

```

import os from flask import Flask, render_template,
redirect, session, url_for, request, abort, flash, jsonify
import db, forms app = Flask(__name__)
app.config.from_mapping( SECRET_KEY='dev',
DATABASE=os.path.join(app.instance_path, 'data.sqlite'), )
app.cli.add_command(db.init_db)
app.teardown_appcontext(db.close_db_con)
@app.route('/') def index(): form = forms.ProfileForm()
form.user_id.data = session['user_id'] return
render_template('index.html')
@app.route('/overlay/<int:group_id>/') def
overlay(group_id): logincheck() db_con =
db.get_db_con() group_row = db_con.execute(
"SELECT id, name FROM groups WHERE id = ?",
(group_id,) ).fetchone() if group_row is None:
abort(404) active_challenge = db_con.execute("""
SELECT c.*, gc.status FROM group_challenges gc
JOIN challenges c ON c.id = gc.challenge_id WHERE
gc.group_id = ? AND gc.status = 'active' ORDER BY
gc.started_at DESC, gc.assigned_at DESC LIMIT 1
""", (group_id,)).fetchone() done_challenges =
db_con.execute(""" SELECT c.*, gc.status FROM
group_challenges gc JOIN challenges c ON c.id =
gc.challenge_id WHERE gc.group_id = ? AND
gc.status = 'done' ORDER BY gc.assigned_at DESC
""", (group_id,)).fetchall() queued_challenges =
db_con.execute(""" SELECT c.*, gc.status FROM
group_challenges gc JOIN challenges c ON c.id =
gc.challenge_id WHERE gc.group_id = ? AND
gc.status = 'queued' ORDER BY gc.assigned_at DESC
""", (group_id,)).fetchall() if request.args.get("json") is
not None: return jsonify({ "group_name":
group_row['name'], "active_challenge":
dict(active_challenge) if active_challenge else None,
"queued_challenges": [dict(ch) for ch in
queued_challenges], "done_challenges": [dict(ch)
for ch in done_challenges] }) else: return
render_template('overlay.html',
group_name=group_row['name'],
active_challenge=active_challenge,

```

```

queued_challenges=queued_challenges,
done_challenges=done_challenges      ) # -----
Challenges ----- @app.route('/challenges/') def
challenges(): db_con = db.get_db_con() challenges =
db_con.execute(      "SELECT id, title, difficulty,
game_name, time_needed FROM challenges ORDER BY
title"      ).fetchall() return
render_template('challenges.html', challenges=challenges)
@app.route('/challenge/<int:challenge_id>/') def
challenge(challenge_id): db_con = db.get_db_con()
challenge_row = db_con.execute(      ""      SELECT id,
title, description, difficulty, game_name, time_needed
FROM challenges      WHERE id = ?      """,
(challenge_id,)      ).fetchone() if challenge_row is None:
abort(404) return render_template('challenge.html',
challenge=challenge_row) # ----- Static pages -----
@app.route('/support/') def support(): return
render_template('support.html') @app.route('/pricing/')
def pricing(): return render_template('pricing.html')
@app.route('/guide/') def guide(): return
render_template('guide.html') # ----- Authentication --
----- @app.route('/login/', methods=['GET', 'POST']) def
login(): db_con = db.get_db_con() form =
forms.LoginForm() if request.method == 'GET':
sql_query = "SELECT id, username FROM users ORDER BY
username"      users =
db_con.execute(sql_query).fetchall() return
render_template('login.html', form=form, users=users)
else: # POST      if form.validate():      username =
form.username.data      password =
form.password.data      user = db_con.execute(
"SELECT id, password FROM users WHERE username = ?",
(username,)      ).fetchone()      if user and
password == user['password']:      session['user_id']
= user['id']      flash('Login successful.', 'success')
return redirect(url_for('index'))      else:
flash('Invalid username or password.', 'error')      else:
flash("Please fill out the form correctly.", 'error') return
render_template('login.html', form=form)
@app.route('/register/', methods=['GET', 'POST']) def
register(): db_con = db.get_db_con() form =
forms.RegisterForm() if request.method == 'GET':
sql_query = "SELECT * FROM users ORDER BY id"

