



SKISERVICE

Back-end für die Website von Jet-Stream

Giorgio Jacomella

Inhalt

Projektablauf	2
Informieren.....	3
Projektanforderungen	3
Technologien	3
Potentielle Wissenslücken	4
Planen.....	4
Datenbank	4
API.....	5
Authentifizierung und Autorisierung.....	5
Frontend Interface.....	5
Entscheiden	6
Versionierung.....	6
Vorgehensvariation.....	6
Alternative Komponente	6
Realisieren	7
Git Repository	7
Initialisierung	7
Datenbank	7
API Schnittstellen.....	7
Client Side Verbinden	7
Endpoints mit JWT Authentifizierung.....	7
Dashboard/Mitarbeiter Tools	8
Kontrollieren.....	9
API Endpoints ohne Authentifizierung	9
API Endpoints mit Authentifizierung	10
Auswerten	11
Nutzeranleitung:.....	12
Quellenverzeichnis.....	12

Projektablauf

Bei diesem Projekt wird IPERKA angewendet, eine im Projektmanagement weit verbreitete Vorgehensweise.

Die einzelnen durchgeführten Schritte werden wie folgt aussehen:

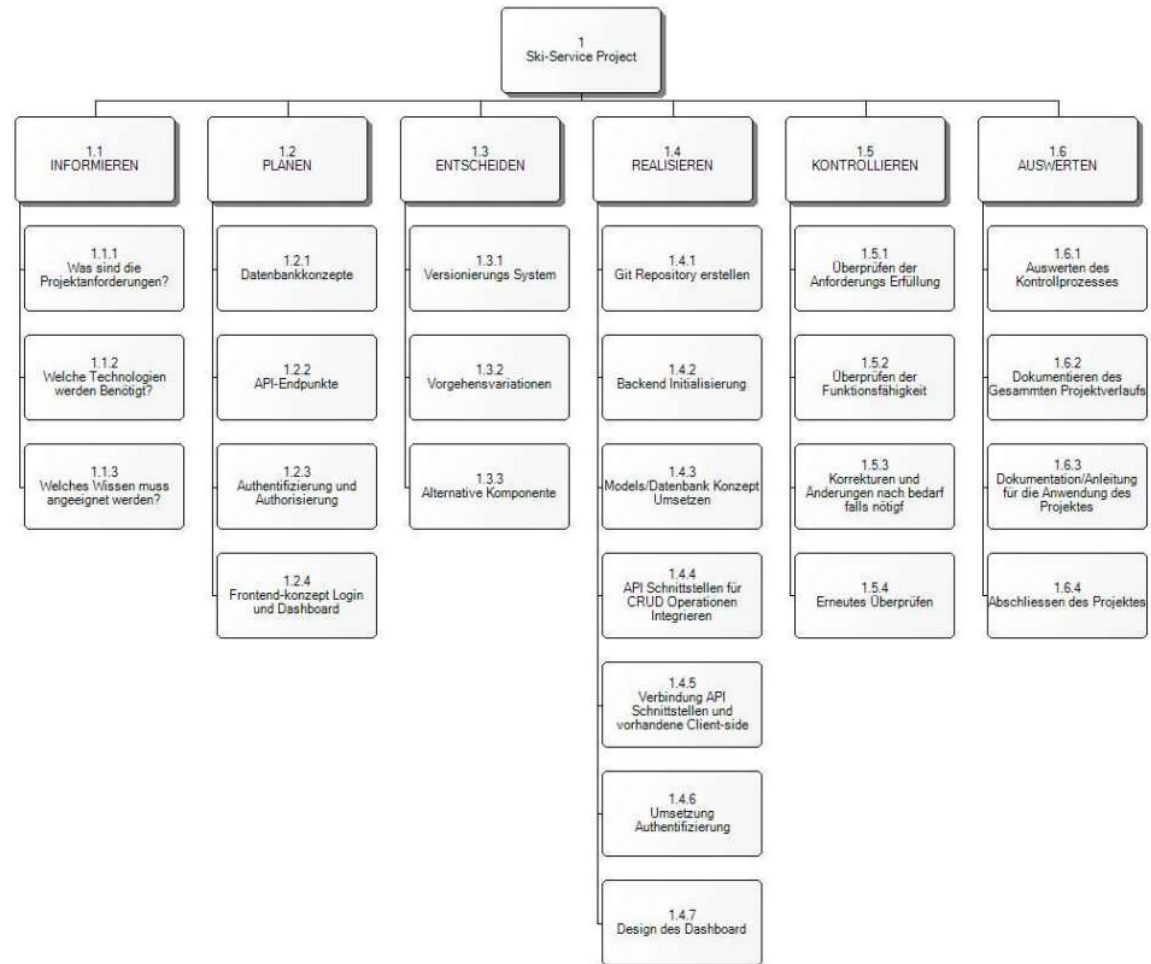


Abbildung 1/ WBSTool Grafik Projektplan

Informieren

In dieser Phase geht es darum, sich einen Überblick über das Projekt zu verschaffen. Die Informationen zu unserem Auftrag lauten wie folgt:

Projektanforderungen

Das Auftragsmanagement muss folgende Funktionen zur Verfügung stellen:

- Login mit Benutzername und Passwort
- Anstehende Serviceaufträge anzeigen (Liste)
- Bestehende Serviceaufträge mutieren. Dazu stehen folgende Status zur Verfügung: Offen, In Arbeit und Abgeschlossen
- Aufträge löschen (ggf. bei Stornierung)

Die Informationen zur Online-Anmeldung, die bereits realisiert wurden, müssen ggf. bei Bedarf wie folgt ergänzt werden:

- Kundenname
- E-Mail
- Telefon
- Priorität

Dienstleistung (Angebot), siehe nachfolgende Auflistung. Pro Serviceauftrag kann immer nur eine Dienstleistung zugeordnet werden.

