

Webentwicklung

1. MVC Pattern (Model View Controller) → **visuell** Backend / Frontend ein Projekt
2. Web API → **datenbasiert**
 - Backend z.B. Flask oder Django mit Python
 - Anwender bekommt nur Daten (meist JSON-Format)

API

API (Application Programming Interface) → Reihe von Definitionen und Protokollen, die es ermöglichen mit einem externen Dienst zu interagieren.

A	Application	Software	<i>Google</i>
P	Programming	führt die Anforderung (in A) aus	<i>Google search</i>
I	Interface	Ort, wo P aufgefördert wird, zu laufen	<i>Eingabe Browser</i>



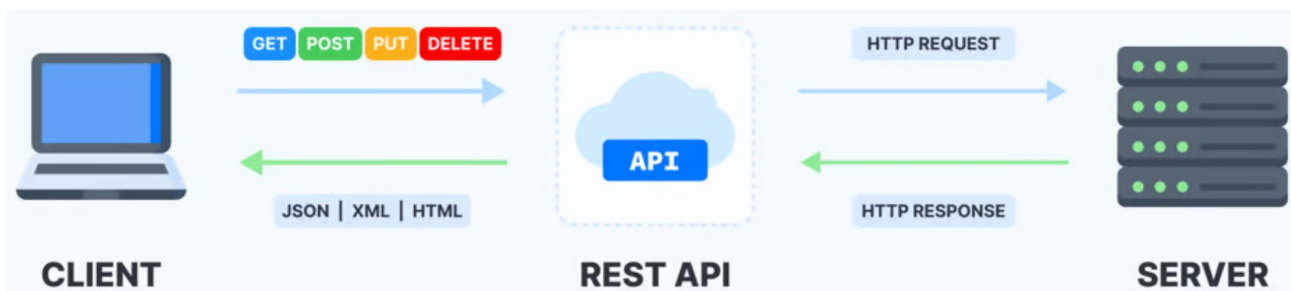
Mehrere API's: www.travelocity.com → API's aller Fluggesellschaften werden abgefragt

API – Typen

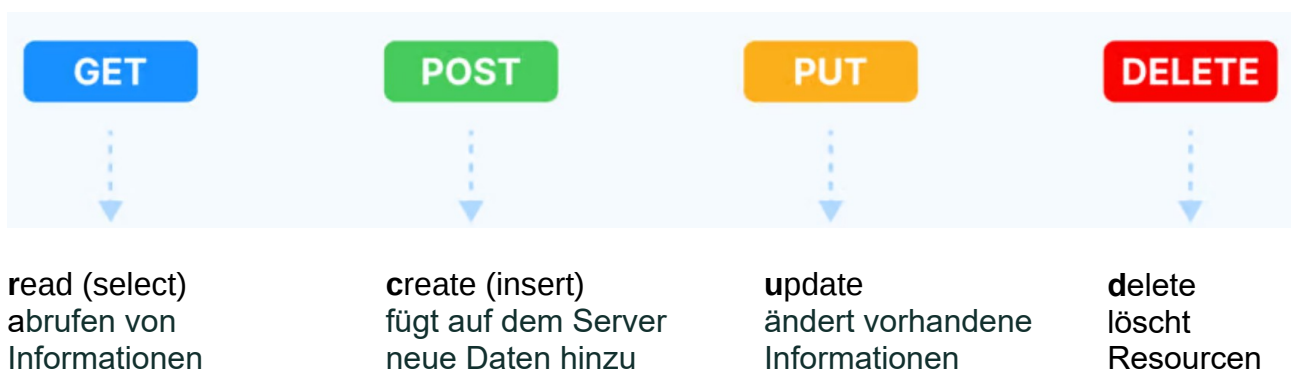
SOAP API	Datenaustausch über das XML – Format mit Hilfe verschiedener Protokolle (HTTP, SMTP...) → sicher
WebSocket API	permanente Verbindung zwischen Client und Server → schnell
GraphQL API	Abfragesprache zur spezifischen Datenübertragung zum Client
REST-API	(Representational State Transfer) → Datenaustausch zwischen Client und Server mit Hilfe der HTTP -Methoden über API-Endpunkte