```

```

users = db_con.execute(sql_query).fetchall()      return
render_template('register.html', form=form, users=users)
else: # POST      if form.validate():      sql_query =
"INSERT INTO users (username, password, email) VALUES
(?, ?, ?);"      username = form.username.data
password = form.password.data      email =
form.email.data      db_con.execute(sql_query,
(username, password, email))      db_con.commit()
flash('Registration successful. Please log in.', 'success')
return redirect(url_for('login'))      else:      flash('Error
in registration form. Please check the input fields.', 'error')
return render_template('register.html', form=form) # ----
---- Profile ----- @app.route('/profile/') def profile():
logincheck() db_con = db.get_db_con() user_id =
session['user_id'] user = db_con.execute("SELECT
id, username, email, abonoment FROM users WHERE id =
?", (user_id,)).fetchone() return
render_template('profile.html', user=user) # -----
Groups ----- @app.route('/groups/') def groups():
db_con = db.get_db_con() #search functionality g
= request.args.get("g", "").strip().lower() groups =
db_con.execute("SELECT id, name FROM groups
WHERE lower(name) LIKE ? ORDER BY name", (f"%{g}%",))
).fetchall() return render_template('groups.html',
groups=groups, g=g) @app.route('/join/<int:group_id>/')
def join_group(group_id): logincheck() db_con =
db.get_db_con() form = forms.JoinGroupForm()
user_id = session['user_id'] # Hole Gruppeninfo (inkl.
password und owner_id) group_row = db_con.execute(
"SELECT id, name, password, owner_id FROM groups
WHERE id = ?", (group_id,)).fetchone() if
group_row is None: abort(404) # Prüfe, ob User
bereits Mitglied ist existing_member = db_con.execute(
"SELECT 1 FROM group_members WHERE user_id = ? AND
group_id = ?", (user_id, group_id)).fetchone() if
existing_member: flash('You are already a member of
this group.', 'info') return redirect(url_for('group',
group_id=group_id)) # Füge User als Mitglied hinzu
und Prüfe Passwort if request.method == 'GET':
return render_template('join_group.html', form=form,
group=group_row) else: # POST      if form.validate():
if form.password.data == group_row['password']:
# Füge User zur Gruppe hinzu      db_con.execute(

```

```

"INSERT INTO group_members (owner_id, user_id,
group_id) VALUES (?, ?, ?)",
(group_row['owner_id'], user_id, group_id)
    )
db_con.commit()
    flash('Successfully joined the
group!', 'success')
    return redirect(url_for('group',
group_id=group_id))
    else:
        flash('Invalid
group password.', 'error')
    else:
        flash('Please
enter the group password.', 'error')
    return
render_template('join_group.html', form=form,
group=group_row)
@app.route('/group/<int:group_id>/',
methods=["GET", "POST"])
def group(group_id):
    logincheck()
    db_con = db.get_db_con()
    user_id = session['user_id']
    form = forms.GroupForm()
    #
    Prüfe, ob User Mitglied oder Owner ist
    is_member = db_con.execute(
        "SELECT 1 FROM group_members
WHERE user_id = ? AND group_id = ?",
        (user_id, group_id)
    ).fetchone()
    group_row = db_con.execute(
        "SELECT id, owner_id, name FROM groups WHERE id = ?",
        (group_id,)
    ).fetchone()
    if group_row is None:
        abort(404)
    owner = db_con.execute(
        "SELECT id, username FROM users WHERE id = ?",
        (group_row["owner_id"],)
    ).fetchone()
    group_challenges = db_con.execute(
        "SELECT * FROM group_challenges WHERE group_id = ?",
        (group_id,)
    ).fetchall()
    group_members = db_con.execute("""
SELECT u.id, u.username, owner_id
FROM group_members gm
JOIN users u ON u.id = gm.user_id
WHERE u.id != owner_id AND gm.group_id = ?
""", (group_id,)).fetchall()
    active_challenge = db_con.execute("""
SELECT c.*, gc.status
FROM group_challenges gc
JOIN challenges c ON c.id = gc.challenge_id
WHERE gc.group_id = ? AND gc.status = 'active'
ORDER BY gc.started_at DESC, gc.assigned_at DESC
LIMIT 1
""", (group_id,)).fetchone()
    done_challenges = db_con.execute("""
SELECT c.*, gc.status
FROM group_challenges gc
JOIN challenges c ON c.id = gc.challenge_id
WHERE gc.group_id = ? AND gc.status = 'done'
ORDER BY gc.assigned_at DESC
""", (group_id,)).fetchall()
    queued_challenges = db_con.execute("""
SELECT c.*, gc.status
FROM group_challenges gc
JOIN challenges c ON c.id = gc.challenge_id
WHERE gc.group_id = ? AND

