ipso! Bildung seit 1876

Modul 295 Lerndokumentation

Nevan Alberola

ipso! Bildungsmarken

HWS ipso! Business School · IBZ ipso! Executive Education · IFA ipso! Haus des Lernens · NSH ipso! International School

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangssituation

Jetstream-Service ist ein Unternehmen, das sich auf Skiservicearbeiten spezialisiert hat. Während der Wintersaison arbeitet das Team an einer Vielzahl von Aufträgen, die bisher manuell organisiert wurden. Um diese Prozesse zu modernisieren, sollte eine datenbankgestützte Lösung entwickelt werden, die es den Mitarbeitern ermöglicht, Serviceaufträge einfach zu verwalten. Ziel war es, ein System zu schaffen, das sowohl Userfreundlich als auch sicher ist.

1.2 Allgemeine Anforderungen

Jetstream-Service benötigt ein stabiles und sicheres Backend-System, das die folgenden Funktionen bietet:

- **Login-System:** Mitarbeiter können sich mit Usernamen und Passwort authentifizieren.
- **Serviceordersverwaltung:** Die Anwendung ermöglicht es, Aufträge anzuzeigen, zu bearbeiten, zu löschen und deren Status zu aktualisieren.
- **Priorisierung und Filterung:** Aufträge können nach Priorität gefiltert werden.
- **Datenbankintegration:** Alle Daten wearden in einer gut strukturierten, normalisierten Datenbank gespeichert, um Effizienz und Konsistenz zu gewährleisten.

1.3 Zielsetzung

Das Ziel des Projekts ist es, die täglichen Arbeitsabläufe der Firma zu erleichtern, indem:

- Die Verwaltung von Serviceaufträgen digitalisiert wird.
- Mitarbeiter einen sicheren Zugang zu den benötigten Informationen erhalten.
- Daten fehlerfrei und effizient verarbeitet werden.

2 Plannung

2.1 IPERKA-Methodologie

Für die Planung und Durchführung des Projekts wurde die IPERKA-Methodik verwendet. Es gibt zwar auch andere Modelle, wie z. B. Hermes, aber IPERKA ist schliesslich das in der Branche am weitesten verbreitete und zeichnet sich durch eine klare Struktur und einen systematischen Ansatz aus. Für die Dokumentation wie bei Zeitplanung, habe ich IPERKA benutzt. Die sieben Phasen - berichten, planen, entscheiden, beschliessen, realisieren, überwachen und bewerten - ermöglichten es, die einzelnen Aufgaben effektiv zu steuern und das Projekt gezielt voranzutreiben. Jede Phase wurde sorgfältig dokumentiert, um Transparenz und Nachvollziehbarkeit zu gewährleisten.

2.2 Projektorganisation

Das Projekt war klar strukturiert, um eine reibungslose Umsetzung zu gewährleisten. Die Verantwortlichkeiten wurden definiert und die Aufgaben priorisiert:

- **Initialisierung:** Analyse der Anforderungen und Erstellung eines Umsetzungsplans.
- **Umsetzung:** Entwicklung und Testen der Backend-Features.
- **Abschluss:** Dokumentation und Evaluierung der Ergebnisse.

2.3 Auswahl der Entwicklungswerkzeuge

Die Auswahl der Entwicklungswerkzeuge spielte eine entscheidende Rolle für die Effizienz des Projekts. Folgende Tools wurden verwendet:

- **Visual Studio Code:** Hauptentwicklungsumgebung für das Schreiben und Verwalten des Codes.
- **SQL Server:** Datenbanksystem für die Speicherung und Verwaltung von Serviceaufträgen und Userdaten.
- **Postman:** Tool für das Testen und Validieren der API-Endpunkte.
- **Swagger:** Dokumentationswerkzeug für die API, um eine übersichtliche und zugängliche Dokumentation zu gewährleisten.

2.4 Datenbankdesign (Code First Ansatz)

Für die Entwicklung der Datenbank wurde der Code First Ansatz von Entity Framework gewählt. Dieser Ansatz ermöglichte eine direkte Modellierung der Datenbank aus den Code-Klassen heraus. Die Datenbankstruktur wurde entsprechend der 3. Normalform gestaltet, um Redundanzen zu minimieren und die referenzielle Integrität zu gewährleisten. Die wichtigsten Tabellen umfassen:

- **User:** Speicherung der Anmeldedaten.
- **Serviceaufträge:** Verwaltung von Kundenaufträgen einschliesslich Priorität, Status und Dienstleistungen.

3 Entscheiden

3.1 Auswahl der Technologien und Frameworks

Die Auswahl der Technologien und Frameworks wurde sorgfältig getroffen, um die Anforderungen des Projekts effizient zu erfüllen. Die wichtigsten Entscheidungen umfassten:

- **ASP.NET Core:** Als Hauptframework für die Entwicklung der Web-API. Es bietet eine robuste Basis für skalierbare und sichere Anwendungen.
- **Entity Framework Core:** Zum Umgang mit der Datenbank, insbesondere für den Code First Ansatz. Es ermöglicht eine einfache Verwaltung der Datenmodelle und Migrationen.
- **MS SQL Server:** Als Datenbanksystem, um eine zuverlässige und leistungsstarke Speicherung der Daten zu gewährleisten.
- **Swagger/OpenAPI:** Für die API-Dokumentation, um Endpunkte klar zu definieren und die Integration durch andere Entwickler zu erleichtern.

3.2 Entscheidung für die Schichtenarchitektur

Das Backend wurde in einer klar strukturierten Schichtenarchitektur entwickelt, um Wartbarkeit und Erweiterbarkeit zu gewährleisten. Diese Architektur umfasst:

- **Controllerschicht:** Verantwortlich für die Verarbeitung von Anfragen und die Weiterleitung an die Logikschicht.
- **Logikschicht:** Enthält die Geschäftslogik, wie z. B. das Verwalten von Userdaten und Serviceaufträgen.
- **Datenzugriffsschicht:** Verwaltung der Datenbankoperationen mithilfe von Entity Framework Core.

Diese Aufteilung ermöglicht es, Änderungen in einer Schicht vorzunehmen, ohne die anderen zu beeinträchtigen.

3.3 Definition der API-Spezifikationen (OpenAPI)

Die API-Spezifikationen wurden mit Hilfe von Swagger definiert. Folgende Aspekte wurden berücksichtigt:

- **Endpoint-Dokumentation:** Jeder API-Endpunkt ist klar dokumentiert, einschliesslich der HTTP-Methoden, Parameter und erwarteten Antworten.
- **Fehlerbehandlung:** Es wurden standardisierte Fehlermeldungen definiert, um die Fehlerlokalisierung zu erleichtern.
- **Testbarkeit:** Die Integration von Swagger UI ermöglicht es, die API-Endpunkte direkt zu testen und zu validieren.

Swagger UI

4 Realisieren

4.1 Einrichtung des Entwicklungsumfelds

Die Erstellung des Projekts begann mit der Konfiguration des ASP.NET Core Web API-Projekts. Dies wurde über die .NET CLI ausgeführt:

1. **Projekt erstellen:**

```
dotnet new webapi -n SkiBackend
```

Dies erstellt ein neues Web-API-Projekt mit dem Namen SkiBackend.

2. **Abhängigkeiten hinzufügen:**
    - Entity Framework Core und das SQL Server-Paket:

```
dotnet add package Microsoft.EntityFrameworkCore
dotnet add package Microsoft.EntityFrameworkCore.SqlServer
```

    - Tools für Migrationen:

```
dotnet add package Microsoft.EntityFrameworkCore.Tools
```

3. **Projekt ausführen:**

```
dotnet run
```

Dadurch wird der lokale Server gestartet, und die API ist über https://localhost:5000 zugänglich.

