Aufgabe 4: TypeScript und WebAssembly

In dieser Aufgabe soll der Chat-Client aus Aufgabe 3 um zwei zusätzliche Features erweitert werden: Eine Rechtschreibprüfung für die aktuelle Nachricht und eine Wortvorhersage. Anders als bei den vorherigen Aufgaben sollen diese nicht selbst in JavaScript implementiert, sondern mit TypeScript und WebAssembly integriert werden. Dazu wurde der Server erneut angepasst, um ein Wörterbuch und ein Vorhersagemodell auszuliefern.

4.1 Vorbereitung

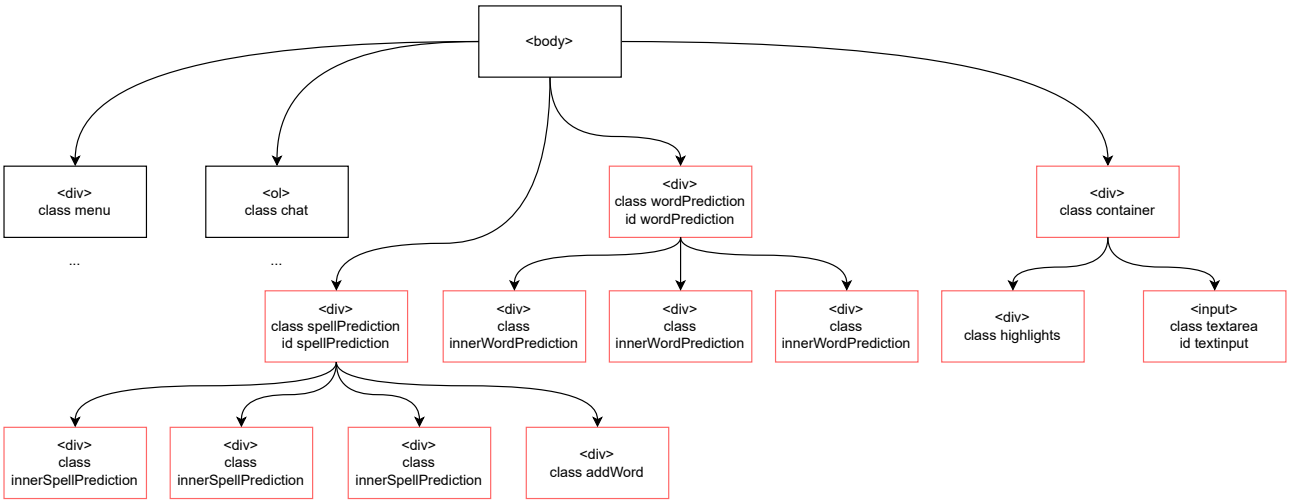

Abbildung 1: Veränderte HTML-Struktur; Achtung wenn ihr aus vorherigen Aufgaben das HTML kopiert, es darf nur einen `input` geben.

Als Basis für diese Aufgabe dient Ihre Chat-Client-Implementierung aus der letzten Aufgabe. Um diese für die aktuelle Aufgabe vorzubereiten, befolgen Sie die folgenden Schritte:

1. Kopieren Sie alle `.html`, `.css` und `.js` Dateien aus der letzten Aufgabe die nicht generiert wurden, sowie die `package.json` und `package.lock.json` in den aktuellen Aufgabenordner. Integrieren Sie die Inhalte ihrer `webpack.config.js` aus Aufgabe 3 zudem in die neu ausgeteilte Datei.

2. Führen Sie im Aufgabenordner den Befehl `npm install --save-dev @wasm-tool/wasm-pack-plugin \ copy-webpack-plugin ts-loader typescript typescript-loader wasm-pack` aus, um die benötigten Pakete für TypeScript und WebAssembly zu installieren.

3. Passen Sie Ihre `webpack.config.js`-Datei an, sodass diese den `ts-loader`[1] und `wasm-pack`[2] verwendet. Für `wasm-pack` soll dabei als extra Argument `--target web` übergeben werden und die Dateien aus `lib/markov` zu WebAssembly kompiliert werden. Das Ergebnis soll danach in dem Ordner `lib/markov/pkg` liegen.

4. Passen Sie die `index.html` so an, dass sie Abbildung 1 entspricht. Dabei müssen nur die roten Elemente hinzugefügt bzw. angepasst werden. Importieren Sie zudem die `lib/check/check.ts` und `lib/check/check.css`-Dateien, und sorgen Sie dafür, dass die exportierte `loadCheckWords`-Funktion nach dem Laden der Seite ausgeführt wird.

Zusätzliche Hinweise:

- Stellen Sie sicher, dass `rustup` installiert ist, bevor Sie `wasm-pack` verwenden.
- Falls die Rust-Toolchain bereits installiert war, muss diese ggf. mit `rustup update` aktualisiert werden.
- Unter Windows: Wenn beim ausführen von `wasm-pack` ein Fehler in `linker.exe` auftritt, kann dieser durch das Installieren der *Build Tools for Visual Studio*[3] (speziell die C++ Tools) behoben werden.
- Nach der Installation können TypeScript mit `npx tsc` und `wasm-pack` mit `npx wasm-pack` zu Testzwecken auch ohne Webpack ausgeführt werden.
- Fügen Sie passenden inneren Text für das div mit der Klasse `AddWord` hinzu.

[1] `https://webpack.js.org/guides/typescript/`
[2] `https://github.com/wasm-tool/wasm-pack-plugin`
[3] `https://visualstudio.microsoft.com/thank-you-downloading-visual-studio/?sku=BuildTools&rel=16`

4.2 Rechtschreibprüfung

Für die Rechtschreibprüfung wird die *Typo*-Library bereitgestellt. Zudem sind in `check.ts` bereits einige Hilfsfunktionen enthalten, die für die Interaktion mit der Library und den neuen HTML-Elementen genutzt werden können. Bei diesen fehlen jedoch teilweise noch die Typinformationen, welche Sie selbst hinzufügen sollen. Orientieren Sie sich dabei an den Fehlermeldungen des TypeScript-Compilers, welcher beim Webpack-Bundling mit `webpack --config webpack.config.js` ausgeführt wird. Sobald dieser keine Fehler mehr ausgibt, wird der generierte Code mit in die *bundle.js* gebündelt.

- Implementieren Sie die Funktionen `handleKeyUp` und `handleKeyDown`, welche für die Markierung fehlerhafter Wörter zuständig sind. `handleKeyUp` soll dabei beim Tippen eines Leerzeichens am Ende des Textes das zuletzt geschriebene Wort überprüfen und bei einem Fehler markieren. Wenn durch ein Leerzeichen ein Wort getrennt wurde, sollen alle Wörter oder optional nur die beiden neu entstandenen erneut überprüft werden.

  Die Funktion `handleKeyDown` hingegen soll das Löschen von Zeichen beim Drücken von Backspace oder Entfernen behandeln. Hierbei sollen die beiden Fälle behandelt werden, dass entweder ein Leerzeichen oder ein Buchstabe eines Wortes entfernt wurde. Bei dem Entfernen eines Leerzeichens können entweder alle oder optional nur das neu entstandene Wort überprüft werden.

  Hilfreiche Hilfsfunktionen sind hier `checkWord`, `checkWords`, `isPunctuation`, `getWordAtIndex` und `deleteFromWrongWords`.

- Implementieren Sie die Funktionen `handleClick`, `handleClickSpellPrediction` und `handleClickAddToDict`, welche Verbesserungsvorschläge für falsch geschriebene Wörter anzeigen und einfügen können. Dabei soll `handleClick` zunächst feststellen, ob der Nutzer ein markiertes Wort angeklickt hat. Ist dies der Fall, sollen für dieses Wort maximal drei Verbesserungsvorschläge gefunden und in den `innerSpellPrediction`-Elementen angezeigt werden. Zudem soll diese Funktion erkennen, ob außerhalb des Textfeldes geklickt wurde und in diesem Fall die Vorschläge ausblenden.

  Relevante Hilfsfunktionen sind hier `getWordAtIndex`, `showSpellPredictions` und die `suggest(string)`-Methode des `dictionary`-Objektes. Mit `handleClickSpellPrediction` soll dann das ausgewählte Wort durch den angeklickten Verbesserungsvorschlag ersetzt werden. `handleClickAddToDict` hingegen soll das ausgewählte Wort in das Wörterbuch aufnehmen, damit es nicht mehr als fehlerhaft markiert wird. Dafür kann die `addWordToDictionary`-Funktion genutzt werden.

Zusätzliche Hinweise:

- Für diese Aufgabe ist es sinnvoll, sich genauer mit dem `HTMLTextAreaElement`[4] auseinanderzusetzen.
- Unter Umständen kann es zu Fehlern kommen, wenn die Rechtschreibprüfung zusammen mit der *Giphy*-Integration bzw. der *Typing Notification* verwendet wird. In diesem Fall können diese aus der Anwendung entfernt werden.
- Zusätzlich zu der verpflichtenden Option `noImplicitAny` können optional noch weitere Checks[5] von TypeScript durchgeführt werden. