wie füge ich eine Datenbanken mit Flask-SQLAlchemy in mein Projekt ein. ich habe Barrierefrei Restaurants in Berlin aufgelistet und möchte dann auch eine Funktion das man weitere hinzufügen kann

Du brauchst im Prinzip drei Bausteine: Datenbank anbinden, ein Modell für deine Restaurants definieren und eine Route + Formular zum Anlegen neuer Einträge.

1. Flask-SQLAlchemy einrichten

```
# app.py
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///restaurants.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

db = SQLAlchemy(app)
```

Damit legst du eine SQLite-Datenbank `restaurants.db` in deinem Projektverzeichnis an.[1] [2]

2. Modell für barrierefreie Restaurants

```
class Restaurant(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(120), nullable=False)
    adresse = db.Column(db.String(255), nullable=False)
    stadtteil = db.Column(db.String(80), nullable=False)
    barrierefrei = db.Column(db.Boolean, default=True)
    bemerkung = db.Column(db.Text)
```

Einmalig beim Start (z.B. in einer Python-Shell) Tabellen anlegen:

```
from app import app, db
with app.app_context():
    db.create_all()
```

3. Liste der Restaurants anzeigen

```
from flask import render_template

@app.route("/")
def index():
    restaurants = Restaurant.query.order_by(Restaurant.name).all()
    return render_template("index.html", restaurants=restaurants)
```

`templates/index.html` (sehr einfaches Beispiel):

```
<!doctype html>
<html>
  <body>
    <h1>Barrierefreie Restaurants in Berlin</h1>
    <a href="{{ url_for('add_restaurant') }}">Neues Restaurant hinzufügen</a>
    <ul>
      {% for r in restaurants %}
        <li>
          <strong>{{ r.name }}</strong>,
          {{ r.adresse }} ({{ r.stadtteil }})
          – {% if r.barrierefrei %}barrierefrei{% else %}nicht barrierefrei{% endif %}
          {% if r.bemerkung %}- {{ r.bemerkung }}{% endif %}
        </li>
      {% endfor %}
    </ul>
  </body>
</html>
```

4. Formular zum Hinzufügen neuer Restaurants

Variante ohne WTForms, nur mit `request.form`:

```
from flask import request, redirect, url_for

@app.route("/add", methods=["GET", "POST"])
def add_restaurant():
    if request.method == "POST":
        name = request.form["name"]
        adresse = request.form["adresse"]
        stadtteil = request.form["stadtteil"]
        barrierefrei = "barrierefrei" in request.form
        bemerkung = request.form.get("bemerkung", "")

        r = Restaurant(
            name=name,
            adresse=adresse,
            stadtteil=stadtteil,
            barrierefrei=barrierefrei,
            bemerkung=bemerkung,
        )
        db.session.add(r)
        db.session.commit()
```

```
        return redirect(url_for("index"))

    return render_template("add_restaurant.html")
```

`templates/add_restaurant.html`:

```
<!doctype html>
<html>
  <body>
    <h1>Neues Restaurant hinzufügen</h1>
    <form method="post">
      <label>Name:</label>
      <input type="text" name="name" required><br>

      <label>Adresse:</label>
      <input type="text" name="adresse" required><br>

      <label>Stadtteil:</label>
      <input type="text" name="stadtteil" required><br>

      <label>Barrierefrei:</label>
      <input type="checkbox" name="barrierefrei" checked><br>

      <label>Bemerkung:</label>
      <textarea name="bemerkung"></textarea><br>

      <button type="submit">Speichern</button>
    </form>
  </body>
</html>
```

5. Nächste Schritte

- Wenn du Validierung und Fehlermeldungen möchtest, kannst du Flask-WTF/WTForms nutzen.[3] [4]
- Für ein größeres Projekt lohnt sich eine Struktur mit `app.py`, `models.py`, `forms.py`, `templates/`.[1] [5]

Wenn du mir kurz zeigst, wie dein aktuelles Projekt (Verzeichnisse, Dateien) aussieht, kann ich dir den Code genau passend umbauen.