Technologien

Um in diesem Projekt effizient voranzukommen, ist es wichtig, einheitliche Technologien zu verwenden.

Datenbanksystem

Bei der Datenbank gibt es eine große Auswahl verschiedener Systeme. Ich habe mich für MS SQL entschieden, da ich mit dem Datenbankserver sowie mit dem Datenbankserver-Manager von Microsoft die meisten Erfahrungen gesammelt habe.

Objektorientierte Programmiersprache

Die Wahl der Programmiersprache fällt bei diesem Projekt auf C# mit .NET.

Frameworks

Bei den Frameworks gibt es einiges, was uns bei diesem Projekt behilflich sein wird. Als erstes komme ich zu ASP.NET, welches für .NET geeignet ist, um API-Schnittstellen zu entwickeln. Dies kombinieren wir noch mit ADO.NET, um mit unserer Datenbank interagieren zu können.

Tools

Ich werde mit folgenden Tools arbeiten, um in diesem Projekt voranzukommen:

- GitHub – Versionierung und Repository-Verwaltung
- Visual Studio 2022 - Editor
- Visual Studio Code - Editor
- MS SQL Server - SQL Server
- MS SQL Server Management Studio 19 - SQL Server-Management-Tool
- Swagger UI – API-Visualisierung und Test-Interface
- .NET 7.0 – Framework für C

Potentielle Wissenslücken

Authentifizierung

Beim Projektbeginn war mir noch nicht zu 100% klar wie ich einen Nutzer Authentifizieren kann. Diese Wissenslücke habe ich jedoch mit Erlernen der Anwendung von JWT geschlossen.

Datenbanken und OOP

Beim Projektbeginn wusste ich noch nicht wie ich Daten aus vorhandenen Datenbanken in meinem OOP Projekt anwenden kann, dies habe ich jedoch durch die Anwendung des ADO.NET Framework und dem Database First / Code First Konzept begriffen.

Planen

Basierend auf den gesammelten Informationen wird ein Projektplan entwickelt.