4.2 Struktur des Projekts

Das Projekt wurde in einer klaren Schichtenarchitektur strukturiert:

- **Controllerschicht:** Verarbeitet HTTP-Anfragen und leitet diese an die Logik weiter.
- **Servicelogik:** Implementiert die Geschäftslogik.
- **Datenzugriffsschicht:** Übernimmt Datenbankoperationen mit Entity Framework Core.

Controller:

```
[HttpGet("{id}")]
public async Task<IActionResult> GetOrderById(int id)
{
    var order = await _context.ServiceOrder.FindAsync(id);
    return order == null ? NotFound() : Ok(order);
}
```

4.3 Authentifizierung und Autorisierung

Die Authentifizierung basiert auf einem einfachen Usernamen und Passwort-Modell. Die Controller-Funktionen für Userregistrierung und Login sichern sensible Bereiche.

Login:

```
[HttpPost("login")]
public async Task<IActionResult> Login([FromBody] LoginRequest request)
{
    var User = await _context.User.SingleOrDefaultAsync(u => u.Username ==
request.Username);
    return User == null || User.Passwort != request.Passwort
        ? Unauthorized("Invalid login data.") : Ok("Successfully
registered.");
}
```

4.4 CRUD-Operationen

CRUD-Operationen für Serviceaufträge umfassen das Erstellen, Lesen, Aktualisieren und Löschen.
Ein Order erstellen:

```
[HttpPost]
public async Task<IActionResult> CreateOrder(ServiceOrder order)
{
    _context.ServiceOrder.Add(order);
    await _context.SaveChangesAsync();
    return CreatedAtAction(nameof(GetOrderById), new { id = order.Id },
order);
}
```

4.5 Datenbankintegration

4.5.1 Datenbankverbindung und Konfiguration

Die Datenbankverbindung wurde in der Datei appsettings.json konfiguriert:

```
  "ConnectionStrings": {
    "MoviesDbConnectionString":
"Server=Nevan\\SQLEXPRESS;Database=EF;Trusted_Connection=True;MultipleActiveR
esultSets=true;TrustServerCertificate=True"
```

Die Verbindung wird in Program.cs initialisiert:

```
builder.Services.AddDbContext<DatabaseContext>(options =>
options.UseSqlServer(builder.Configuration.GetConnectionString
("MoviesDbConnectionString")));
```

4.5.2 Entity Framework und Migrationen

Migrationen wurden genutzt, um die Datenbank zu erstellen und zu aktualisieren:

1. **Migration erstellen:**

```
dotnet ef migrations add InitialCreate
```

2. **Migration anwenden:**

```
dotnet ef database update
```

3. **Migration:**

```
protected override void Up(MigrationBuilder migrationBuilder)
{
    migrationBuilder.CreateTable(
        name: "ServiceOrder",
        columns: table => new
        {
            Id = table.Column<int>(nullable: false)
                .Annotation("SqlServer:Identity", "1, 1"),
            CustomerName = table.Column<string>(nullable: false),
            Prioritaet = table.Column<string>(nullable: false),
            Status = table.Column<string>(nullable: true)
        },
        constraints: table => table.PrimaryKey("PK_ServiceOrder", x =>
x.Id));
}
```

5 Kontrollieren

5.1 Teststrategie

Um die Qualität und Funktionalität des Systems sicherzustellen, wurde eine umfassende Teststrategie entwickelt. Diese besteht aus zweu Hauptkomponenten:

- **Integrationstests:** Sicherstellung, dass verschiedene Module korrekt zusammenarbeiten.
- **API-Tests:** Validierung der API-Endpunkte mit Tools wie Postman.

