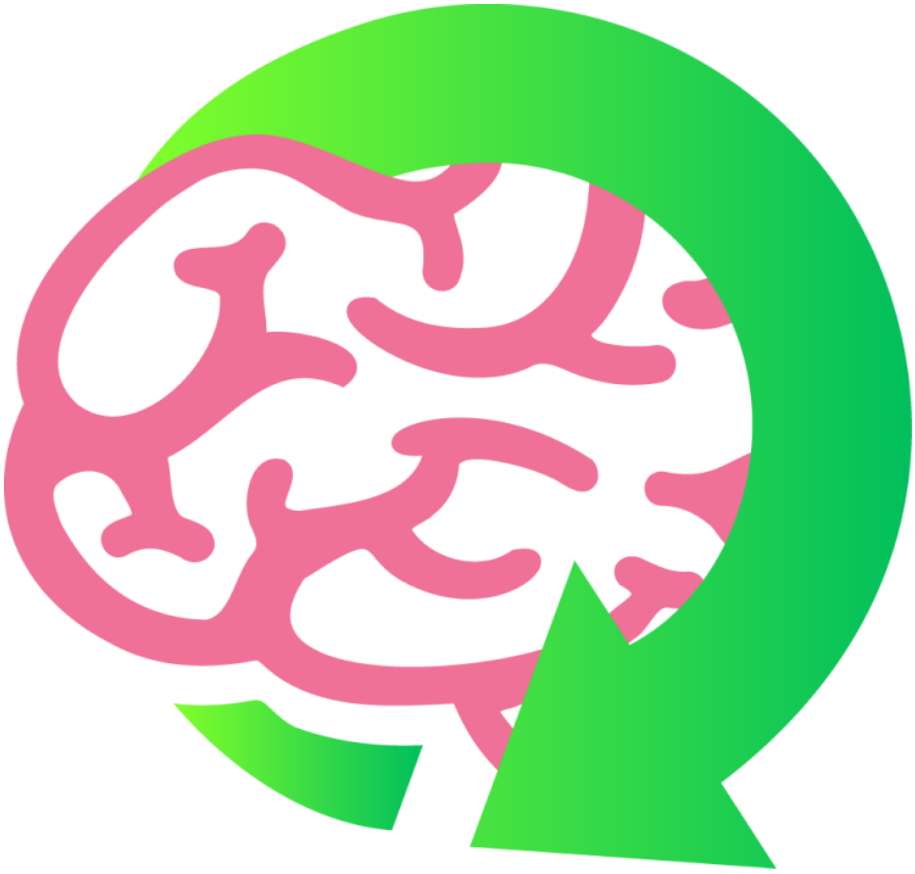
Chunking Interval Learning



Projekt erstellt von Anton Guliaev

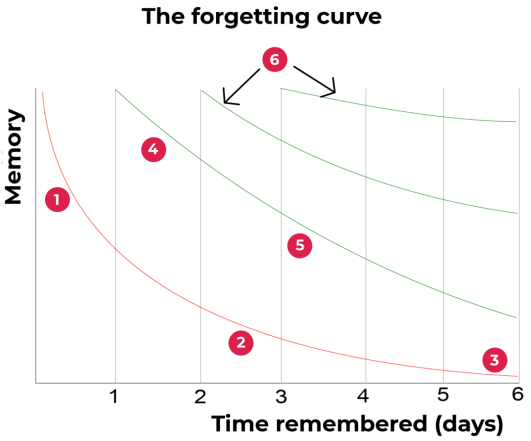
GitHub-Link

Chunking Interval Learning

Motivation und Anforderungen

Motivation

Die Idee für dieses Projekt entstand vor etwa vier Jahren, als ich während des Abiturs zum ersten Mal von der Methode des Intervalllernens von **Ebbinghaus** erfahren habe. Diese Methode, die darauf abzielt, die Information über regelmäßige Abstände hinweg zu wiederholen, um das **Long-term memory (LTM)** zu verbessern.



Das **Ebbinghaus Vergessenskurve**[1] ist eine Lernmethode, die auf den Forschungen des deutschen Psychologen Hermann Ebbinghaus basiert. Seine Ergebnisse besagen grob, dass wir bereits **20** Minuten nach dem Lernen nur noch **60 %** des Gelernten abrufen können. Nach einer Stunde sind nur noch **45** % und nach einem Tag gar nur **34** % des Gelernten im Gedächtnis. Sechs Tage nach dem Lernen wiederum ist das Erinnerungsvermögen bereits auf **23** % geschrumpft; dauerhaft werden nur **15** % des Erlernten gespeichert.

In einem **Experiment**[2] mit 250 Studenten, die Konzepte der Immunologie und Reproduktionsphysiologie lernten, zeigte sich, dass das Lernen mit erweiternden Intervallen (Tage 1, 6, 16 ohne Verzögerung) effektiver ist als das Lernen in konstanten Abständen. Die Gruppen, die erweiternde Intervalle nutzten, erinnerten sich signifikant besser an Informationen am 29. Tag als diejenigen, die konstante Intervalle verwendeten. Die durchschnittliche Punktzahl der Gruppe mit erweiternden Intervallen war **42,57 (SD 1,8)** im Vergleich zu **34,1 (SD 1,36)** der Gruppe mit konstanten Intervallen. Im Vergleich zur Kontrollgruppe **(21,26, SD 1,4)** bestätigen die Ergebnisse, dass die Praxis des Wiederabrufens ein Hauptbeitrag zum erfolgreichen Lernen ist und dass das Lernen in erweiternden Intervallen das Lernen weiter verbessert.

Vor einem halben Jahr wurde diese Idee weiterentwickelt, nachdem ich über die **Chunking-Methode**[3] in einem Podcast gehört habe. Die Idee der **Chunking-Methode** basiert auf kognitiven Psychologie Forschungen und ist eng verbunden mit dem Konzept der Arbeitsgedächtniskapazität. Der Begriff "Chunking" wurde in den 1950er Jahren vom Psychologen George A. Miller in seinem Artikel **"The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information"**[4] geprägt. Miller stellte fest, dass das menschliche Gedächtnis Informationseinheiten (oder "Chunks") effizienter verarbeiten und speichern kann, wenn diese in kleinere, bedeutungsvolle Gruppen unterteilt werden.

Die **Kombination** dieser **beiden** Techniken inspirierte mich dazu, eine App zu entwickeln, die diese Ansätze implementiert, um das Lernen von Fremdsprachenwörtern leichter zu machen und zu optimieren.

Anforderungen

Hier sind die Wichtigsten Prinzipien, die am besten für meine App passen, aus **"Software Engineering"**[5] von Jochen Ludewig und Horst Lichter Teil.1. Grundlagen - 5. Software-Qualität.

Für Users:

- **Benutzerfreundlichkeit:** Die App soll so gestaltet sein, dass sie einfach und intuitiv für Users ist.
- **Effizienz:** Durch die Kombination von Intervalllernen und Chunking soll die App das Lernen von Fremdwörtern verbessern
- **Verständlichkeit**: Die Funktionen sind leicht zu begreifen.

Für Entwicklern:

- **Lesbarkeit**: Der Code ist leicht verständlich.
- **Simplizität**: Einfachheit der Struktur.

Planung und Entwurf

Agile:

Obwohl ich allein den Projekt implementiert habe, habe ich trotzdem Agile Methodologien in manchen Aspekten verwendet.

Ich habe meine Arbeit in **kleine, handhabbare Aufgaben** unterteilt, die ich in kurzen Zeitabschnitten, sogenannten **Sprints**, abgeschlossen habe (sie haben meistens 2-3 Tagen anstatt einer Woche gedauert).

Ich konzentrierte mich darauf, die **wichtigsten Funktionen** meiner App zuerst zu entwickeln, was mir half, schnell einen funktionierenden **Prototyp** zu erstellen. Habe zuerst Frontend und Backend, ohne die zu verknüpfen, was mir danach die Zeit gespart hatte.

Durch die Anwendung dieser agilen Prinzipien konnte ich effizient und zielgerichtet an meinem Projekt arbeiten, auch ohne ein Team im Rücken.

Programmieren Sprachen:

Für meinen Projekt werde ich **Python**[6] und **JavaScript**[7] benutzen, zwei sehr beliebte und vielseitige Sprachen, die sich durch ihre einzigartigen Stärken und Einsatzgebiete auszeichnen. **Python** ist einfach zu **lernen** und zu **verwenden**, und gut für **Backend** passt. **JavaScript** ist **universell**, weil JavaScript läuft auf fast allen modernen Webbrowsern ohne die Notwendigkeit für Plugins oder Erweiterungen, und hat **reichhaltiges Ökosystem** mit einer riesigen Auswahl an Bibliotheken und Frameworks.

Entwicklungsumgebung:

Genutzt wird integrierte Entwicklungsumgebung (IDE) - **Visual Studio Code**[8], die Unterstützung für Vue.js, Typescript und Flask bietet. Diese IDE ist leistungsstarkes Tool zur Codevervollständigung, zum Debugging und zur Versionskontrolle, was die Entwicklungsprozesse beschleunigen wird.

Frameworks:

Frontend: Wird verwendet **Vue.js**[9], ein progressives JavaScript-Framework, das für seine Einfachheit und Flexibilität bekannt ist. Dies ist die beste Wahl für kleinen Projekten. Vue hilft, interaktive und dynamische Benutzeroberflächen leicht zu erstellen. Mit **Vuex**[10] für den Zustandsmanagement und **Vue-**

Router[11] für die Navigation zwischen den Seiten kann eine reiche Benutzererfahrung gebeten werden. Die Nutzung von **Typescript**[12] ermöglicht es, den Code sicherer und einfacher zu warten zu machen.

Backend: Für das Backend wird **Flask**[13], ein leichtgewichtiges Python-Framework, das uns die Flexibilität bietet, **RESTful APIs** zu entwickeln, verwendet. Dies ist wichtig für die Kommunikation zwischen Frontend und dem Backend. **Flask-Cors**[14] wird verwendet, um Cross-Origin Resource Sharing zu ermöglichen, damit die Frontend-App auf die im Backend gehosteten Ressourcen zugreifen kann.

Verwendete Bibliotheken:

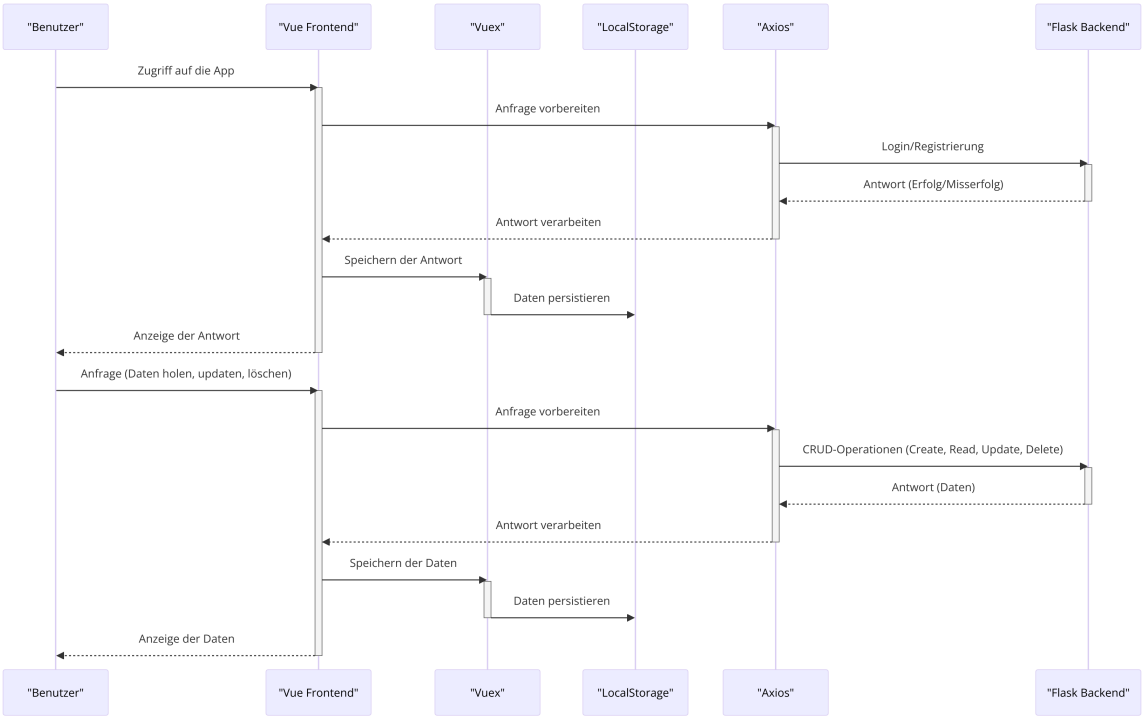
Axios[15] wird im Frontend für HTTP-Anfragen verwendet, um mit dem Backend zu kommunizieren. Es ist einfach zu verwenden und unterstützt Promises, was den asynchronen Code sauber und lesbar macht.

SQLAlchemy[16] ist unsere Wahl für das ORM (Object-Relational Mapping) im Backend, um die Datenbankoperationen zu vereinfachen.

Version Control:

Versionskontrolle, ist ein wichtiger Teil der Softwareentwicklung. Es hilft dabei, Änderungen am Code zu verfolgen, zu verwalten und notfalls Änderungen rückgängig zu machen. Für den Projekt habe ich mich für **Git**[17] und **GitHub**[18] entschieden, um diesen Prozess zu erleichtern.

Entwurf vom Ablauf der Kommunikation zwischen einem Benutzer und App:



1. **Benutzerzugriff:** Der Benutzer öffnet die App und startet eine Interaktion.
2. **Vue Frontend:** Das Frontend, erstellt mit Vue.js, ist die Schnittstelle, die der Benutzer sieht und mit der er interagiert.
3. **Vuex:** Wird verwendet, um den Zustand der Anwendung zu verwalten. Es hilft dabei, die Daten konsistent zu halten und reagiert auf Benutzeraktionen.

4. **LocalStorage:** Dies ist eine Möglichkeit, Daten im Browser des Benutzers zu persistieren. Hier werden Daten zwischen den Sitzungen erhalten, um nicht nach jedem Update mit dem Server zu kommunizieren.
5. **Axios:** Es bereitet die Anfragen vor und verarbeitet die Antworten.
6. **Anfragen für Login/Registrierung:** Der Benutzer kann sich anmelden oder registrieren. Diese Anfrage wird von Axios vorbereitet und an das Backend gesendet.
7. **Flask Backend:** Das Backend, entwickelt mit Flask, ist verantwortlich für die Verarbeitung der Anfragen. Es führt die Login- oder Registrierungsvorgänge und CRUD-Operationen durch und gibt eine Antwort, ob es erfolgreich war oder nicht, mit der Daten (wenn es notwendig ist) zurück.
8. **CRUD-Operationen:** CRUD steht für "Create, Read, Update, Delete". Diese Operationen sind grundlegende Funktionen, die das Backend ausführt, um Daten zu verarbeiten. Axios bereitet diese Anfragen vor und das Backend führt sie aus.
9. **Antwortverarbeitung:** Axios empfängt die Antwort mit der Daten vom Backend und verarbeitet sie.
10. **Antworten speichern und anzeigen:** Nachdem das Backend eine Antwort gesendet hat, wird diese im Vuex gespeichert und kann im LocalStorage für spätere Sitzungen persistiert werden. Schließlich zeigt das Vue Frontend die verarbeiteten Daten dem Benutzer an.

Entwicklung

In meinem Projekt habe ich mich auf die Entwicklung und Integration der drei wichtigsten Funktionen konzentriert, diese Funktionen sind die Basis für die Idee, die ich in das Projekt eingebracht habe.

Login und Registrierung:

Der erste Schritt für Benutzer meiner App ist die Möglichkeit, sich zu **registrieren** und **einzuloggen**. Kommunikation geschieht durch Asynchrone Anfragen mit **Axios** benutzend **Routen** von **Flask**

- **Login** erfolgt durch eine asynchrone Anfrage an Server. Bei erfolgreicher Authentifizierung wird die Benutzerdaten sowohl im lokalen Speicher als auch im Vuex-Store gespeichert und der Benutzer wird dann auf die Hauptseite umgeleitet. Falls die Authentifizierung fehlschlägt, wird eine Fehlermeldung angezeigt, die darauf hinweist, dass der Benutzer nicht gefunden wurde.

```
async login() {
  try {
    const response = await axios.post('http://localhost:5000/login', {
      username: this.username,
      password: this.password,
    });
    this.setStore(response.data.user, response.data.cards);
    this.setLocalStorage(response.data.user, response.data.cards);
    this.$router.push('/');
  } catch (error) {
    this.userNotFound = true;
  }
}
```

```
@app.route("/login", methods=["POST"])
def login():
    username = request.json.get("username")
    password = request.json.get("password")
    user = User.query.filter_by(name=username).first()
    if user is None or not user.check_password(password):
        return jsonify({"message": "Invalid username or password"}), 401
    cards = Card.query.filter_by(user=user).all()
    cards_to_response = {}
    for id, card in enumerate(cards):
        cards_to_response[id] = {
            "card_id": card.id,
            "front": card.front,
            "back": card.back,
            "labels": card.labels,
            "next_review": card.next_review,
            "repeat_count": card.repeat_count,
        }
    return jsonify({"user": user.name, "cards": cards_to_response}), 200
```

- **Registrierungsprozess** erfolgt ebenfalls über eine **asynchrone Anfrage** an den **Server**. Nach erfolgreicher Registrierung wird der Benutzer zur **Anmeldeseite** umgeleitet. Auf dem Server bei der Registrierung wird ein neuer Benutzer angelegt, sofern der Benutzername noch nicht existiert, ansonsten wird das Fehler an Benutzer zurückgegeben.

```vue
async register() {
  try {
    await axios.post('http://localhost:5000/register', {
      username: this.username,
      password: this.password,
    });
    this.$router.push('/login');
  } catch (error: any) {
    console.log(error);
    if (error.response.status === 409) {
      this.userAlreadyExists = true;
    }
  }
}
```

```python
@app.route("/register", methods=["POST"])
def register():
    username = request.json.get("username")
    password = request.json.get("password")

    user = User(username, password)

    if db.session.query(User.id).filter_by(name=username).scalar() is not None:
        return jsonify({"message": "User already exists"}), 409

    db.session.add(user)
    db.session.commit()

    return jsonify({"message": "User created successfully"}), 201
```

Hinzufügen und Löschen von Flashcards:

Das Hinzufügen und Löschen von Lernkarten ist eine Kernfunktion meiner Anwendung, die den Benutzern ermöglicht, ihren Lernprozess aktiv zu gestalten.

- **Hinzufügen von Lernkarten:** Benutzer können neue Lernkarten über ein Formular hinzufügen, in dem sie die **Vorder**- und **Rückseite** der Karte sowie eventuelle **Kategorien** (Labels) angeben. Sobald die Daten eingegeben sind, wird eine **asynchrone Anfrage** an den **Server** gesendet, um die neue Karte mit einer **eindeutigen ID**, den **Inhaltsangaben**, **Benutzernamen** und dem nächsten **Überprüfungsdatum** zu speichern. Die Karte wird im **Vuex-Store** hinzugefügt, um den Zustand der Anwendung zu aktualisieren. Dies ermöglicht eine **sofortige Anzeige** der neuen Karte in der **Benutzeroberfläche** ohne **Neuladen** der Seite.
  Auf dem
  Server werden die Daten einer neuen Karte entgegengenommen, überprüft und in der **Datenbank** gespeichert

```vue
async addCard() {
  try {
    const card_id =
      this.cards.cards.length === 0 ? 1 : this.cards.cards.length + 1;
    const front = this.front;
    const back = this.back;
    const labels = this.convert_labels();
    const username = this.store.state.user['user'];
    const next_review = new Date().toISOString();

    await axios.post('http://localhost:5000/cards', {
      card_id,
      front,
      back,
      labels,
      username,
      next_review,
    });
    this.store.commit('addCard', {
      card_id,
      front,
      back,
      labels,
      username,
      next_review,
      repeat_count: 0,
    });
    (this.front = ''), (this.back = ''), (this.labels = '');
  } catch (error) {
    console.log(error);
  }
}
```

```python
@app.route("/cards", methods=["POST"])
def add_card():
    front = request.json.get("front")
    back = request.json.get("back")
    labels = request.json.get("labels")
    username = request.json.get("username")
    next_review = request.json.get("next_review")

    labels = "".join(labels)

    if not all([front, back, labels, username, next_review]):
        return jsonify({"message": "Missing data"}), 400

    user = User.query.filter_by(name=username).first()
    if not user:
        return jsonify({"message": "User not found"}), 404

    card = Card(
        user_id=user.id, front=front, back=back, labels=labels, next_review=next_review
    )

    db.session.add(card)
    db.session.commit()

    return jsonify({"message": "Card created successfully"}), 201
```

