TYPINGSTORIES

Worte verbinden Welten.

EXPOSEE

Eine Webanwendung, die das Verfassen und Teilen von Geschichten mit einem integrierten Schreibtrainer kombiniert.

Praewphan Jamnongnok

INHALTSVERZEICHNIS

1 EINFÜHRUNG / PROJEKTIDEE

Projekttitel: TypingStories

Motto: *«Words connect worlds»*

TypingStories ist eine Webapplikation, die das kreative Schreiben von Geschichten mit gezieltem Tipptraining kombiniert. Die Plattform ermöglicht es Nutzern, eigene Texte online zu verfassen, zu speichern und zu teilen. Jede Geschichte kann dabei mit strukturierten Bewertungen versehen werden, zum Beispiel in Kategorien wie Spannung, Kreativität oder Lesbarkeit. Diese Scores helfen die User dabei, gezieltes Feedback zu bekommen und ihre Texte zu verbessern.

Das Besondere an TypingStories ist der integrierte Typing-Trainer: jede gespeicherte Geschichte direkt in einem Übungsmodus abtippen. Dabei misst die App die Tippgeschwindigkeit(WPM) und die Fehlerquote. Diese Ergebnisse werden gespeichert und können vom Nutzer eingesehen werden. So wird der Inhalt der Community gleichzeitig zu persönlichen Lernressource. Ziel ist es, eine Plattform zu schaffen, auf der Schreiben, Lesen Bewerten und das Trainieren von Schreibfertigkeiten nahtlos ineinandergreifen.

1.1 ZWECK UND MOTIVATION

Die App soll kreative Menschen motivieren, mehr zu schreiben und ihre Texte qualitativ zu verbessern. Gleichzeitig bietet sie einen niederschwelligen Ansatz, die eigene Tippgeschwindigkeit und -genauigkeit zu steigern. Gerade in Schule, Ausbildung oder Beruf sind diese Fähigkeiten wichtig. TypingStories richtet sich an Hobby-Autoren, Lernende, Schreibgruppen, Lehrer und alle, die Freude an Sprache und Schreiben haben.

1.2 ZENTRALE MERKMALE

- CRUD-Storyverwaltung
- Rest-API
- Strukturierte Scores/Bewertungen pro Story
- Lesemodus
- Typing-Übungsmodus
- WPM- und Fehleranalyse
- Speichern und Verwalten von Übungsergebnissen

2 ANFORDERUNGSKATALOG

- Übersicht der Kernfeatures
    - Geschichten erstellen, lesen, aktualisieren und löschen.
    - Stories werden in Genres organisiert
    - Jede Story kann mit Scores bewertet werden
    - Stories können im Tipptrainer-Modus abtippen
    - Tipptrainer speichert Tippergebnisse für Auswertung
    - Listenansicht aller Stories mit Such und Filterfunktion
    - Reader-Ansicht für Geschichten
    - Verwaltung der Genres

- Abgeleitete User Stories Beispiel
    - ❖ «Als Leser möchte ich eine Story bewerten können, damit andere Nutzer sehen können, wie gut sie ist.»
        - Kriterien:
            - Benutzer kann Scores für Kriterien anlegen
            - Scores werden in der Detailansicht einer Story angezeigt
            - Scores sind pro Story gespeichert und abrufbar
    - ❖ «Ich möchte eine neue Story erstellen, bearbeiten oder löschen können, damit ich meine eigenen Texte in der Plattform verwalten kann.»
        - Kriterien:
            - Kann eine Story mit Titel, Genre und Inhalt speichern
            - Kann bestehende Stories bearbeiten
            - Kann Stories löschen
            - Validierung verhindert leere Felder
    - ❖ «Ich möchte eine Story im Tipptrainer abtippen können, damit ich meine Tippgeschwindigkeit und Genauigkeit trainieren kann.»
        - Kriterien:
            - Kann eine Story auswählen und den Tipptrainer starten
            - Im Tipptrainer wird der Storytext angezeigt
            - Gibt Text ein und erhält sofort Feedback (WPM, Fehler)
            - Ergebnis wird mit Zeitspempel gespeichert

3 USE-CASES UND ARBEITSABLÄUFE

3.1 BESCHREIBUNG DER HAUPTPROZESSE

Use Case 1: Story erstellen und verwalten

- Story verfassen und auf der Plattform veröffentlichen. Dabei gibt man Titel, Genre und Textinhalt ein. Nach dem Speichern kann man die Story jederzeit bearbeiten oder löschen.
    - Arbeitsablauf:
        - Navigation zur Seite «Create Story»
        - Gibt Titel an, Genre auswählen, Inhalt schreiben
        - Speichern klicken -> Story wird in den Datenbank gespeichert
        - Über «Edit Story» kann man Änderungen vornehmen
        - Story löschen
    - Betroffene Daten:
        - Story (CRUD)
        - Genre(FK)

Use Case 2: Tipptrainer verwenden

- Tippfähigkeiten verbessern, indem er eine gespeichert Story abtippt. Dabei misst die Anwendung Tippgeschwindigkeit (WPM) und Fehler.
    - Arbeitsablauf:
        - Wählt eine Story aus
        - Startet Tipptrainer-Modus
        - Originaltext wird angezeigt
        - Tippen der Text
        - App berechnet WPM und Fehleranzahl live
        - Ergebnis wird mit Datum/Uhrzeit in der Datenbank gespeichert
    - Betroffene Daten:
        - Story (Lesen)
        - TypingResult (Create)

Use Case 3: Story bewerten

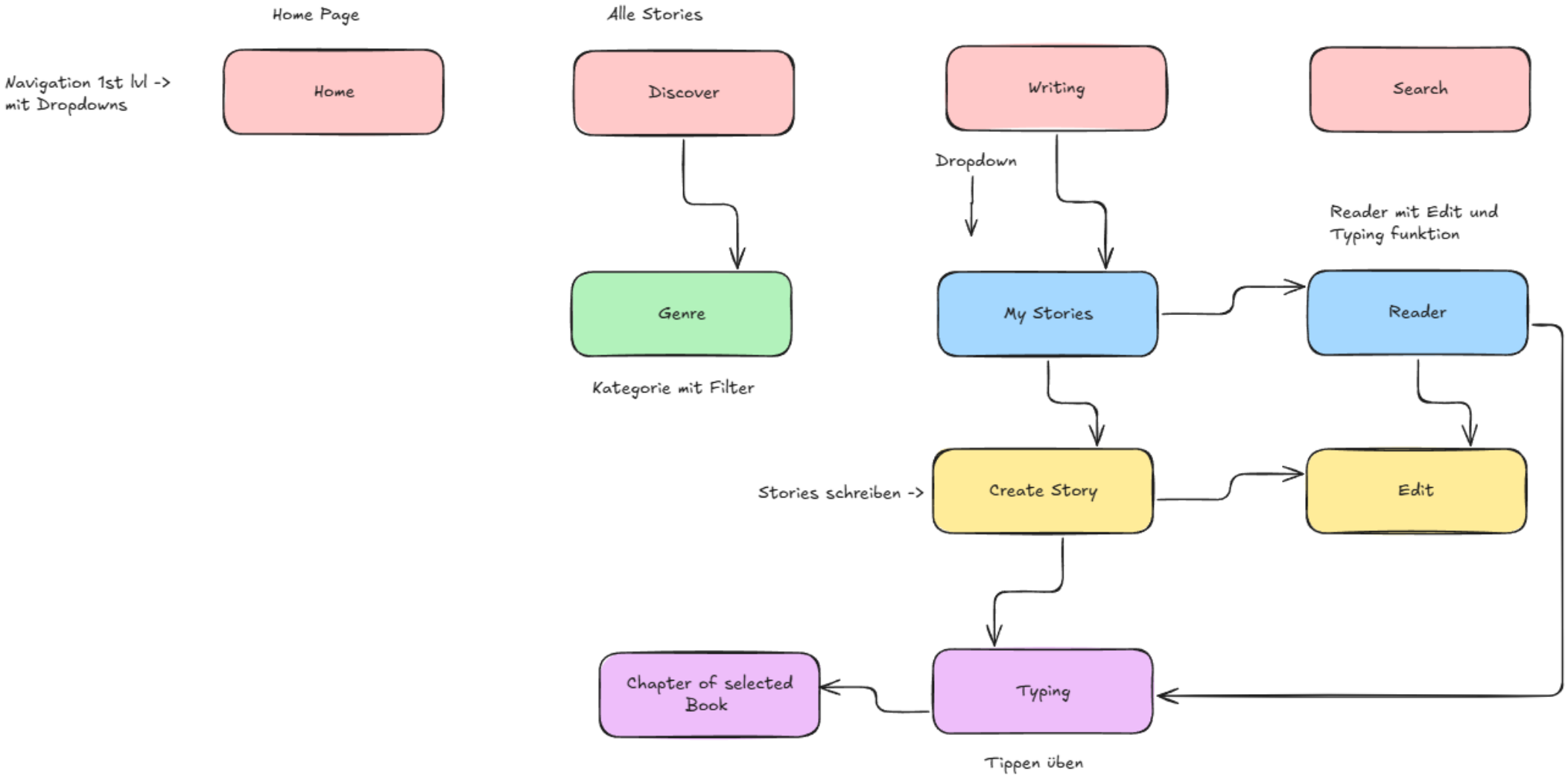
- Stories können bewertet werden. Scores werden zu Aspekten wie Kreativität, Spannung, Lesbarkeit vergeben.
    - Arbeitsablauf:
        - Navigation zur Detailansicht einer Story
        - Gibt Scores für verschiedene Kriterien ein
        - Klickt auf «Bewertung speichern»
        - Scores werden in der Datenbank gespeichert und in der Detailansicht angezeigt
    - Betroffene Daten:
        - Story (Lesen)
        - Score (Create)

4 STORYBOARD / NAVIGATIONSPLAN

TypingStories besteht aus mehreren Seiten, die über eine clientseitige Navigation miteinander verbunden sind. Das Routing ermöglicht es den Benutzern, gezielt zwischen den Kernfunktionalitäten zu wechseln.

- Seitenübersicht
    - Home:
        - Startseite
        - Begrüssungstext und kurze Erklärung der Funktionen
        - Navigation zur anderen Pages
    - Discover:
        - Zeigt alle verfügbaren Stories
        - Suchfeld mit Filter
        - Jede Story hat Buttons zum Lesen, Bearbeiten
    - Writing:
        - Zeigt Reader an mit vollständigen Stories
        - Listet vorhandene Scores auf
        - Buttons «Typing»
        - Create Story
    - Create Story:
        - Eingabemaske zum Erstellen einer neuen Story
        - Felder: Titel, Genre, Inhatstext
        - Editieransicht für bestehende Stories mit vorgefüllten Feldern
        - Speichern und Abbrechen Buttons
    - Typing:
        - Modus zum Abtippen des Textes
        - Zeigt Originaltext an
        - Live-Anzeige von WPM und Fehlerzahl
        - Ergebnis wird gespeichert und angezeig

4.1 SCHEMATISCHER ABLAUF



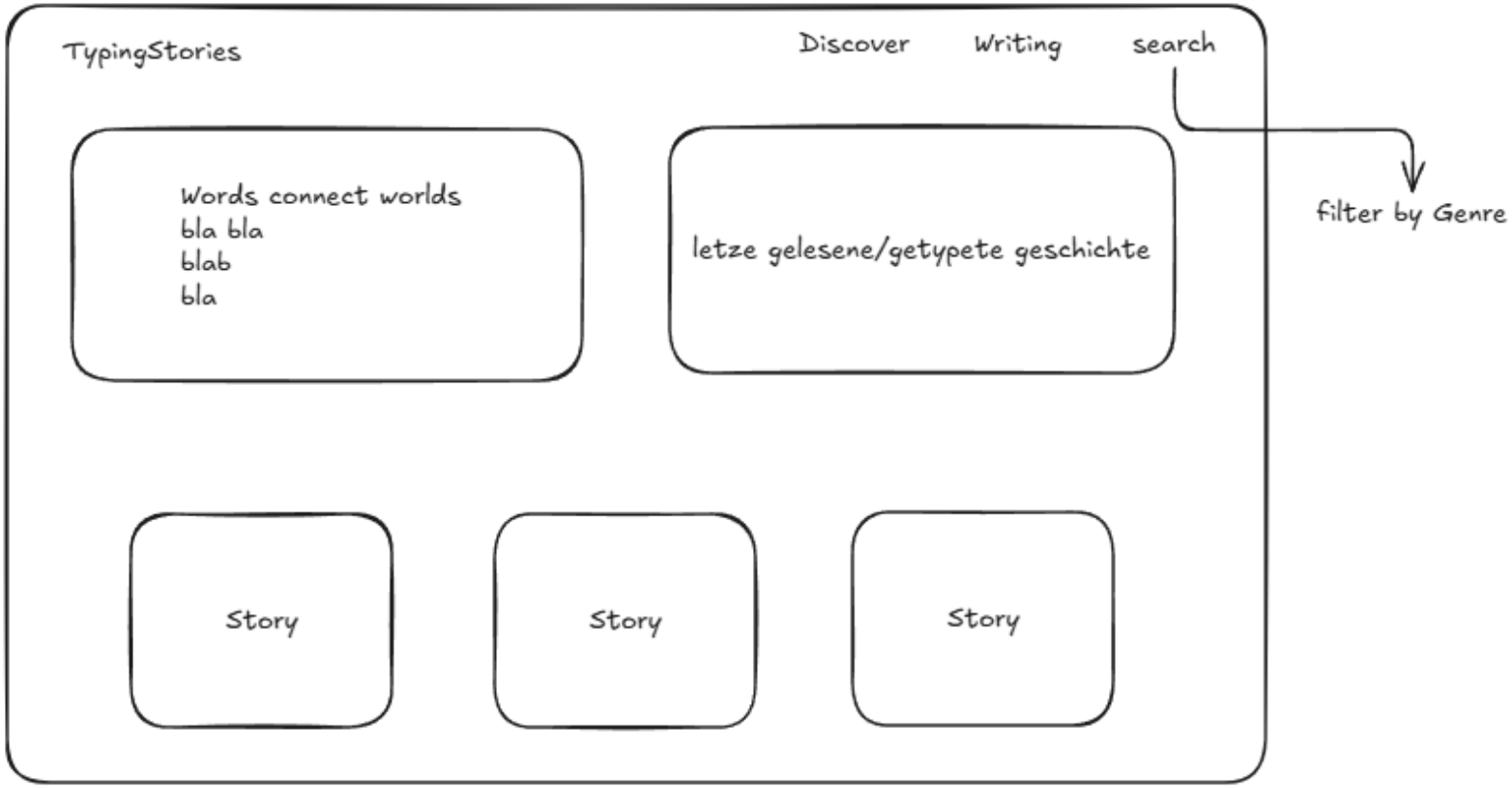
5 SCREEN-MOCKUPS / WIREFRAMES

Die folgenden Skizzen zeigen geplante Aufbau und das Layout der Hauptseiten der Applikation. Die Mockups dienen der Planung und Orientierung und sollen sicherstellen, dass alle Funktionen klar Strukturiert und erreichbar sind.

5.1 HOME

Beschreibung:

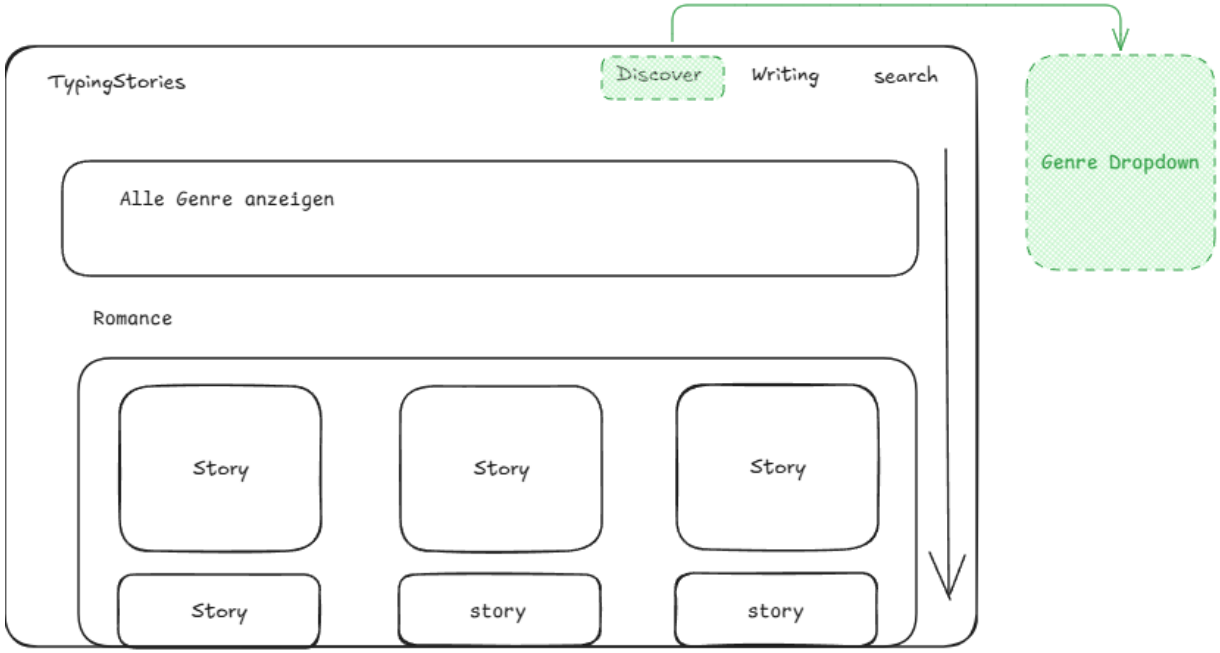
- Willkommenstext / Logo
- Navigation
- Einstiegspunkt für User



5.2 DISCOVER

Beschreibung:

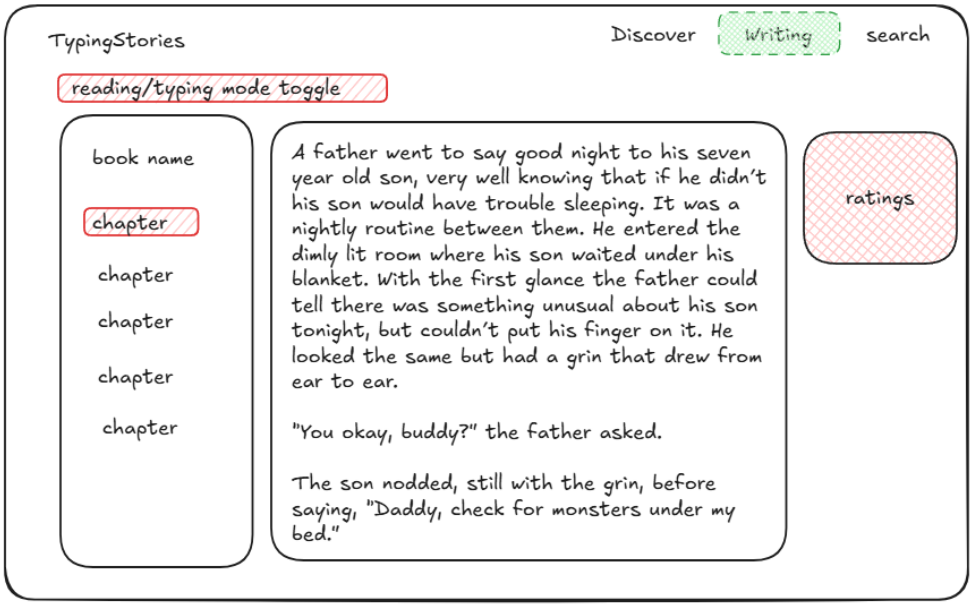
- Suchfeld mit Filter
- Liste aller Stories
- Buttons pro Story: Lesen, Bearbeiten, Tippen



5.3 READER

Beschreibung:

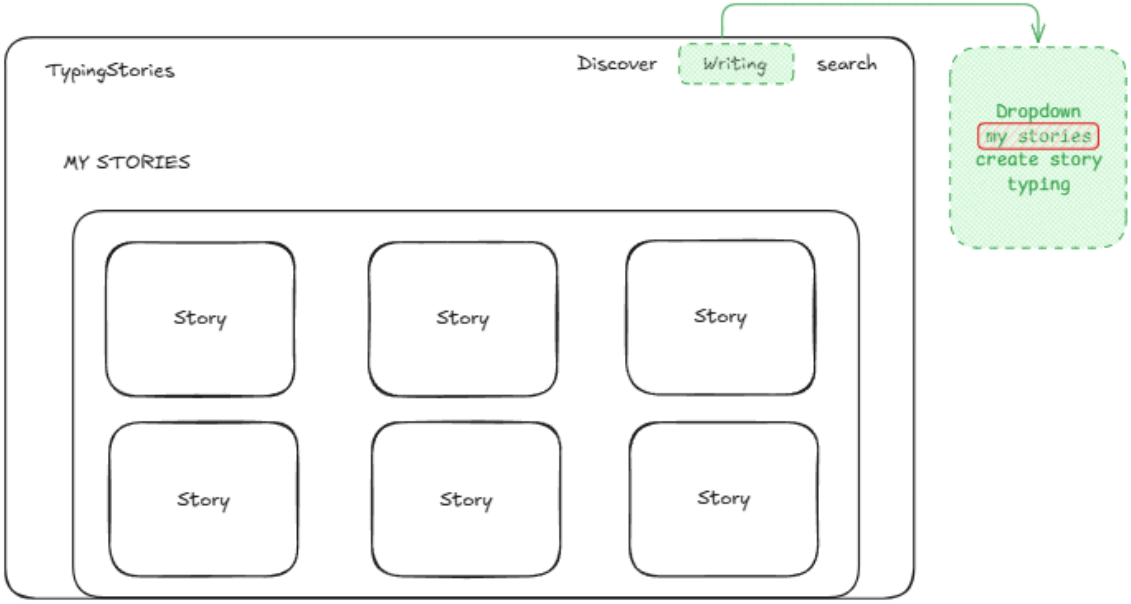
- Vollständige Story
- Typing Button
- Bewerten der Story



5.4 MY STORIES

Beschrebung:

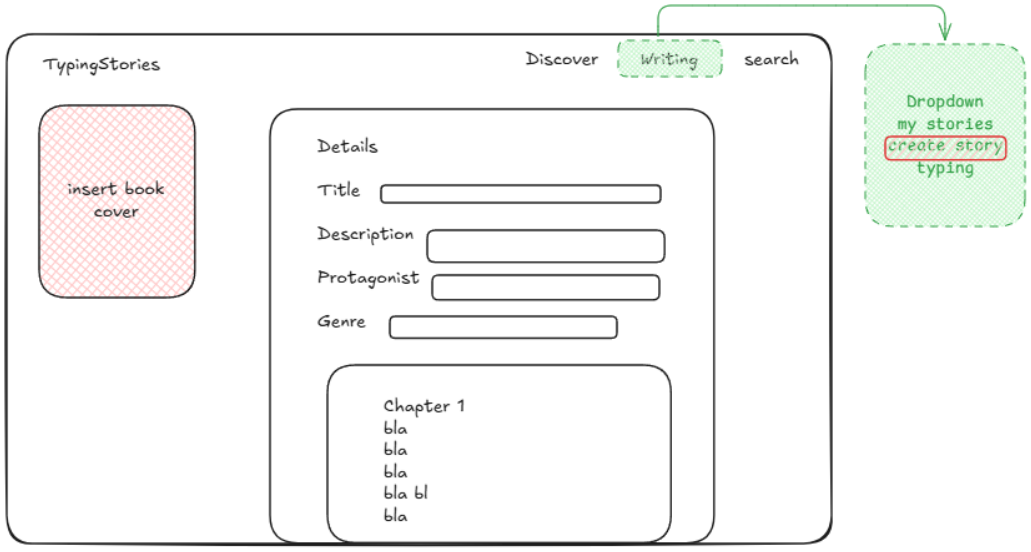
- Alle geschriebene Stories
- Edit
- Delete
- Read



5.5 CREATE STORY

Beschreibung:

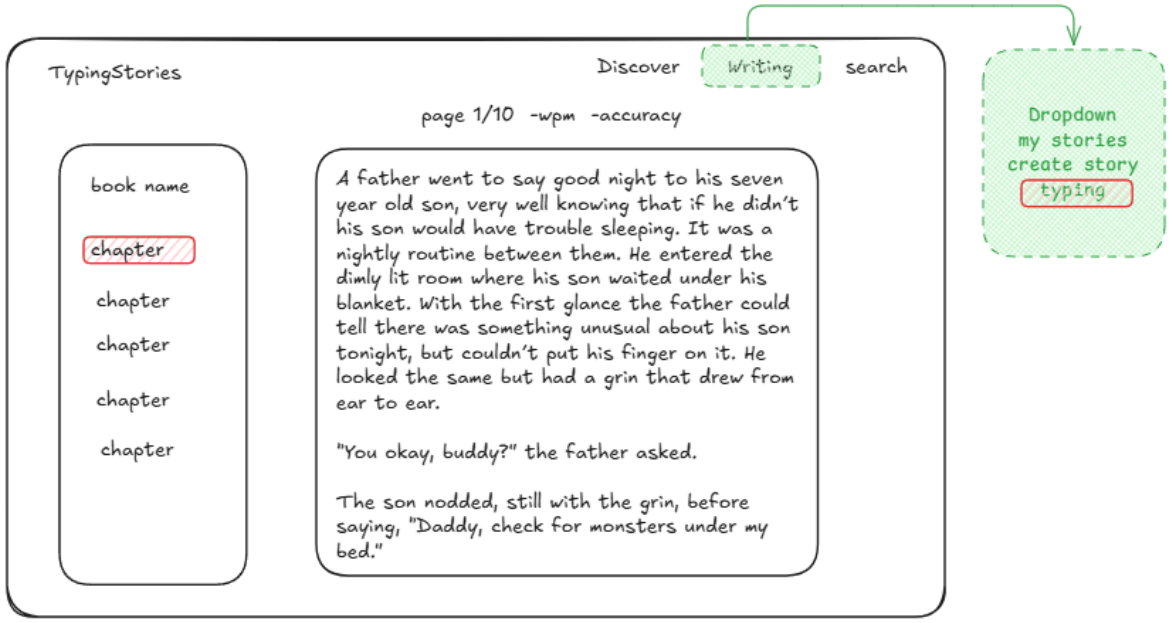
- Formular zum Erstellen von Stories
- Felder: Titel, Genre, Inhalt
- Verschiedene Buttons: Speichern, Abbrechen



5.6 TYPING

Beschreibung:

- Story anzeigen
- Live-Anzeige von WPM und Fehler
- Start und Reset



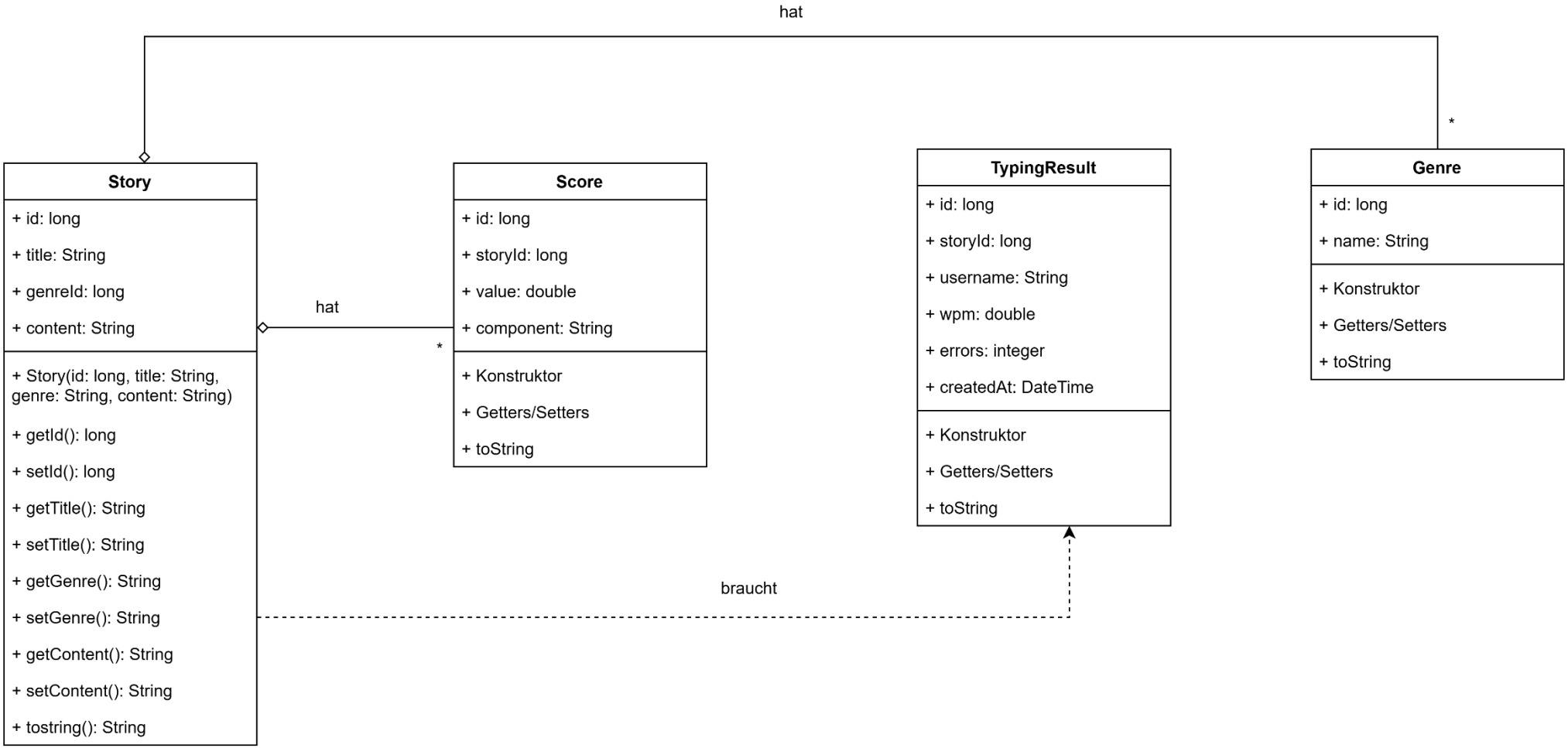
6 DATENMODELL / KLASSENDIAGRAMM

Die App arbeitet mit einer relationalen Datenbank. Das Datenmodell besteht aus vier Hauptentitäten, die miteinander in Beziehung stehen. Jede Entität wird in der Datenbank als eigene Tabelle umgesetzt. Die Beziehungen werden über Fremdschlüssel abgebildet.

6.1 BEZIEHUNGEN IM DATENMODELL

- Story <-> Score:
  - Eine Story kann mehrere Scores haben
  - Score.storyId verweist auf Story.id
- Story <-> TypingResult:
  - Eine Story kann mehrere TypingResults haben
  - TypingResult.storyId verweist auf Story.id
- Genre <-> Story:
  - Genre kann mehrere Stories enthalten
  - Story.genreId verweist auf Genre.id

6.2 KLASSENDIAGRAMM



6.3 FRONT UND BACKEND ABLAUF

Frontend → Services

- Discover.jsx → genreService.fetchAllGenres()
- Practice.jsx → storyService.fetchStoryById(), scoreService.fetchScoresByStory(), typingService.fetchAllTypingResults()
- RatingsPanel.jsx → scoreService
- CreateStory.jsx & MyStories.jsx → storyService

Services → Controller

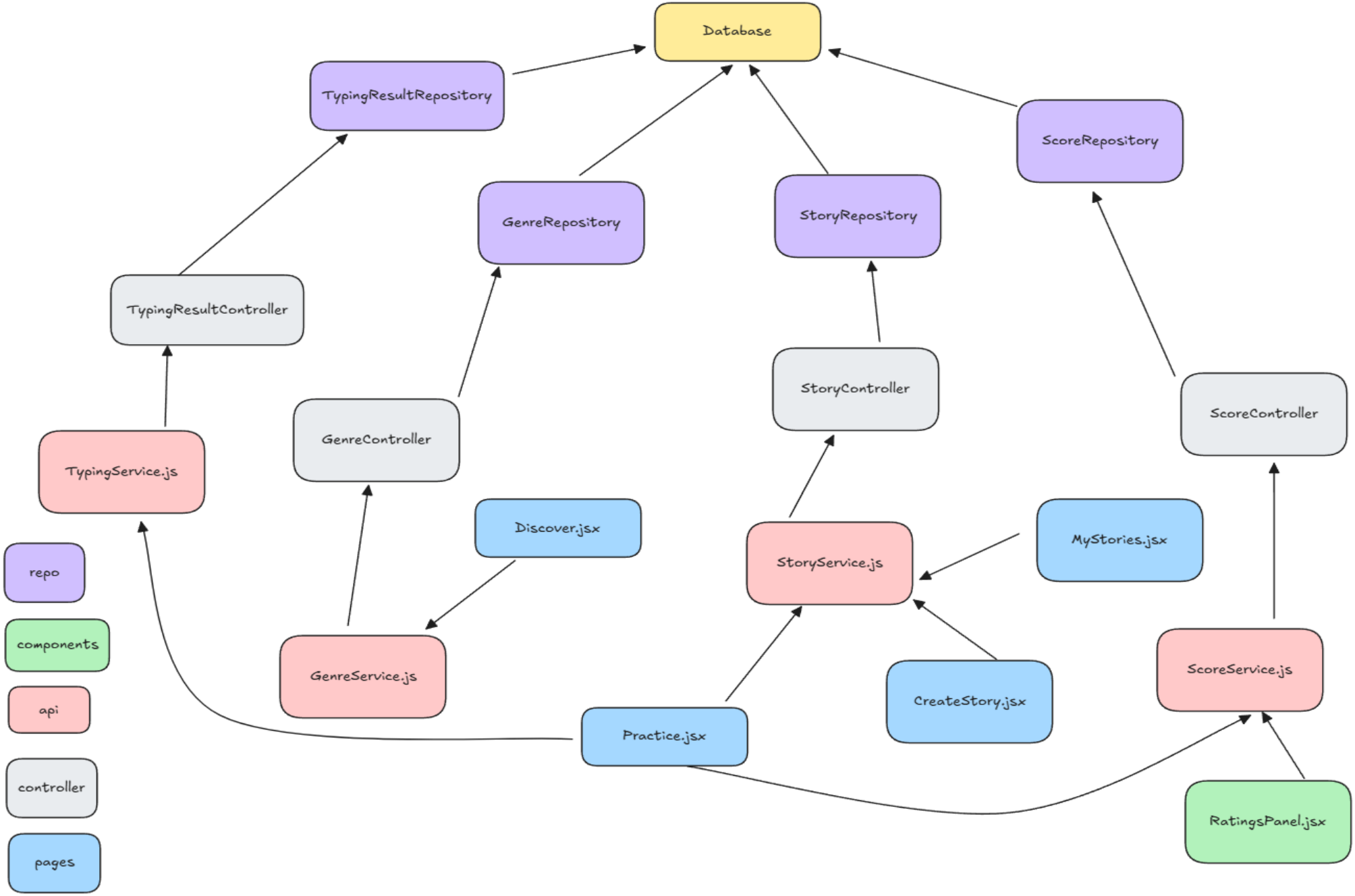
- genreService → GenreController (/api/genres)
- storyService → StoryController (/api/stories)
- scoreService → ScoreController (/api/scores)
- typingService → TypingResultController (/api/typingresults)

Controller → Repository

- GenreController → GenreRepository
- StoryController → StoryRepository
- ScoreController → ScoreRepository
- TypingResultController → TypingResultRepository

Repository → Database

- JPA-Repository-Aufrufe werden in SQL übersetzt, gegen die Datenbank ausgeführt und liefern Entitäten zurück.


Database
TypingResultRepository
ScoreRepository
GenreRepository
StoryRepository
TypingResultController
StoryController
ScoreController
TypingService.js
GenreController
Discover.jsx
MyStories.jsx
StoryService.js
repo
components
GenreService.js
ScoreService.js
api
CreateStory.jsx
Practice.jsx
controller
RatingsPanel.jsx
pages

7 REST-API-SPEZIFIKATION

TypingStories bietet REST-API, um alle Kernobjekte zu verwalten. Die Schnittstellen sind CRUD-fähig und folgen REST-Konventionen. Alle Endpunkte sind typischerweise unter dem Präfix /api erreichbar.

7.1 ÜBERSICHT ALLER RESSOURCEN

- /stories
- /stories/{id}
- /stories/{storyId}/scores
- /stories/{storyId}/typing-results
- /genres
- /genres/{id}

7.2 ENDPUNKTE

Story Ressource

- GET /api/stories
  - Alle Stories abrufen
  - Beispiel:

```
[
  {
    "id": 1,
    "title": "Adventure in Space",
    "genreId": 2,
    "content": "Once upon a time..."
  }
]
```

- GET /api/stories/{id
  - Story nach ID abruffen
  - Response Beispiel:

```
{
  "id": 1,
  "title": "Adventure in Space",
  "genreId": 2,
  "content": "Once upon a time..."
}
```

- POST /api/stories
  - Neue Story erstellen
  - Request Beispiel:

```
{
  "title": "New Story",
  "genreId": 2,
  "content": "The quick brown fox..."
}
```

- Put /api/stories/{id}
  - Existierende Story aktualisieren
  - Request Beispiel:

```
{
  "title": "Updated Title",
  "genreId": 2,
  "content": "Updated content..."
}
```

- DELETE /api/stories/{id}
  - Story löschen
  - Response: 204 No Content

8 VALIDIERUNGSKONZEPT

Prüfung der Benutzereingaben sowohl auf der Client-Seite (Frontend) als auch auf der Server-Seite (Backend). Ziel ist es, fehlerhafte oder unvollständige Eingaben abzufangen und saubere Daten in der Datenbank zu speichern.

8.1 VALIDIERTE FELDER

Story:

- Title (Pflichtfeld)
- genreId (muss existierendes Genre sein)
- content (Pflichtfeld)

Score:

- Component (Pflichtfeld)
- Value (zwischen 0-10 liegen)

TypingResult:

- Wpm (muss positiv sein)
- Errors (muss ≥ 0 sein)

Genre:

- Name (Pflichtfeld)

8.2 BEISPIEL FÜR VALIDIERUNGSREGELN

Frontend (Client-seitig)

- Title darf nicht leer sein → required
- Content darf nicht leer sein → required
- Value im Score-Formular muss zwischen 0 und 10 liegen → min/max
- Username im Typing-Result maximal 50 Zeichen → maxlength
- Genre-Auswahl darf nicht leer bleiben

Backend (Server-seitig)

- @NotNull / @NotBlank für Pflichtfelder
- @Size(max=50) für Username
- @Min(0) / @Max(10) für Scores
- Referentielle Integrität (genreId muss existieren)

