



PROJEKTARBEIT SKI-SERVICE MANAGEMENT

Azin Ildas



IPSO BILDUNG AG
IBZ Basel

Abbildungsverzeichnis

Abbildung 1: Logo Jetstream Ski-Service	2
Abbildung 2: Ansicht meiner Web-Apis.	4
Abbildung 3: Ansicht des Post Login Endpunktes.....	5
Abbildung 4: Ansicht des Post Login Endpunktes nach dem ausführen.....	5
Abbildung 5: Ansicht des Put Login Endpunktes.	6
Abbildung 6: Ansicht des Delete Customer Endpunktes.	7
Abbildung 7: Ansicht des Get Endpunktes.....	7
Abbildung 8: Ansicht des Post Endpunktes.....	8
Abbildung 9: Ansicht des Endpunkts zum Abrufen einer Bestellung anhand ihrer Id..	9
Abbildung 10: Ansicht des Endpunkts zum Aktualisieren einer Bestellung anhand ihrer Id.....	10
Abbildung 11: Ansicht des Endpunkts zum Löschen einer Bestellung anhand ihrer Id.	10
Abbildung 12: Ansicht meiner SQL Datenbank.....	11
Abbildung 13: Ansicht meiner appsettings.json Datei.....	12

Versionsverzeichnis

Version	Datum	Änderung
1	28.10.2023	Projektdokumentation Erstelldatum
1.1	28.10.2023	Erste Einträge in die Dokumentation
1.2	21.11.2023	Projektdokumentation Abschlussdatum

1 Einleitung

1.1 Was war der Auftrag?

Die Firma Jetstream-Service plant seine Ski-Service-Auftragsverwaltung zu modernisieren. Sie möchten alles über eine webbasierte Anwendung abwickeln und diese erweitern, um Aufträge besser zu verwalten. Bis zu 10 Mitarbeiter sollen Zugang zu den Aufträgen haben und diese bearbeiten können.



Abbildung 1: Logo Jetstream Ski-Service

1.2 Rollen

1.2.1 Der Auftraggeber

Der Auftraggeber für das Projekt ist die Firma Jetstream Service.

1.2.2 Der Auftragnehmer

Der Auftragnehmer, welcher die Anforderungen von dem Auftraggeber entgegennimmt, ist Azin Ildas.

2 Informieren

2.1 Was soll getan werden?

Das Projekt umfasst die Entwicklung des Backend-Teils, einschliesslich einer sicheren Anmeldung, Datenbankdesign, Tests und die eigentliche Umsetzung der Anwendung. Mitarbeiter müssen sich anmelden, um Auftragsdaten zu ändern, während das Anzeigen von Aufträgen keine Anmeldung erfordert.

3 Planen

3.1 Welche Lösungswege gibt es und wie kann ich vorgehen?

1. Das ganze selbst Entwickeln.
 - a. Web-API erstellen welches den Anforderungen des Frontend entspricht.
 - b. Die komplette SQL Datenbank in die Web-API implementieren, entweder mit "Code First" oder "Database First".
 - c. OpenAPI-Dokumentation erstellen um die Web-API klar zu dokumentieren.
2. Ein vorhandenes Programm benutzen welches die benötigten Kriterien beinhaltet.
 - a. Die Software nach den Kriterien anpassen.
 - b. Die Software Testen.
 - c. Die Software Dokumentieren und die Mitarbeiter von Jetstream-Service einschulen.

4 Entscheiden

4.1 Für welches vorgehen entscheide ich mich?

Für dieses Projekt entscheide ich mich für den ersten Weg, weil ich das Projekt erfolgreich abschließen und gleichzeitig neue Dinge lernen möchte. Mein Plan sieht folgendermassen aus:

1. Web-API erstellen welches den Anforderungen des Frontend entspricht.
2. Die komplette SQL Datenbank in die Web-API implementieren, entweder mit "Code First" oder "Database First".
3. OpenAPI-Dokumentation erstellen um die Web-API klar zu dokumentieren.

