PRAXISARBEIT

Davud Ponjevic

Inhaltsverzeichnis

Informieren

Auftrag & Ausgangslage

Jetstream-Service, ein KMU im Skiservice, strebt die Digitalisierung ihrer internen Auftragsverwaltung an. Während der Wintersaison sollen bis zu 10 Mitarbeiter über eine webbasierte Anwendung auf Ski-Serviceaufträge zugreifen können. Das Backend-Projekt beinhaltet die Erstellung eines sicheren Web-APIs mit Authentifikation, Datenbankdesign und Implementierung (Code First oder Database First), die Entwicklung eines Testprojekts (Unit-Test) und die Realisierung der Anwendung gemäß den Anforderungen. Dabei wird bei datenlesenden Operationen keine Authentifikation benötigt, während Änderungen an Auftragsdaten nur von authentifizierten Mitarbeitern vorgenommen werden können.

Anforderungen an das Projekt

Nr.	Beschreibung
A1	Login Dialog mit Passwort für den autorisierten Zugang der Mitarbeiter (Datenänderungen).
A2	In der Datenbank müssen die Informationen des Serviceauftrags und die Login Daten der Mitarbeiter verwaltet sein.
A3	Erfasste Serviceaufträge abrufbar sein.
A4	Die erfassten Serviceaufträge müssen selektiv nach Priorität abrufbar sein.
A5	Mitarbeiter können eine Statusänderung eines Auftrages vornehmen.
A6	Mitarbeiter können Aufträge löschen (z.B. bei Stornierungen)
A7	Die aufgerufenen API-Endpoints müssen zwecks Fehlerlokalisierung protokolliert sein (DB oder Protokolldatei).
A8	Datenbankstruktur muss normalisiert in der 3.NF sein inkl. referenzieller Integrität
A9	Für die Web-API Applikation muss ein eigener Datenbankbenutzerzugang mit eingeschränkter Berechtigung (DML) zur Verfügung gestellt werden (Benutzer root bzw. sa ist verboten).
A10	Das Web-API muss vollständig nach Open-API (Swagger) dokumentiert sein.
A11	Das Softwareprojekt ist über ein Git-Repository zu verwalten.
A12	Ganzes Projektmanagement muss nach IPERKA dokumentiert sein

Planen

Zeitplan

Aufgabe	Zeit
Informieren	1 std
Planen	4 std
Entscheiden	2 std
Realisieren	40 std
Kontrollieren	4 std
Auswerten	4 std
Summe	55 std

Was wird in den schritten gemacht

Informieren

In diesem Schritt setzte ich mich mit den Anforderungen auseinander. Ich las die Ausgangslage und notierte grob, was ich tun musste.

Planen

Hier erstelle ich den Zeitplan und erstelle das Projekt. Hier spreche ich mit den Mitschülern auch darüber, wie man etwas lösen kann, zum Beispiel, und was man etwa machen muss. Ich fange hier mit den Gedanken an, welche NuGet-Pakete ich alles brauche, und installiere diese dann.

Entscheiden

Hier werde ich mich entscheiden, wie ich etwa vorgehen kann. Hier werde ich die Klassenordner erstellen und werde entscheiden, welches Prinzip ich wähle (Code First oder Database First).

Realisieren

In diesem Schritt werde ich anfangen zu programmieren. Als erstes werde ich hier programmieren, dass ich die Daten vom Frontend an das Backend senden kann, und ich werde diese Daten dann in der Datenbank sehen. Der nächste Schritt wird sein, die Benutzer zu erstellen, das heißt, alle Benutzer anzulegen, mit denen ich mich dann auch einloggen kann. Anschließend werde ich die Klassen in der Datenbank so strukturieren, dass sie in der 3. Normalform sind.

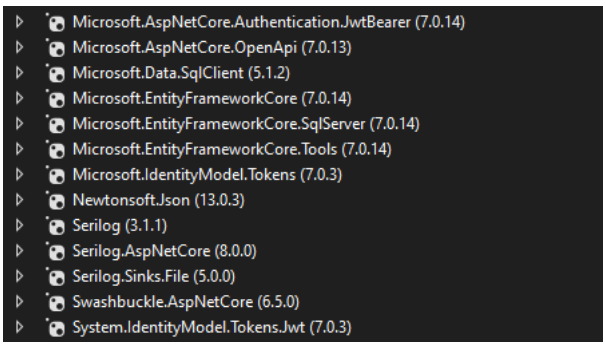
Kontrollieren

Hier werde ich den Code mit den Anforderungen überprüfen und werde, wenn nötig, das Programm dann umschreiben.

Auswerten

Hier präsentiere ich das Projekt und gebe das Projekt am besprochenen Datum ab mit dem GitHub link.

NuGet Packete welche ich installiert habe



- Microsoft.AspNetCore.AuthenticationJwtBearer (7.0.14):
  Authentifizierung mit JSON Web Tokens in ASP.NET Core.

- Microsoft.AspNetCore.OpenApi (7.0.13):
  Integration von Swagger/OpenAPI für API-Dokumentation in ASP.NET Core.

- Microsoft.DataSqlClient (5.12):
  Datenzugriffsbibliothek für SQL-Server und andere Datenquellen.

- Microsoft.EntityFrameworkCore (7.0.14):
  Object-Relational Mapping (ORM)-Framework für Datenbankzugriff.

- Microsoft.EntityFrameworkCoreSqlServer (7.0.14):
  SQL-Server-spezifische Erweiterungen für Entity Framework Core.

- Microsoft.EntityFrameworkCore.Tools (7.0.14):
  Tools für die Entwicklung mit Entity Framework Core, einschließlich Migrations.

- Microsoft.IdentityModel.Tokens (7.0.3):
  Klassen für die Arbeit mit Sicherheitstoken, einschließlich JWTs.

- NewtonsoftJson (13.0.3):
  Bibliothek für die Verarbeitung von JSON-Daten in .NET.

- Serilog (3.1.1):
  Logging-Framework für .NET-Anwendungen.

- Serilog.AspNetCore (8.0.0):
  Integration von Serilog für ASP.NET Core-Logging.

- SerilogSinks.File (5.0.0):
  Serilog-Sink für das Logging von Ereignissen in Dateien.

- Swashbuckle.AspNetCore (6.5.0):
  Integration von Swagger/Open API in ASP.NET Core für API-Dokumentation.

- System.IdentityModel.TokensJwt (7.0.3):
  Klassen für die Arbeit mit JWTs im .NET-Framework.

Entscheiden

Was ich hier genau mache/entscheide?

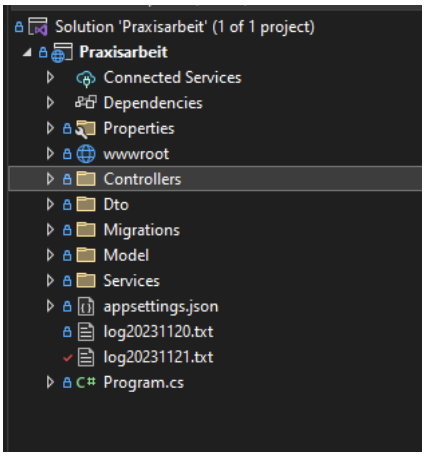
In diesem wichtigen Schritt denke ich darüber nach, wie ich mein Vorgehen planen will. Hier erstelle ich die Ordner für die Klassen und überlege, ob ich Code First oder Database First nutzen möchte.

Ich fokussiere mich zunächst darauf, die Klassenordner zu erstellen, um mein Projekt gut zu organisieren. Die große Entscheidung betrifft die Herangehensweise an die Datenbankentwicklung. Beim Code First-Ansatz definiere ich die Datenbankstruktur direkt im Code und erstelle dann die Datenbank. Beim Database First-Ansatz erstelle ich zuerst die Datenbank und generiere dann den Code basierend auf der vorhandenen Struktur.

Diese Entscheidung beeinflusst, wie ich meine Anwendung entwickle und wie flexibel sie für zukünftige Änderungen ist. Daher überlege ich genau, welche Methode am besten zu den Anforderungen des Projekts, der Skalierbarkeit und den Vorlieben meines Entwicklungsteams passt.

Zusammengefasst lege ich in diesem entscheidenden Schritt die Grundlagen für mein Projekt. Ich strukturiere die Klassenordner und wähle das grundlegende Prinzip für die Datenbankentwicklung aus. Diese Entscheidungen sind entscheidend für den weiteren Verlauf meiner Entwicklungsarbeit.