```

```

gc.status = 'queued'      ORDER BY gc.assigned_at DESC
"", (group_id,)).fetchall()    #search functionality    q =
request.args.get("q", "").strip().lower()    challenges =
db_con.execute("""        SELECT id, title, game_name
FROM challenges        WHERE title LIKE ? AND id NOT IN (
SELECT challenge_id FROM group_challenges WHERE
group_id = ?        );    "", (f"%{q}%", group_id)).fetchall()
if request.method == 'GET':        return render_template(
"group.html",        group=group_row,
owner=owner,        group_members=group_members,
active_challenge=active_challenge,
done_challenges=done_challenges,
queued_challenges=queued_challenges,
is_member=is_member,        challenges=challenges,
q=q,        form=form,
group_challenges=group_challenges    )    else:
#request.method == 'POST'        if
form.validate_on_submit():        if
form.add_challenge.data:        sql_query = ""
INSERT OR IGNORE INTO group_challenges (group_id,
challenge_id)        VALUES (?, ?);        ""
db_con.execute(sql_query, [group_id,
form.challenge_id.data])        db_con.commit()
flash('Challenge has been added', 'success')        if
form.delete_challenge.data:        sql_query = ""
DELETE FROM group_challenges        WHERE
group_id = ? AND challenge_id = ?;        ""
db_con.execute(sql_query, [group_id,
form.challenge_id.data])        db_con.commit()
flash('Challenge has been deleted', 'success')        if
form.set_active.data:        # alte active
deaktivieren        db_con.execute("""
UPDATE group_challenges        SET status =
'queued', started_at = NULL        WHERE group_id
= ? AND status = 'active';        "", (group_id,))
db_con.commit()        # neue active setzen
db_con.execute("""        UPDATE group_challenges
SET status = 'active', started_at = CURRENT_TIMESTAMP
WHERE group_id = ? AND challenge_id = ?;        "",
(group_id, form.challenge_id.data))
db_con.commit()        flash('Active challenge has
been updated.', 'success')        if
form.completed_challenge.data:        # Markiere die

```

```

aktive Challenge als 'done' db_con.execute("""
UPDATE group_challenges SET status = 'done',
finished_at = CURRENT_TIMESTAMP WHERE
group_id = ? AND challenge_id = ?; """,
(group_id, form.challenge_id.data))
db_con.commit() flash('Challenge marked as
completed.', 'success') if
form.start_session.data: db_con.execute("""
UPDATE groups SET session_start =
CURRENT_TIMESTAMP WHERE id = ?;
""", (group_id,)) db_con.commit()
flash('Session has been started', 'success') if
request.args.get("json") is not None: return
jsonify({"status": "success"}) else: return
redirect(url_for('group', group_id=group_id)) #if
request.args.get("json"): #return
redirect(url_for('group', group_id=group_id)) # -----
Create Group ----- @app.route('/create_group/',
methods=['GET', 'POST']) def create_group():
logincheck() db_con = db.get_db_con() form =
forms.CreateGroupForm() form.user_id.data =
session.get('user_id') if request.method == 'GET':
return render_template('create_group.html', form=form)
else: #request.method == 'POST' if form.validate():
sql_query = 'INSERT INTO groups (name, password,
owner_id) VALUES (?, ?, ?);'
db_con.execute(sql_query, [form.name.data,
form.password.data, form.user_id.data])
db_con.commit() flash('Group has been created.',
'success') else: flash('Error creating group.
Please check the input fields.', 'error') return
redirect(url_for('groups')) # groups fürs debugging -->
/group/<int:group_id>/ direkt zur erstellten
@app.route('/insert/sample/') def run_insert_sample():
db.insert_sample() return "Sample data inserted." def
logincheck(): if 'user_id' not in session: flash('Please
log in to see the content.') return
redirect(url_for('login')) und <!DOCTYPE html>
<html lang="de">
<head>
<meta charset="UTF-8">
<title>{% block title %}Overlay {{ group_name }} {%
endblock %}</title>

