



wie kann man bei VSC eine adminfunktion in das projekt miteinbauen? was sind wichtige punkte die ich beachten muss und auf jeden fall drin haben muss damit es funktioniert?

Du baust die Adminfunktion nicht in **VS Code** ein, sondern in dein Projekt (Backend/Frontend-Code), VS Code ist nur der Editor. Ich zeige dir die wichtigsten Bausteine, damit ein Adminbereich sauber und sicher funktioniert.

1. Rollen und Rechte planen

Bevor du Code schreibst, brauchst du ein Konzept für Rollen und Berechtigungen (Role-Based Access Control, RBAC).^[1] ^[2]

- Definiere Rollen, z.B. `user`, `admin`, evtl. `editor`.^[3] ^[2]
- Lege fest, welche Aktionen jede Rolle darf: z.B. Admin: Nutzer verwalten, Einstellungen ändern, alle Daten sehen.^[2] ^[1]
- Erstelle dir eine kleine „Role-Permission-Matrix“ (Tabelle), damit du im Code genau weißt, was erlaubt ist.^[2]

Beispiel-Matrix

Rolle	Darf...
Admin	Nutzer anlegen/löschen, alles sehen ^[2]
Editor	Inhalte bearbeiten, ansehen ^[2]
User	Eigene Daten ansehen, Basisfunktionen ^[2]

2. Authentifizierung (Login) einbauen

Ohne Login kein sinnvoller Adminbereich, weil du nicht weißt, wer der Nutzer ist.

- Implementiere ein Login-System (z.B. mit JWT oder Session-Cookies), das nach erfolgreichem Login eine User-ID und die Rolle liefert.^[4] ^[3]
- Speichere im Token/Session mindestens: `id`, `email`, `role`.^[4] ^[3]
- Stelle sicher, dass jede Anfrage an geschützte Routen zuerst prüft, ob der User eingeloggt ist.^[4] ^[2]

Beispiel (Express + JWT, stark vereinfacht):

```
// Middleware: prüft, ob der Nutzer eingeloggt ist
function authenticate(req, res, next) {
  const token = req.headers['authorization']?.split(' ')[1];
  if (!token) return res.status(401).json({ error: 'Nicht eingeloggt' });

  jwt.verify(token, process.env.JWT_SECRET, (err, decoded) => {
    if (err) return res.status(403).json({ error: 'Token ungültig' });
    req.user = decoded; // enthält z.B. { id, email, role }
    next();
  });
}
```

3. Autorisierung (Adminrechte prüfen)

Jetzt kommt die eigentliche Adminfunktion: nur bestimmte Rollen dürfen bestimmte Routen/Seiten sehen.

- Schreibe eine Middleware, die prüft, ob `req.user.role` in einer Liste erlaubter Rollen ist. [\[5\]](#) [\[2\]](#) [\[4\]](#)
- Hänge diese Middleware an alle Admin-Routen (z.B. `/admin/users`, `/admin/settings`). [\[2\]](#) [\[4\]](#)

Beispiel:

```
// Middleware: prüft Rollen
function authorize(roles = []) {
  return (req, res, next) => {
    if (!req.user) return res.status(401).json({ error: 'Nicht eingeloggt' });
    if (roles.length && !roles.includes(req.user.role)) {
      return res.status(403).json({ error: 'Keine Berechtigung' });
    }
    next();
  };
}

// Nutzung
app.get('/admin/users',
  authenticate,
  authorize(['admin']),
  (req, res) => {
    // Code nur für Admins
  }
);
```

Wichtige Punkte:

- Immer zuerst **authenticate**, dann **authorize** aufrufen. [\[2\]](#) [\[4\]](#)
- Niemals nur im Frontend verstecken (z.B. Button ausblenden), Backend muss immer prüfen. [\[1\]](#) [\[2\]](#)

4. Admin-UI sauber trennen

Für Übersichtlichkeit lohnt sich eine Trennung des Adminbereichs im Code.