Die Ordner



Was in den Ordner drinnen ist

- Properties
  Build-Einstellungen, Konfigurationsdateien

- wwwroot:
  Frontend

- Controllers:
  HTTP-Anfragenverarbeitung, Anwendungsflusssteuerung

- Dto
  Datenübertragung, Strukturierte Datenmodelle

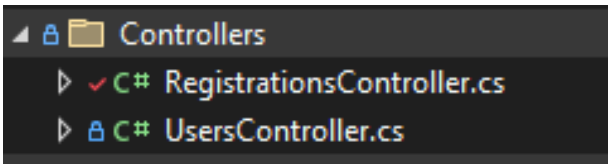
- Migrations:
  Datenbankschemaaktualisierung, Migrationsdateien

- Model:
  Datenbankentitäten, Geschäftslogikmodelle

- Services:
  Geschäftslogikdienste, Datenzugriffsdienste

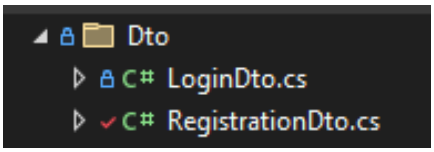
Realisieren

In diesem schritt habe ich angefangen mit dem Programmieren ich erkläre kurz was in jedem Ordner programmiert wurde



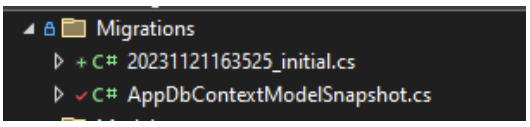
- RegistrationController
  einfache Datenbankoperationen für Registrierungsdaten ermöglicht, wie das Abrufen, Erstellen, Aktualisieren und Löschen.

- UsersController
  Der Code implementiert einen ASP.NET Core Web API-Controller namens UsersController für Benutzerlogins. Bei erfolgreicher Authentifizierung wird ein Token mit dem Benutzernamen zurückgegeben, andernfalls erfolgt eine Fehlermeldung.

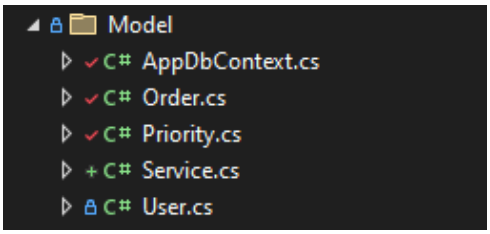


- LoginDto
  Der Code enthält die DTO-Klasse LoginDto im Namespace Praxisarbeit.Dto. Diese Klasse hat Eigenschaften für Benutzernamen (UserName) und Passwort (Password) und wird verwendet, um Anmeldedaten zwischen verschiedenen Teilen der Anwendung zu übertragen

- RegistrationDto
  Der Code enthält die DTO-Klasse RegistrationDto im Namespace Praxisarbeit.Dto. Diese Klasse dient dazu, strukturierte Informationen für Registrierungsvorgänge zu übermitteln, darunter Name, E-Mail, Telefonnummer und relevante IDs sowie Datumsangaben.

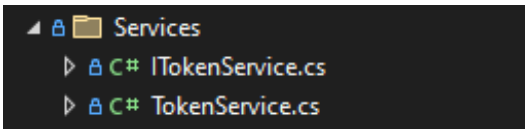


Der "Migrations"-Ordner speichert in ASP.NET Core mit Entity Framework Core die Code-Dateien für Schritte zur Aktualisierung oder Erstellung des Datenbankschemas. Es erleichtert die Verwaltung der Datenbankstruktur während der Entwicklung.



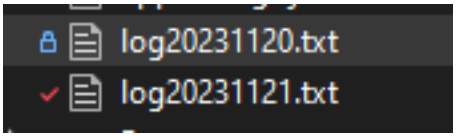
- AppDbContext.cs
  Der Code definiert den Datenbankkontext AppDbContext, erbt von DbContext, und enthält DbSet-Eigenschaften für "Registrations" (Aufträge), "Priorities" (Prioritäten) und "Users" (Benutzer). In der Methode OnModelCreating werden Daten für Benutzer und Prioritäten hinzugefügt.

