|  |  |
| --- | --- |
| 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 2](#_Toc202623374)

[1.1 Zweck und Motivation 2](#_Toc202623375)

[1.2 Zentrale Merkmale 2](#_Toc202623376)

[2 Anforderungskatalog 3](#_Toc202623377)

[3 Use-Cases und Arbeitsabläufe 4](#_Toc202623378)

[3.1 Beschreibung der Hauptprozesse 4](#_Toc202623379)

[4 Storyboard / Navigationsplan 5](#_Toc202623380)

[4.1 Schematischer Ablauf 6](#_Toc202623381)

[5 Screen-Mockups / Wireframes 7](#_Toc202623382)

[5.1 Home 7](#_Toc202623383)

[5.2 Discover 8](#_Toc202623384)

[5.3 Reader 8](#_Toc202623385)

[5.4 Create Story 9](#_Toc202623386)

[5.5 Typing 9](#_Toc202623387)

[6 Datenmodell / Klassendiagramm 10](#_Toc202623388)

[6.1 Beziehungen im Datenmodell 10](#_Toc202623389)

[6.2 Klassendiagramm 11](#_Toc202623390)

[7 REST-API-Spezifikation 12](#_Toc202623391)

[7.1 Übersicht aller Ressourcen 12](#_Toc202623392)

[7.2 Endpunkte 12](#_Toc202623393)

[8 Validierungskonzept 14](#_Toc202623394)

[8.1 Validierte Felder 14](#_Toc202623395)

[8.2 Beispiel für Validierungsregeln 14](#_Toc202623396)

[8.3 Beispiel für Fehlermeldungen 15](#_Toc202623397)

[8.4 Fehlerbehandlung im Frontend 15](#_Toc202623398)

[8.5 Fehlerbehandlung im Backend 15](#_Toc202623399)

[9 Testplan 16](#_Toc202623400)

[9.2 Beispiel-Ergebnisse (Screenshots) 16](#_Toc202623401)

[9.3 Fehlerhandling getestet 16](#_Toc202623402)

[9.4 Unit-Tests 16](#_Toc202623403)

[10 Installationsanleitung 17](#_Toc202623404)

[10.1 Voraussetzungen 17](#_Toc202623405)

[11 Hilfestellungen und Quellen 18](#_Toc202623406)

## 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.

### 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.

### 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

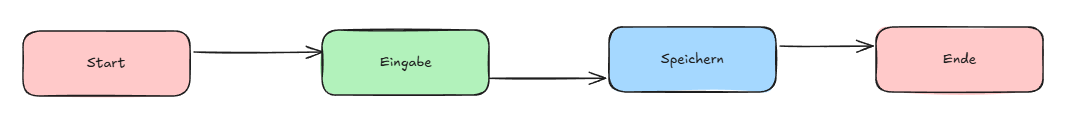
* Story verfassen und auf der Plattform veröffentlichen. Dabei gibt man Titel, Genre und Textinhalt ein. Nach dem Speichern kann man die Story jederzeit bearbeiten oder löschen.
  + Arbeitsablauf:
    - Navigation zur Seite «Create Story»
    - Gibt Titel an, Genre auswählen, Inhalt schreiben
    - Speichern 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

Ein Bild, das Text, Diagramm, Screenshot, Plan enthält.

KI-generierte Inhalte können fehlerhaft sein.

## 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

Ein Bild, das Text, Screenshot, Diagramm, Rechteck enthält.

KI-generierte Inhalte können fehlerhaft sein.Beschreibung:

* Willkommenstext / Logo
* Navigation
* Einstiegspunkt für User

### Ein Bild, das Text, Screenshot, Diagramm, Zahl enthält. KI-generierte Inhalte können fehlerhaft sein.5.2 Discover

Beschreibung:

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

### Ein Bild, das Text, Screenshot, Zahl, Schrift enthält. KI-generierte Inhalte können fehlerhaft sein.5.3 Reader

Beschreibung:

* Vollständige Story
* Typing Button
* Bewerten der Story

### Ein Bild, das Text, Screenshot, Diagramm, Design enthält. KI-generierte Inhalte können fehlerhaft sein.5.4 Create Story

Beschreibung:

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

### 5.5 Typing

Ein Bild, das Text, Screenshot, Schrift, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.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

Ein Bild, das Screenshot, Schwarz enthält.

KI-generierte Inhalte können fehlerhaft sein.

## 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
  + Ein Bild, das Text, Screenshot, Schrift, Design enthält.

    KI-generierte Inhalte können fehlerhaft sein.Beispiel:
* GET /api/stories/{id
  + Story nach ID abruffen
  + Ein Bild, das Text, Screenshot, Schrift enthält.

    KI-generierte Inhalte können fehlerhaft sein.Response Beispiel:
* POST /api/stories
  + Neue Story erstellen
  + Ein Bild, das Text, Schrift, Screenshot enthält.

    KI-generierte Inhalte können fehlerhaft sein.Request Beispiel:
* Put /api/stories/{id}
  + Existierende Story aktualisieren
  + Ein Bild, das Text, Schrift, Screenshot enthält.

    KI-generierte Inhalte können fehlerhaft sein.Request Beispiel:
* 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
* Ein Bild, das Text, Screenshot, Schrift, Design enthält.

  KI-generierte Inhalte können fehlerhaft sein.Fehlerantwort im JSON-Format:

## 9 Testplan

| **Test-Nr** | **Titel** | **Beschreibung** | **Erwartetes Ergebnis** | **Ergebnis Dokumentation** |
| --- | --- | --- | --- | --- |
| 1 | Story anlegen | Neue Story mit Titel, Genre und Content speichern | Story wird in Liste angezeigt, Daten sind korrekt | funktioniert |
| 2 | Story bearbeiten | Vorhandene Story ändern und speichern | Änderungen sind in Detailansicht sichtbar | funktioniert |
| 3 | Story löschen | Eine Story aus der Liste löschen | Story verschwindet aus der Liste | funktioniert |
| 4 | Tipptrainer starten | Story auswählen und Tippübung beginnen | Text wird angezeigt, Tippfeld aktiv, WPM/Fehler berechnet | funktioniert |
| 5 | Tippresultat speichern | Tippübung abschließen | Ergebnis wird mit WPM/Fehlerzahl gespeichert | funktioniert |
| 6 | Score vergeben | Story bewerten (Kreativität, Spannung) | Bewertung wird gespeichert und angezeigt | funktioniert |
| 7 | Genre anlegen | Neues Genre im Admin-Bereich hinzufügen | Genre erscheint in Genre-Auswahl | funktioniert |
| 8 | Genre-Auswahl beim Erstellen | Genre beim Story-Erstellen auswählen | Genre wird korrekt zugewiesen | funktioniert |

### 9.2 Beispiel-Ergebnisse (Screenshots)

*(optional, kannst du später einfach Screenshots einkleben)*  
✅ Screenshot: Erfolgreich gespeicherte Story in Liste  
✅ Screenshot: Tipptrainer-Ansicht mit Ergebnis  
✅ Screenshot: Bewertungsformular ausgefüllt

### 9.3 Fehlerhandling getestet

✅ Leere Felder werden blockiert (Client-seitig).  
✅ Servervalidierung verhindert falsche Daten (z. B. Score über 10).  
✅ Fehlermeldungen werden im Frontend angezeigt.

### 9.4 Unit-Tests

Für das Backend wurden Unit-Tests mit JUnit geschrieben. Beispiel:

## 10 Installationsanleitung

Diese Anleitung beschreibt die Schritte, um die Applikation TypingStorise 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

## 11 Hilfestellungen und Quellen

* In diesem Abschnitt werden alle Hilfestellungen, Ideenquellen und Recherchen aufgelistet, die bei der Entwicklung der Applikation **TypingStories** genutzt wurden.
* **✅ 11.1 Unterricht & Dozenten**
* M294 und M295 Modulblätter (WISS)
* Unterrichtsmaterial und Live-Demos der Lehrperson
* Fragen und Erklärungen während der Präsenzzeit
* **✅ 11.2 Austausch mit Mitschülern**
* Gemeinsames Brainstorming zur Projektidee
* Feedback zu Storyboard und Klassendiagramm
* Tipps zur Fehlerbehebung in React / Spring Boot
* **✅ 11.3 Online-Recherche & Tutorials**
* React-Dokumentation: <https://react.dev/>
* Spring Boot Guides: <https://spring.io/guides>
* Java Bean Validation: https://hibernate.org/validator/
* MySQL-Dokumentation: <https://dev.mysql.com/doc/>
* Stack Overflow-Beiträge zu spezifischen Fehlern
* YouTube-Tutorials zu React-CRUD und Spring Boot-APIs
* **✅ 11.4 Genutzte Tools**
* draw.io für Diagramme
* Visual Studio Code (Frontend)
* Eclipse / IntelliJ (Backend)
* Postman (API-Tests)
* GitHub / Git zur Versionierung
* https://excalidraw.com/

Ein Bild, das Text, Screenshot, Software, Schrift enthält.

KI-generierte Inhalte können fehlerhaft sein.

Endless json entry in scores in story class @JsonManageReference und Score , typingresult @JsonBackreference reintun