Karol Osko
G4C

Notenverwaltung Webapp

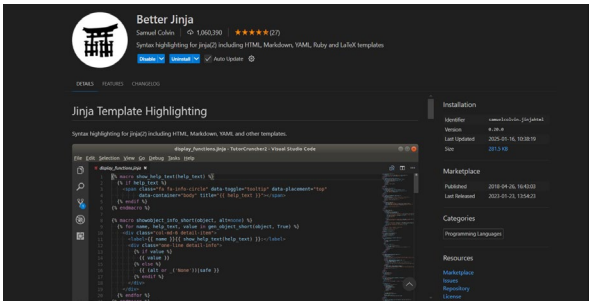
Beschreibung des Resultats

Das Resultat meiner Arbeit erfüllt alle Anforderungen, die ich mir gesetzt habe. Die Anwendung ermöglicht die Verwaltung von Noten in verschiedenen Fächern. Noten können mit Fachname, Prüfungsdatum und Gewichtung gespeichert und in einer Datenbank abgelegt werden, wobei nur gültige Werte zwischen 1 und 6 akzeptiert werden. Gespeicherte Noten werden pro Fach angezeigt, und sowohl der Fach- als auch der Gesamtdurchschnitt werden berechnet. Ein zusätzliches Feature erlaubt die Simulation der nächsten Note: Die Anwendung berechnet in Echtzeit, welche Mindestnote benötigt wird, um eine genügende Durchschnittsnote (z. B. 4.0) zu erreichen. Zudem können Noten nach Datum sortiert werden. Noten und Fächer lassen sich bearbeiten oder löschen, um Korrekturen vorzunehmen oder Einträge zu entfernen. Darüber hinaus habe ich freiwillig in meiner Freizeit an einem Account System gearbeitet, so dass verschiedene User ihre eigene Notenlisten haben können, sich registrieren und bzw. einloggen können. Dieses System wurde gegen IDOR angriffe mit einigen Sicherheitsmassnahmen gesichert. Dabei habe ich auch einiges in Richtung in den Basics von Cybersecurity in solchen Anwendungen gelernt aber auch meine Fähigkeiten mit Datenbanken umzugehen geschärft. Die Anwendung ermöglicht es aber auch die Importierung der Noten von der Schulnetz Noten Tabelle, das war aber auch freiwillig und folglich nicht in meinen Anforderungen. Deswegen werde ich nicht drauf eingehen. Der Frontend wurde mit BetterJinja und JavaScript gemacht, da ich schon einige Erfahrungen in JavaScript habe von meinem Projekt im Grundlagenfach.

Nötige Schritte

Es wurden mehrere Libraries verwendet. Diese muss man herunterladen, sowohl wie die VS-Code Extension BetterJinja. Diese wurde benutzt, um mir das Leben in den HTML Templates einfacher zu machen.

1. Schritt:
   Im Terminal eingeben: pip install flask flask-sqlalchemy beautifulsoup4 werkzeug flask-login
2. Schritt:
   Die Better Jinja Extension herunterladen. Sonst muss man nichts mehr machen. Der Syntax der HTML Templates wird automatisch durch die settings.json Datei im .vscode Folder geändert.



Anforderungen

Anforderungen

1. **Erstellung und Speicherung von Noten (Create):**

- **Die Anwendung muss es ermöglichen, Noten in verschiedenen Fächern zu speichern, inklusive Fachname, Prüfungsdatum und Note (inkl. Gewichtung) und Bearbeiten. Die Daten sollten überprüft werden, ob sie valid sind. D.h. Nur 1 bis 6.**