Merkmale REST-API:

- Client und Server sind unabhängig voneinander
- es werden keine Daten vom Client auf der Serverseite gespeichert (zustandslos)
- Cachefähig: zwischenspeichern der Daten von/zur API
- Sitzungsstatus wird clientseitig gespeichert
- Client-Server-Entkopplung → Die einzige Information, die die Client-Anwendung kennen sollte, ist der URI der angeforderten Ressource.
- HTTP-Protokoll für die Kommunikation.
- Sehr gut skalierbar



HTTP Methoden



PATCH ändert, ohne zu ersetzen (<- -> PUT)

Der Endpunkt enthält einen URI – Uniform Resource Identifier, der angibt, wie und wo die Ressource gefunden werden kann. Eine URL oder Uniform Resource Location ist der häufigste URI-Typ, der eine vollständige Webadresse darstellt.

HTTP Aufbau

1. Startline:

	Request	Response
HTTP Version	HTTP/1.1	HTTP/1.1
Method	GET, POST, PUT, DELETE, etc.	-
API Program Folder *	/search	-
Parameters *	?p=REST	-
Status Code	-	z.B. 200 OK
Example	GET/search?p=REST HTTP/1.1	HTTP/1.1 200 OK

* = optional

2. Header:

GET /beispielseite.html HTTP/1.1	Startline
HOST: beispieldomain.tld	Header
Accept-Language: de	Der Server <i>beispieldomain.tld</i> soll <i>beispielseite.html</i> zurücksenden, möglichst in deutsch und optimiert für Safari / iPhone.
User-Agent: Mozilla/5.0 (iPhone; CPU iPhone OS 125_06_1 like Mac OS X) AppleWebKit/605.1.15 (KHTML, like Gecko) Version/125.06 Mobile/15E148 Safari/604.1	

[Header Fields](#) Anschauen (Firefox): rechte Maustaste → Untersuchen
→ Netzwerkanalyse → links HTTP-Anfrage rechts Header

3. Blankline:

trennt Header von Body

4. Body:

Inhalt: Webseite, Bild, Video, **JSON / XML** von / zur API

Beispiel: Online – Pizzabestellung

- Request:** Aufruf Web-Seite Methode: **GET** **Response:** Content der Seite
- Request:** Pizzabelegung Methode: **POST** **Response:** Bestätigung
- Request:** Belegung ändern Methode: **PUT** **Response:** Bestätigung
- Request:** Status abrufen Methode: **GET** **Response:** Status
- Request:** Bestellung stornieren Methode: **DELETE** **Response:** „Bestellung gelöscht“

Anforderung von Wetterdaten (ohne API-Key) über einen Python-Client mit Auswertung und Anzeige der JSON-Daten: *wettervorhersage.py*

HTTP-Eingabe-API über Python – Client an [emoncms](https://emoncms.org/) (CMS zur Darstellung von Messwerten)

Ordner/API-Endpunkt JSON-Objekt als Parameter Key zur Authentifizierung

```
1 from urllib.request import urlopen
2
3 urlopen('https://emoncms.org/input/post?node=Gewaechshaus_1&fulljson={"lum":400,"temp":12.22}&apikey=.....')
```

Inputs

Gewaechshaus_1

<input type="checkbox"/> lum	68 mins	400	
<input type="checkbox"/> temp	68 mins	12.22	