5 Realisieren

5.1 Mein Web-API

Mein Web Api im Überblick:

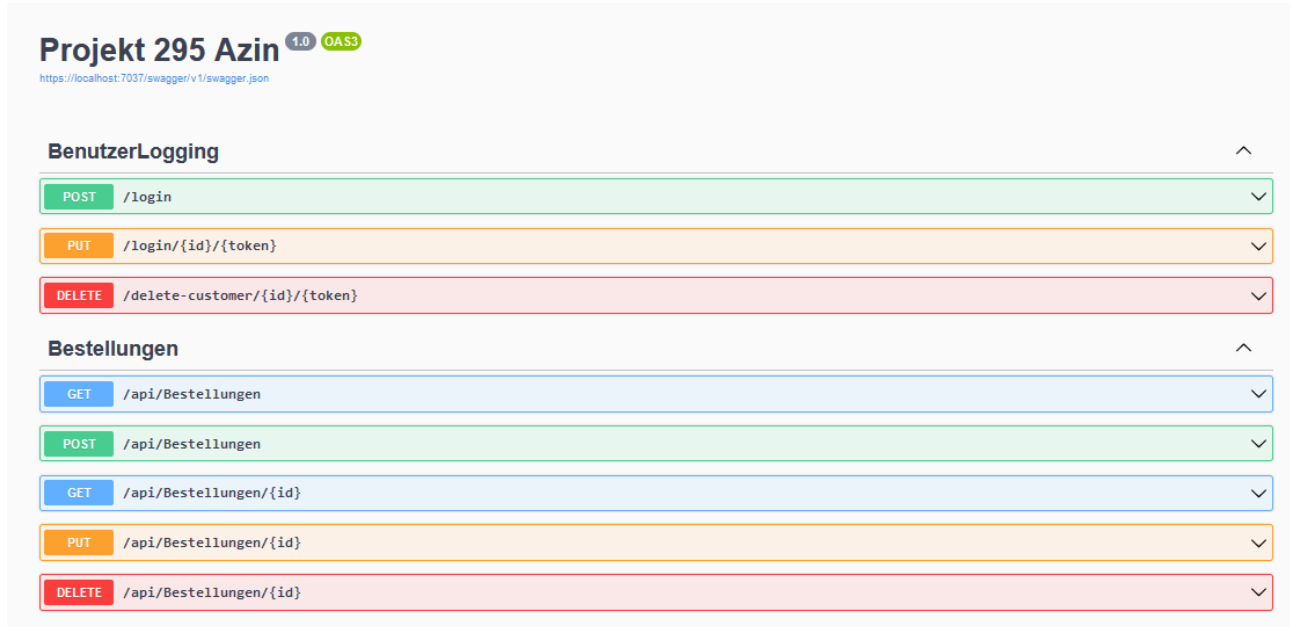


Abbildung 2: Ansicht meiner Web-Apis.

5.1.1 Post/login

Der POST Login API-Endpunkt ermöglicht das Hochladen eines Json Web Tokens in unsere SQL-Datenbank:

1. Im Body Kasten die Persönlichen Daten angeben.
2. Auf «Execute» klicken um den Command auszuführen.

POST

/login

⌵

Parameters

Cancel

Reset

No parameters

Request body

application/json

⌵

```
{
  "username": "TestUser",
  "password": "TestPasswort"
}
```

Execute

Abbildung 3: Ansicht des Post Login Endpunktes.

- Nachdem wir auf «Execute» geklickt haben, erhalten wir den Json Web Token welche in die Datenbank gespeichert wird und uns die Aktualisierung und Löschung von Kunden ermöglicht.

Curl	<pre>curl -X 'POST' \ 'https://localhost:7037/login' \ -H 'accept: */*' \ -H 'content-type: application/json' \ -d '{ "username": "Testuser", "password": "TestPasswort" }'</pre>
Request URL	<pre>https://localhost:7037/login</pre>
Server response	
Code	Details
200	Response body
	<pre>{ "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIjUzI2NDVWVXNlciIsImp0eSI6IiYwMTI2MDQ2LWE0MGItTGMyOS1lZmE1LTJlZDZlZDZGRHMSIiwiaW4CI6MTcwMDQ4MTU5NCiwiaXNzIjoiaHR0bG93b2Nhbgvc306N2AzNyIsImF1dGUiOiJhbmh0dHAgY8x0jcjUwPC4wLjE6NTUuMCJ3.5cmUEcp9Xj-jj9Iolk9x0xCw_K846GsiPtTKw5Hn8" }</pre>
Response headers	<pre>content-type: application/json; charset=utf-8 date: Tue, 21 Nov 2023 15:16:34 GMT server: Kestrel x-firefox-spyd: h2</pre>
Responses	

Abbildung 4: Ansicht des Post Login Endpunktes nach dem ausführen.

5.1.2 Put/login/{id}/{token}

Der PUT Login API-Endpunkt ermöglicht das Aktualisieren eines Json Web Tokens in unsere SQL-Datenbank:

1. Bei den Parameters Id und den Token angeben welche wir mit dem Post bekommen haben.
2. Im Body Kasten die Persönlichen Daten zum aktualisieren angeben.
3. Auf «Execute» klicken um den Command auszuführen.

The screenshot shows a REST client interface for the PUT endpoint `/login/{id}/{token}`. The interface is divided into several sections:

- Parameters:** A table with two parameters:

Name	Description
id * required integer(\$int32) (path)	<input type="text" value="3"/>
token * required string (path)	<input type="text" value="Ecp9rXj-ij9olkq9xWCw_K846GsptLTkw5Hn8"/>
- Request body:** A text area containing a JSON object:


```
{
  "bestellungsId": 3,
  "name": "Eduard",
  "emailaddress": "user@example.com",
  "telephone": "12333443",
  "birthdate": "2023-11-21T15:21:30.346Z",
  "service": "string"
}
```
- Media type:** A dropdown menu set to `application/json`.
- Execute:** A large blue button at the bottom.

Abbildung 5: Ansicht des Put Login Endpunktes.

5.1.3 Delete/delete-customer/{id}/{token}

Der DELETE Login API-Endpunkt ermöglicht das Löschen eines Json Web Tokens in unsere SQL-Datenbank:

1. Bei den Parameters Id und den Token angeben welche wir mit dem Post bekommen haben.
2. Auf «Execute» klicken um den Command auszuführen.

DELETE /delete-customer/{id}/{token}

Parameters

Name	Description
id * required integer(\$int32) (path)	3
token * required string (path)	Ecp9rXj-ij9iolkq9xWCw_K846GslptLTkw5Hn8

Execute

Abbildung 6: Ansicht des Delete Customer Endpunktes.

5.1.4 Get/api/Bestellungen

Der GET API-Endpunkt ermöglicht es, alle Aufträge abzurufen, die in meiner SQL-Datenbank gespeichert werden:

1. Auf «Execute» klicken um den Command auszuführen.

GET /api/Bestellungen

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7037/api/Bestellungen' \
  -H 'accept: text/plain'
```

Request URL

https://localhost:7037/api/Bestellungen

Server response

Code	Details
200	<p>Response body</p> <pre>{ "bestellungsId": 8, "name": "essssquerf", "emailadresse": "aqweaaadua@ASFD.CH", "telefonnummer": "0231423", "lieferdatum": "2024-11-16T00:00:00", "service": "RennskiService" }, { "bestellungsId": 9, "name": "esrquerf", "emailadresse": "aqweaaadua@ASFD.CH", "telefonnummer": "0231423", "lieferdatum": "2023-11-18T00:00:00", "service": "RennskiService" }]</pre>

Abbildung 7: Ansicht des Get Endpunktes.

5.1.5 Post/api/Bestellungen

Der POST API-Endpunkt ermöglicht das Hochladen eines Auftrags in meine SQL-Datenbank:

4. Im weissen Kasten die Persönlichen Daten angeben.
5. Auf «Execute» klicken um den Command auszuführen.

The screenshot shows a REST client interface for a POST request to the endpoint `/api/Bestellungen`. The interface is divided into several sections:

- Parameters:** A section with a "Cancel" button and a "Reset" button. It currently shows "No parameters".
- Request body:** A section with a dropdown menu set to `application/json`. Below it is a large text area containing a JSON object:


```
{
  "bestellungsid": 0,
  "name": "string",
  "emailadresse": "string",
  "telefonnummer": "string",
  "lieferdatum": "2023-11-16T09:09:56.811Z",
  "service": "string"
}
```
- Buttons:** At the bottom of the request body section are two buttons: "Execute" (highlighted in blue) and "Clear".
- Responses:** A section at the bottom, currently empty, with a "Curl" label and a dark background showing the equivalent curl command:

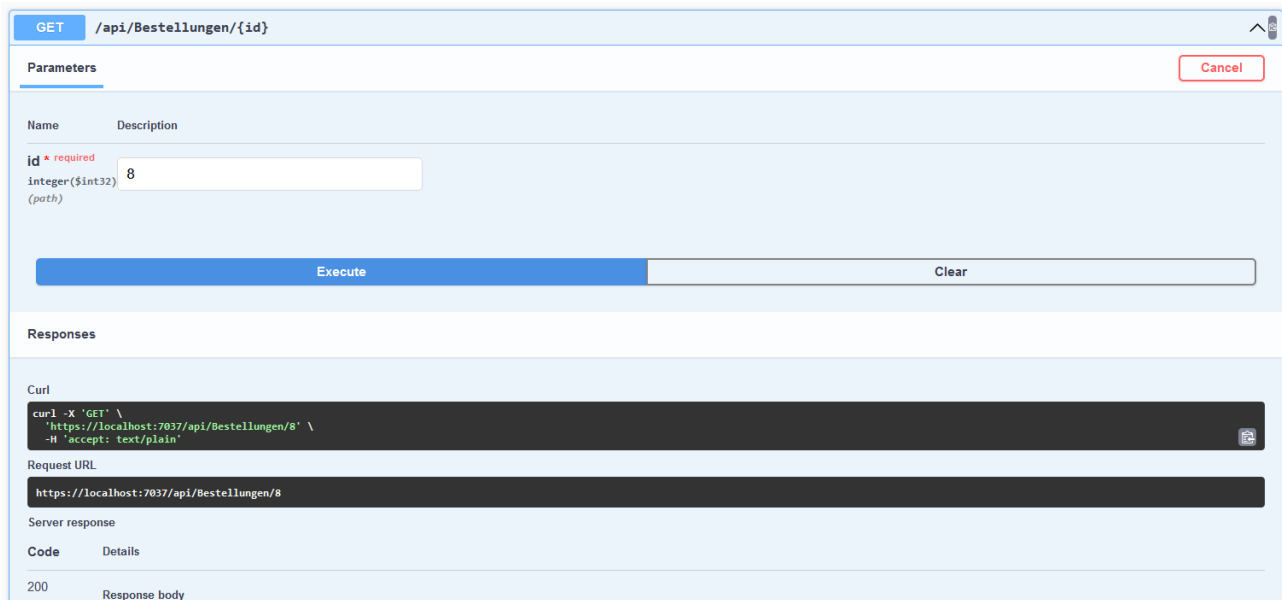

```
curl -X 'POST' \
  "https://localhost:7027/api/Bestellungen" \
  -H 'accept: text/plain' \
  -H 'Content-Type: application/json' \
```

Abbildung 8: Ansicht des Post Endpunktes.

5.1.6 Get/api/Bestellungen/{id}

Der GET API-Endpunkt nach Id ermöglicht es, einen Auftrag anhand seiner in meiner SQL-Datenbank gespeicherten Id abzurufen:

1. Beim Kasten wo «id required» steht, eine vorhandene id eingeben.
2. Auf «Execute» klicken um den Command auszuführen.



GET /api/Bestellungen/{id}

Parameters

Name	Description
id * required integer(\$int32) (path)	8

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'https://localhost:7037/api/Bestellungen/8' \
  -H 'accept: text/plain'
```

Request URL

```
https://localhost:7037/api/Bestellungen/8
```

Server response

Code	Details
200	Response body

Abbildung 9: Ansicht des Endpunkts zum Abrufen einer Bestellung anhand ihrer Id..

5.1.7 Put/api/Bestellungen/{id}

Der PUT API-Endpunkt nach Id ermöglicht das Aktualisieren eines Auftrags anhand seiner in meiner SQL-Datenbank gespeicherten Id:

1. Beim Kasten wo «id required» steht, eine vorhandene id in unserer Datenbank eingeben.
2. Im unteren Kasten die Daten welche geändert werden sollen eingeben.
3. Auf «Execute» klicken um den Command auszuführen.

The screenshot shows a web interface for the PUT API endpoint `/api/Bestellungen/{id}`. The interface has a light orange header with the method `PUT` and the endpoint path. Below the header, there are two buttons: `Cancel` (red) and `Reset` (grey). The main area is divided into two sections: `Parameters` and `Request body`. In the `Parameters` section, there is a table with two columns: `Name` and `Description`. The first row shows `id` with a red asterisk and the text `required`, followed by `integer($int32)` and `(path)`. The value `8` is entered in the input field. The `Request body` section has a dropdown menu set to `application/json` and a text area containing a JSON object:

```
{
  "bestellungsId": 8,
  "name": "essssquerf",
  "emailadresse": "aqueaaaadua0@ASFD.CH",
  "telefonnummer": "0231423",
  "lieferdatum": "2024-11-16T09:43:31.504Z",
  "service": "RennskiService"
}
```

. At the bottom, there are two buttons: `Execute` (blue) and `Clear` (grey).

Abbildung 10: Ansicht des Endpunkts zum Aktualisieren einer Bestellung anhand ihrer Id.

5.1.8 Delete/api/Bestellungen/{id}

Der DELETE API-Endpunkt nach Id ermöglicht das Löschen eines Auftrags anhand seiner in meiner SQL-Datenbank gespeicherten Id:

2. Beim Kasten wo «id required» steht, eine vorhandene id in unserer Datenbank eingeben.
3. Auf «Execute» klicken um den Command auszuführen.