- Die weiteren Klassen
  Die Codes definieren Modellklassen für eine Anwendung. Die Order-Klasse repräsentiert Bestellungen mit Beziehungen zu Benutzern, Prioritäten und Services. Die Priority-Klasse enthält Typ und Zeit zur Fertigstellung für Prioritäten, die Service-Klasse beschreibt verschiedene Services, und die User-Klasse speichert Benutzerinformationen. Alle Klassen verwenden Datenannotations für Datenbank- und Validierungsanforderungen.



- ITokenService
  Der Code definiert ein Interface namens ITokenService im Namespace Praxisarbeit.Services. Dieses Interface enthält eine Methode CreateToken, die für die Erstellung von Tokens für Benutzernamen verantwortlich ist.

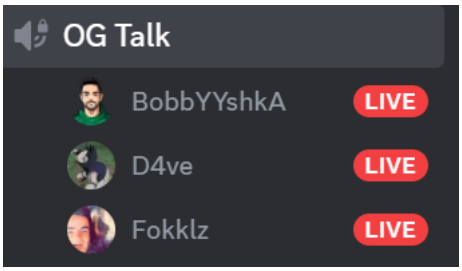
- TokenService
  Der Code erstellt in der Klasse TokenService ein JWT-Token für einen Benutzernamen. Der Token-Schlüssel stammt aus der Konfigurationsdatei, Claims für den Benutzernamen werden erstellt, und ein SecurityTokenDescriptor wird konfiguriert. Das JWT-Token wird anschließend erstellt und zurückgegeben.



- Log dateien
  In den Log-Dateien deines Projekts können Informationen wie Fehlermeldungen, Warnungen, allgemeine Anwendungsabläufe, Debugging-Details, Sicherheitsaudits, Leistungsdaten und Benutzeraktionen protokolliert sein. Die genaue Art der protokollierten Informationen hängt von der Logging-Konfiguration und Implementierung in deiner Anwendung ab.

Kontrollieren

Ich und mein Mitschüler haben uns zusammen den Code angeguckt. Wir haben Sachen repariert, die nicht funktioniert haben, und versucht, den Code besser zu machen. Wir haben darauf geachtet, dass der Code gut lesbar ist und dass er effizient arbeitet. Wir haben nicht nur Fehler behoben, sondern versucht, den Code insgesamt zu verbessern.



Ich habe mit diesen Mitschülern drüber geschaut

Auswerten

Fazit

Insgesamt bin ich stolz darauf, dass ich an meiner Praxisarbeit gearbeitet habe und in der Lage war, Probleme zu identifizieren und Lösungen zu finden. Es macht mich zufrieden, dass ich viele Schwierigkeiten allein bewältigen konnte. Auch wenn meine Kenntnisse noch begrenzt sind, denke ich, dass ich durch Zusammenarbeit und eigenständiges Lernen in der Lage war, etwas Bedeutendes zu schaffen. Es ermutigt mich, weiterhin meine Fähigkeiten zu verbessern und mich neuen Herausforderungen zu stellen.

Lessons learned

Aus dieser Praxisarbeit habe ich viel gelernt, insbesondere im Umgang mit einer API und der Verbindung einer Datenbank mit einem Programm. Die Erfahrung, diese Technologien in der Praxis anzuwenden, hat mir wertvolle Einblicke verschafft. Ich hoffe, weitere Projekte in ähnlichem Umfang umsetzen zu können, um meine Fähigkeiten weiter zu vertiefen und mein Verständnis für die Entwicklung von Anwendungen zu erweitern.

Wichtig

Wenn man nach einer längeren zeit das Projekt wieder starten will muss man die NuGet Packete aktualisieren. Dann gibt es Befehle die in die Packet manager console kommen um die Verbindung zu gewährleisten:

Drop-Database = Drop-Database löscht die gesamte Datenbank und trennt alle aktiven Verbindungen falls eine besteht. Stelle sicher, dass du diesen Befehl nur dann ausführst, wenn du die Datenbank wirklich entfernen möchtest, da alle Daten unwiderruflich gelöscht werden.

Add-Migration InitialCreate = Der Befehl Add-Migration erstellt eine neue Migration mit dem Namen InitialCreate.

Update-Database = Der Befehl Update-Database führt die Migrationen auf die Datenbank aus und erstellt die Tabellen und Beziehungen. Bei einer neuen Datenbank wird die gesamte Struktur entsprechend der Migrationen aufgebaut.

Mit diesen 3 Befehlen sollte man das System zum laufen bringen können.