Das Backend verwendet eine dynamische Route zur Verwaltung von Fächern (fach) und Noten (note). Die Route handle_item() behandelt sowohl das Erstellen als auch das Bearbeiten von Fächern und Noten. Die Route wurde dynamisch gestaltet da es flexibler, sauberer und erweiterbarer ist.

```
@app.route("/<string:type>/<string:action>", methods=["GET", "POST"])
@app.route("/<string:type>/<string:action>/<int:fach_id>", methods=["GET",
"POST"])
def handle_item(type, action, fach_id=None):
    if type == "fach":
        item = Fach.query.filter_by(id=fach_id,
user_id=current_user.id).first() if fach_id else None
        if fach_id and not item:
            return "Unauthorized", 403


        if request.method == "POST":
            name = request.form['name']
            if action == "add":
                new_fach = Fach(name=name, user_id=current_user.id)
                db.session.add(new_fach)
            elif action == "edit" and item:
                item.name = name
            db.session.commit()
            return redirect(url_for('index'))
        return render_template("handle.html", type="Fach", action=action,
data=item)
```

Funktionsweise:

Fächer erstellen (add): Erstellt ein neues Fach mit einem Namen und speichert es für den aktuellen Nutzer.

Fächer bearbeiten (edit): Erlaubt das Ändern des Namens eines bestehenden Faches.

Die handle_item()-Funktion behandelt auch das Hinzufügen und Bearbeiten von Noten. Hierbei werden verschiedene Validierungen durchgeführt.

```
elif type == "note":
        item = Note.query.get(fach_id) if fach_id and action == "edit" else
None
        fach = Fach.query.get(item.fach_id) if item else
Fach.query.get(fach_id)

        if not fach:
            return "Fach nicht gefunden", 404

        if fach.user_id != current_user.id:
            return "Unauthorized", 403

        existing_grades = [note.wert for note in fach.noten] if fach.noten
else []
        existing_weights = [note.gewichtung for note in fach.noten] if
fach.noten else []

        if request.method == "POST":
            try:
                wert = float(request.form['wert'])
                if not (1 <= wert <= 6):
                    return "Die Note muss zwischen 1 und 6 liegen", 400

                datum = datetime.strptime(request.form['datum'], '%Y-%m-
%d').date()
                gewichtung = float(request.form.get('gewichtung', 1.0))

                if gewichtung <= 0:
                    return "Die Gewichtung muss größer als 0 sein", 400

                if action == "add":
                    new_note = Note(
                        wert=wert,
                        datum=datum,
                        gewichtung=gewichtung,
                        fach_id=fach_id,
                        user_id=current_user.id
                    )
                    db.session.add(new_note)
                elif action == "edit" and item:
                    item.wert = wert
                    item.datum = datum
                    item.gewichtung = gewichtung
```

```
            db.session.commit()
            return redirect(url_for('index'))

        except ValueError:
            return "Ungültige Eingabe", 400
        except Exception as e:
            return str(e), 500
```

- **Die Daten sollen in einer Datenbank gespeichert werden.**

```
class Fach(db.Model):
    __tablename__ = 'faecher'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False)
    noten = db.relationship('Note', backref='fach', lazy=True, cascade="all, delete")
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
class Note(db.Model):
    __tablename__ = 'noten'
    id = db.Column(db.Integer, primary_key=True)
    wert = db.Column(db.Float, nullable=False)
    datum = db.Column(db.Date, nullable=False)
    gewichtung = db.Column(db.Float, default=1.0
    fach_id = db.Column(db.Integer, db.ForeignKey('faecher.id'), nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
```

Die Datenbank wird mit SQLAlchemy verwaltet. Es werden zwei Tabellen definiert: faecher (Fach) und noten (Note). Die Daten werden über Beziehungen miteinander verknüpft.

Die Fach-Klasse repräsentiert die Schulfächer eines Nutzers:

id (Primärschlüssel): Eindeutige ID des Fachs.
name: Name des Fachs (z. B. "Mathematik").
user_id (Fremdschlüssel): Verknüpfung mit dem Nutzer (users.id).
noten: Beziehung zur Note-Tabelle mit backref='fach', sodass Noten eines Fachs einfach abgerufen werden können. Die cascade="all, delete"-Option sorgt dafür, dass beim Löschen eines Fachs auch alle zugehörigen Noten entfernt werden.

Die Note-Klasse speichert einzelne Noteneinträge:

id (Primärschlüssel): Eindeutige ID der Note.
wert: Der Notenwert als Float.
datum: Datum der Notenvergabe als Date.
gewichtung: Standardmässig 1.0, um unterschiedlich gewichtete Noten zu ermöglichen.

fach_id (Fremdschlüssel): Verknüpfung zur faecher-Tabelle.
user_id (Fremdschlüssel): Verknüpfung zur users-Tabelle, um die Noten einem Nutzer zuzuordnen.

Beziehungen wie backref='fach' ermöglichen einfache Abfragen, beispielsweise um alle Noten eines Fachs abzurufen.

2. Anzeige und Berechnung von Durchschnittsnoten (Read):

- **Die Anwendung soll die gespeicherten Noten pro Fach anzeigen**

```python
@app.route("/", methods=["GET", "POST"])
def index():



    faecher = Fach.query.filter_by(user_id=current_user.id).all()
    faecher_data = []
    for fach in faecher:
        fach.average = fach.calculate_average()
        fach.minimum_grade = fach.calculate_minimum_grade(4.0)
        faecher_data = [fach.to_dict() for fach in faecher]
    total_average = 0
    total_pluspunkte = 0
    faecher_mit_noten = [fach for fach in faecher if len(fach.noten) > 0]
    fach_count = len(faecher_mit_noten)

    for fach in faecher:
        total_average += fach.calculate_average()
        total_pluspunkte += fach.calculate_pluspunkte()


    if fach_count > 0:
        total_average /= fach_count
    else:
        total_average = 0.0

    username = current_user.username

    return render_template(
        "index.html",
        faecher=faecher,
        total_average=round(total_average, 3),
        total_pluspunkte=total_pluspunkte,
        show_login = False,
        faecher_data=faecher_data,
        username=username
```

```
    )
```

Hier sind die user.ids enthalten, für das Loginsystem, das ich in meiner Freizeit gemacht habe, jedoch ist das Prinzip des Anzeigens sehr einfach. Die Fächer werden geladen und ans index.html Template geschickt.

- **und den Durchschnitt pro Fach**
- **sowie einen Gesamtdurchschnitt berechnen.**

Berechnung des Durchschnitts pro Fach

Die Methode calculate_average() wird in der Klasse Fach definiert und berechnet den gewichteten Durchschnitt der Noten eines Faches.

```
class Fach(db.Model)

#--Andere Felder….—

def calculate_average(self):
        total_weighted_sum = sum(note.wert * note.gewichtung for note in
self.noten)
        total_weight = sum(note.gewichtung for note in self.noten)
        if total_weight == 0:
            return 0.0
        return round(total_weighted_sum / total_weight, 3)
```

Funktionsweise:

- Die Methode summiert die gewichteten Notenwerte.
- Die Summe der Gewichtungen wird berechnet.
- Falls keine Noten vorhanden sind (total_weight == 0), wird 0.0 zurückgegeben.
- Ansonsten wird der gewichtete Durchschnitt berechnet und auf drei Dezimalstellen gerundet.

Berechnung des Gesamtdurchschnitts:

```
 for fach in faecher:
        total_average += fach.calculate_average()
        total_pluspunkte += fach.calculate_pluspunkte()

    if fach_count > 0:
        total_average /= fach_count
    else:
        total_average = 0.0
```

- Die Calculate_average Funktion wird für jedes Fach durchgeführt, die Durchschnitte werden summiert und durch den Fach_count geteilt, um den gesamten Durchschnitt zu bekommen.