Datenbank

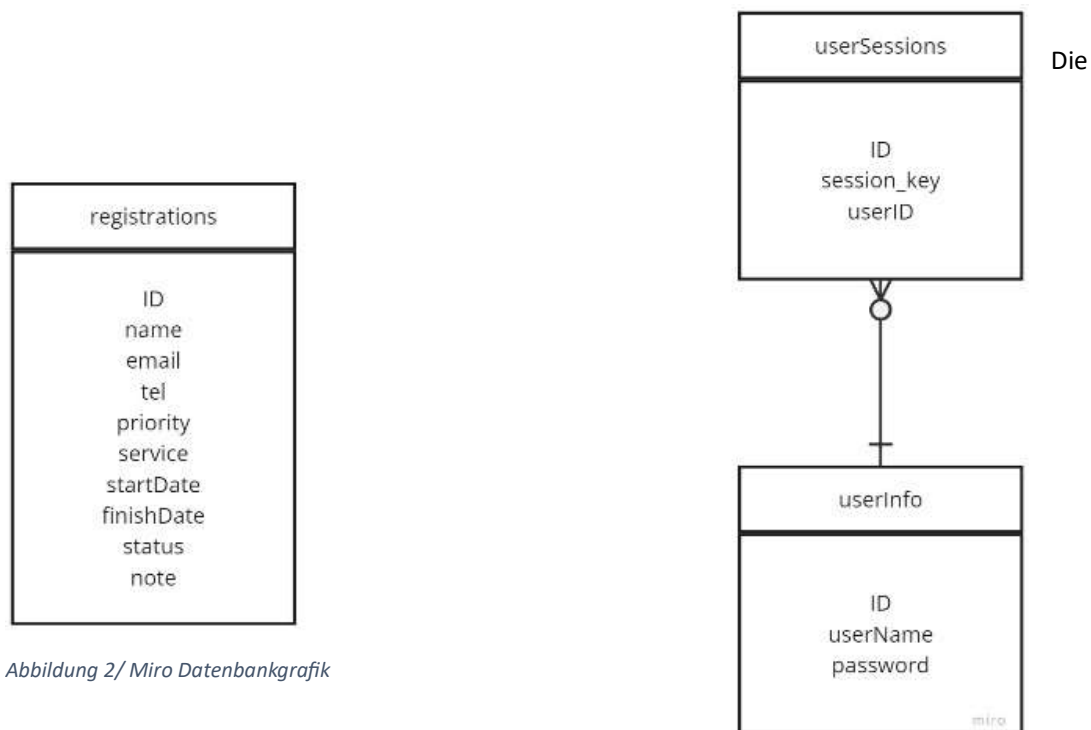


Abbildung 2/ Miro Datenbankgrafik

Datenbank verfügt über einen Table indem Kundenaufträge gespeichert werden, diesen habe ich «registrations» genannt, dieser Table enthält:

- ID(Primary Key)
- Namen des Kunden
- Email des Kunden
- Telefonnummer des Kunden
- Priorität des Auftrags
- Wann der Auftrag aufgegeben wurde
- Wann der Auftrag erfüllt sein muss
- Ob der Auftrag erfüllt wurde
- Eine mögliche zusätzliche Notiz

Der Nächste Table ist «userInfo», der zwecks von diesem Table ist es die Nutzerdaten der Mitarbeiter auf zu wahren so dass diese sich Einloggen können um die Aufträge Anzusehen, zu Bearbeiten oder zu löschen, dieser Table enthält:

- ID(Primary Key)
- userName
- password

Um nach einem Login eines Mitarbeiters diesen Wieder erkennen zu können Speichern wir die Relevanten Informationen im Table «userSessions», dieser enthält:

- ID(Primary Key)
- Session_key
- userID(PK aus dem Table «userInfo»)

API

Die API wird verschiedene einzelne Funktionen, diese wären:

- POST(Neuen Auftrag registrieren)
- GET(Alle Aufträge anzeigen)
- GET{id}(Auftrag mit einer bestimmten ID anzeigen)
- PUT{id}(Auftrag mit bestimmter ID bearbeiten, nur zu demonstrationszwecken)
- DELETE{id}(Auftrag mit bestimmter ID löschen, nur zu demonstrationszwecken)

Und folgende API's mit JWT Authentifizierung:

- POST(Nutzer Anmelden)
- PUT{id}(Auftrag mit bestimmter ID bearbeiten)
- DELETE{id}(Auftrag mit bestimmter ID löschen)

Authentifizierung und Autorisierung

Zur Authentifizierung werden JWT Verwendet, für jede Anmeldung wird ein JWT generiert. Der JWT wird einerseits in der Datenbank Gespeichert und kann andererseits auch vom Client gespeichert werden, beispielsweise im Local Storage.

Frontend Interface

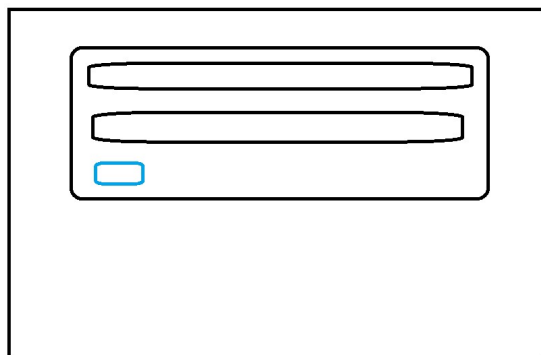


Abbildung 4/ Login Interface für Mitarbeiter Skizze

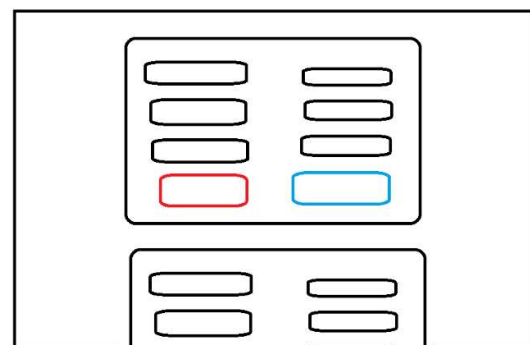


Abbildung 3/ Erste Skizze für Interface der Bestellungen aus Mitarbeiter Sicht

Entscheiden

Versionierung

Meine Wahl bei den Tools für Versionierung war GitHub und Git. Diese sind geeignet zur Hilfe bei Versionierung und dem Entwicklung Prozess.

Vorgehensvariation

Die Vorgehensvariante für die ich mich entschieden habe ist wird folgenderweise ablaufen:

1. Git Repository erstellen.
2. Initialisierung ASP.Net Projekt.
3. Datenbank Umsetzen und Models Importieren(Database First).
4. API Schnittstellen Realisieren.
5. Client Side Verbinden.
6. Entwicklung JWT Authentifizierung Schnittstellen.
7. Dashboard/Mitarbeiter Tools entwickeln.

Alternative Komponente

Die Komponente ohne Authentifizierung mit veränderndem Einfluss auf die Datenbank werden nur zu Demonstrationszwecken Implementiert

Ohne Authentifizierung:

- PUT{id}(Auftrag mit bestimmter ID bearbeiten, nur zu demonstrationszwecken)
- DELETE{id}(Auftrag mit bestimmter ID löschen, nur zu demonstrationszwecken)

Mit Authentifizierung:

- POST(Nutzer Anmelden)
- PUT{id}(Auftrag mit bestimmter ID bearbeiten)
- DELETE{id}(Auftrag mit bestimmter ID löschen)

Realisieren

In diesem Teil des IPERKA Prozesses wird das Schlussendlich Relevante Ergebnis Entwickelt.

Git Repository

<https://github.com/GiorgioJacomella/Ski-Service>

Initialisierung

Als erstes habe ich ein .gitignore File erstellt wo ich bestimmt habe welche dateien nicht von Git beeinflusst werden und auch nicht auf meinem Repository erscheinen werden.

