ipso! Bildung seit 1876

SkiService-Backend mit MongoDB

David Mesic, Beren Atici

ipso! Bildungsmarken

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

ipso! Bildung seit 1876

Inhaltsverzeichnis

1 Einleitung

1.1 Ziel und Zweck des Projekts

In diesem Projekt ging es darum, ein Backend-System zu entwickeln, das die internen Abläufe der Firma Jetstream-Service einfacher und effizienter macht. Mit dem neuen System können Ski-Service-Aufträge besser organisiert und die Digitalisierung des Unternehmens vorangetrieben werden. Dabei lag der Fokus auf einer sicheren Web-API, einem flexiblen Datenbankdesign und einer benutzerfreundlichen Verwaltung für die Mitarbeiter.

1.2 Ausgangssituation

Jetstream-Service ist ein Unternehmen, das sich auf Skiservice während der Wintersaison spezialisiert hat. Die bisherigen Aufträge wurden hauptsächlich manuell oder mit separaten digitalen Tools bearbeitet. Mit der zunehmenden Bedeutung der Digitalisierung hat die Firma beschlossen, eine zentrale Web- und Datenbank-basierte Lösung zu entwickeln. Ziel war es, die internen Prozesse zu vereinfachen, die Arbeit der Mitarbeiter effizienter zu gestalten und eine klarere Nachvollziehbarkeit der Auftragsdaten sicherzustellen.

1.3 Grund für die Wahl der IPERKA-Methode

Für die Umsetzung und Dokumentation des Projekts haben wir uns entschieden, die IPERKA-Methode zu verwenden. Diese Methode ist nicht nur vorgegeben, sondern auch einfach anzuwenden und ermöglicht eine strukturierte und nachvollziehbare Dokumentation aller Arbeitsschritte.

2 Informieren

Das Projekt begann mit einer klaren Analyse der Anforderungen. Ziel war es, ein modernes Backend-System zu entwickeln, das die internen Prozesse der Firma Jetstream-Service effizienter macht.

2.1 Ziele

- Sichere Authentifizierung für autorisierte Mitarbeiter.
- Gut strukturierte MongoDB-Datenbank mit minimaler Datenredundanz.
- Benutzerfreundliche API mit klarer Dokumentation.

2.2 Wichtige Aspekte

2.2.1 Authentifizierung

- Nur autorisierte Mitarbeiter dürfen auf sensible Daten und Funktionen zugreifen.
- Das System sollte einfach genug sein, um die tägliche Nutzung zu erleichtern.

2.2.2 Datenbankdesign & API-Dokumentation

- Effizientes Speichern und Verarbeiten von Informationen.
- Reduktion von Datenredundanzen durch eine optimierte Datenstruktur.
- Nutzung von Swagger zur Erstellung einer leicht zugänglichen Dokumentation.
- Fehlerprotokollierung zur einfachen Wartung und Weiterentwicklung.

3 Planen

Die Planung war ein zentraler Bestandteil unseres Projekts, um alle Schritte klar zu definieren und effizient umzusetzen. Dabei haben wir die Aufgaben in überschaubare Abschnitte aufgeteilt und einen realistischen Zeitplan erstellt.

3.1 Zeitplan

Arbeitspaket	verant-wortlich	geplanter Zeitaufwand	effektiver Zeitaufwand
Informieren			
Aufgabenstellung analysieren	DA & BE	1	1
Anforderungen stellen	DA & BE	1	1
Planen			
Zeitplanung			
	BE	0,5	0,5
Planung der Migration	DA	**0,5**	**1**
Erstellung eines Datenmodells	DA	1	0,5
Entscheiden			
Entscheidung für Struktur	BE	1	1
Umsetzungsschritte	BE	1	1,5
Realisieren			
Migration durchführen	DA & BE	3	3
API-Anpassen	DA & BE	1,5	1
CRUD Operationen	BE	0,5	0,5
Kontrollieren			
Testing mit Postman	DA	0,5	0,5
Validierung Teste	BE	0,5	0,5
Auswerten			
Präsentation Erstellen	BE	**1**	**1**
Dokumentation(lessons learned, Fazit)	DA & BE	2	2
		15	15