- Lege eigene Routen/Namensräume an, z.B. `/admin/...` oder ein eigenes „Admin-Layout“ im Frontend. ^[6] ^[7]
- Halte Admin-Komponenten (z.B. „UserManagement“, „Settings“) in eigenen Ordnern, damit dein Projekt wartbar bleibt. ^[8] ^[6]

Beispielstruktur:

- `routes/admin/users.js` (Backend)
- `src/admin/pages/UsersPage.tsx` (Frontend)

^[7] ^[6]

5. Sicherheits-Basics, die du beachten musst

Damit dein Adminbereich wirklich sicher ist:

- **Least Privilege:** Gib jeder Rolle nur genau die Rechte, die sie braucht, nicht mehr. ^[9] ^[1]
- Monitoring/Logging: Logge Admin-Aktionen (z.B. Nutzer gelöscht, Rolle geändert), um Probleme nachverfolgen zu können. ^[1]
- Regelmäßige Rechte-Reviews: Prüfe regelmäßig, ob Rollen noch sinnvoll sind und ob jemand zu viele Rechte hat. ^[9] ^[1]
- Fehlercodes korrekt nutzen: 401 (nicht eingeloggt), 403 (eingeloggt, aber keine Berechtigung). ^[4] ^[2]

6. Was du konkret „auf jeden Fall drin haben“ solltest

Kurzcheckliste:

- Login mit sicherer Passwortspeicherung und einem Token/Session-System. ^[3] ^[4]
- Usermodell mit Feld `role` (z.B. `'user' | 'admin'`). ^[3] ^[5]
- Middleware `authenticate` zum Prüfen des Logins. ^[3] ^[4]
- Middleware `authorize(['admin'])` für Admin-Routen. ^[2] ^[4]
- Eigener Routenbereich für Admin (z.B. `/admin/*`) plus passende UI. ^[6] ^[7]
- Logging wichtiger Admin-Aktionen und Anwendung des Least-Privilege-Prinzips. ^[9] ^[1]

Wenn du mir sagst, mit welchem Stack du arbeitest (z.B. Node/Express, Django, Laravel, React, Vue usw.), kann ich dir ein kleines, konkretes Codebeispiel für genau dein Setup schreiben.

ich schreiben in python macht das ein unterschied?

Ja, es macht einen Unterschied – aber eher im „Wie“, nicht im „Was“.

Die wichtigen Konzepte (Login, Rollen, Rechte prüfen, Admin-Routen schützen) bleiben gleich, egal ob du in Python, JavaScript oder etwas anderem programmierst. In Python änderst du nur die konkrete Umsetzung, z.B.:

- Web-Framework:
 - Django: hat eingebautes Auth-System, User-Model, Gruppen/Rollen und ein fertiges Admin-Interface.
 - Flask/FastAPI: du baust Auth und Rollenprüfungen selbst (oder mit Extensions).
- Typische Umsetzung in Python:
 - User hat ein Feld wie `is_admin` oder `role` in der Datenbank.
 - Nach dem Login speicherst du die User-ID und Rolle in der Session oder in einem Token.
 - Du schreibst einen Decorator/Middleware, der prüft: „Ist eingeloggt?“ und „Hat Rolle admin?“.

Beispiel (stark vereinfacht, Flask-Style):

```
from functools import wraps
from flask import session, redirect, abort

def login_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        if "user_id" not in session:
            return redirect("/login")
        return f(*args, **kwargs)
    return wrapper

def admin_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        if "user_role" not in session or session["user_role"] != "admin":
            abort(403)
        return f(*args, **kwargs)
    return wrapper

@app.route("/admin")
@login_required
@admin_required
def admin_dashboard():
    return "Adminbereich"
```

Wichtig ist also:

- Saubere Authentifizierung (Login mit sicherer Passwortspeicherung).
- Eine Rolle/Flag beim User, die Admin kennzeichnet.

- Einen zentralen Check (Decorator/Middleware), den du auf alle Admin-Routen legst.
- Niemals nur im Frontend „verstecken“, sondern immer im Backend prüfen.

Wenn du mir sagst, ob du z.B. Django, Flask oder FastAPI benutzt (oder etwas anderes), kann ich dir ein fertiges, kleines Beispiel genau für dein Setup schreiben.

✱

es soll nur 2 admins geben wie kann ich diese festlegen? anhand der email oder woran genau und gebe ein beispiel wie ich es umsetzen könnte

Du kannst die 2 Admins am einfachsten **anhand der E-Mail-Adresse** oder eines Rollenfelds (`role = "admin"`) festlegen – beides ist okay. Für „genau 2 Admins“ ist E-Mail oft am übersichtlichsten, weil du die Liste hart im Code oder in der DB definierst. ^[21] ^[22]

Variante 1: Admins über E-Mail-Liste festlegen

Du definierst eine feste Liste von Admin-E-Mails und prüfst bei jeder Anfrage, ob der eingeloggte User darin ist.

```
# z.B. config.py
ADMIN_EMAILS = {
    "admin1@example.com",
    "admin2@example.com",
}
```

```
# helpers.py
from functools import wraps
from flask import session, abort
from config import ADMIN_EMAILS

def admin_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        email = session.get("email")
        if email is None:
            # nicht eingeloggt
            abort(401)
        if email not in ADMIN_EMAILS:
            # eingeloggt, aber kein Admin
            abort(403)
        return f(*args, **kwargs)
    return wrapper
```

```
# routes.py
from flask import Flask
from helpers import admin_required
```

```

app = Flask(__name__)

@app.route("/admin")
@admin_required
def admin_dashboard():
    return "Adminbereich nur für die 2 Admins"

```

Wichtig:

- Beim Login speicherst du `session["email"] = user.email`. [\[23\]](#) [\[21\]](#)
- Nur die E-Mails in `ADMIN_EMAILS` haben Zugriff auf alle Admin-Routen. [\[22\]](#) [\[21\]](#)
- Wenn du später einen Admin ändern willst, änderst du einfach die Liste.

Variante 2: Adminflag im Usermodell + Begrenzung auf 2

Statt E-Mail-Liste kannst du im Usermodell ein Feld `is_admin` oder `role` nutzen und dafür sorgen, dass nur 2 User dieses Flag haben. [\[24\]](#) [\[22\]](#)

Beispiel Usermodell (vereinfacht):

```

class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    email = db.Column(db.String(120), unique=True, nullable=False)
    password_hash = db.Column(db.String(128), nullable=False)
    is_admin = db.Column(db.Boolean, default=False)

```

Beim Hochstufen zum Admin prüfst du:

```

def make_admin(user: User):
    admin_count = User.query.filter_by(is_admin=True).count()
    if admin_count >= 2:
        raise ValueError("Es dürfen nur 2 Admins existieren")

    user.is_admin = True
    db.session.commit()

```

Decorator:

```

from functools import wraps
from flask import session, abort

def admin_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        user_id = session.get("user_id")
        if user_id is None:
            abort(401)
        user = User.query.get(user_id)
        if not user or not user.is_admin:
            abort(403)
    return wrapper

```

```
    return f(*args, **kwargs)
return wrapper
```

Welche Variante für dich?

- Nutze E-Mail-Liste, wenn:
 - das ein kleines Projekt ist,
 - die Admins sich selten ändern,
 - du schnell eine Lösung willst. [\[21\]](#) [\[22\]](#)
- Nutze `is_admin/role`, wenn:
 - du eine Datenbank hast,
 - du später evtl. mehr Rollen willst,
 - du Adminrechte im UI ändern möchtest. [\[24\]](#) [\[23\]](#)

Wenn du mir sagst, ob du Flask, Django oder etwas anderes verwendest, passe ich dir das Beispiel 1:1 an dein Framework an.

✱

kannst du mir youtubevideos empfehlen die das erklären

Hier sind die besten YouTube-Videos, die genau dein Thema (Python/Flask Admin-Berechtigungen) erklären:

Top-Empfehlungen (Flask + Admin-Rechte)

1. "Python Flask Tutorial #10: Restrict access to admin panel"

<https://www.youtube.com/watch?v=3rr3pGX7OsY> [\[31\]](#)

Perfekt für dich! Zeigt Schritt-für-Schritt:

- Admin-Panel mit Flask-Admin erstellen
- Nur bestimmte User (wie deine 2 Admins) reinlassen
- `is_accessible()` Methode für Rollenprüfung
- User/Rollen-Model erstellen

Dauer: ~20min, super praxisnah

2. "Basic Admin Page - Flask Fridays #34"

<https://www.youtube.com/watch?v=WqHtmz8lbn8> [\[32\]](#)

Zeigt:

- Einfache Admin-Seite bauen
- Nur Admins sehen lassen (`if current_user.is_admin`)

- Admin-Link verstecken für normale User

Ideal für Anfänger, 8 Minuten

3. "Flask python tutorial flask-admin roles and permission"

<https://www.youtube.com/watch?v=bjclAKuRiJw>^[33]

Tiefergehend:

- Verschiedene Rollen (CEO, Secretary, Programmer)
- `is_admin` Feld in DB
- Komplette Flask-Admin mit Berechtigungen

Sehr detailliert

Deutsche Videos

4. "Python Flask Tutorial deutsch [Kompletter Einsteigerkurs]"

<https://www.youtube.com/watch?v=eTLM5SgEEhA>^[34]

Auf Deutsch! Flask REST-API Kurs, behandelt auch Authentifizierung.

Lernpfad (empfohlen)

1. **Zuerst Video #1** (Flask Tutorial #10) – das ist EXAKT dein Use-Case
2. **Dann Video #2** – für die Basics
3. **Optional Video #3** – wenn du mehr Rollen willst

Tipp: Das erste Video (#10) der Serie schauen (https://www.youtube.com/watch?v=8BB3UK_pQy8) gibt dir den Admin-Panel-Kontext für Video #10.^[35]

Alle Videos zeigen genau den Code, den du brauchst: Decorator, `is_admin` Check, Session-Prüfung – perfekt für deine "nur 2 Admins per E-Mail" Lösung!

✱

dankeeee

Gerne! Viel Erfolg mit deinem Admin-Bereich! 😊

ich will role anwenden und habe eine lokale datenbank. aber irgendwas funktioniert nicht. reicht es wenn ich die rolen nur bei mir festlege oder muss ich das auf dem server meiner partnerin auch festlegen. ich hab es im code eingefügt und es funktioniert nur bei mir

Das ist dein Problem!

Rollen liegen NICHT im Code, sondern in der Datenbank. Wenn du die Rollen nur bei dir lokal festgelegt hast, haben sie deine Partnerin nicht – jeder braucht seine **eigene** Datenbank mit den richtigen Rollen-Einträgen.

Das Problem genau erklärt

```
Du (lokal):
├─ app.py (Code: ✓ hat Rollen-Check)
├─ database.db (✓ hat deine Admin-Rolle gespeichert)
└─ funktioniert

Partnerin:
├─ app.py (Code: ✓ hat Rollen-Check)
└─ database.db (✗ hat KEINE Admin-Rollen gespeichert!)
    → Fehler 403!
```

Lösung 1: Datenbank synchronisieren (schnellste Lösung)

Option A: DB-Datei kopieren

```
Deine DB: database.db → an Partnerin schicken
Partnerin: database.db ersetzen → fertig!
```

Option B: SQL-Einfüge-Statements

```
-- Für deine Partnerin in ihrer DB ausführen:
UPDATE users SET role = 'admin' WHERE email = 'ihre-email@example.com';
-- oder
INSERT INTO roles_users (user_id, role_id) VALUES (2, 1); -- user_id und role_id anpassen
```

Lösung 2: Automatische Admin-Erstellung (sauberste Lösung)

```
# In app.py, beim App-Start
def ensure_admins_exist():
    with app.app_context():
        # Deine Email
        user1 = User.query.filter_by(email="deine-email@example.com").first()
```

```

    if user1:
        user1.role = "admin"

    # Email deiner Partnerin
    user2 = User.query.filter_by(email="partnerin-email@example.com").first()
    if user2:
        user2.role = "admin"

    db.session.commit()

# Aufruf beim Start
ensure_admins_exist()

```

Lösung 3: Admin-Emails im Code festlegen (einfachste Lösung)

```

# config.py
ADMIN_EMAILS = {
    "deine-email@example.com",
    "partnerin-email@example.com"
}

# Decorator nutzt das statt DB:
def admin_required(f):
    @wraps(f)
    def wrapper(*args, **kwargs):
        email = session.get("email")
        if email not in ADMIN_EMAILS:
            abort(403)
        return f(*args, **kwargs)
    return wrapper

```

Sofort-Test für Partnerin

1. Öffne ihre Datenbank (z.B. mit DB Browser for SQLite)
2. Schaue in Tabelle `users`
3. Prüfe ihre Email-Zeile: Hat sie `role = 'admin'`?
4. Falls nein → per SQL fixen:

```
UPDATE users SET role = 'admin' WHERE email = 'ihre-email@...';
```

Empfehlung

Für **Entwicklung mit Partnerin: Lösung 3** (Email-Liste im Code) – keine DB-Probleme mehr!

1. `ADMIN_EMAILS = {"deine@email.com", "partnerin@email.com"}`
2. Partnerin braucht nur Login mit ihrer Email
3. ✓ Funktioniert sofort überall

Soll ich dir den kompletten Code für Lösung 3 zeigen? Oder erstmal die DB deiner Partnerin fixen?

✱

perfekt danke ich versuch es mal

Gerne! Viel Erfolg – melde dich wenn's hakt! ☺

wo liegt der fehler ? {# Bewertung löschen (nur Admins) #}

```
{% if bewertungen %}
<div class="row g-3">
  {% for b in bewertungen %}
    <div class="col-md-6">
      <div class="card h-100">
        <div class="card-body d-flex flex-column h-100">

          {{ b.nutzer.benutzername if b.nutzer else "Anonym" }}

          {{ "★" * b.sterne }}{{ "☆" * (5 - b.sterne) }}

          {{ b.erstellt_am.strftime("%d.%m.%Y") }}

          {{ b.text or "" }}

        </div>
      </div>
    </div>
  {% endfor %}
</div>
{% endif %}
```

```

</div>
    </div>
</div>
{% endfor %}
</div>
{% else %}
    <div class="text-muted">Es gibt noch keine Bewertungen.</div>
{% endif %}

```

Der Fehler ist hier:

```
{% if session.get("role") == "admin" %}
```

Flask/Jinja2 kann `session.get()` NICHT direkt im Template ausführen! `session` ist eine Flask-spezifische Funktion, die **nur im Python-Code** funktioniert.

3 Lösungen (einfach bis sauber)

Lösung 1: Im Python-View die Rolle mitgeben (SCHNELLST!)

```

# In deiner View-Funktion (reviews.html rendern)
@app.route('/reviews')
def reviews():
    bewertungen = Bewertung.query.all()

    # ← ← ← DAS FEHLT!
    is_admin = session.get('role') == 'admin'

    return render_template('reviews.html',
                           bewertungen=bewertungen,
                           is_admin=is_admin) # ← HIER mitgeben!

```

Template dann:

```

{% if is_admin %}
    <form method="post" action="{{ url_for('review_delete', id=b.id) }}" ...>
        <button class="btn btn-sm btn-outline-danger">Löschen</button>
    </form>
{% endif %}

```

Lösung 2: Session als global verfügbar machen (sauber)

```

# In app.py ganz oben
@app.context_processor
def inject_session():
    return dict(session=session,
                is_admin=session.get('role') == 'admin')

```

Template bleibt fast gleich:

```
{% if is_admin %}  
    <!-- Löschen Button -->  
{% endif %}
```

Lösung 3: Custom Template-Filter (profi)

```
# In app.py  
@app.template_filter('is_admin')  
def is_admin_filter():  
    return session.get('role') == 'admin'
```

Template:

```
{% if is_admin() %}  
    <!-- Löschen Button -->  
{% endif %}
```

▮ Sofort-Lösung (kopier das!)

1. View anpassen:

```
@app.route('/reviews')  
def reviews():  
    bewertungen = Bewertung.query.all()  
    return render_template('reviews.html',  
                           bewertungen=bewertungen,  
                           is_admin=session.get('role') == 'admin')
```

2. Template ändern:

```
{% if is_admin %}  
<form method="post" action="{{ url_for('review_delete', id=b.id) }}" ...>  
    <button class="btn btn-sm btn-outline-danger">Löschen</button>  
</form>  
{% endif %}
```

→ **Funktioniert in 30 Sekunden!** ▮

Test: `print(session.get('role'))` in der View – siehst du "admin"? Dann ist die Session korrekt, nur das Template war falsch.



funktioniert immer noch nicht

Debugging - wir finden den Fehler JETZT!

1. DEBUG: Was steht wirklich in der Session?

In deiner **VIEW-Funktion** (wo `render_template('reviews.html'...` steht), füge **DIESE ZEILEN** ein:

```
@app.route('/reviews')
def reviews():
    bewertungen = Bewertung.query.all()

    # 1 DEBUGGING - WICHTIG!
    print("1 SESSION INHALT:", dict(session))
    print("1 ROLE:", session.get('role'))
    print("1 EMAIL:", session.get('email'))

    is_admin = session.get('role') == 'admin'
    print("1 IS_ADMIN:", is_admin)

    return render_template('reviews.html',
                           bewertungen=bewertungen,
                           is_admin=is_admin)
```

Starte die App neu, gehe zu `/reviews` und schau in die **KONSOLE**.
Was steht da? Kopiere mir die Ausgabe!

2. MÖGLICHE Ursachen + FIXES

Problem A: Session ist leer

```
1 SESSION INHALT: {}
1 ROLE: None
```

→ **Du bist nicht eingeloggt!** Login neu machen.

Problem B: Falscher Session-Key

```
1 SESSION INHALT: {'user_id': 1, 'email': 'test@example.com'}
1 ROLE: None ← FEHLT!
```

→ **Rolle wurde beim Login nicht gespeichert!** Login-Funktion fixen:

```
@login.route('/login', methods=['POST'])
def login():
    # ... login logik ...
```

```
session['role'] = user.role # ← DAS FEHLT!  
return redirect(url_for('reviews'))
```

Problem C: is_admin ist False

```
▮ ROLE: user  
▮ IS_ADMIN: False
```

→ **Rolle ist nicht "admin"!** DB fixen:

```
UPDATE users SET role = 'admin' WHERE email = 'deine-email';
```

▮ 3. Template-Debug (zusätzlich)

Ändere dein Template TÄTIG:

```
<!-- DEBUG INFO (löschen später) -->  
<p>DEBUG: Role={{ session.role if session.role else "KEINE" }} | is_admin={{ is_admin }}</p>  
  
{% if is_admin %}  
    ✓ Admin-Button sichtbar  
{% else %}  
    ✗ Kein Admin  
{% endif %}
```

▮ SOFORT-TEST:

1. Obige Debug-Prints in View
2. Template mit Debug-Info
3. Seite laden
4. **KONSOLE-Ausgabe + HTML-Ausgabe** hier posten

```
Beispiel erwartete Ausgabe:  
▮ SESSION INHALT: {'role': 'admin', 'email': 'test@example.com'}  
▮ IS_ADMIN: True  
DEBUG: Role=admin | is_admin=True  
✓ Admin-Button sichtbar
```

Sag mir exakt, was du siehst – dann weiß ich in 10 Sekunden, wo der Fehler liegt! ▮

✱

ich habe eine funktion das nutzer restaurant hinzufügen können. der admin hat dann die funktion das restaurant zu bearbeiten: müssen alle daten bearbeitbar sein oder reicht name und adresse, Koordinaten

Für Admin-Bearbeitung reicht meist Name + Adresse + Koordinaten.

▮ Kern-Daten (Admin MUSS bearbeiten können)

- ✓ Name ← wichtigste Info
- ✓ Adresse ← Kunden brauchen das
- ✓ Koordinaten ← Karte/Map-Integration

Das sind die 3 absoluten Must-Haves.

△ Optionale Felder (je nach Projekt)

Feld	Wann bearbeiten?	Warum wichtig?
Telefon	Wenn Kunden anrufen sollen	Direktkontakt
Öffnungszeiten	Bei Reservierungen/Bestellungen	Verfügbarkeit
Beschreibung	Marketing	Kunden-Entscheidung
Bilder	Visuelle Apps	Attraktivität
Bewertung	Automatisch berechnet	Nicht manuell!
Preisklasse	Optional	Filter

▮ Empfohlenes Admin-Formular

```
<!-- Admin Edit Restaurant -->
<form method="post">
  <div class="mb-3">
    <label>Name *</label>
    <input name="name" value="{{ restaurant.name }}" required>
  </div>

  <div class="mb-3">
    <label>Adresse *</label>
    <input name="adresse" value="{{ restaurant.adresse }}" required>
  </div>

  <div class="row">
    <div class="col-md-6">
      <label>Breitengrad *</label>
      <input name="lat" value="{{ restaurant.lat }}" step="any" required>
    </div>
```



```

    <div class="col-md-6">
      <label>Längengrad *</label>
      <input name="lng" value="{{ restaurant.lng }}" step="any" required>
    </div>
  </div>

  <!-- Optional: -->
  <div class="mb-3">
    <label>Telefon</label>
    <input name="telefon" value="{{ restaurant.telefon }}">
  </div>

  <button type="submit" class="btn btn-primary">Speichern</button>
</form>