Danach fuhr ich mit der Initialisierung der API fort. Das Projekt basiert auf ASP.net, zusätzlich Installieren wir die benötigten wir die NuGet Pakete um mit unserem Spezifischen SQL Server kommunizieren zu können(in meinem fälle ist dies MS SQL).

Um mit der Datenbank arbeiten zu Können musste unter anderem noch ein Connection String in den App Settings Festgelegt werden.

Datenbank

Da ich mit dem Database First System am besten arbeiten konnte begann ich damit die Datenbank zu realisieren, diese Datenbank habe ich dann mithilfe von Folgendem Command in die Models Importiert:

```
Scaffold-DbContext "Server=.;Database=skiService;Trusted_Connection=True;TrustServerCertificate=True;"  
Microsoft.EntityFrameworkCore.SqlServer -OutputDir Models -Context BlogContext
```

Somit kann ich von nun an mit OOP Techniken mit der Datenbank interagieren.

API Schnittstellen

Als erstes habe ich für jede CRUD Operation zu den Registrierten Aufträgen eine Schnittstelle entwickelt.

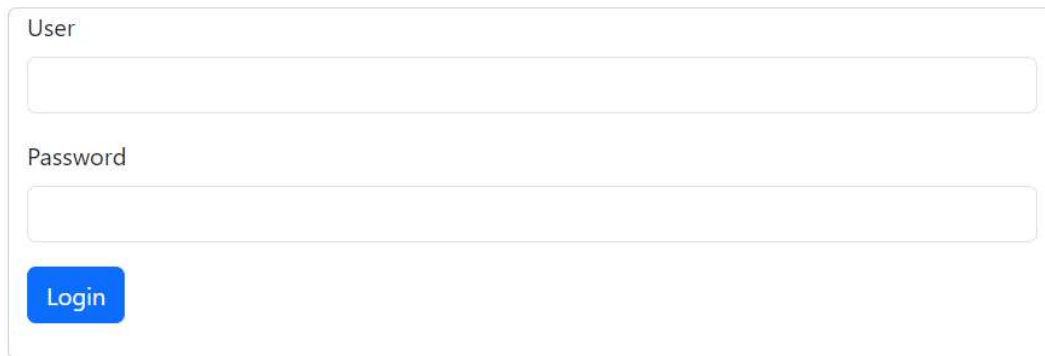
Client Side Verbinden

In diesem Schritt wird der Fetch Post API Call so angepasst das die Eingabe Daten eines Kunden an den Korrekten API End Point weiter Gesendet werden.

Endpoints mit JWT Authentifizierung

Als Erstes musste ein Login End Point entwickelt werden, bei diesem Können Nutzer/Mitarbeiter Ihre Anmeldedaten eingeben, der Back-end Server Überprüft danach die Eingabedaten, sind diese Korrekt wird ein JWTToken Generiert welcher auf der Datenbank Gespeichert wird und auch an den Client gesendet, der Client kann nun auch die weiteren API's welche einen JWT Benötigen verwenden und ist somit berechtigt Daten der Registrierten Kundenbestellungen abzuändern und sogar Kundenbestellungen zu Löschen.

Dashboard/Mitarbeiter Tools



A login form with two input fields and a button. The first field is labeled 'User' and the second is labeled 'Password'. Below the password field is a blue button labeled 'Login'.