3.2 Technologieauswahl

Für die Entwicklung des Backends entschieden wir uns für ASP.NET Core, da es leistungsstark und flexibel ist. MySQL wurde als Datenbank ausgewählt, um eine stabile und skalierbare Speicherung zu gewährleisten. JWT diente als Methode für sichere Authentifizierung, und Swagger wurde genutzt, um die API leicht verständlich zu dokumentieren



4 Entscheiden

In der Entscheidungsphase haben wir uns intensiv mit der Auswahl der Technologien und Tools beschäftigt, um die Anforderungen des Projekts bestmöglich umzusetzen. Unser Ziel war es, leistungsstarke und zuverlässige Lösungen zu finden, die sowohl skalierbar als auch benutzerfreundlich sind.

4.1 Herausforderungen bei der Auswahl

Während der Entscheidungsphase mussten wir sicherstellen, dass alle gewählten Technologien reibungslos miteinander funktionieren und zukunftssicher sind. Die Integration von JWT für die Authentifizierung erforderte besondere Aufmerksamkeit, um Sicherheitslücken zu vermeiden und gleichzeitig die Nutzererfahrung so einfach wie möglich zu gestalten.

4.2 Ausgewählte Technologien und Tools

Diese Entscheidungen legten die Grundlage für die erfolgreiche Umsetzung des Projekts. Ausgewählte Technologien:

- **Backend:** ASP.NET Core für ein leistungsstarkes und flexibles Framework.
- **Datenbank:** MySQL für eine stabile und effiziente Speicherung von Daten.
- **Authentifizierung:** JWT (JSON Web Token) zur Sicherung sensibler Daten.
- **API-Dokumentation:** Swagger, um eine einfache und klare Dokumentation der API bereitzustellen
- **Testen:** Postman für eine sicheres Testing und Funktionierung für das Backend

5 Realisieren

5.1 Umsetzungsphase

Die Umsetzungsphase konzentrierte sich auf die Entwicklung und Implementierung der zentralen Komponenten des Backends, einschließlich der Datenbank, der API und der Authentifizierung. Ziel war es, ein stabiles und sicheres System bereitzustellen, das die Anforderungen des Projekts vollständig erfüllt.

5.2 Datenbankdesign (MongoDB)

Die Datenbank wurde so entworfen, dass alle relevanten Daten effizient und sicher gespeichert werden können. Da MongoDB eine dokumentenbasierte NoSQL-Datenbank ist, wurden die Daten in JSON-ähnlichen Dokumenten innerhalb von Collections gespeichert. Die Haupt-Collections sind:

5.2.1 Mitarbeiter (Collection: users)

```
{
  "_id": ObjectId,
  "username": "string",
  "password": "string",
  "role": "string"  // z. B. "admin" oder "employee"
}
```

Zweck: Verwaltung der Benutzerzugänge und Berechtigungen.

5.2.2 Aufträge (Collection: orders)

```
{
  "_id": ObjectId,
  "customerName": "string",
  "service": "string",
  "priority": "string",  // z. B. "hoch", "mittel", "niedrig"
  "status": "string",    // z. B. "offen", "in Bearbeitung", "abgeschlossen"
  "comment": "string",
  "createdAt": ISODate
}
```

Zweck: Speicherung und Verwaltung aller Serviceaufträge.