- **Löschen von Lernkarten:** Benutzer können jede Lernkarte aus ihrer Sammlung entfernen. Eine **asynchrone Anfrage** wird an den **Server** gesendet, um die spezifizierte Karte basierend auf ihrer **ID** und dem **Benutzernamen** zu löschen. Nach erfolgreicher Löschung wird die Karte aus dem **Vuex-Store** entfernt, was die Benutzeroberfläche aktualisiert und die Karte nicht mehr anzeigt.
  Die Löschung erfolgt ebenfalls
  serverseitig, wobei die Karte basierend auf ihrer **ID** und dem Benutzernamen identifiziert wird. Nach erfolgreicher Löschung wird eine **Bestätigungsnachricht** zurückgegeben.

```vue
  async deleteCard(cardId: number, username: string) {
    try {
      console.log(`Deleting card with id ${cardId} for user ${username}`);
      await axios.delete('http://localhost:5000/cards', {
        data: {
          card_id: cardId,
          username: username,
        },
      });
      this.store.commit('deleteCard', cardId);
    } catch (error) {
      console.log(error);
    }
  }
}
</script>
```

```python
@app.route("/cards", methods=["DELETE"])
def delete_card():
    card_id = request.json.get("card_id")
    username = request.json.get("username")

    if not all([card_id, username]):
        return jsonify({"message": "Missing data"}), 400

    user = User.query.filter_by(name=username).first()
    if not user:
        return jsonify({"message": "User not found"}), 404

    card = Card.query.filter_by(user_id=user.id, id=card_id).first()
    if not card:
        return jsonify({"message": "Card not found"}), 404

    db.session.delete(card)
    db.session.commit()

    return jsonify({"message": "Card deleted successfully"}), 200
```

Lernen und Filtern von Lernkarten:

- **Update der Lernkarten:** Benutzer interagieren mit ihren Lernkarten durch einen Überprüfungsprozess, bei dem sie die Karte als "**bekannt**" oder "**unbekannt**" markieren können. Der Überprüfungszeitpunkt einer Karte wird basierend auf der **Anzahl der Wiederholungen** angepasst, wobei jede **erfolgreiche Wiederholung** den nächsten **Überprüfungszeitpunkt** weiter in die **Zukunft** verschiebt. Dies folgt dem Prinzip des **Intervalllernens**, wobei die Abstände zwischen den Wiederholungen progressiv vergrößert werden.
  Die Logik zur Festlegung des nächsten Überprüfungsdatums basiert auf der Anzahl der Wiederholungen: von einer Stunde nach der ersten Wiederholung bis zu mehreren Tagen nach mehreren Wiederholungen. Dies fördert das langfristige Behalten der Informationen. Bei jeder Überprüfung wird die
  `repeat_count` der Karte aktualisiert und das nächste Überprüfungsdatum neu berechnet und sowohl im **Vuex-Store** als auch auf dem **Server** aktualisiert.
  Wenn eine Karte als "
  bekannt" markiert wird, aktualisiert der Server die Karte in der **Datenbank** mit dem neuen Überprüfungsdatum und der aktualisierten Wiederholungsanzahl.

```vue
  markKnown(known: boolean) {
    const updateBasedOnRepeat: { [key: number]: number } = {
      0: 1,
      1: 8,
      2: 24,
      3: 64,
      4: 192,
    };

    const next_review = new Date(
      new Date().getTime() +
      updateBasedOnRepeat[this.currentCard.repeat_count] * 60 * 60 * 1000
    );

    const repeat_count = this.currentCard.repeat_count + 1;

    if (known) {
      this.store.commit('updateCard', {
        card_id: this.currentCard.card_id,
        next_review,
        repeat_count,
      });

      axios.put('http://localhost:5000/cards', {
        card_id: this.currentCard.card_id,
        username: this.store.state.user['user'],
        next_review: next_review,
        repeat_count: repeat_count,
      });

      if (this.currentCardIndex == this.cards.length - 1) {
        this.$router.push('/');
      }
    }
    this.nextCard();
  }
```

```python
@app.route("/cards", methods=["PUT"])
def updateCard():
    card_id = request.json.get("card_id")
    username = request.json.get("username")
    next_review = request.json.get("next_review")
    repeat_count = request.json.get("repeat_count")

    if not all([card_id, username, next_review, repeat_count]):
        return jsonify({"message": "Missing data"}), 400

    user = User.query.filter_by(name=username).first()
    if not user:
        return jsonify({"message": "User not found"}), 404

    card = Card.query.filter_by(user_id=user.id, id=card_id).first()
    if not card:
        return jsonify({"message": "Card not found"}), 404

    card.next_review = next_review
    card.repeat_count = repeat_count
    db.session.commit()

    return jsonify({"message": "Card updated successfully"}), 200
```

- **Kategorisierung:** Die Karten werden nach Kategorien (**Labels**) gefiltert und danach werden basierend auf ihren Labels gruppiert, um das gezielte Lernen innerhalb spezifischer Themenbereiche zu erleichtern. Dies folgt dem Prinzip des **Chunking-Method.** Die gruppierte Karten können danach in eine Reihenfolge basierend auf Label gelernt werden, was das Lernen von Fremdwörter effizienter macht.

```
41  get cards(): Card[] {
42    return this.store.state.cards.cards;
43  }
44
45  get labels(): { [key: string]: number } {
46    const labels: { [key: string]: number } = {};
47    for (const card of this.cards) {
48      if (labels[card.labels]) {
49        labels[card.labels]++;
50      } else {
51        labels[card.labels] = 1;
52      }
53    }
54    return labels;
55  }
56
57  get groupedCards(): Card[] {
58    const groupedCards: [Card[]] = [[]];
59    for (const label in this.labels) {
60      const cards = this.cards.filter((card: Card) => {
61        return card.labels === label;
62      });
63      groupedCards.push(cards);
64    }
65    return groupedCards.flat();
66  }
```

Technische Herausforderungen

Bei der Entwicklung des Projekts traten verschiedene technische Herausforderungen auf.

Eine der schwierigsten Herausforderungen war die **Optimierung der Kommunikation** zwischen dem **Server** (**Backend**) und den **Benutzern** (**Frontend**). Es gab ständige Fehlermeldungen, die durch falsch übertragene Daten verursacht wurden, falsch gespeicherte Daten oder die IDs der Karten, die wie durch Magie verschwanden. Dies führte dazu, dass die Karten nicht gelöscht oder manipuliert werden konnten.

Ein weiteres Problem war die **Beibehaltung der Benutzerdaten** und **des Status beim Neuladen der Seite**. Um dieses Problem zu lösen, wurde **LocalStorage**[19] eingesetzt. **LocalStorage** ermöglicht es, die Daten im **JSON-Format zu speichern**, jedes Mal wenn Änderungen am Zustand der Anwendung vorgenommen werden. Zusätzlich wurde der Login-Status des Benutzers gespeichert, um zu bestimmen, wohin der Benutzer beim Öffnen der Anwendung geleitet werden soll. Bei jeder `mutation` Funktion (Änderung im State) in Vuex wurde diese Methode implementiert `localStorage.setItem('cards', JSON.stringify(state.cards));`

Inbetriebnahme

Um die Anwendung zu starten, kann die Anleitung auf GitHub genutzt werden, doch zusätzlich wird sie hier bereitgestellt.

Voraussetzungen

Bevor das Projekt gestartet wird, müssen alle erforderlichen Abhängigkeiten für das Backend und das Frontend installiert werden. Detaillierte Anweisungen zur Installation dieser Abhängigkeiten finden sich in den jeweiligen README-Dateien in den Ordnern `backend` und `frontend`.

Installation

Das Repository soll auf den lokalen Rechner geklont werden: `git clone https://github.com/nmkzzztos/Chunking-Interval-Learning`

Die Installationsanweisungen für die **Backend**- und **Frontend**-Abhängigkeiten, wie in ihren jeweiligen README-Dateien erwähnt, sollen befolgt werden.

Ausführen des Projekts

Nach der Installation aller notwendigen Abhängigkeiten:

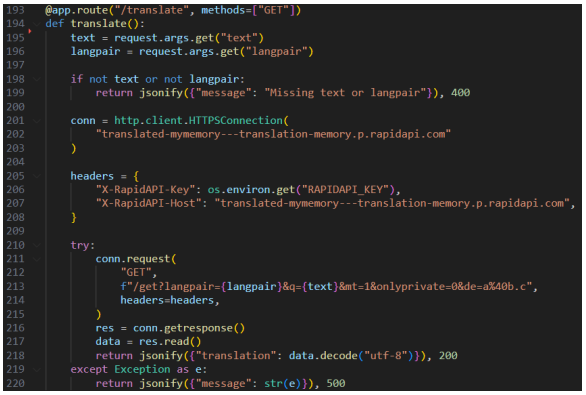
- Um den **Backend-Server** zu starten und die **Frontend-Anwendung** zu starten, soll einfach der Befehl `./run.cmd` ausgeführt werden:
- Oder sie können selbst gestartet werden:
  1. Zum Starten des Backend-Servers soll zum Ordner `backend` navigiert werden und der Befehl `python app/app.py` ausgeführt werden.
  2. Zum Starten der Frontend-Anwendung soll ein neues Terminalfenster geöffnet und zum Ordner `frontend` navigiert werden und der Befehl `npm run serve` ausgeführt werden.