The screenshot shows a web interface for the DELETE API endpoint `/api/Bestellungen/{id}`. The interface has a light red header with the method `DELETE` and the endpoint path. Below the header, there is a `Cancel` button (red). The main area is divided into two sections: `Parameters` and `Request body`. In the `Parameters` section, there is a table with two columns: `Name` and `Description`. The first row shows `id` with a red asterisk and the text `required`, followed by `integer($int32)` and `(path)`. The value `8` is entered in the input field. The `Request body` section is empty. At the bottom, there is a single button: `Execute` (blue).

Abbildung 11: Ansicht des Endpunkts zum Löschen einer Bestellung anhand ihrer Id.

5.2 Die SQL Datenbank

Hier können wir die Eigenschaften der Datenbank einsehen, in der die Aufträge in meiner Tabelle «Bestellungen» gespeichert werden:

```

1  -- Datenbank dropen falls es schon existiert
2  Use Master;
3  Drop Database if Exists BackendSkiService;
4
5  --Datenbank BackendSkiService erstellen
6  Create Database BackendSkiService;
7  Use BackendSkiService;
8
9  -- Tabelle Bestellungen
10 CREATE TABLE Bestellungen (
11     BestellungsID INT IDENTITY(1, 1) PRIMARY KEY,
12     Name VARCHAR(255) NOT NULL,
13     Emailadresse VARCHAR(255) NOT NULL,
14     Telefonnummer VARCHAR(20),
15     Lieferdatum DATE,
16     Service VARCHAR(50)
17 );
18
19 -- Um inhalt zu checken
20 Select * From Bestellungen;

```

105 %

Results Messages

	BestellungsID	Name	Emailadresse	Telefonnummer	Lieferdatum	Service
1	8	essssqwerf	aqweaaadwaD@ASFD.CH	0231423	2024-11-16	RennskiService
2	9	esrqwerf	aqweadwaD@ASFD.CH	0231423	2023-11-18	RennskiService
3	10	esrqwerf	aqweadwaD@ASFD.CH	0231423	2023-11-18	RennskiService
4	11	string	string	string	2023-11-14	string
5	12	string	string	string	2023-11-15	string
6	13	string	string	string	2023-11-15	string
7	14	string	string	string	2023-11-15	string

Abbildung 12: Ansicht meiner SQL Datenbank.

5.3 Die Datenbank in die Web-API implementieren

Mithilfe meiner «appsettings.json» Datei gebe ich meine Datenbank an und verbinde sie mit meinem Web-API.

```
{
  // Konfiguration der Protokollierung (Logging) für die Anwendung
  "Logging": {
    "LogLevel": {
      "Default": "Information", // Standard-Loglevel für die meisten Log-Nachrichten
      "Microsoft": "Warning", // Loglevel für Nachrichten aus Microsoft-Namespace
      "Microsoft.Hosting.Lifetime": "Information" // Loglevel für Lebenszyklus-Ereignisse der Hosting-Umgebung
    }
  },
  // Definiert, welche Hosts Zugriff auf die Anwendung haben. "*" erlaubt alle Hosts.
  "AllowedHosts": "*",
  // Verbindungszeichenfolgen für Datenbanken
  "ConnectionStrings": {
    // Verbindungszeichenfolge für die 'BackendSkiService'-Datenbank
    "BackendSkiService": "Server=azin22;Database=BackendSkiService;Trusted_Connection=True;"
  }
}
```

Abbildung 13: Ansicht meiner appsettings.json Datei.

6 Kontrollieren

6.1 Ist der Auftrag fachgerecht und auftragsgerecht ausgeführt?

Die Ausführung des Auftrags war im Grossen und Ganzen sehr in Ordnung. Die fachgerechte und auftragsgerechte Umsetzung war gut erkennbar, jedoch gäbe es kleine Verbesserungsmöglichkeiten. Die erbrachte Arbeit erfüllte die gestellten Anforderungen, obwohl es noch Raum zur Optimierung geben könnte.

7 Auswerten

7.1 Wie war es, was muss nächstes Mal besser gemacht werden?

7.1.1 Fazit

Im allgemeinen hat mir das Projekt sehr spass gemacht.

Ich musste viel Zeit und Aufwand einstecken und bin mit dem Ergebnis zufrieden.

Durch manche Fehler habe ich für die Zukünftige Projekte viel wichtiges erlernt.