Input API Help

Erstellen einer REST-API mit Python Flask

Flask ist ein Mikroframework, mit dem Web-Anwendungen geschrieben und getestet werden.

```
1 from flask import Flask
2 #Erstellen einer Instanz von Flask
3 app = Flask(__name__)
4
5 #Route/ URL festlegen, Standardmethode:GET
6 @app.route("/")
7 def startseite():
8     return "Startseite"
9
10 #Route/ URL festlegen, Methode:GET
11 @app.get("/produkte")
12 def produktseite():
13     return "Produktseite"
14
15 #Route/ URL festlegen, Methoden:GET/POST
16 @app.route("/test", methods=['POST', 'GET'])
17 def produkt():
18     return "Hallo"
```

app.py

Die Funktion *startseite()* - wird ausgeführt, wenn die Basis-URL aufgerufen wird.

die Funktion *produkt()* reagiert sowohl auf GET- als auch POST-Anfragen.

Die Dekoratoren in den Zeilen 6, 11 und 16 verknüpfen bestimmte URLs (Routen) mit Funktionen, die beim Zugriff auf diese URLs ausgeführt werden.

Übergeben von Variablen über die URL

app2.py

```
5 @app.route('/home', methods=['POST', 'GET'], defaults={'name' : 'Werner'})
6 @app.route('/home/<string:name>', methods=['POST', 'GET'])
7 def home(name):
8     return '<h1>Hello {}, you are on the home page!</h1>'.format(name)
```



Hello Hans, you are on the home page!

Übergabe int-Wert:

```
6 @app.route('/home/<int:name>',
```

Übergabe von Daten über das request – Objekt

```
10 @app.route('/query')
11 def query():
12     name = request.args.get('name')
13     location = request.args.get('location')
14     return '<h1>Hi {}. You are from {}. You are on the query page!</h1>'.format(name, location)
```

name und *location* sind Parameter (Argumente) des request-Objekts

127.0.0.1:5000/query?name=Hans&location=Mannheim

Übergabe von Daten über ein HTML – Formular

app2.py

```
16 @app.route('/form', methods=['GET', 'POST'])
17 def form():
18     if request.method == 'GET':
19         return '''<form method="POST" action="/process">
20             <input type="text" name="name">Name <p>
21             <input type="text" name="location">Ort <p>
22             <input type="submit" value="Submit">
23         </form>'''
24     else:
25         name = request.form['name']
26         location = request.form['location']
27         return '<h1>Hello {}. You are from {} <h1>'.format(name, location)
28
29 @app.route('/process', methods=['POST'])
30 def process():
31     name = request.form['name']
32     location = request.form['location']
33     return '<h1>Hello {}. You are from {} <h1>'.format(name, location)
```

HTTP – Request mit Insomnia erstellen

The image shows a sequence of four screenshots from the Insomnia application, illustrating the steps to create an HTTP request. Red arrows connect the steps across the different screenshots.

1. HTTP Request erstellen
The first screenshot shows the top toolbar with a '+' icon. An arrow points to this icon, indicating the first step: creating a new HTTP request.

2. Namen vergeben mit Rename
The second screenshot shows the 'ACTIONS' menu. An arrow points to the 'Rename' option, indicating the second step: naming the request.

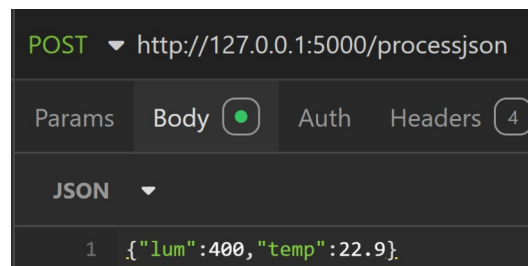
3. Endpunkt angeben: URL und HTTP-Methode
The third screenshot shows the request editor. The top bar displays 'POST' and the URL 'http://127.0.0.1:5000/form'. An arrow points to the URL field, indicating the third step: specifying the endpoint.

4. ev. Daten für den HTTP-Body eingeben
The fourth screenshot shows the 'Body' tab with 'Multipart' selected. A dropdown menu is open, showing options like 'Form Data', 'Form URL Encoded', 'GraphQL', 'JSON', and 'XML'. An arrow points to the 'Form Data' option, indicating the fourth step: specifying the data format for the request body.