3. Simulation der nächsten Note (Feature):

- **Es soll berechnet werden, welche Note in der nächsten Prüfung mindestens erreicht werden muss, um noch eine genügende Durchschnittsnote (z. B. 4.0) im jeweiligen Fach zu haben.**

Diese Anforderung wird durch einen Javascript Skript im Frontend erfüllt. Ich habe auf den Backend verzichtet, aus folgenden Gründen:

1. Die Berechnung Im Frontend ermöglicht eine sofortige Aktualisierung der Daten in Echtzeit die angezeigt werden.
2. Keine sensible Daten und keine Verbindung zum Backend ermöglicht mir das Verzichten auf aufwendige Sicherheitsmassnahmen.

```
      <script>

        document.addEventListener("DOMContentLoaded", function () {
            const faecherData = {{ faecher_data | tojson | safe }};
            const fachDropdown = document.getElementById("fachDropdown");
            const desiredAverageInput =
document.getElementById("desiredAverage");
            const newWeightInput = document.getElementById("newWeight");
            const resultDisplay =
document.getElementById("minimumGradeResult");

            function calculateMinimumGrade(grades, weights, desiredAverage,
newWeight) {
                const totalWeightedSum = grades.reduce((sum, grade, i) => sum
+ grade * weights[i], 0);
                const totalWeight = weights.reduce((sum, weight) => sum +
weight, 0) + newWeight;

                const minimumGrade = (desiredAverage * totalWeight -
totalWeightedSum) / newWeight;
                return(minimumGrade);
            }

            function validateInputs(desiredAverage, newWeight) {
                if (isNaN(desiredAverage) || desiredAverage < 1 ||
desiredAverage > 6) {
                    return "Die gewünschte Durchschnittsnote muss zwischen 1
und 6 liegen!";
                }
                if (isNaN(newWeight) || newWeight <= 0) {
                    return "Die Gewichtung muss größer als 0 sein!";
                }
                return null;
            }
```

```
            function updateMinimumGrade() {
                const fachId = fachDropdown.value;
                const desiredAverage = parseFloat(desiredAverageInput.value);
                const newWeight = parseFloat(newWeightInput.value);


                const error = validateInputs(desiredAverage, newWeight);
                if (error) {
                    resultDisplay.textContent = error;
                    resultDisplay.style.color = "red";
                    return;
                }

                if (!fachId) {
                    resultDisplay.textContent = "Bitte ein Fach auswählen!";
                    resultDisplay.style.color = "red";
                    return;
                }


                const selectedFach = faecherData.find(fach => fach.id ==
fachId);
                const grades = selectedFach.noten.map(note => note.wert);
                const weights = selectedFach.noten.map(note =>
note.gewichtung);


                const minimumGrade = calculateMinimumGrade(grades, weights,
desiredAverage, newWeight);


                if (minimumGrade > 6.0) {
                    resultDisplay.textContent = "Die benötigte Note ist höher
als 6 und kann nicht erreicht werden.";
                    resultDisplay.style.color = "red";
                } else if (minimumGrade < 1.0) {
                    resultDisplay.textContent = "Die benötigte Note ist
kleiner als 1 und kann nicht erreicht werden.";
                    resultDisplay.style.color = "red";
                } else {
                    resultDisplay.textContent = `Die Mindestnote ist:
${minimumGrade.toFixed(3)}`;
                    resultDisplay.style.color = "black";
                }
            }


            fachDropdown.addEventListener("change", updateMinimumGrade);
            desiredAverageInput.addEventListener("input", updateMinimumGrade);
            newWeightInput.addEventListener("input", updateMinimumGrade);
```

```
            updateMinimumGrade();
        });
```

- **Die Pluspunkte sollen in Echtzeit berechnet werden beim Eingeben der Note und dem Nutzer angezeigt werden.**