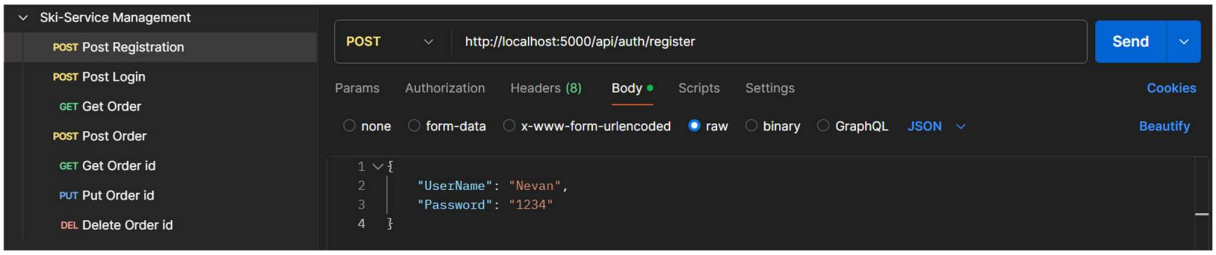
5.2 Nutzung von Postman für API-Tests

Postman wurde für automatisierte und manuelle Tests der API verwendet:

- **GET-Anfragen:** Test der Endpunkte für das Abrufen von Daten (z. B. /api/order).
- **POST-Anfragen:** Validierung der Erstellung neuer Serviceaufträge.
- **PUT/DELETE-Anfragen:** Sicherstellung der korrekten Aktualisierung und Löschung von Daten.

Postman-Collection:

- Post /api/Auth/register (mit einem JSON-Body)
- POST /api/Auth/login
- GET /api/Order/{id}
- PUT /api/Order/{id}
- DELETE /api/Order/{id}



5.3 Validierung der funktionalen Anforderungen

Die Tests überprüften, ob alle Anforderungen des Systems erfüllt wurden:

- User können sich registrieren und einloggen.
- Serviceaufträge können erstellt, angezeigt, bearbeitet und gelöscht werden.
- Nur authentifizierte User können sensible Operationen durchführen.

6 Auswerten

6.1 Erfüllung der Projektziele

Diesen sind alle die festgelegten Ziele:

- **Digitale Verwaltung von Serviceaufträgen:** Die Web-API ermöglicht eine effiziente Bearbeitung und Verwaltung von Kundenaufträgen.
- **Userfreundlichkeit:** Eine klare Struktur und API-Dokumentation erleichtern die Nutzung und Integration.
- **Sicherheit:** Sensible Funktionen wie das Bearbeiten oder Löschen von Aufträgen sind durch Authentifizierung geschützt.
- **Robuste Datenbankintegration:** Die Daten werden in einer normalisierten Struktur gespeichert, wodurch Konsistenz und Effizienz gewährleistet sind.

6.2 Herausforderungen während der Umsetzung

Während der Entwicklung traten mehrere Herausforderungen auf:

1. **Datenbankmigrationen:** Bei der Erstellung und Aktualisierung von Migrationen traten Kompatibilitätsprobleme auf, die durch eine sorgfältige Planung und Tests gelöst wurden.
2. **API-Sicherheit:** Die Implementierung der Authentifizierung und Autorisierung war komplex und erforderte detaillierte Tests, um unbefugten Zugriff zu verhindern.

6.3 Fazit

Das Projekt bot wertvolle Erkenntnisse und Lernerfahrungen:

- **Wichtigkeit der Planung:** Eine klare Struktur und Planung erleichterten die Umsetzung erheblich.
- **Iterative Entwicklung:** Durch kontinuierliche Tests und Anpassungen konnte die Qualität des Codes verbessert werden.
- **Fehlerbehandlung:** Die Bedeutung einer durchdachten Fehlerbehandlung wurde deutlich, insbesondere bei der Kommunikation zwischen Client und API.

Zusammenfassend hat das Projekt gezeigt, wie wichtig eine gut durchdachte Struktur und ein iterativer Ansatz bei der Softwareentwicklung sind. Es wurde nicht nur ein funktionales Produkt entwickelt, sondern auch wertvolle Erfahrungen gesammelt, die in zukünftigen Projekten angewendet werden können.