```

▮ NIE bearbeitbar machen:

- id (Primary Key)
- erstellt_am (Audit)
- bewertung (automatisch berechnen)
- nutzer_id (wer hat es erstellt)

▮ Profi-Tipp: Validierung

```

# Backend: Immer prüfen!
if not name or not adresse or not (lat and lng):
    flash("Name, Adresse und Koordinaten sind Pflicht!")
    return redirect(url_for('admin_edit_restaurant', id=id))

```

Fazit: Name + Adresse + Koordinaten = 95% der Admin-Fälle abgedeckt. Rest je nach Feature-Set deines Projekts.

✱

1. <https://blog.pixelfreestudio.com/how-to-use-role-based-access-control-for-web-security/>
2. <https://webpeak.org/blog/role-based-access-control-in-web-applications/>
3. <https://supertokens.com/blog/user-authentication-in-node>
4. <https://coreui.io/answers/how-to-implement-role-based-auth-in-nodejs/>
5. <https://stackoverflow.com/questions/58067672/how-to-implement-role-based-authentication-with-no-dejs>
6. <https://www.youtube.com/watch?v=u3R7hUEYW54>
7. https://www.reddit.com/r/reactjs/comments/1gp8exv/separated_admin_and_casual_user_frontends_wit_h/
8. <https://dev.to/progrium/turning-vs-code-into-a-product-framework-28hg>
9. <https://www.skillfuel.com/configuring-role-based-access-in-ats-checklist/>
10. https://www.youtube.com/watch?v=HHuiV841g_w

11. <https://www.gutefrage.net/frage/wie-gibt-man-python-admin-rechte>
12. https://www.reddit.com/r/learnpython/comments/1agr0mk/how_do_i_download_python_on_computers_without/
13. https://www.reddit.com/r/learnpython/comments/13gx6a7/how_to_make_python_script_run_as_administrator/?tl=de
14. <https://www.python-forum.de/viewtopic.php?t=48894>
15. <https://www.python-forum.de/viewtopic.php?t=55355>
16. https://developer.mozilla.org/de/docs/Learn_web_development/Extensions/Server-side/Django/Authentication
17. https://www.reddit.com/r/learnpython/comments/18d05sf/cant_use_admin_privileges_when_installing_pyexe/
18. https://www.reddit.com/r/learnpython/comments/apjvnp/how_to_provide_admin_rights_when_executing_a/
19. <https://labex.io/de/tutorials/nmap-how-to-resolve-python-server-permissions-418757>
20. <https://www.drwindows.de/xf/threads/berechtigungsprobleme-trotz-administrator.167968/>
21. <https://blog.teclado.com/learn-python-defining-user-access-roles-in-flask/>
22. <https://stackoverflow.com/questions/78190441/how-can-i-restrict-access-in-fields-in-web-application-with-flask>
23. <https://www.geeksforgeeks.org/python/flask-role-based-access-control/>
24. <https://www.permit.io/blog/implement-role-based-access-control-in-flask>
25. <https://flask-user.readthedocs.io/en/latest/authorization.html>
26. https://books.agiliq.com/projects/django-admin-cookbook/en/latest/specific_users.html
27. <https://stackoverflow.com/questions/63549476/how-to-implement-role-based-access-control-in-flask>
28. <https://www.youtube.com/watch?v=YobFICatF-s>
29. <https://www.youtube.com/watch?v=3rr3pGX7OsY>
30. https://www.reddit.com/r/django/comments/mo19nw/restrict_access_to_views_based_on_users_role_and/
31. <https://www.youtube.com/watch?v=3rr3pGX7OsY>
32. <https://www.youtube.com/watch?v=WqHtmz8Ibn8>
33. <https://www.youtube.com/watch?v=bjclAKuRiJw>
34. <https://www.youtube.com/watch?v=eTLM5SgEEhA>
35. https://www.youtube.com/watch?v=8BB3UK_pQy8
36. https://www.reddit.com/r/flask/comments/q8spq6/how_to_restrict_access_to_admin_panel_in_flask/
37. <https://www.youtube.com/watch?v=hqhc2Wxq3W4>
38. https://www.youtube.com/watch?v=1j3k-_DqobU
39. <https://www.youtube.com/watch?v=t9zA1gvrTvo>
40. <https://www.youtube.com/watch?v=0cySORlhkCg>
41. <https://stackoverflow.com/questions/63549476/how-to-implement-role-based-access-control-in-flask>
42. https://www.reddit.com/r/flask/comments/da45lt/writing_my_own_role_based_access_control/
43. <https://www.permit.io/blog/implement-role-based-access-control-in-flask>