```

```

<link rel="stylesheet" href="{{ url_for('static',
filename='overlay.css') }}">
</head>

<body>
    {% if active_challenge %}
        <h1>Aktive Challenge</h1>
        <li><strong>{{ active_challenge.title }}</strong> </li>
        <p>{{ active_challenge.description }}</p>
    {% else %}
        <p>Gerade keine aktive Challenge</p>
    {% endif %}

    <h2>Restliche Challenges</h2>
    {% if queued_challenges %}

        <ul>
            {% for ch in queued_challenges %}
                <li>{{ ch.title }} ({{ ch.status }})</li>
            {% endfor %}
        </ul>
    {% endif %}

</ul>
<footer>
    <p>&copy; StreamQuest </p>
</footer>
</body>
</html>

```

Ja – die **Grundidee** (Route `/overlay/<group_id>/` lädt Group + Challenges und rendert ein Overlay-Template oder liefert JSON) macht absolut Sinn. Aber in deinem Code sind ein paar Stellen, die dir sehr wahrscheinlich Bugs / unerwartetes Verhalten verursachen.

1) Wichtigster Bug: `logincheck()` funktioniert so nicht

Du machst:

```
python
```

 Code kopieren

```
logincheck()
```

Aber `logincheck()` gibt im Fehlerfall ein `redirect(...)` zurück – **du gibst dieses Redirect aber nicht zurück**. D. h. wenn nicht eingeloggt, läuft die Funktion trotzdem weiter und knallt danach z. B. bei `session['user_id']` oder zeigt Daten ohne Login.

So muss das aussehen:

python

 Code kopieren

```
rv = logincheck()
if rv:
    return rv
```

Oder (sauberer): `logincheck()` soll **abort(401/403)** oder du nutzt `@login_required` - Decorator.

Außerdem ist in deiner `index()` -Route das hier kritisch:

python

 Code kopieren

```
form.user_id.data = session['user_id']
```

Wenn nicht eingeloggt → `KeyError`. Also auch dort `Logincheck` oder `session.get` .

2) `logincheck()` selbst: Redirect “verpufft”

Aktuell:

python

 Code kopieren

```
def logincheck():
    if 'user_id' not in session:
        flash('Please log in to see the content.')
        return redirect(url_for('login'))
```

Das ist okay, wenn du das Result immer zurückgibst (siehe oben). Sonst nicht.

3) Template `overlay.html` : ein paar HTML/Logik-Kleinigkeiten

- Du hast ein **überflüssiges** `</u1>` (du öffnest unten kein zweites `<u1>` , schließt aber eins).
- Wenn `queued_challenges` leer ist, wird einfach gar nichts angezeigt. Vielleicht willst du ein “keine restlichen challenges”.

Beispiel fix:

html

```

<h2>Restliche Challenges</h2>
{% if queued_challenges %}
    <ul>
        {% for ch in queued_challenges %}
            <li>{{ ch.title }} ({{ ch.status }})</li>
        {% endfor %}
    </ul>
{% else %}
    <p>Keine weiteren Challenges in der Queue.</p>
{% endif %}

```

4) active_challenge / queued_challenges / done_challenges Query-Design ist ok

Deine SQLs sind konsistent:

- active nimmt die "neueste" aktive (started_at/assigned_at) und limitiert auf 1.
- queued/done listen alle.

Das passt für ein Overlay super.

5) JSON-Mode: passt, aber Achtung bei dict(row)

Das klappt, wenn dein SQLite cursor row_factory korrekt auf dict-artige Rows steht (z. B. sqlite3.Row). Sonst crasht dict(active_challenge) .

Wenn's bei dir läuft: okay.

6) Noch ein paar "lohnt sich" Hinweise (nicht zwingend, aber sinnvoll)

- Passwörter werden im Klartext gespeichert/verglichen → für echte Nutzung unbedingt hashen (werkzeug.security).
- SECRET_KEY='dev' nur für lokal.
- In join_group ist die Route standardmäßig nur GET, aber du behandelst POST. Dann musst du methods=['GET', 'POST'] setzen, sonst kommt nie POST an.