⁂

meine Routen stehen schon ich müsste jetzt nur das sql einrichten

Du musst im Wesentlichen nur drei Dinge ergänzen: Erweiterung installieren, in deiner `app` konfigurieren und deine Models + `db.create_all()` einbauen.

1. Flask-SQLAlchemy installieren

Im Terminal im Projektordner:

```
pip install flask-sqlalchemy
```

2. In deiner Flask-App konfigurieren

In der Datei, in der du dein `app = Flask(__name__)` hast (oft `app.py` oder `__init__.py`):

```
from flask import Flask
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)

# SQLite-Datei im Projektordner
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///restaurants.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

db = SQLAlchemy(app)
```

Wenn du ein Factory-Pattern nutzt (`def create_app()`), sieht es so aus:

```
db = SQLAlchemy()

def create_app():
    app = Flask(__name__)
    app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///restaurants.db"
    app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

    db.init_app(app)
    return app
```

3. Dein Restaurant-Model anlegen

Zum Beispiel in `models.py` oder direkt in `app.py`:

```
from yourpackage import db  # ggf. anpassen

class Restaurant(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(120), nullable=False)
    adresse = db.Column(db.String(255), nullable=False)
    stadtteil = db.Column(db.String(80), nullable=False)
    barrierefrei = db.Column(db.Boolean, default=True)
    bemerkung = db.Column(db.Text)
```

4. Tabellen einmalig erstellen

Ganz wichtig: einmal vor dem ersten Start ausführen (z.B. unten in `app.py` oder in einem kleinen Skript):

```
if __name__ == "__main__":
    with app.app_context():
        db.create_all()  # erzeugt restaurants.db + Tabellen
    app.run(debug=True)
```

Ab dann kannst du in deinen bestehenden Routen einfach `Restaurant.query...`, `db.session.add(...)`, `db.session.commit()` verwenden, ohne an den Routen selbst viel zu ändern. [11] [12] [13]

Wenn du magst, poste kurz deine `app.py` (oder die Datei mit `app = Flask(__name__)`), dann passe ich dir den Code exakt für deine Struktur an.

⁂

import click

from datetime import datetime
from flask_sqlalchemy import SQLAlchemy
from werkzeug.security import generate_password_hash

db = SQLAlchemy() ist das richtig

`db = SQLAlchemy()` ist als erster Schritt richtig – entscheidend ist, **wo** und **wie** du das machst.

1. Minimal richtiges Setup

In einer zentralen Datei (z.B. `extensions.py` oder `models.py`):

```
from flask_sqlalchemy import SQLAlchemy

db = SQLAlchemy()
```

In deiner App-Datei (wo `app = Flask(__name__)` steht):

```
from flask import Flask
from .extensions import db  # oder von dort, wo du db definiert hast

app = Flask(__name__)
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///restaurants.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False

db.init_app(app)
```

Damit ist `db` mit deiner Flask-App verbunden.

2. Models verwenden

In `models.py`:

```
from .extensions import db

class Restaurant(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(120), nullable=False)
    # weitere Felder ...
```

3. Tabellen einmalig erzeugen

Zum Beispiel in `app.py`:

```
if __name__ == "__main__":
    with app.app_context():
        db.create_all()
    app.run(debug=True)
```

Wenn du mir kurz zeigst, wie deine Dateien strukturiert sind (welcher Code in welcher Datei), kann ich dir das exakt passend einbauen.

⁂

Restaurant hinzufügen (nur wenn "eingeloggt")

def _to_bool(name: str) -> bool:
return request.form.get(name) == "on"

@app.route("/restaurants/new", methods=["GET", "POST"])
def restaurant_new():
if not session.get("logged_in"):
flash("Bitte einloggen, um ein Restaurant einzureichen.", "warning")
return redirect(url_for("login", next=request.path))

```
if request.method == "POST":
    name = (request.form.get("name") or "").strip()
    if not name:
        flash("Bitte gib einen Namen für das Restaurant an.", "danger")
        return render_template("new.html")

    # Website auslesen + optional normalisieren
    website = (request.form.get("website") or "").strip() or None
    if website and not website.startswith(("http://", "https://")):
        website = "https://" + website

    r = Restaurant(
        name=name,
```

```
        website=website,
        strasse=(request.form.get("strasse") or "").strip() or None,
        hausnummer=(request.form.get("hausnummer") or "").strip() or None,
        postleitzahl=(request.form.get("postleitzahl") or "").strip() or None,
        stadt=(request.form.get("stadt") or "").strip() or None,
        beschreibung=(request.form.get("beschreibung") or "").strip() or None,
        status="pending",
        erstellt_von_nutzer_id=session["user_id"],
    )

    r.website = website

    # optional: Koordinaten
    bg = (request.form.get("breitengrad") or "").strip()
    lg = (request.form.get("laengengrad") or "").strip()
    try:
        r.breitengrad = float(bg) if bg else None
        r.laengengrad = float(lg) if lg else None
    except ValueError:
        flash("Bitte gültige Koordinaten eingeben (z.B. 52.5200 und 13.4050).", "danger")
        return render_template("new.html")

    r.merkmale = BarrierefreieMerkmale(
        stufenloser_eingang=_to_bool("stufenloser_eingang"),
        rampe=_to_bool("rampe"),
        barrierefreies_wc=_to_bool("barrierefreies_wc"),
        breite_tueren=_to_bool("breite_tueren"),
        unterfahrbare_tische=_to_bool("unterfahrbare_tische"),
        behindertenparkplatz=_to_bool("behindertenparkplatz"),
    )

    db.session.add(r)
    db.session.flush()

    file = request.files.get("titelbild")
    if file and file.filename:
        if not allowed_file(file.filename):
            flash("Nur Bilddateien (png, jpg, jpeg, webp) sind erlaubt.", "danger")
            return redirect(request.url)

        filename = secure_filename(file.filename)
        upload_dir = app.config["UPLOAD_FOLDER"]
        os.makedirs(upload_dir, exist_ok=True)

        save_path = os.path.join(upload_dir, filename)
        if os.path.exists(save_path):
            filename = f"{int(datetime.utcnow().timestamp())}_{filename}"
            save_path = os.path.join(upload_dir, filename)

        file.save(save_path)

        rel_path = f"static/uploads/{filename}"
        r.fotos = [Foto(dateipfad=rel_path, titelbild=True)]

    db.session.commit()
```

```
    flash("Restaurant wurde gespeichert", "success")
    return redirect(url_for("detail", id=r.id))

return render_template("new.html")
```

das ist meine Route zu hinzufügen eines Restaurants hast du Verbesserungsvorschläge ?

Die Route ist schon ziemlich solide, du nutzt sauber PRG-Pattern (POST → Redirect) und validierst einige Felder. Ich würde vor allem bei Struktur, Wiederverwendung und Fehlerfällen etwas nachschärfen.[31] [32]

1. Validierung und Fehlerausgabe verbessern

Statt bei Fehlern nur `render_template("new.html")` zu machen, kannst du die bisher eingegebenen Werte wieder ins Formular geben, damit der Nutzer nicht alles neu tippen muss.[33]

```
form_data = request.form.to_dict()

# bei Fehler:
return render_template("new.html", form=form_data)
```

Im Template dann `value="{{ form.name }}"` etc.[33]

2. Doppelte Logik vermeiden

`website` setzt du zweimal:

```
website = (request.form.get("website") or "").strip() or None
if website and not website.startswith(("http://", "https://")):
    website = "https://" + website

r = Restaurant(
    name=name,
    website=website,
    ...
)

# das hier ist überflüssig:
r.website = website
```

Die letzte Zuweisung kannst du löschen, sie ändert nichts mehr.[34]

3. Upload-Logik etwas härten

Du machst schon `secure_filename` und einen einfachen Kollisionsschutz, das ist gut. Typische Ergänzungen:[35] [36] [37]

- Größe begrenzen mit `app.config["MAX_CONTENT_LENGTH"]`.[37]

- Eindeutigere Dateinamen (z.B. UUID) statt ursprünglichem Namen, um Kollisionen sicher zu vermeiden.