44. <https://app.studyraid.com/en/read/14935/515038/implementing-role-based-access-control-in-flask>
45. <https://github.com/mikecolbert/flask-rbac>
46. <https://www.geeksforgeeks.org/python/flask-role-based-access-control/>
47. <https://www.youtube.com/watch?v=9v2NZ2dGB60>
48. <https://www.cerbos.dev/blog/authorization-in-flask>
49. <https://flask-rbac.readthedocs.io>
50. https://www.reddit.com/r/flask/comments/jd4r43/recommendations_for_implementing_rolebased_access/
51. <https://www.geeksforgeeks.org/python/flask-role-based-access-control/>
52. <https://stackoverflow.com/questions/36213748/flask-admin-role-based-access-modify-access-based-on-role>
53. <https://stackoverflow.com/questions/31662098/flask-admin-role-based-resource-permissions>
54. <https://github.com/mattupstate/flask-security/issues/518>
55. <https://github.com/pallets-eco/flask-security-3.0/issues/518>
56. https://www.reddit.com/r/flask/comments/1490dmb/how_to_handle_roles_with_flask_and_flask/
57. <https://stackoverflow.com/questions/66810286/show-flask-security-roles-in-jinja2>
58. <https://flask-admin.readthedocs.io/en/stable/introduction/>
59. <https://flask-admin.readthedocs.io/en/v1.0.9/templates/>
60. <https://stackoverflow.com/questions/42013067/how-to-access-session-variables-in-jinja-2-flask>
61. <https://droid.io/framework-diagnosis-knowledge/python-flask-jinja2-templatesyntaxerror>
62. https://www.youtube.com/watch?v=L9it54_C7kE
63. https://www.reddit.com/r/MailDevNetwork/comments/1gotx5g/resolving_jinja2_templatenotfound_error_in_flask/
64. <https://stackoverflow.com/questions/76115311/using-session-within-template-in-flask-jinja2-application>
65. <https://stackoverflow.com/questions/18361151/using-flask-login-session-with-jinja2-templates>
66. https://www.reddit.com/r/learnpython/comments/9pys3o/template_not_found_with_jinja_using_flask/
67. https://www.reddit.com/r/flask/comments/g34asa/session_variable_not_being_registered_as_html/
68. <https://github.com/yogthos/Selmer/issues/204>
69. <https://testdriven.io/tips/e88ab5a0-7f4b-4791-a687-378658250b67/>
70. <https://www.mycompiler.io/view/KLwNZtOHw9L>
71. https://www.reddit.com/r/Database/comments/17tm709/need_help_in_structuring_a_database_to_store/
72. <https://www.geeksforgeeks.org/dbms/how-to-design-a-database-for-online-restaurant-reservation-and-food-delivery/>
73. <https://stackoverflow.com/questions/20931082/how-would-i-organize-a-postgres-db-for-a-list-of-restaurants-and-their-menu-item>
74. <https://stackoverflow.com/questions/38872119/database-scheme-for-stores-and-restaurants>
75. <https://companydata.com/de/datenbank/restaurant/>
76. <https://www.tutorials24x7.com/mysql/guide-to-design-database-for-restaurant-order-system-in-mysql>
77. <https://github.com/scottsimpson/restaurant-database/blob/master/restaurant.sql>

78. https://www.reddit.com/r/Database/comments/14e5o0c/struggling_with_designing_a_restaurant_database/

79. https://www.reddit.com/r/datasets/comments/1bidvj2/how_do_i_create_a_database_of_restaurant_menus/