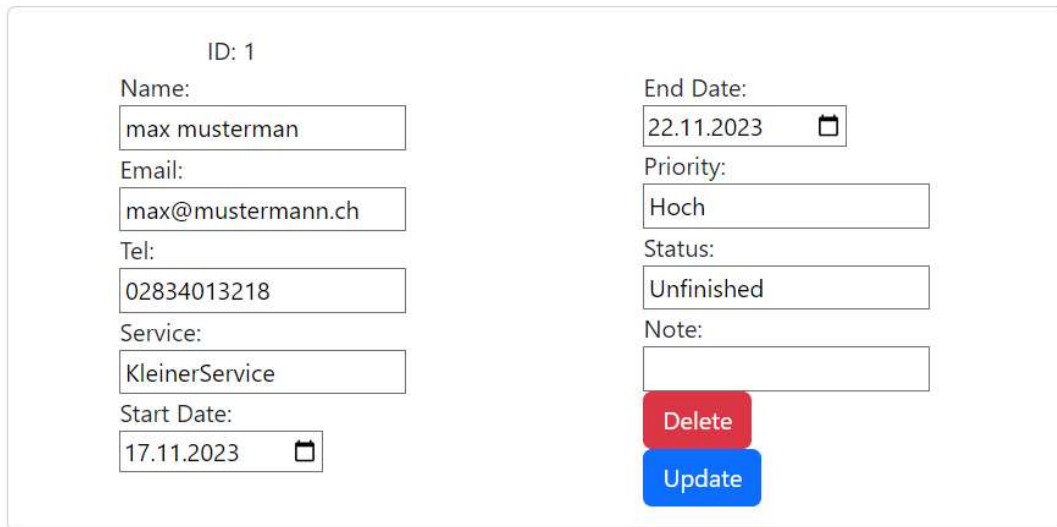
User

Password

Login

Abbildung 5/ Login Form für Mitarbeiter

Bei dem Login Form(Abbildung 5) Können sich die Mitarbeiter Anmelden um auf Bestellungen zuzugreifen und diese auch zu Bearbeiten.



A form showing details for an employee (ID: 1) and their order. The form is divided into two columns. The left column contains fields for Name, Email, Tel, Service, and Start Date. The right column contains fields for End Date, Priority, Status, and Note. At the bottom right are two buttons: 'Delete' (red) and 'Update' (blue).

ID: 1

Name: max musterman

Email: max@mustermann.ch

Tel: 02834013218

Service: KleinerService

Start Date: 17.11.2023

End Date: 22.11.2023

Priority: Hoch

Status: Unfinished

Note:

Delete

Update

Abbildung 6/ Darstellung der Bestellungen aus der Sicht eines Mitarbeiters

Kontrollieren

Hier werden wir die einzelnen Wichtigen Funktionen der API überprüfen, hierbei werde ich Swagger zur Hilfe nehmen.

API Endpoints ohne Authentifizierung

Get api/registration

Keine Parameter

Responses:

200

```
[
  {
    "id": 1,
    "name": "max musterman",
    "email": "max@mustermann.ch",
    "tel": "02834013218",
    "priority": "Hoch",
    "service": "KleinerService",
    "startDate": "2023-11-17T00:00:00",
    "finishDate": "2023-11-22T00:00:00",
    "status": "Unfinished",
    "note": ""
  }
]
```

Post api/registration

Request Body:

```
{
  "name": "string",
  "email": "user@example.com",
  "tel": "0123458",
  "priority": "Hoch",
  "service": "KleinerService",
  "startDate": "2023-11-21T14:06:32.728Z",
  "finishDate": "2023-11-21T14:06:32.728Z",
  "status": "Unfinished",
  "note": ""
}
```

Responses:

201

```
{
  "id": 5,
  "name": "string",
  "email": "user@example.com",
  "tel": "0123458",
  "priority": "Hoch",
  "service": "KleinerService",
  "startDate": "2023-11-21T14:06:32.728Z",
  "finishDate": "2023-11-21T14:06:32.728Z",
  "status": "Unfinished",
  "note": ""
}
```

Get api/registration{id}

ID: 5

Responses:

200

```
{
  "id": 5,
  "name": "string",
  "email": "user@example.com",
  "tel": "0123458",
  "priority": "Hoch",
  "service": "KleinerService",
  "startDate": "2023-11-21T14:06:32.728Z",
  "finishDate": "2023-11-21T14:06:32.728Z",
  "status": "Unfinished",
  "note": ""
}
```