Beispiel:

```
import uuid

filename = secure_filename(file.filename)
ext = filename.rsplit(".", 1)[-1].lower()
filename = f"{uuid.uuid4().hex}.{ext}"
```

4. Beziehung für `BarrierefreieMerkmale` und `Foto`

Stell sicher, dass deine Models die Beziehungen passend definieren, z.B. One-to-One für `Merkmale` und One-to-Many für `Foto`.[38] [39]

```
class Restaurant(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    # ...
    merkmale = db.relationship("BarrierefreieMerkmale", backref="restaurant", uselist=Fal
    fotos = db.relationship("Foto", backref="restaurant", lazy=True)

class BarrierefreieMerkmale(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    restaurant_id = db.Column(db.Integer, db.ForeignKey("restaurant.id"))
    # ...

class Foto(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    restaurant_id = db.Column(db.Integer, db.ForeignKey("restaurant.id"))
    dateipfad = db.Column(db.String(255))
    titelbild = db.Column(db.Boolean, default=False)
```

Dann funktioniert `r.merkmale = BarrierefreieMerkmale(...)` und `r.fotos = [Foto(...)]` wie gedacht.[38] [39]

5. Kleine Lesbarkeits-Refactorings

- Die Umwandlung der Koordinaten kannst du in eine kleine Hilfsfunktion auslagern, dann bleibt die Route übersichtlicher.[34]
- `_to_bool` ist gut, ggf. umbenennen in etwas sprechenderes wie `get_checkbox`.
- Du könntest einen Service/Helper nutzen, der aus `request.form` direkt ein `Restaurant`-Objekt baut, um die Route schlanker zu machen.[34]

Wenn du möchtest, kann ich dir im nächsten Schritt den kompletten Routen-Code in eine leicht refaktorierte Version schreiben (gleiche Funktionalität, nur aufgeräumter) und dazu passende Model-Definitionen skizzieren.

⁂

Restaurant-Liste nach Hinzufügen anzeigen

Um nach dem Hinzufügen direkt die Restaurant-Liste zu sehen, brauchst du:

1. Eine Route, die alle Restaurants listet.
2. In deiner `restaurant_new`-Route nach `db.session.commit()` ein `redirect` genau auf diese Liste (das machst du im Prinzip schon mit `detail`, du änderst nur das Ziel).

1. Route für die Liste

Beispiel:

```
@app.route("/restaurants")
def restaurant_list():
    restaurants = Restaurant.query.order_by(Restaurant.name).all()
    return render_template("list.html", restaurants=restaurants)
```

In `list.html` zeigst du dann die Restaurants an.

2. Nach dem Speichern zu dieser Liste umleiten

In deiner bestehenden Route ersetzt du:

```
flash("Restaurant wurde gespeichert", "success")
return redirect(url_for("detail", id=r.id))
```

durch

```
flash("Restaurant wurde gespeichert", "success")
return redirect(url_for("restaurant_list"))
```

Wenn du beides haben willst (erst Detailansicht, dann Liste), kannst du z.B. in der Detailseite einen gut sichtbaren Button „Zur Übersicht“ mit Link auf `restaurant_list` einbauen.[42]

⁂

kannst du mir das genauer erklären

Du hast schon fast alles eingebaut – es geht jetzt nur darum, die „Liste“ als eigene Seite zu haben und nach dem Speichern genau dorthin zu springen. Ich zerlege dir das Schritt für Schritt.

1. Was passiert beim Hinzufügen?

In deiner `restaurant_new`-Route machst du am Ende:

```
db.session.commit()

flash("Restaurant wurde gespeichert", "success")
return redirect(url_for("detail", id=r.id))
```

Bedeutung:

- `db.session.commit()` speichert das neue Restaurant in der Datenbank.
- `flash(...)` legt eine Nachricht in der Session ab, die du im Template anzeigen kannst.
- `redirect(url_for("detail", id=r.id))` sagt: „Schick den Browser auf die Route mit dem Namen `detail` und übergib `id=r.id` als URL-Parameter.“

Das nennt man „Post-Redirect-Get“: nach einem POST-Request gibt es keinen direkten HTML-Antwort-Body, sondern eine Weiterleitung. So wird ein doppeltes Absenden beim Neuladen verhindert.

2. Route für die Restaurant-Liste

Damit du nach dem Hinzufügen eine Liste sehen kannst, brauchst du eine Route, die alle Restaurants aus der Datenbank holt und in ein Template steckt.

Beispiel:

```
@app.route("/restaurants")
def restaurant_list():
    restaurants = Restaurant.query.order_by(Restaurant.name).all()
    return render_template("list.html", restaurants=restaurants)
```

- `Restaurant.query.order_by(...).all()` holt alle Restaurants.
- `render_template("list.html", restaurants=restaurants)` rendern das Template und übergibt die Liste als Variable `restaurants`.

Ein mögliches `list.html`:

```
<!doctype html>
<html>
  <body>
    {% with messages = get_flashed_messages(with_categories=true) %}
      {% if messages %}
        <ul>
          {% for category, message in messages %}
            <li class="{{ category }}">{{ message }}</li>
          {% endfor %}
        </ul>
      {% endif %}
```

```
    {% endwith %}

    <h1>Barrierefreie Restaurants</h1>

    <a href="{{ url_for('restaurant_new') }}">Neues Restaurant hinzufügen</a>

    <ul>
      {% for r in restaurants %}
        <li>
          <strong>{{ r.name }}</strong>
          {% if r.stadt %}- {{ r.stadt }}{% endif %}
          – <a href="{{ url_for('detail', id=r.id) }}">Details</a>
        </li>
      {% endfor %}
    </ul>
  </body>
</html>
```

Wichtig: Hier siehst du auch, wie die `flash`-Meldung oben eingeblendet wird.

3. Nach dem Speichern auf die Liste umleiten

Jetzt änderst du in deiner `restaurant_new`-Route nur die letzte Zeile:

Statt Detailansicht:

```
flash("Restaurant wurde gespeichert", "success")
return redirect(url_for("detail", id=r.id))
```

Weiterleitung auf die Liste:

```
flash("Restaurant wurde gespeichert", "success")
return redirect(url_for("restaurant_list"))
```

Damit passiert folgendes:

1. User füllt Formular aus und klickt „Speichern“.
2. Server erstellt `Restaurant`-Objekt, speichert es mit `db.session.commit()`.
3. Server schickt HTTP-Redirect zur URL von `restaurant_list`.
4. Browser ruft diese URL auf, `restaurant_list` holt alle Restaurants inklusive dem neuen und zeigt sie in `list.html` an.
5. Die Flash-Nachricht wird ganz oben angezeigt.

4. Kombi-Variante: erst Detail, dann Liste

Falls du lieber direkt die Detailansicht sehen willst, aber trotzdem eine Liste anbieten willst:

- Lass die Weiterleitung auf `detail` wie bisher.
- Baue in dein Detail-Template einen Button/Link:

```
<a href="{{ url_for('restaurant_list') }}">Zur Restaurant-Übersicht</a>
```

Dann ist der Ablauf:

- Speichern → Detailseite des neuen Restaurants
- Von dort kann man mit einem Klick zur Gesamtliste.

Wenn du mir sagst, wie deine aktuelle Startseite aussieht (welche Route du als „Hauptliste“ nutzt), kann ich dir exakt sagen, auf welche Funktion du aus `restaurant_new` am besten redirectest.