Fazit

Zum Abschluss des Projekts kann ich feststellen, dass alle geplanten Funktionen erfolgreich implementiert wurden, wodurch die Anwendung ihre Aufgaben effizient erfüllt. Die bewährte Lernmethoden wie **Flashcards**, **Intervalllernen** und der **Chunking-Methode** wurden implementiert, damit die Benutzern effizient Fremdsprachen lernen können.

Für die Zukunft des Projekts gibt es spannende **Möglichkeiten zur Erweiterung** der Anwendung:

1. **Automatischer Übersetzungsservice beim Hinzufügen von Karten:** Benutzer werden die Möglichkeit haben, den Text, den sie hinzufügen möchten, **automatisch** durch die Nutzung eines bereits implementierten **API**[20] in eine von ihnen gewählte Sprache **übersetzen** zu lassen.



```python
@app.route("/translate", methods=["GET"])
def translate():
    text = request.args.get("text")
    langpair = request.args.get("langpair")

    if not text or not langpair:
        return jsonify({"message": "Missing text or langpair"}), 400

    conn = http.client.HTTPSConnection(
        "translated-mymemory---translation-memory.p.rapidapi.com"
    )

    headers = {
        "X-RapidAPI-Key": os.environ.get("RAPIDAPI_KEY"),
        "X-RapidAPI-Host": "translated-mymemory---translation-memory.p.rapidapi.com",
    }

    try:
        conn.request(
            "GET",
            f"/get?langpair={langpair}&q={text}&mt=1&onlyprivate=0&de=a%40b.c",
            headers=headers,
        )
        res = conn.getresponse()
        data = res.read()
        return jsonify({"translation": data.decode("utf-8")}), 200
    except Exception as e:
        return jsonify({"message": str(e)}), 500
```

2. **Integration eines API von OpenAI**[21] **für die Texterstellung:** Eine weitere Idee ist die Nutzung eines API von **OpenAI**, um beim Lernen von Wörtern **kleine Texte zu generieren**. Ein Hauptvorteil der Anwendung – das **Chunking-Method**, also das Zusammenfassen von Wörtern nach Labels – ermöglicht es, in kurzer Zeit mehr Informationen zu lernen. Durch die Anfrage bei einem **Large Language Model (Chat GPT)** könnten kurze, zusammenhängende Sätze generiert werden, die den Benutzern ein besseres Verständnis und eine Nutzung der Wörter bieten. Dies könnte potenziell auch das Merken der Informationen entwickeln, obwohl es hierzu noch keine eindeutigen Studien (in **PubMed** habe ich keine gefunden) gibt.
3. **Einführung einer neuen View für die Bearbeitung statt Löschen:** Dies ermöglicht es den Benutzern, mit ihren erstellten Karten effizienter zu arbeiten, indem sie diese bearbeiten, weitere Labels hinzufügen oder Fehler korrigieren können.

References

1. Die Vergessenskurve von dem deutschen Psychologen Hermann Ebbinghaus
2. How to learn effectively in medical school: test yourself, learn actively, and repeat in intervals
3. Chunking (Psychologie) von George A. Miller
4. The magical number seven, plus or minus two: Some limits on our capacity for processing information
5. Software Engineering, 3rd Edition by Jochen Ludewig, Horst Lichter
6. Python
7. JavaScript
8. Visual Studio Code
9. Vue JS (The Progressive JavaScript Framework)
10. Vuex (State Management)
11. Vue Router (The official Router for Vue.js)
12. TypeScript
13. Flask
14. Flask-Cors (Cross Origin Resource Sharing)
15. Axios - GitHub
16. SQLAlchemy (Python SQL toolkit and Object Relational Mapper)
17. Git
18. GitHub
19. LocalStorage
20. MyMemory is the world's largest Translation Memory
21. OpenAI API