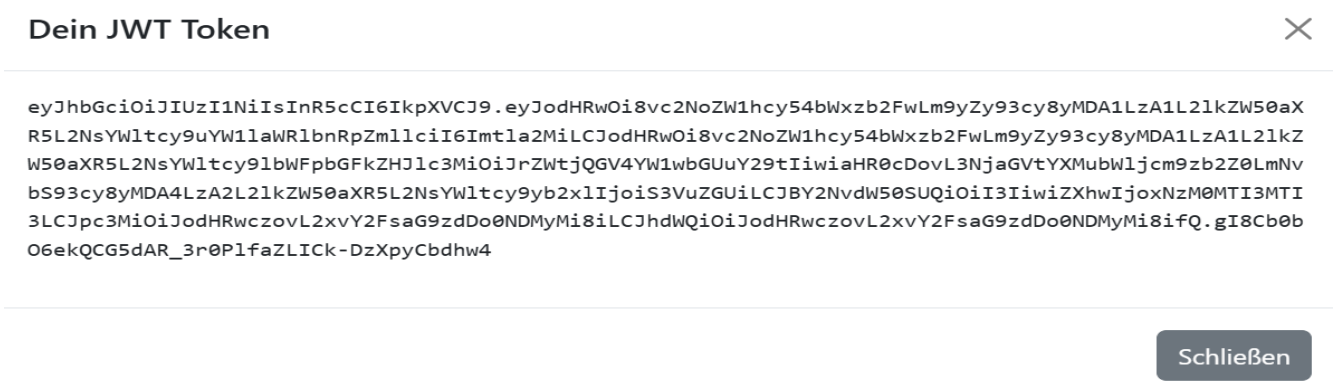
5.3 API-Entwicklung

Die API wurde so gestaltet, dass sie die wichtigsten Funktionen des Systems abdeckt und gleichzeitig sicher und benutzerfreundlich ist. Die implementierten Endpunkte umfassen:

- **POST /login** – Authentifizierung der Mitarbeiter mit JWT.
- **GET /orders** – Abrufen aller Aufträge aus der MongoDB-Datenbank.
- **GET /orders/{id}** – Zugriff auf einen einzelnen Auftrag anhand seiner ObjectId.
- **PUT /orders/{id}** – Bearbeiten eines bestimmten Auftrags.
- **DELETE /orders/{id}** – Löschen eines bestimmten Auftrags.
- **PATCH /orders/status** – Änderung des Status eines Auftrags.

5.3.1 Authentifizierung

- Zur Absicherung geschützter Endpunkte wurde JWT verwendet. Dies ermöglicht eine sichere Verwaltung der Benutzerzugriffe.
- Lesende Endpunkte wie das Abrufen der Auftragsliste bleiben öffentlich zugänglich, um die Nutzung zu vereinfachen.



5.4 Fehlerprotokollierung

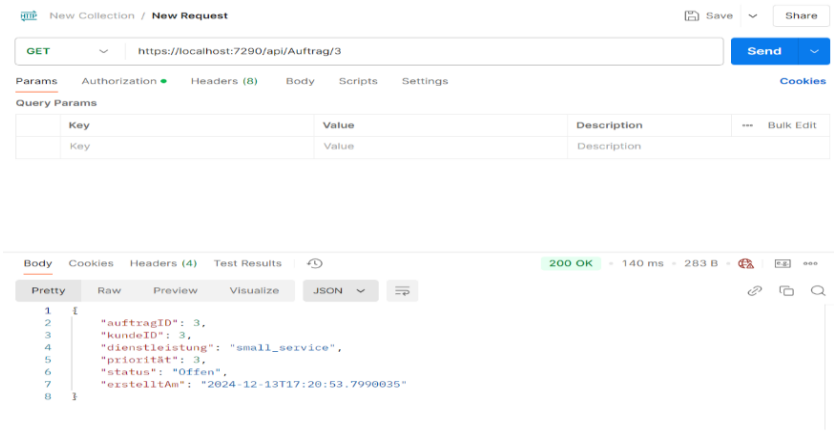
Fehlerprotokollierung:

- Alle API-Aufrufe und auftretenden Fehler werden in einer Protokolldatei gespeichert, um die Nachvollziehbarkeit zu gewährleisten.
- Zusätzlich erfolgt eine Protokollierung in der Datenbank für eine detaillierte Analyse.

```
[2024-11-27 23:15:07.945 +01:00  INF]  Now listening on: https://localhost:7251
[2024-11-27 23:15:07.966 +01:00  INF]  Now listening on: http://localhost:5244
[2024-11-27 23:15:07.998 +01:00  INF]  Application started. Press Ctrl+C to shut down.
[2024-11-27 23:15:07.998 +01:00  INF]  Hosting environment: Development
[2024-11-27 23:15:07.998 +01:00  INF]  Content root path: C:\Users\GerätvonDavid\Documents\Ipso\ICT-Module\Modul 295 (ÜK)\Projektmappe-Modul-295\Serillog
[2024-11-27 23:15:08.637 +01:00  INF]  Request starting HTTP/2 GET https://localhost:7251/swagger/index.html - null null
[2024-11-27 23:15:08.819 +01:00  INF]  Request finished HTTP/2 GET https://localhost:7251/swagger/index.html - 200 null text/html;charset=utf-8
185.8843ms
[2024-11-27 23:15:08.855 +01:00  INF]  Request starting HTTP/2 GET https://localhost:7251/_framework/aspnetcore-browser-refresh.js - null null
[2024-11-27 23:15:08.855 +01:00  INF]  Request starting HTTP/2 GET https://localhost:7251/swagger/index.js - null null
[2024-11-27 23:15:08.859 +01:00  INF]  Request finished HTTP/2 GET https://localhost:7251/swagger/index.js - 200 null
application/javascript;charset=utf-8 3.4983ms
[2024-11-27 23:15:08.861 +01:00  INF]  Request starting HTTP/2 GET https://localhost:7251/_vs/browserLink - null null
[2024-11-27 23:15:08.867 +01:00  INF]  Request finished HTTP/2 GET https://localhost:7251/_framework/aspnetcore-browser-refresh.js - 200 13768
application/javascript; charset=utf-8 12.2356ms
[2024-11-27 23:15:08.901 +01:00  INF]  Request finished HTTP/2 GET https://localhost:7251/_vs/browserLink - 200 null text/javascript; charset=UTF-8
39.5867ms
[2024-11-27 23:15:08.966 +01:00  INF]  Request starting HTTP/2 GET https://localhost:7251/swagger/v1/swagger.json - null null
[2024-11-27 23:15:08.971 +01:00  INF]  Request finished HTTP/2 GET https://localhost:7251/swagger/v1/swagger.json - 200 null
application/json;charset=utf-8 5.8369ms
[2024-11-27 23:15:38.357 +01:00  INF]  Request starting HTTP/2 GET https://localhost:7251/WeatherForecast - null null
[2024-11-27 23:15:38.391 +01:00  INF]  Executing endpoint 'Serillog.Controllers.WeatherForecastController.Get (Serillog)'
[2024-11-27 23:15:38.403 +01:00  INF]  Route matched with {action = "Get", controller = "WeatherForecast"}. Executing controller action with signature
System.Collections.Generic.IEnumerable`1[Serillog.WeatherForecast] Get() on controller Serillog.Controllers.WeatherForecastController (Serillog).
[2024-11-27 23:15:38.513 +01:00  INF]  Executing action method Serillog.Controllers.WeatherForecastController.Get (Serillog) - Validation state: "Valid"
[2024-11-27 23:15:38.518 +01:00  INF]  Executed action method Serillog.Controllers.WeatherForecastController.Get (Serillog), returned result
Microsoft.AspNetCore.Mvc.ObjectResult in 3.2199ms.
[2024-11-27 23:15:38.522 +01:00  INF]  Executing ObjectResult, writing value of type 'Serillog.WeatherForecast[]'.
[2024-11-27 23:15:38.538 +01:00  INF]  Executed action Serillog.Controllers.WeatherForecastController.Get (Serillog) in 132.2777ms
[2024-11-27 23:15:38.538 +01:00  INF]  Executed endpoint 'Serillog.Controllers.WeatherForecastController.Get (Serillog)'
[2024-11-27 23:15:38.538 +01:00  INF]  Request finished HTTP/2 GET https://localhost:7251/WeatherForecast - 200 null application/json; charset=utf-8
183.059ms
[2024-11-27 23:16:04.220 +01:00  INF]  Request starting HTTP/2 GET https://localhost:7251/WeatherForecast - null null
[2024-11-27 23:16:04.222 +01:00  INF]  Executing endpoint 'Serillog.Controllers.WeatherForecastController.Get (Serillog)'
```