Post api/register{id}

ID: 5

```
{
  "id": 5,
  "name": "string",
  "email": "user@example.com",
  "tel": "0123458",
  "priority": "Hoch",
  "service": "KleinerService",
  "startDate": "2023-11-21T14:06:32.728Z",
  "finishDate": "2023-11-21T14:06:32.728Z",
  "status": "Unfinished",
  "note": "Eine einfache notitz"
}
```

Responses:

204

API Endpoints mit Authentifizierung

Post api/login

```
{
  "userName": "admin",
  "password": "admin"
}
```

Responses:

200

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbSI6ImFmFmNGFmNjE3LWM1ZGQtNGJjNy05MDU4LTA2NDZmZjkyMmIwYSIsImV4cCI6MTcwMDU4NDAY0CwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo1MDkyIiwiaXVkiOjoiHR0cDovLzEyNy4wLjAuMT01NTAwIn0.7f0e5fQwu7QKR9UI2716eVI6QU7T5GiaEiFVZVt_0kc
```

Put api/mitarbeiter/registrations{id}

ID: 5

Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbilzImp0aSI6ImFmNGFmNjE3LWM1ZGQtNGJjNy05MDU0LTA2NDZmZjkyMmlwYSIsImV4cCI6MTcwMDU0NDYwOCwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo1MDkylwiYXVkljoiaHR0cDovLzEyNy4wLjAuMT01NTAwIn0.7f0e5fQwu7QKR9UI27I6eVI6QU7T5GiaeIFVZVt_0kc

Body:

```
{
  "id": 5,
  "name": "string",
  "email": "user@example.com",
  "tel": "0123458",
  "priority": "Hoch",
  "service": "GrosserService",
  "startDate": "2023-11-21T14:06:32.728Z",
  "finishDate": "2023-11-21T14:06:32.728Z",
  "status": "Unfinished",
  "note": "Eine einfache notitz"
}
```

Responses:

200

```
{
  "id": 5,
  "name": "string",
  "email": "user@example.com",
  "tel": "0123458",
  "priority": "Hoch",
  "service": "GrosserService",
  "startDate": "2023-11-21T14:06:32.728Z",
  "finishDate": "2023-11-21T14:06:32.728Z",
  "status": "Unfinished",
  "note": "Eine einfache notitz"
}
```

[Delete api/mitarbeiter/registrations{id}](#)

ID: 5

Token:

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJhZG1pbilzImp0aSI6ImFmNGFmNjE3LWM1ZGQtNGJjNy05MDU0LTA2NDZmZjkyMmlwYSIsImV4cCI6MTcwMDU0NDYwOCwiaXNzIjoiaHR0cDovL2xvY2FsaG9zdDo1MDkylwiYXVkljoiaHR0cDovLzEyNy4wLjAuMT01NTAwIn0.7f0e5fQwu7QKR9UI27I6eVI6QU7T5GiaeIFVZVt_0kc

Response: 204

Auswerten

Ein Endprodukt welches ein Funktionsfähiges Login mit Benutzername und Passwort hat, Anstehende Serviceaufträge in einer Bearbeitbaren Liste anzeigt. Zusätzlich kann der Status Gesetzt oder bearbeitet werden wie auch eine Notiz hinzugefügt werden. Ausserdem JWT Konnten Funktionsfähig Implementiert werden.

Nutzeranleitung:

1. Clonen des Github Repositorys
2. Ausführen Datenbank Query
3. Anpassen des Connection string und der Appsettings oder auch Localhost Ports
4. Ausführen des Programm.cs
5. Ausführen des Front-ends als Live Server Öffnen

Quellenverzeichnis

Grafiken: <https://miro.com/>

Swagger: <https://swagger.io/>

GitHub: <https://github.com/>

Git: <https://git-scm.com>

Gitignore Vorlage: <https://gist.github.com/vmandic/ac2ecc9c24f6899ee0ec46e4ce444a0e>

Dotnet: <https://dotnet.microsoft.com/>