Minimaler Fix, damit Login wirklich greift (Overlay als Beispiel)

python

 Code kopieren

```

@app.route('/overlay/<int:group_id>/')
def overlay(group_id):
    rv = logincheck()

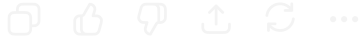
```

```

if rv:
    return rv
...

```

Wenn du magst, kann ich dir auch kurz einen kleinen `@login_required` -Decorator zeigen, damit du diese zwei Zeilen nicht in jede Route schreiben musst.



aber wird overlay aktuell über json generiert?

Kurzfassung: Ja – dein Overlay *kann* aktuell über JSON generiert werden, aber nur wenn du es explizit so aufrufst. Standardmäßig wird HTML gerendert, nicht JSON.

Was dein Code aktuell macht

In deiner Route:

python

```

@app.route('/overlay/<int:group_id>/')
def overlay(group_id):
    ...
    if request.args.get("json") is not None:
        return jsonify({
            "group_name": group_row['name'],
            "active_challenge": dict(active_challenge) if active_challenge else None,
            "queued_challenges": [dict(ch) for ch in queued_challenges],
            "done_challenges": [dict(ch) for ch in done_challenges]
        })
    else:
        return render_template(
            'overlay.html',
            group_name=group_row['name'],
            active_challenge=active_challenge,
            queued_challenges=queued_challenges,
            done_challenges=done_challenges
        )

```

 Code kopieren

Bedeutet konkret:

Aufruf	Ergebnis
<code>/overlay/3/</code>	HTML (<code>overlay.html</code>)
<code>/overlay/3/?json=1</code>	JSON
<code>/overlay/3/?json=true</code>	JSON
<code>/overlay/3/?json=</code>	JSON

➔ JSON wird NICHT automatisch benutzt, sondern nur, wenn der Query-Parameter `json` existiert.

Wird das Overlay „über JSON generiert“?

✗ Nein (aktuell)

- Dein HTML-Overlay wird serverseitig mit Jinja gerendert
- Kein JavaScript lädt JSON nach
- Kein Fetch / AJAX / Polling

✓ Ja (optional / vorbereitet)

- Du hast eine API-ähnliche JSON-Ausgabe eingebaut
- Perfekt für:
 - OBS Browser Source
 - JavaScript Overlay
 - Polling / Live Updates

Wann würdest du wirklich „JSON-basiert“ sein?

Erst wenn du im `overlay.html` JavaScript verwendest, z. B.:

```
html
```

```
<script>
async function loadOverlay() {
  const res = await fetch('/overlay/3/?json=1');
  const data = await res.json();
}
```

```

document.querySelector('#active').innerText =
  data.active_challenge ? data.active_challenge.title : 'Keine aktive Challenge'
}

setInterval(loadOverlay, 5000);
loadOverlay();
</script>

```

 Code kopieren

Dann wäre:

- Backend = reine Daten
- Frontend = dynamisches Overlay
- Kein Reload nötig → ideal für Streaming



Empfehlung (für Stream-Overlay 💡)

Dein Ansatz ist sehr gut. Ich würde:

1. /overlay/<id>/?json=1 als offizielle Overlay-API
2. overlay.html nur als JS-Shell
3. Polling alle 2–5 Sekunden oder WebSocket später

Kurzantwort auf deine Frage

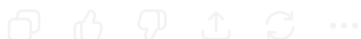
aber wird overlay aktuell über json generiert?

👉 Nein, standardmäßig über HTML.

JSON nur, wenn du ?json anhängst.

Wenn du willst, baue ich dir:

- ein minimalistisches OBS-Overlay (HTML + JS)
- oder eine saubere Trennung /api/overlay/<id> vs /overlay/<id>



änder die overlay.html datei so, das sie json nutzt, damit
Daten, die in group/group_id geändert werden gepatched
werden bei Änderungen: <!DOCTYPE html>

```

<html lang="de">
<head>
  <meta charset="UTF-8">
  <title>{% block title %}Overlay {{ group_name }} {%
endblock %}</title>
  <link rel="stylesheet" href="{{ url_for('static',
filename='overlay.css') }}">
</head>

<body>
  {% if active_challenge %}
    <h1>Aktive Challenge</h1>
    <li><strong>{{ active_challenge.title }}</strong></li>

    <div class="challenge">
      <p>Gerade keine aktive Challenge</p>
    {% endif %}

    <h2>Restliche Challenges</h2>
    {% if queued_challenges %}

      <ul>
        {% for ch in queued_challenges %}
          <li>{{ ch.title }} ({{ ch.status }})</li>
        {% endfor %}
      </ul>
    {% endif %}

  </ul>
</body>
</html>

```

ChatGPT 5.2 ▾

Klar — hier ist eine **overlay.html**, die **keine Jinja-Daten mehr rendert**, sondern die Daten **regelmäßig per JSON** von `/overlay/<group_id>/?json=1` holt und nur die geänderten Teile im DOM "patcht".