5.5 Tests

- **Unit-Tests:** Diese wurden für die Controller-Methoden erstellt, um die Funktionalität sicherzustellen.
- **Postman-Tests:** Mit Postman wurde die API auf ihre korrekte Funktionsweise geprüft.
- **Sicherheitsüberprüfung:** Die JWT-Authentifizierung wurde auf potenzielle Schwachstellen getestet.



Mit diesen Schritten konnten wir ein robustes Backend-System entwickeln, das alle Anforderungen erfüllt und gleichzeitig flexibel genug ist, um zukünftige Erweiterungen zu unterstützen.

6 Kontrollieren

Die Kontrollphase war ein wesentlicher Schritt, um sicherzustellen, dass alle Komponenten des Systems zuverlässig und sicher arbeiten. Dabei wurden alle zentralen Funktionen gründlich getestet und validiert.

6.1 Tests und Validierung

- **Datenbank:** Wir haben überprüft, ob die Dokumentenstruktur korrekt modelliert wurde und die Konsistenz der gespeicherten Daten gewährleistet ist. Zudem wurden alle Abfragen und Indexierungen getestet, um sicherzustellen, dass die Performance den Anforderungen entspricht.

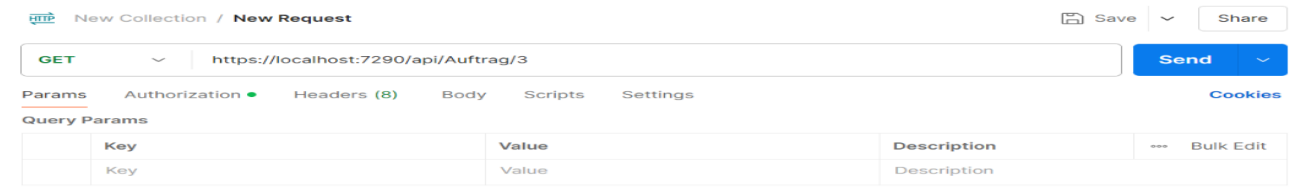
- **API:** Alle API-Endpunkte wurden mit Postman getestet, um sicherzustellen, dass sie wie geplant funktionieren. Dabei wurden auch mögliche Randfälle berücksichtigt, um Fehlfunktionen zu vermeiden. Insbesondere wurde getestet, ob CRUD-Operationen korrekt mit MongoDB interagieren und fehlerhafte Eingaben ordnungsgemäß behandelt werden.

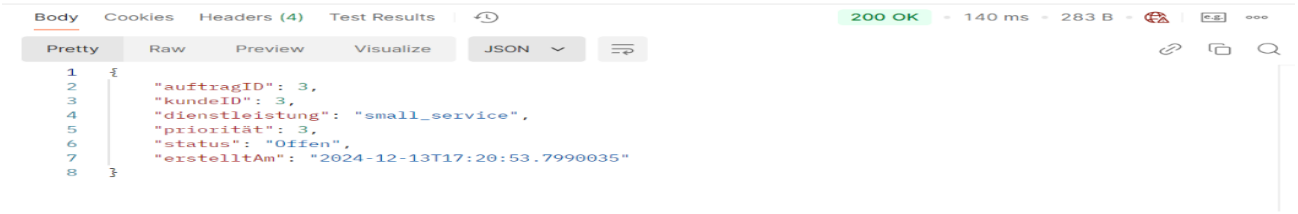
- **Authentifizierung:** Die JWT-Token-Erzeugung und -Verwaltung wurde getestet, um sicherzustellen, dass nur autorisierte Benutzer auf geschützte Endpunkte zugreifen können.