8.3 BEISPIEL FÜR FEHLERMELDUNGEN

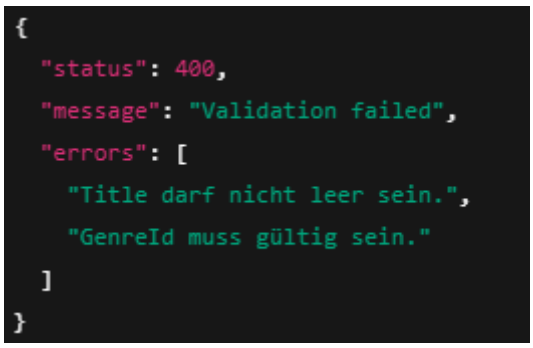
Feld	Regel/Fehlerfall	Fehlermeldung
title	leer	„Bitte einen Titel eingeben.“
genreId	nicht ausgewählt/ungültig	„Bitte ein gültiges Genre wählen.“
content	leer	„Story-Inhalt darf nicht leer sein.“
value	<0 oder >10	„Bewertung muss zwischen 0 und 10 liegen.“
username	>50 Zeichen	„Benutzername darf max. 50 Zeichen haben.“
wpm	negativ	„WPM muss positiv sein.“
errors	negativ	„Fehleranzahl darf nicht negativ sein.“
genre.name	leer	„Genre-Name darf nicht leer sein.“

8.4 FEHLERBEHANDLUNG IM FRONTEND

- Formulare zeigen Inline-Fehler (rote Labels oder Meldungen)
- Buttons werden deaktiviert, solange Pflichtfelder fehlen
- Benutzer erhält sofort Feedback beim Tippen

8.5 FEHLERBEHANDLUNG IM BACKEND

- Spring Boot wirft ConstraintViolationException
- ExceptionHandler fängt Fehler ab
- Fehlerantwort im JSON-Format:



Test-Nr	Titel	Beschreibung	Erwartetes Ergebnis	Ergebnis Dokumentation
F1	NavBar	Links und Suchfeld korrekt anzeigen	Menüeinträge wie «Discover» sind sichtbar	Funktioniert
F2	NavBar	Suchfeld kann befüllt werden	Suchtext wird korrekt angezeigt	Funktioniert
F3	GenreBadge	Genre-Button zeigt Label und Link	Link zeigt Genre und verweist korrekt	Funktioniert
F4	ToggleSwitch	Auswahloptionen reagieren auf Klick	onChange wird mit korrektem Wert aufgerufen	Funktioniert
F5	RatingsPanel	Bewertungsdaten werden angezeigt	Durchschnitt wird berechnet und angezeit	Funktioniert
F6	TypingPage	Kapiteltext wird angezeigt, Tippeingabe möglich	Zeichenanzeige, Tippverlauf, WPM & ACC funktionieren	Funktioniert
B1	POST /api/stories	Neue Story mit gültigen Daten speichern	Status 201 + ID zurückgegeben	Funktioniert
B2	POST /api/stories	Ungültige Story ablehnen	Status 400 Bad Request	Funktioniert
B3	GET /api/stories	Liste aller Stories abrufen	Rückgabe als JSON-Array	Funktioniert
B4	GET /api/stories/{id}	Einzelne Story abrufen	Rückgabe mit korrekten Feldern	Funktioniert
B5	DELETE /api/stores/1	Vorhandene Story löschen	Status 204 No Content	Funktioniert
B6	DELETE /api/stories/999	Nicht vorhandene Story löschen	Status 404 Not Found	Funktioniert

9.2 TESTWERKZEUGE

Frontend: Jest, React Testing Library

Backend: Junit 5, Spring Boot Test, MockMvc

10 INSTALLATIONSANLEITUNG

Diese Anleitung beschreibt die Schritte, um die Applikation TypingStories lokal zu installieren und auszuführen. Das Projekt besteht aus einem Frondend mit React und einem Backend mit Spring Boot.

10.1 VORAUSSETZUNGEN

- Node.js (empfohlen ≥ 16)
- npm oder yarn
- Java JDK 21
- Maven
- MySQL (oder Docker für Datenbankcontainer)
- Git für Clone

10.2 INSTALLATIONSSCHRITTE

1. Repository klonen

git clone https://github.com/preojam/typingStories.git

2. Datenbank einrichten

- Erstelle eine Datenbank z. B. via MySQL Workbench:
    - CREATE DATABASE typingstories;
    -
    - CREATE USER 'typinguser'@'localhost' IDENTIFIED BY 'typingpass';
    - GRANT ALL PRIVILEGES ON typingstories.* TO 'typinguser'@'localhost';
    - FLUSH PRIVILEGES;

Stelle sicher, dass der Benutzername und das Passwort mit der application.properties Datei übereinstimmen (z. B. root / password).

3. Backend starten

mvn spring-boot:run

Port: Standardmäßig unter http://localhost:8080 erreichbar.

4. Frontend installieren und starten

npm install

npm run dev

Port: Standardmäßig unter http://localhost:5173 (Vite)

5. API DOKUMENTATION

Sobald das Backend läuft, kann die API-Dokumentation unter

http://localhost:8080/swagger-ui/index.html

aufgerufen werden (OpenAPI/Swagger ist eingebunden).

11 HILFESTELLUNGEN UND QUELLEN

Austausch mit Mitschülern

- Arvin, Thomas, Chris
- Gemeinsames Brainstorming zur Projektidee
- Feedback zu Storyboard und Klassendiagramm
- Tipps zur Fehlerbehebung in React / Spring Boot

Online-Recherche & Tutorials

- React-Dokumentation: https://react.dev/
- Spring Boot Guides: https://spring.io/guides
- Java Bean Validation: https://hibernate.org/validator/
- Stack Overflow-Beiträge zu spezifischen Fehlern
- YouTube-Tutorials für Tests, Spring, React, Typing App

Genutzte Tools

- ChatGPT
- draw.io für Diagramme
- Visual Studio Code
- IntelliJ
- Insomnia (API-Tests)
- GitHub / Git zur Versionierung
- https://excalidraw.com/
- MySQL Workbench
- Swagger