lehrstühle abbilden und unterschiedliche Nutzer verwalten zu können. Die Hauptaspekte des Projekts sind hier kurz zusammengefasst:

- Ziel ist die Entwicklung einer gemeinsamen Thesisverwaltung für mehrere Lehrstühle.
- Dabei können Sie auf einer bestehende Webanwendung für einen Lehrstuhl aufbauen.
- Das Projekt soll auf die aktuelle ASP.NET-Core-Version portiert werden.
- Die Anwendung soll mandantenfähig werden, so dass mehrere Lehrstühle auf einer Instanz mit jeweils eigener Sicht auf die Daten arbeiten können (Nutzerverwaltung mit ASP Identity).
- Die bisherige Anwendung verwendet kommerzielle UI-Komponenten von Telerik, auf die in Zukunft verzichtet werden soll.
- Über die bisherige Funktionalität hinaus soll der Upload von Dateien (insb. PDF einer abgegebenen Thesis) möglich sein.

Zur Entwicklung eines Prototyps sollen .NET-Technologien eingesetzt werden. Verwenden Sie dazu die in der Lehrveranstaltung vorgestellten Konzepte von C#, ASP.NET MVC, HTML, CSS und JavaScript.

Das Projekt ist in mehrere Teile unterteilt, die im Folgenden genauer erläutert werden.

Teil 1: Bestehendes Projekt überführen

Machen Sie sich mit der bestehenden Anwendung vertraut (Projekt „Thesis MVC“ in WueCampus). Diese können Sie als Grundlage für ihr Projekt nutzen und viele bestehende MVC-Komponenten übernehmen. Dazu erstellen Sie möglichst ein neues **leeres** Projekt (die notwendigen Inhalte können nachträglich manuell aus dem bestehenden Projekt kopiert werden). Um einen langfristigen Support zu gewährleisten soll das Projekt von Version 2.1 auf Version 5.0 von ASP.NET Core überführt werden. Achten Sie darauf, die Authentifizierungskomponente ASP Identity nicht gleich zu Beginn hinzuzufügen, da sonst die Anpassung des User-Models deutlich schwieriger wird.

Teil2: Erweiterung der Models

Das bestehende Projekt verfügt bereits über folgende Model-Klassen:

- **Thesis** beinhaltet alle Angaben zur einzelnen Abschlussarbeit, d.h. Titel, Aufgabenstellung,

Anmeldedatum, Betreuer, Abgabedatum.

- **Supervisor** umfasst den Vor- und Nachnamen, die E-Mail-Adresse sowie den aktuellen Status des Mitarbeiters.
- **Programme** enthält Informationen (Name) zu den Studiengängen, z.B. „Master Business Management“, „Bachelor Wirtschaftsinformatik“ usw.

Zentrales Model der Anwendung ist die einzelne Abschlussarbeit („Thesis“), die durch Titel, Aufgabenstellung, Anmeldedatum usw. charakterisiert wird. Auch die Inhalte des Gutachtens werden innerhalb dieser Entität gespeichert.

Darüber hinaus gibt es zwei weitere Verknüpfungen, die der Einfachheit halber aber nicht als separate Models/Tabellen implementiert sind, sondern als Datenfelder mit einem Enumerationswert, da sich die entsprechenden Inhalte im Zeitverlauf nicht ändern:

- Der *Status* einer Thesis gibt deren Position im Lebenszyklus an:
  - Frei = Das Thesisthema ist bislang von niemandem belegt worden.
  - Reserviert = Für die Thesis wurde ein/e Student/in vorgemerkt, die Arbeit ist aber noch nicht angemeldet.
  - Angemeldet = Die Thesis wurde offiziell beim Prüfungsamt angemeldet und befindet sich in Bearbeitung.
  - Abgegeben = Die Thesis wurde abgegeben und muss begutachtet werden.
  - Bewertet = Das Gutachten für die Thesis wurde erstellt.
- Der *Typ* einer Arbeit gibt an, für welche Art von Studiengang das Thesisthema geeignet ist:
  - Bachelor
  - Master

Nun zu ihrer Aufgabe: Sie können alle bestehenden Models übernehmen bis auf das Supervisor-Model. Letzteres soll durch Identity ersetzt werden, damit sich die einzelnen Nutzer (= Betreuer von Thesisthemen) mit ihrem eigenen Account anmelden können (dazu mehr im Teil zu Identity). Zunächst müssen Sie lediglich alle Verknüpfungen der bestehenden Models zu der Klasse Supervisor entfernen.