  Dies wird ebenfalls in Javascript im Frontend in der handle.html datei gemacht, da es sich in der handle route befindet, wenn man eine Note hinzufügt oder bearbeitet. Gleiche Begründung wie vorher: Die Daten werden in Echtzeit berechnet. Hier gibt es aber jedoch eine separate Berechnung Im Backend als Sicherheitsmassnahme. Die Daten vom Frontend werden nicht ans Backend geschickt. Deswegen ist die Berechnung in Frontend und Backend separat, im Frontend dient sie nur als ein bequemes Feature für den User.

```
 document.addEventListener("DOMContentLoaded", function () {
            {% if type == "Note" %}
            const existingGrades = JSON.parse(decodeURIComponent("{{
existing_grades | tojson | urlencode }}")) || [];
            const existingWeights = JSON.parse(decodeURIComponent("{{
existing_weights | tojson | urlencode }}")) || [];
            const noteIndex = {{ note_index }};
            {% endif %}

            const wertInput = document.getElementById("wert");
            const gewichtungInput = document.getElementById("gewichtung");
            const pluspunkteField = document.getElementById("pluspunkte");

            function calculatePluspunkte(grades, weights, newGrade = null,
newWeight = null, index = -1) {
                const baseGrade = 4.0;
                let totalWeightedSum = 0.0;
                let totalWeight = 0.0;


                for (let i = 0; i < grades.length; i++) {
                    if (i === index && newGrade !== null && newWeight !==
null) {

                        totalWeightedSum += newGrade * newWeight;
                        totalWeight += newWeight;
                    } else {
                        totalWeightedSum += grades[i] * weights[i];
                        totalWeight += weights[i];
                    }
                }


                if (index === -1 && newGrade !== null && newWeight !== null) {
```

```
                    totalWeightedSum += newGrade * newWeight;
                    totalWeight += newWeight;
                }


                if (totalWeight === 0) {
                    return "Keine gültigen Daten";
                }


                const average = totalWeightedSum / totalWeight;


                const roundedAverage = Math.round(average * 2) / 2;


                if (roundedAverage >= baseGrade) {
                    return (roundedAverage - baseGrade).toFixed(2);
                } else {
                    return ((roundedAverage - baseGrade) * 2).toFixed(2);
                }
            }


            function updatePluspunkte() {
                const grade = parseFloat(wertInput?.value || 0);
                const weight = parseFloat(gewichtungInput?.value || 1);

                if (!isNaN(grade) && grade >= 1 && grade <= 6 &&
!isNaN(weight) && weight > 0) {

                    const calculated = calculatePluspunkte(
                        existingGrades,
                        existingWeights,
                        grade,
                        weight,
                        noteIndex
                    );
                    pluspunkteField.value = calculated;
                } else {
                    pluspunkteField.value = "Ungültige Eingabe";
                }
            }


            wertInput?.addEventListener("input", updatePluspunkte);
            gewichtungInput?.addEventListener("input", updatePluspunkte);


            updatePluspunkte();
```

```
        });
    </script>
```

- **Die Noten sollen pro Fach nach Datum sortiert werden können, entweder neuste oder alte zuerst werden.**