  Protokollierung: Die Log-Funktionalität wurde überprüft, um sicherzustellen, dass alle Fehler und API-Aufrufe korrekt aufgezeichnet werden.

6.1.1 Einsatz von Postman

Postman war ein unverzichtbares Tool für die Durchführung der API-Tests. Mit seiner Hilfe konnten wir gezielt HTTP-Requests an die Endpunkte senden und die Antworten analysieren. Dadurch konnten wir sicherstellen, dass die Daten korrekt an die Datenbank übermittelt werden. Fehlerhafte Eingaben ordnungsgemäß abgefangen und mit verständlichen Fehlermeldungen beantwortet werden.





7 Auswerten

Die abschließende Auswertungsphase bot uns die Gelegenheit, über das gesamte Projekt zu reflektieren, unsere Fortschritte zu bewerten und wertvolle Erkenntnisse für zukünftige Projekte zu gewinnen.

7.1 Lessons Learned

- Während des Projekts haben wir nicht nur technische Kenntnisse erweitert, sondern auch viele praktische Einsichten gewonnen:
- **Struktur und Planung:** Eine klare Struktur und eine detaillierte Planung waren ausschlaggebend für den Projekterfolg. Sie haben uns geholfen, auch komplexe Aufgaben effizient umzusetzen und flexibel auf Herausforderungen zu reagieren.
- **JWT-Authentifizierung:** Die Arbeit mit JWT hat uns die Bedeutung einer sicheren Implementierung deutlich gemacht. Dieses Verfahren bietet viele Vorteile, erfordert jedoch besondere Aufmerksamkeit, um Schwachstellen zu vermeiden.
- **Zusammenarbeit von MongoDB und API:** Wir haben gelernt, dass die enge Abstimmung zwischen der MongoDB-Datenbank und der API-Entwicklung entscheidend für eine nahtlose Funktionalität ist. Diese Verknüpfung sollte von Anfang an gut durchdacht sein.
- **Wichtigkeit von Tests:** Durch die Nutzung von Postman konnten wir die Stabilität und Sicherheit der API umfassend testen. Dies hat uns gezeigt, wie wichtig es ist, kontinuierlich zu testen und direkt auf potenzielle Fehler zu reagieren.

7.2 Projekterfolg

Das Projekt wurde erfolgreich abgeschlossen, und alle Ziele wurden erreicht:

- **Umsetzung:** Alle geplanten Funktionen, einschließlich optionaler Features wie Kommentarfunktion und Login-Sperrung, wurden vollständig realisiert.
- **Stabilität und Sicherheit:** Das System ist zuverlässig und sicher, bereit für den praktischen Einsatz.
- **Flexibilität:** Die Architektur erlaubt künftige Erweiterungen und Anpassungen, was das System nachhaltig macht.

7.3 Reflektion

Mit diesem Vorhaben haben wir uns auf verschiedenen Ebenen bereichert. Wir hatten die Möglichkeit, neue Technologien wie die JWT-Authentifizierung und moderne NoSQL-Datenbankarchitekturen kennenzulernen und in die Praxis umzusetzen. Es hat uns auch verdeutlicht, welche Bedeutung eine klare Planung und ein strukturierter Ansatz für die Bewältigung komplexer Herausforderungen haben. Die Implementierung der Authentifizierung und die Optimierung der Datenbankabfragen waren besonders herausfordernd. Aber genau diese Aufgaben haben uns gezeigt, wie wichtig es für die Schaffung eines stabilen und sicheren Systems ist, saubere Implementierungen und vorausschauendes Denken anzuwenden. Wir sind im Nachhinein stolz auf die Ergebnisse. Das System bietet nicht nur einen echten Mehrwert durch die optionalen Erweiterungen, sondern erfüllt auch alle gestellten Anforderungen. In künftigen Projekten werden wir durch diese Erfahrungen noch effektiver und zielgerichteter arbeiten können.