Aus- und Eingabe von JSON

Die Flask-Funktion `jsonify()` wandelt eine Python-Datenstruktur in ein korrekt formatiertes JSON- Objekt um und setzt automatisch einen passenden HTTP-Header.

```
36 @app.route('/json')
37 def json():
38     return jsonify({"name": "Franz", "hobby": ["Python", "Java", "C++"]})
39
40
41 @app.route('/processjson', methods=['POST', 'GET'])
42 def processjson():
43     data = request.get_json()
44     temp = data['temp']
45     lum = data['lum']
46
47     return jsonify({'result' : 'Success!', 'temp' : temp, 'lum' : lum})
```



Simulation einer kleinen Shop-API: <https://insomnia.rest>

Endpunkt = Methodenpfadpaar (welche HTTP-Methode, welcher Pfad (auf dem Server))

Bei Aufruf der URL `/shop` wird die Datenstruktur zurück gegeben.

anfängliche Datenstruktur

```
20 @store_app.get("/shop")
21 def get_shop():
22     return {"shop": abteilung}
```

```
7 abteilung = [
8     {
9         "abteilung": "Möbel",
10        "items": [
11            {
12                "name": "Stuhl",
13                "preis": 49.90
14            }
15        ]
16    }
17 ]
```

GET – Anfrage:

GET ▼ `http://127.0.0.1:5000/shop`

Ausgabe Browser:

```
▼ shop:
  ▼ 0:
    abteilung:    "Möbel"
    ▼ items:
      ▼ 0:
        name:     "Stuhl"
        preis:    49.9
```

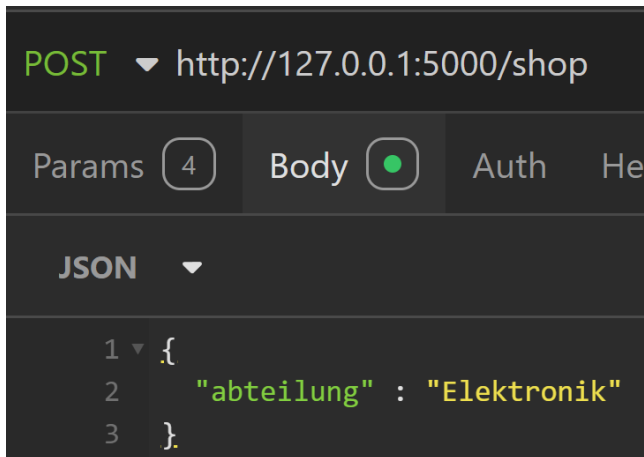
Eine neue Abteilung anlegen

Die Methode `create_store()` wird aufgerufen, wenn der Endpunkt `shop` mit HTTP POST aufgerufen wird.

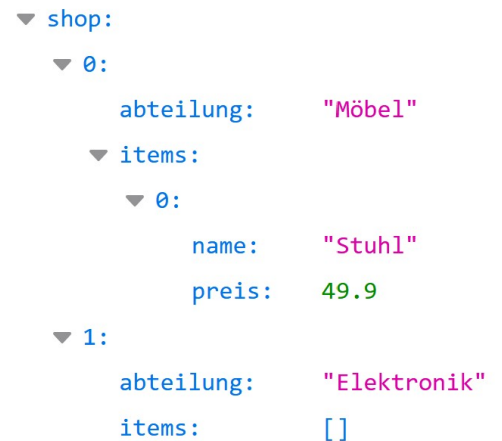
```
25 @store_app.post("/shop")
26 def create_store():
27     request_data = request.get_json()
28     abteilung_neu = {"abteilung": request_data["abteilung"], "items": []}
29     abteilung.append(abteilung_neu)
30     return abteilung_neu, 200
```

Die POST-Daten (Zugriff über `request`) müssen ein JSON-Objekt mit dem Feld `abteilung` enthalten. Es wird eine neue Abteilung ohne Artikel erstellt und der `abteilung`-Liste hinzugefügt.