Fügen Sie zum Schluss noch in *OnModelCreating* einige Zeilen ein, um die DB mit Testdaten zu befüllen. Erstellen Sie anschließend eine Datenbank und legen darin mit den Befehlen "add-migration" und "update-database" das Datenmodell an. Prüfen Sie, ob Ihre Testdaten auch tatsächlich in den Tabellen gespeichert wurden. Gegeben falls müssen Sie über den NuGet-Paket-Manager die Pakete Microsoft.Entity.FrameworkCore (v5.0.0) und Microsoft.EntityFrameworkCore.Tools (zum Migrieren) installieren, um die Datenbank wie in den Vorlesungsvideos gezeigt zu initialisieren.

Teil 3: Identity

Im bisherigen Projekt gab es nur einen Nutzeraccount mit dem sich alle Lehrstuhlmitarbeiter eingeloggt haben (die Mitarbeiter wurden über die Modelklasse Supervisor verwaltet). Ihre Aufgabe ist es, eine Benutzeroberfläche zu erstellen, um Nutzer anzulegen und ihnen Rollen zuzuweisen. Diese sollen sich dann mit einem zugeteilten Passwort und Nutzernamen anmelden können (aufgrund von Sicherheitsaspekten wird hier auf eine Implementierung der Registrierung verzichtet). Fügen Sie dazu wie in den Videos beschrieben Identity (neue Klasse „AppUser“, welche Identity erbt) hinzu und achten sie darauf die bereits bestehende Datenbank(-Connection) zu nutzen (https://docs.microsoft.com/de-de/aspnet/core/security/authentication/scaffold-identity?view=aspnetcore-5.0&tabs=visual-studio#scaffold-no-locidentity-into-an-mvc-project-without-existing-authorization).

Um mehrere Lehrstühle abbilden zu können muss das Datenmodell um die Model-Klasse Lehrstuhl (BWL 10, BWL 11 und BWL 12, ähnlich wie die Model-Klasse „Programme“) erweitert werden. Die jeweiligen AppUser müssen mit den Lehrstühlen verknüpft werden, modellieren Sie dafür die notwendigen Entitätsbeziehungen. Zudem soll der User mit dem Thesis-Model verknüpft werden, sodass der eingeloggte User automatisch der von ihm angelegten Thesis zugeordnet wird (VL: 8.1. Integration mit ASP Identity).

Der Administrator soll neue User über Identity hinzuzufügen und bearbeiten können. Der folgende Link veranschaulicht eine beispielhafte Umsetzung und soll als Hilfestellung dienen: https://www.yogihosting.com/aspnet-core-identity-create-read-update-delete-users/#create-users

Achten Sie hierbei darauf, keine zusätzliche User-Klasse anzulegen, sondern die benötigten Datenfelder in die AppUser-Klasse zu integrieren (anders als im Link gezeigt!). Beim Anlegen der User, soll mit Hilfe von Drop-Down dem User ein Lehrstuhl zugeordnet werden können.

Zusätzlich dazu müssen Sie die Benutzeroberfläche noch dahingegen absichern, dass nur bestimmte Rollen (Admin) Nutzer anlegen dürfen. Um den angelegten Nutzern Rollen zuweisen zu können, benötigen Sie eine weitere Benutzeroberfläche. Sie können sich beim Erstellen an folgendem Link orientieren:

https://www.codewithmukesh.com/blog/user-management-in-aspnet-core-mvc/

Ab der Überschrift **„Add / View Available Roles“** wird der relevante Code gezeigt, um neue Rollen anzulegen und um eine Verknüpfung zum User herzustellen. Für unsere einfache Anwendung genügen zwei Rollen: Admin und User.

Sichern Sie abschließend alle Funktionen zum Erstellen und Bearbeiten von Usern sowie Rollen ab, sodass diese Funktionen nur für die Admin-Rolle sichtbar und verfügbar sind.

Teil 3: Thesisverwaltung

Nachdem die Modelklassen und die Datenbank stehen, müssen Sie in einem zweiten Schritt eine Benutzeroberfläche erstellen, die den Lebenszyklus einer Thesis von der anfänglichen Themendefinition bis zur Begutachtung und abschließenden Archivierung abbildet. Zu diesem Zweck erzeugen Sie Controller bzw. Views per Scaffolding, die anschließend angepasst werden. Dazu können Sie sich an den bestehenden Thesis Controllern und Views orientieren.