Wichtig: Damit das funktioniert, muss Flask beim Rendern dieser Seite **nur** `group_id` (optional) reinreichen. Also in deiner Route im HTML-Branch z. B.:

python

```
return render_template('overlay.html', group_id=group_id)
```

 Code kopieren

(Alles andere holt sich die Seite per JSON.)

html

```
<!DOCTYPE html>
<html lang="de">
<head>
  <meta charset="UTF-8">
  <title>Overlay</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='overlay.css') }}">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>

<body>
  <header>
    <h1 id="groupTitle">Overlay</h1>
  </header>

  <main>
    <section id="activeSection">
      <h2>Aktive Challenge</h2>
      <div id="activeContainer">
        <p>lade...</p>
      </div>
    </section>

    <section id="queuedSection">
      <h2>Restliche Challenges</h2>
      <ul id="queuedList"></ul>
      <p id="queuedEmpty" style="display:none;">Keine weiteren Challenges in der
    </section>

    <section id="doneSection" style="display:none;">
      <h2>Erledigte Challenges</h2>
      <ul id="doneList"></ul>
      <p id="doneEmpty" style="display:none;">Noch keine erledigten Challenges.</
    </section>
```

```
</main>

<footer>
  <p>&copy; StreamQuest</p>
</footer>

<script>
  // group_id wird serverseitig nur als Zahl eingesetzt
  const GROUP_ID = {{ group_id|int }};
  const JSON_URL = `/overlay/${GROUP_ID}/?json=1`;

  const els = {
    groupTitle: document.getElementById('groupTitle'),

    activeContainer: document.getElementById('activeContainer'),

    queuedList: document.getElementById('queuedList'),
    queuedEmpty: document.getElementById('queuedEmpty'),

    doneSection: document.getElementById('doneSection'),
    doneList: document.getElementById('doneList'),
    doneEmpty: document.getElementById('doneEmpty'),
  };

  // Letzten Zustand merken, um nur Änderungen zu patchen
  let lastStateJson = null;

  function escapeHtml(s) {
    return String(s ?? '')
      .replaceAll('&', '&amp;')
      .replaceAll('<', '&lt;')
      .replaceAll('>', '&gt;')
      .replaceAll('"', '&quot;')
      .replaceAll("'", '&#039;');
  }

  function renderActive(active) {
    if (!active) {
      els.activeContainer.innerHTML = `<p>Gerade keine aktive Challenge</p>`;
      return;
    }
    const title = escapeHtml(active.title);
    const desc = escapeHtml(active.description);
```

```

els.activeContainer.innerHTML = `
  <div class="active-challenge">
    <div class="active-title"><strong>${title}</strong></div>
    <div class="active-desc">${desc}</div>
  </div>
`;
}

function renderList(listEl, emptyEl, items, formatter) {
  if (!items || items.length === 0) {
    listEl.innerHTML = '';
    emptyEl.style.display = 'block';
    return;
  }
  emptyEl.style.display = 'none';
  listEl.innerHTML = items.map(formatter).join('');
}

function liChallenge(ch) {
  const title = escapeHtml(ch.title);
  const status = escapeHtml(ch.status);
  return `<li>${title} (${status})</li>`;
}

function patchDom(data) {
  // Titel + Gruppenname
  const groupName = data.group_name ? `Overlay - ${data.group_name}` : 'Overl
  if (els.groupTitle.textContent !== groupName) {
    els.groupTitle.textContent = groupName;
    document.title = groupName;
  }

  // Active
  renderActive(data.active_challenge);

  // Queued
  renderList(els.queuedList, els.queuedEmpty, data.queued_challenges, liChall

  // Done (optional anzeigen)
  const hasDone = (data.done_challenges && data.done_challenges.length > 0);
  els.doneSection.style.display = hasDone ? 'block' : 'none';
  renderList(els.doneList, els.doneEmpty, data.done_challenge:  Code kopieren ;
}