POST – Anfrage:



Ausgabe Browser:



Einen neuen Artikel im Shop anlegen

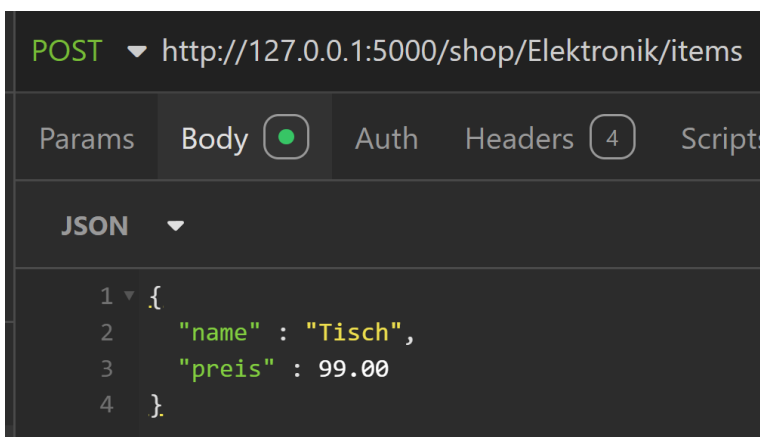
Die Methode `create_item()` wird aufgerufen, wenn der Endpunkt `shop/Parameter/items` mit HTTP POST aufgerufen wird:

```
43 #Neuer Artikel in Abteilung anlegen
44 @store_app.post("/shop/<string:name>/items") #http://127.0.0.1:5000/shop/Möbel/items
45 def create_item(name):
46     request_data = request.get_json()
47     for abt in abteilung:
48         if abt["abteilung"] == name:
49             new_item = {"name": request_data["name"], "preis": request_data["preis"]}
50             abt["items"].append(new_item)
51             return new_item, 201
52     return {"message": "departmenmt not found"}, 404
```

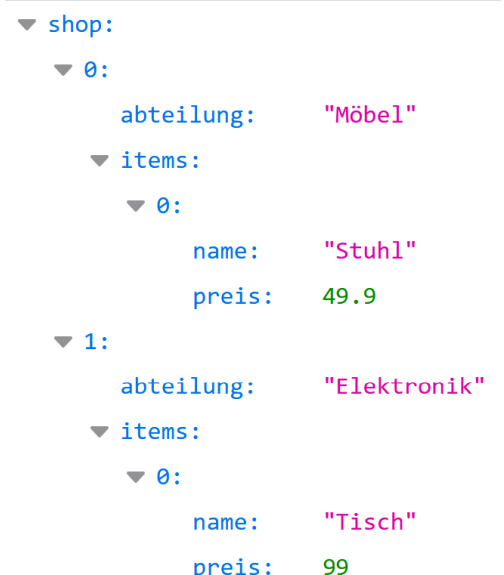
Die Funktion `create_item(name)` fügt einen neuen Artikel zu einer spezifischen Abteilung hinzu, die durch den URL-Pfad `name` angegeben wird.

Die POST-Daten müssen den Artikelnamen und den Preis enthalten. Wenn die Abteilung existiert, wird der Artikel zur entsprechenden Abteilung hinzugefügt.

POST – Anfrage:



Ausgabe Browser:



Artikel löschen

Die Methode `delete_item()` wird aufgerufen, wenn der Endpunkt `shop/ <abteilung>/items` mit HTTP DELETE aufgerufen wird:

```
33 @store_app.delete("/shop/<string:abteilung_name>/items")
34 def delete_item(abteilung_name):
```

DELETE ▼ http://127.0.0.1:5000/shop/Elektronik/items?item=Tisch

Send ▼

URL und Parameter:

- **<string:abteilung_name>**: Name der Abteilung, aus der ein Artikel gelöscht werden soll.
- **Query-Parameter item**: Der Artikelname wird über einen Query-Parameter (? item=Artikelname) übergeben.