Ändern Sie die Ansicht der Themenliste ab, sodass sie sortierbar wird. Es sollten in der Liste zudem nur relevante Felder angezeigt werden (Titel, Status, Student, Anmeldung, Abgabe, Typ, **Betreuer (d.h. AppUser), Lehrstuhl** und die relevanten Bearbeitungsbuttons). Achten Sie dabei auf eine anschauliche Darstellung der Tabelle.

Um die Thesisdaten in den verschiedenen Lebenszyklen zu verwalten, ist eine Modifikation der per Scaffolding erzeugten Thesis-View Edit notwendig. Hierbei können Sie sich stark an den bestehenden Ansichten und Controllern orientieren. Legen Sie dafür die folgenden Bereiche in der GUI an:

- Thema: Titel, Description, Supervisor, Bachelor, Master
- Bearbeitung: Status, StudentName, StudentId, StudentEmail, Type, ProgrammeID, Registration, Filing.
- Bewertung: alle restlichen Felder Summary, …, RichenessWt, die Gewichtungen sollten direkt neben den dazugehörigen Werten stehen (2 Spalten Tabelle).
- Gesamtnote besteht lediglich aus 2 Feldern:
    - Notenvorschlag: Berechnung eines Notenvorschlags. Dazu soll eine Funktion

implementiert werden, die auf Basis der Einzelbewertungen einen Notenvorschlag berechnet (Code dafür ist in dem alten Projekt bereits vorhanden). Die Einzelbewertungen bewegen sich jeweils auf einer Skala von 1-5 (z.B. realisiert als Dropdown List oder mehrere Radio Buttons). Zusätzlich ist jede Teilbewertung mit einer Gewichtung zwischen 0 und 100 versehen. Bauen Sie in die View einen zusätzlichen Button und ein read-only-Textfeld ein. Wird der Button geklickt, berechnet eine Javascript-Funktion auf dem Client einen Notenvorschlag als gewichteten Durchschnitt der Einzelbewertungen. Falls keine Berechnung möglich ist, wird eine Fehlermeldung angezeigt (über die Javascript-Anweisung „`alert`“). Der Notenvorschlag kann auch krumme Werte ergeben (z.B. „1,87“), soll aber auf zwei Nachkommastellen gerundet werden. Stellen Sie im Theses-Controller sicher, dass die Summe der Gewichtungsfaktoren bei einer Speicherung der Bewertung immer 100% ergibt, sonst soll eine Warnung ausgegeben werden ("Alle Gewichtungsfaktoren müssen angegeben werden und in Summe 100% ergeben.").

- Note/Grade: „Grade“ soll als Dropdownliste mit fest vorgegebenen Notenwerten (1,0, 1,3, 1,7, 2,0 ..., 5,0) und ist unabhängig von dem Feld Notenvorschlag.

Textfelder, deren Inhalte längere Texte umfassen (z. B. Description, Summary …), sollten mit mehreren Zeilen angezeigt werden. Da in den Views auch Fließkommazahlen und Datumsangaben vorkommen, muss die Client- Validierung um eine deutsche Lokalisierung erweitert werden. Sorgen Sie außerdem dafür, dass die Eingabe des Abgabedatums für die Hausaufgabe durch ein Web-Steuerelement (z.B. Kendo UI Datepicker, Gijgo Datepicker) vereinfacht wird.

Nachdem die Bewertung eingerichtet wurde, fehlt noch eine Funktion um daraus ein Gutachten zu erstellen. Nutzen Sie dazu Details-View des Model Thesis und nehmen sie entsprechende Änderungen vor.

Löschen Sie alle Felder aus den Views, die dort nicht benötigt werden. Löschen Sie auch jeweils aus *Create* und *Edit* das Feld *LastModified*. Stattdessen soll dieses Feld jeweils im Controller direkt vor `SaveChanges()` mit dem aktuellen Zeitstempel (`DateTime.Now`) neu gesetzt werden.

Bis zu diesem Punkt sollte alles nur für die registrierten Nutzer bearbeitbar/sichtbar sein, d.h., durch die Benutzerauthentifizierung geschützt sein.