```



```

async function fetchJson() {
  const res = await fetch(JSON_URL, { cache: 'no-store' });
  if (!res.ok) throw new Error(`HTTP ${res.status}`);
  return await res.json();
}

async function tick() {
  try {
    const data = await fetchJson();
    const current = JSON.stringify(data);

    // Nur patchen, wenn sich etwas geändert hat
    if (current !== lastStateJson) {
      patchDom(data);
      lastStateJson = current;
    }
  } catch (err) {
    // Bei Fehler nicht alles leeren, nur Hinweis setzen
    els.activeContainer.innerHTML = `

Overlay offline / Fehler beim Laden</p>`;
    console.error(err);
  }
}

// Polling-Intervall (2s ist fürs Streaming meist okay)
tick();
setInterval(tick, 2000);
</script>
</body>
</html>


```

Was du damit bekommst

- Overlay lädt beim Start die Daten per JSON
- alle 2 Sekunden werden Updates geholt
- es wird **nur neu gerendert**, wenn sich das JSON wirklich geändert hat

Wenn du willst, kann ich dir auch eine Variante bauen, die **nur einzelne DOM-Teile** updated (z. B. per IDs der Challenges), statt per `JSON.stringify` auf „alles oder nichts“.



jinja2.exceptions.UndefinedError: 'group_id' is undefined

Das passiert, weil du beim Rendern von `overlay.html` **kein** `group_id` **an das Template übergibst**. In deinem JS steht aber:

js

```
const GROUP_ID = {{ group_id|int }};
```

 Code kopieren

Wenn `group_id` im Template-Kontext fehlt → Jinja `UndefinedError`.

Fix (empfohlen): `group_id` beim Rendern übergeben

Ändere in deiner `overlay()` -Route den HTML-Teil so:

python

```
return render_template('overlay.html', group_id=group_id)
```

 Code kopieren

Also komplett:

python

```
@app.route('/overlay/<int:group_id>/')
def overlay(group_id):
```

```
    rv = logincheck()
```

```
    if rv:
```

```
        return rv
```

```
    db_con = db.get_db_con()
```

```
    group_row = db_con.execute(
```

```
        "SELECT id, name FROM groups WHERE id = ?",
```

```
        (group_id,)
```

```
    ).fetchone()
```

```
    if group_row is None:
```

```
        abort(404)
```

```
    active_challenge = db_con.execute(""" ... """, (group_id,)).fetchone()
```

```
    done_challenges = db_con.execute(""" ... """, (group_id,)).fetchone()  Code kopieren
```

```
    queued_challenges = db_con.execute(""" ... """, (group_id,)).fetchall()
```

```

if request.args.get("json") is not None:
    return jsonify({
        "group_name": group_row['name'],
        "active_challenge": dict(active_challenge) if active_challenge else N
        "queued_challenges": [dict(ch) for ch in queued_challenges],
        "done_challenges": [dict(ch) for ch in done_challenges]
    })

return render_template('overlay.html', group_id=group_id)

```

Alternative (ohne Template-Variable): group_id aus der URL ziehen

Wenn du *gar kein* Jinja für `group_id` willst, kannst du im `overlay.html` statt `{{ group_id }}` einfach die ID aus `window.location.pathname` extrahieren:

Ersetze im Script:

js

 Code kopieren

```

const GROUP_ID = {{ group_id|int }};
const JSON_URL = `/overlay/${GROUP_ID}/?json=1`;

```

durch:

js

 Code kopieren

```

const match = window.location.pathname.match(/\/overlay\/(\d+)\//);
const GROUP_ID = match ? match[1] : null;

if (!GROUP_ID) {
    throw new Error("Konnte group_id nicht aus der URL lesen.");
}

const JSON_URL = `/overlay/${GROUP_ID}/?json=1`;

```

Damit brauchst du **gar nichts** mehr in `render_template(...)` zu übergeben.

Wenn du OBS Browser Source nutzt, ist die **URL-Parsing-Variante** oft am robustesten, weil du dann nie vergisst `group_id` zu übergeben.