Artikelpreis aktualisieren

Hier wird eine neue Route hinzugefügt, die es ermöglicht, den Preis eines Artikels in einer bestimmten Abteilung zu ändern.

```
66 @store_app.put("/shop/<string:abteilung_name>/items/<string:item_name>")
67 def update_item_price(abteilung_name, item_name):
```

PUT ▼ http://127.0.0.1:5000/shop/Möbel/items/Stuhl

Params

Body

Auth

Headers

4

Scripts

JSON ▼

```
1 {
2   "preis": 59.90
3 }
```

Datenbankeinträge über HTML-Formular

app_db.py

```
1 # Importiert die Flask-Bibliothek für Webanwendungen und die MySQL-Connector-Bibliothek zur Datenbankbindung.
2 from flask import Flask, render_template, request
3 import sys, mysql.connector
4
5 # Erstellt eine Flask-App-Instanz.
6 app = Flask(__name__)
7
8 # Definiert die Startseite der Anwendung.
9 @app.route('/')
10 def hello():
11     # Lädt und zeigt die HTML-Datei "index.html" an.
12     return render_template("index.html")
13
14 # Definiert die Route für die Registrierung ("signUp"), die sowohl POST- als auch GET-Methoden unterstützt.
15 @app.route('/signUp', methods=['POST', 'GET'])
16 def signUp():
17     # Liest Eingabewerte aus dem HTTP-Request-Formular (POST-Methoden).
18     vorname = request.form['firstname']
19     nachname = request.form['lastname']
20     mail = request.form['mail']
21     msg = request.form['message']
22
23     # Werte, die über URL-Parameter (GET) übergeben werden
24     # vorname = request.args.get('firstname')
25     # nachname = request.args.get('lastname')
26     # mail = request.args.get('mail')
27     # msg = request.args.get('message')
28
29     try:
30         # Versucht, eine Verbindung zur MySQL-Datenbank herzustellen.
31         connection = mysql.connector.connect(
32             host="localhost",
33             user="root",
34             passwd=""
35         )
36     except:
37         # Gibt eine Fehlermeldung aus, falls die Verbindung fehlschlägt.
38         print("Keine Verbindung zum Server")
39         return "Fehler: Verbindung zur Datenbank konnte nicht hergestellt werden."
40
41     # Erstellt einen Cursor zum Ausführen von SQL-Befehlen.
42     cursor = connection.cursor()
43
44     # Wählt die Datenbank "user".
45     cursor.execute("USE user")
46     connection.commit()
47
48     # Führt einen INSERT-Befehl aus, um die Benutzerdaten in die Tabelle "tbl_user" einzufügen.
49     cursor.execute("INSERT INTO tbl_user VALUES(NULL,%s,%s,%s,%s);", [vorname,nachname,mail,msg])
50     connection.commit()
51
52     # Zeigt die HTML-Seite "index2.html" an und übergibt den Nachnamen des Nutzers als Variable.
53     return render_template("index2.html", nachname=nachname)
```

Übergabe HTML – Formular
an Python

Übergabe Python an
Datenbank

Bei Datenbankoperationen gilt für eine Rest API bei der Benutzung der HTTP-Methoden:

GET	Anzeigen von Daten aus der Datenbank (SELECT)
POST	Einfügen von Daten / Erstellen neuer Tabellen (INSERT / CREATE)
PUT	Aktualisieren von Daten (UPDATE)
DELETE	Löschen von Daten (DELETE)

Die HTTP-Methoden **PUT/DELETE** sind normalerweise nicht direkt im Browser nutzbar,

da Browser standardmäßig GET-Anfragen unterstützen, und POST-Anfragen häufig nur durch Formulare ausgelöst werden. Es gibt jedoch mehrere Möglichkeiten, um z.B. eine **PUT**-Anfrage zu senden:

- Postman oder Insomnia
- Verwendung von curl
- Verwendung von JavaScript im Browser (z. B. über die Konsole)
- Verwendung von Erweiterungen im Browser RESTer (Firefox) oder Talend API Tester (Chrome)
- Das Standardverhalten HTML-Formulars (GET oder POST) mit JavaScript ändern
 - siehe *formular2.html / app_db_2.py*

HTML-Formular

index.html

```
1 <html>
2 <head>
3     <!-- Der Titel der Webseite wird in der Registerkarte des Browsers angezeigt. -->
4     <title>HTML Formular</title>
5
6     <!-- Interne CSS-Styles für das Layout der Seite. -->
7     <style>
8         /* Stellt sicher, dass die Labels eine Mindestbreite haben und inline angezeigt werden. */
9         label {
10             min-width: 100px; /* Mindestbreite für Labels, um eine saubere Ausrichtung zu gewährleisten. */
11             display: inline-block; /* Labels werden neben den Eingabefeldern positioniert. */
12         }
13     </style>
14 </head>
15 <body>
16     <!-- Beginnt das HTML-Formular. -->
17     <form method="post" action="/signUp">
18         <!-- Methode "post": Daten werden im Body des HTTP-Requests gesendet.
19         Aktion "/signUp": Daten werden an die definierte Route im Flask-Backend gesendet. -->
20
21         <!-- Eingabefeld für den Vornamen des Benutzers -->
22         <label for="firstname">Vorname</label>
23         <input type="text" id="firstname" name="firstname"/>
24         <br/><br/>
25
26         <!-- Eingabefeld für den Nachnamen des Benutzers -->
27         <label for="lastname">Nachname</label>
28         <input type="text" id="lastname" name="lastname"/>
29         <br/><br/>
30
31         <!-- Eingabefeld für die E-Mail-Adresse des Benutzers -->
32         <label for="mail">E-Mail Adresse</label>
33         <input type="email" id="mail" name="mail"/>
34         <br/><br/>
35
36         <!-- Textbereich für eine optionale Nachricht -->
37         <textarea rows="4" cols="16" name="message"></textarea>
38         <!-- rows: Die Anzahl der Zeilen im Textfeld.
39         cols: Die Anzahl der Zeichen pro Zeile. -->
40         <br/><br/>
41
42         <!-- Schaltfläche zum Absenden des Formulars -->
43         <input type="submit" value="Absenden"/>
44         <!-- Beim Klicken werden die Formulardaten an die angegebene Aktion gesendet. -->
45     </form>
46 </body>
47 </html>
```

Vorname

Nachname

E-Mail Adresse

Nachricht

_____ = Übergabe HTML an Python

Attribut for:

- Der Wert for="firstname" verweist auf das id-Attribut des zugehörigen Eingabefelds.
- Diese Verknüpfung hat zwei Vorteile:
 1. **Barrierefreiheit:** Screenreader erkennen die Beziehung zwischen Label und Eingabefeld und können dem Benutzer vorlesen, was er ausfüllen soll.
 2. **Benutzerfreundlichkeit:** Ein Klick auf das Label fokussiert automatisch das verknüpfte Eingabefeld.

Attribut id:

- Das id-Attribut hat den Wert firstname.
- Dieses id muss eindeutig sein und wird verwendet, um das Eingabefeld zu identifizieren (z. B. bei CSS, JavaScript oder in der Verknüpfung mit einem Label).
- **Attribut name:**
- Der Wert firstname wird als Schlüssel verwendet, wenn das Formular gesendet wird.
- Beim Senden des Formulars wird der eingegebene Wert mit dem Schlüssel firstname an das Backend übermittelt. Im Python-Backend kann dieser Wert mit `request.form['firstname']` ausgelesen werden.