Sorgen sie dafür, dass auch nicht angemeldete Nutzer eine Liste der freien bzw. reservierten Themen einsehen können. Legen Sie dazu eine neue View (Public) an, die jeweils nicht noch nicht bearbeiteten Einträge in der Thesis-Themenliste eines Lehrstuhls anzeigt. Modifizieren Sie den Thesis-Controller entsprechend, sodass die View nicht alle freien Thesisthemen anzeigt, sondern nach einem Lehrstuhl filtert. Die Liste soll den Titel (in Klammern ThesisType) anzeigen und die restlichen Informationen in einem Dropdown (mit Klick auf den Titel) anzeigen: Beschreibung, Status und Betreuer. “). Die E-Mail-Adresse des Betreuers (*ApplicationUser_EMail*) soll nicht als eigene Spalte angezeigt, sondern als Link hinter den Namen gelegt werden, sodass man per Klick auf den Namen eine Mail an den Betreuer senden kann (d.h. ein Link im Format „mailto …“). Die Public-Views basieren nicht auf dem Layout-Template der Anwendung, d.h., es werden keine Navigationsleiste, kein Login-Link usw. angezeigt.

Teil 4: Abschließendes Finetuning

- Die Liste aller Themen (Index-View Thesis) soll filterbar und sortierbar werden. Bisher wird

dies mit der Grid-Komponente von Telerik realisiert, die durch eine Tabelle im Eigenbau ersetzt werden soll. Bauen Sie die Index-View zu diesem Zweck, wie in einem der Videos gezeigt um. Die Überblickstabellen auf den Index-Seiten sind aufgrund der kleinen Zahl der Einträge normalerweise eher übersichtlich. Die Liste der bewerteten Arbeiten hingegen wird im Zeitverlauf so viele Einträge aufweisen, dass sie nicht mehr auf einer einzelnen Seite angezeigt werden kann/sollte. Nutzen Sie daher einen Pager, um die Tabelle in Seiten von je 10 Einträgen zu zerlegen. Dazu können Sie den bestehenden Code nutzen.

- Die Detail-View der bewerteten Thesen soll nicht nur als Webseite angezeigt werden, sondern einen Button enthalten um eine PDF-Datei zu erzeugen. Integrieren Sie dazu z.B. das NuGet-Paket „Rotativa PDF“ (oder eine Alternative) in Ihr Projekt und passen die *Detail*-Methode des *Thesis*-Controllers entsprechend an. Implementieren Sie einen Actionlink in der Detail-View, der auf eine Methode zur Generierung des PDF verweist. Genauere Informationen zu Rotativa finden Sie auf der Website des Projekts: https://github.com/webgio/Rotativa. Zusätzlich sollte sich noch ein Button in der Index-View des Thesis Models befinden um eine PDF des Gutachtens zu erzeugen. Auch hierzu können Sie sich am bestehenden Code orientieren.
- Sichern Sie alle Controller bis auf *Home* und *Public* gegen unautorisierte Zugriffe ab. Ein Zugriff ist nur für eingeloggte Benutzer erlaubt.
- Sorgen Sie zudem dafür, dass der Nutzer nach dem Login als Erstes seine eigenen Einträge in die Thementabelle sieht (mit den noch offenen oder in Bearbeitung befindlichen Thesen ganz zu Beginn).
- Zuletzt fügen Sie noch ein Zusatzfeature ein, welches es ermöglicht die Arbeit eines Studenten zum jeweiligen Thema während des Bewertungsprozesses in die Datenbank zu laden. Die Upload-Funktion geht über die vermittelten Kenntnisse des Kurses hinaus und Sie können hier ihre Transferfähigkeiten unter Beweis stellen. Der folgende Link: https://docs.microsoft.com/de-de/aspnet/core/mvc/models/file-uploads?view=aspnetcore-2.1, ist eine erste Hilfestellung zur Implementierung der Upload-Funktion. Es geht darum ein **einfaches** File-Input-Feld in die View einzubauen (d.h. Upload mehrerer Dateien ist nicht notwendig) und das Model sowie den Controller entsprechend anzupassen, sodass die Datei in der Datenbank selbst in einem zusätzlichen Datenfeld gespeichert wird.

Prof. Thiesse und Alexander Dürr wünschen Ihnen viel Erfolg mit der Programmieraufgabe.