  Hier wird ebenfalls Javascript eingesetzt, und die Sortierung wird in der localstorage gespeichert, so dass wenn man in eine andere Route geht, oder die Seite refreshed, die Sortierung auch so bleibt wie sie war. Dieses Skript ermöglicht das Sortieren von Tabellen basierend auf der dritten Spalte, die ein Datum enthält. Nach dem Laden der Seite wird überprüft, ob bereits eine Sortierreihenfolge in localstorage gespeichert ist. Falls ja, wird diese angewendet. Die Funktion sortTable sammelt alle Zeilen der Tabelle, extrahiert das Datum aus der dritten Spalte, wandelt es in ein Date-Objekt um und sortiert die Zeilen entweder aufsteigend oder absteigend. Anschliessend werden die sortierten Zeilen wieder ins tbody eingefügt.
  Die Funktion applySortOrder stellt sicher, dass die gespeicherte Sortierreihenfolge für jede Tabelle direkt nach dem Laden der Seite angewendet wird. Dabei wird auch der entsprechende Sortierbutton visuell hervorgehoben. Die Event-Listener für die Buttons .sort-oldest und .sort-newest ermöglichen es dem Nutzer, die Sortierung manuell zu ändern. Wird ein Button geklickt, wird die Tabelle entsprechend sortiert, die Auswahl in localstorage gespeichert und die Buttons optisch aktualisiert. Dadurch bleibt die gewählte Sortierreihenfolge auch nach einem neuladen der Seite erhalten.

```
        document.addEventListener("DOMContentLoaded", function () {
        const localStorageKey = "sortOrder";
        let sortOrder = JSON.parse(localStorage.getItem(localStorageKey)) ||
{};

        function sortTable(table, ascending) {
            const rows = Array.from(table.querySelectorAll("tbody tr"));
            rows.sort((a, b) => {
                const dateA = new Date(a.cells[2].textContent.trim());
                const dateB = new Date(b.cells[2].textContent.trim());
                return ascending ? dateA - dateB : dateB - dateA;
            });

            rows.forEach(row =>
table.querySelector("tbody").appendChild(row));
        }

        function applySortOrder() {
            document.querySelectorAll("table").forEach(table => {
                const fachId = table.getAttribute("data-fach-id");
                const order = sortOrder[fachId] || "desc";

                const ascending = order === "asc";
                sortTable(table, ascending);
```

```
                const buttons = document.querySelectorAll(`[data-fach-
id="${fachId}"]`);
                buttons.forEach(btn => btn.classList.remove("active-sort"));
                const activeButton = document.querySelector(
                    `.sort-${ascending ? "oldest" : "newest"}[data-fach-
id="${fachId}"]`
                );
                if (activeButton) activeButton.classList.add("active-sort");
            });
        }

        document.querySelectorAll(".sort-oldest").forEach(button => {
            button.addEventListener("click", function () {
                const fachId = this.getAttribute("data-fach-id");
                const table = document.querySelector(`table[data-fach-
id="${fachId}"]`);
                sortTable(table, true);
                sortOrder[fachId] = "asc";
                localStorage.setItem(localStorageKey,
JSON.stringify(sortOrder));


                document.querySelectorAll(`[data-fach-
id="${fachId}"]`).forEach(btn => btn.classList.remove("active-sort"));
                this.classList.add("active-sort");
            });
        });

        document.querySelectorAll(".sort-newest").forEach(button => {
            button.addEventListener("click", function () {
                const fachId = this.getAttribute("data-fach-id");
                const table = document.querySelector(`table[data-fach-
id="${fachId}"]`);
                sortTable(table, false);
                sortOrder[fachId] = "desc";
                localStorage.setItem(localStorageKey,
JSON.stringify(sortOrder));


                document.querySelectorAll(`[data-fach-
id="${fachId}"]`).forEach(btn => btn.classList.remove("active-sort"));
                this.classList.add("active-sort");
            });
        });


        applySortOrder();
    });
```

```
    </script>
</body>
{% endif %}
</html>
```

4. **Löschen von Noten (Update/Delete):**

- **Die Anwendung muss es ermöglichen, Noten und Fächer zu löschen**

```
@app.route("/<string:type>/delete/<int:item_id>")
def delete_item(type, item_id):
    if type == "fach":

        fach = Fach.query.filter_by(id=item_id,
user_id=current_user.id).first()
        if not fach:
            return "Unauthorized", 403


        db.session.delete(fach)

    elif type == "note":


        note = Note.query.get_or_404(item_id)
        if note.fach.user_id != current_user.id:
            return "Unauthorized", 403


        db.session.delete(note)

    else:
        return "Invalid type", 400

    db.session.commit()
    return redirect(url_for('index'))
```

Hier sind die user.ids enthalten, für das Loginsystem, das ich in meiner Freizeit gemacht habe, jedoch ist das Prinzip des Löschens sehr einfach. Durch das Cascade in der Klasse Fach werden alle zugehörige Noten des Fachs auch gelöscht, und man kann auch die Noten einzeln löschen, durch queries und db.session.delete.

Reflexion

Kurz gesagt bin ich mit diesem Projekt sehr zufrieden, da es meiner Meinung nach auf dem Level von ähnlichen Notenverwaltung-Apps auf dem App Store ist, von denen einige man sogar kaufen muss. Ich habe sehr vieles gelernt, sei das von dem Referenzieren 2 verschieden Klassen, in diesem Fall genauer gesagt 2 Tabellen (Fach und Note) mithilfe von Beziehungen und Primärschlüsseln. Ich habe gelernt, wie man Parameter wie Backref, cascade und lazy benutzt.

```
class Fach(db.Model):
    __tablename__ = 'faecher'
    id = db.Column(db.Integer, primary_key=True)
    name = db.Column(db.String(255), nullable=False)
    noten = db.relationship('Note', backref='fach', lazy=True, cascade="all,
delete")
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
class Note(db.Model):
    __tablename__ = 'noten'
    id = db.Column(db.Integer, primary_key=True)
    wert = db.Column(db.Float, nullable=False)
    datum = db.Column(db.Date, nullable=False)
    gewichtung = db.Column(db.Float, default=1.0)
    fach_id = db.Column(db.Integer, db.ForeignKey('faecher.id'),
nullable=False)
    user_id = db.Column(db.Integer, db.ForeignKey('users.id'), nullable=False)
```

Mir wurden die Beziehungen zwischen Frontend und Backend viel klarer. Ich habe auch viele neue Erkenntnisse in JavaScript (Vor allem in dem Syntax von Javascript, und auch die Vorteile von Better Jinja) und den Design von Websites machen können. Besonders stolz bin ich auf die Sortierfunktion im Frontend, da ich dort am meisten Zeit und Aufwand eingesteckt habe. Dort habe ich auch am meisten gelernt, auch wenn ich manchmal an der Grenze der Verzweiflung war. Die Arbeit hat mir aber schlussendlich sehr Spass gemacht, und darum habe ich mir auch zusätzliche Ziele gesetzt und sehr viel in meiner Freizeit an dieser WebApp gearbeitet.

Was ich besser machen könnte, ist definitiv meinen Code besser zu ordnen, und Kommentare hinter zulassen. Wo ich nach 2 und ein halb Wochen, also nach den Ferien, mir den Code angeschaut habe, hatte ich keine Ahnung mehr was wo war und wieso alles funktionierte. Dies war teilweise eben durch die Chaotische Anordnung des Codes, da ich mir beim Implementieren neuen Features nicht viel gedacht habe, wo ich den Code einsetzen soll, sondern einfach losgeschrieben habe. Zweitens hätten eben auch Kommentare geholfen.

Darüber hinaus wäre es interessant, diese WebApp in eine tatsächliche Webseite umzusetzen, wo jeder sich einloggen kann und seine eigene Notenliste hat. Ich werde definitiv mich noch da informieren da es eine gute Art «Vertiefungsarbeit» wäre.

Quellen

Dieses Projekt wurde selbstständig erstellt, ausser das Login System und die Notenextrahierung, wo ich vermehrt ChatGPT benutzt habe. Diese Funktionen waren aber nicht in meinen Anforderungen und wurden aus Interesse in der Freizeit programmiert, um mein Wissen mehr zum Vertiefen.

Natürlich habe ich ChatGPT auch vermehrt zum Debugging und Säuberung des Codes benutzt, wenn ich nicht mehr drausgekommen bin, sowohl wie zur Logik von z.b Notenberechnung, da ich dort oft Bugs hatte, und auch als Helfer für die Frontend-Skripte. Bei den Javascript Skripten und vor allem HTML habe ich oft auch kleinere Code Ausschnitte oder Informationen aus Foren online geholt. Auch wurde ChatGPT für die Kosmetik der Website, vor allem das CSS (Bootstrap) benutzt. Durch ChatGPT bin ich auch überhaupt auf die Extension Better Jinja gestossen, da ich Syntax-Fehler in den HTML-Dateien hatte und es Better Jinja vorgeschlagen hat. ChatGPT wurde aber viel mehr als ein Assistent und Lehrmittel benutzt, wo ich gefragt habe, wie ich mit dem Programmieren vorgehen soll, was ich alles brauche usw., wenn ich mir unsicher war, um Zeit und Nerven zu sparen.