1. https://www.youtube.com/watch?v=nC7vQn0jZXw
2. https://flask-sqlalchemy.readthedocs.io/en/stable/quickstart/
3. https://www.geeksforgeeks.org/python/flask-wtf-explained-how-to-use-it/
4. https://riptutorial.com/flask/example/31761/a-simple-form
5. https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application
6. https://stackoverflow.com/questions/65566823/flask-sqlalchemy-quickstart
7. https://stackoverflow.com/questions/60750880/how-do-i-import-a-database-made-with-sqlalchemy-to-flask-app
8. https://muneebdev.com/flask-sqlalchemy-quickstart/
9. https://www.youtube.com/watch?v=j5lQl4aW9ZU
10. https://www.youtube.com/watch?v=aH647Ex5lgA
11. https://www.geeksforgeeks.org/python/connect-flask-to-a-database-with-flask-sqlalchemy/
12. https://flask-sqlalchemy.readthedocs.io/en/stable/quickstart/
13. https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application
14. https://www.reddit.com/r/flask/comments/11k3wm7/dbcreate_all/
15. https://flask.palletsprojects.com/en/stable/patterns/sqlalchemy/
16. https://www.pythontutorials.net/blog/confusion-about-uri-path-to-configure-sqlite-database/
17. https://flask-sqlalchemy.readthedocs.io/en/stable/contexts/
18. https://stackoverflow.com/questions/56416437/confusion-about-uri-path-to-configure-sqlite-database
19. https://stackoverflow.com/questions/73961938/flask-sqlalchemy-db-create-all-raises-runtimeerror-working-outside-of-applicat
20. https://flask.palletsprojects.com/en/stable/patterns/sqlite3/

21. https://stackoverflow.com/questions/78466962/flask-sqlalchemy-how-to-properly-call-the-app-to-initialize-sqlalchemy-databa
22. https://www.pythontutorials.net/flask/using-sqlalchemy-with-flask-an-orm-guide/
23. https://flask.palletsprojects.com/en/stable/patterns/sqlalchemy/
24. https://github.com/swagger-api/swagger-codegen/issues/7307
25. https://flask-sqlalchemy.readthedocs.io/en/stable/quickstart/
26. https://stackoverflow.com/questions/43459182/proper-sqlalchemy-use-in-flask
27. https://www.reddit.com/r/flask/comments/14fq5lv/use_sqlalchemy_without_flasksqlalchemy_or/
28. https://github.com/jaimin-bariya/Flask-SQLAlchemy-Guide
29. https://www.digitalocean.com/community/tutorials/how-to-use-flask-sqlalchemy-to-interact-with-databases-in-a-flask-application
30. https://www.reddit.com/r/flask/comments/ccenkb/flasksqlalchemy_best_practices_sessions_create/
31. https://www.reddit.com/r/flask/comments/gjfzdc/redirect_vs_render_template/
32. https://stackoverflow.com/questions/21668481/difference-between-render-template-and-redirect
33. https://www.reddit.com/r/flask/comments/rsnkqt/how_do_i_render_template_and_redirect_to_login/
34. https://flask.palletsprojects.com/en/stable/patterns/sqlalchemy/
35. https://stackoverflow.com/questions/59887362/flask-do-i-need-to-use-secure-filename-on-uploads-to-s3-google-cloud/59888836
36. https://www.codingeasypeasy.com/blog/flask-file-uploads-a-comprehensive-guide-with-code-examples
37. https://flask.palletsprojects.com/en/stable/patterns/fileuploads/
38. https://dev.to/freddiemazzilli/flask-sqlalchemy-relationships-exploring-relationship-associations-igo
39. https://github.com/w8s/flask-model-relationships
40. https://www.youtube.com/watch?v=dPM2Ndy9ID8
41. https://www.hackerone.com/blog/secure-file-uploads-flask-filtering-and-validation-techniques
42. https://www.perplexity.ai/search/b2aa1167-7829-4b60-9241-bc8b58d6e9e5
43. https://www.dish.co/DE/de/blog/restaurant-eintragen-google-maps/
44. https://www.speisekartex.de
45. https://www.tripadvisor.com/business/de-de/insights/manage-tripadvisor-restaurant-listing-guide
46. https://www.speisekarte.de/restaurantsuche
47. https://www.reddit.com/r/webdev/comments/1g2yvmf/best_way_to_retrieve_all_restaurants_and_cafes_in/
48. https://www.reddit.com/r/FoodNYC/comments/10x9lq5/is_there_a_site_where_you_can_make_a_list_of/
49. https://www.opentable.de/nearby
50. https://www.resmio.com/spoon-bytes/suchmaschinenwerbung-fuer-restaurants/
51. https://www.speisekarte.de
52. https://www.reddit.com/r/GoogleMaps/comments/10kwrjb/is_there_a_way_to_view_all_restaurants_in_the/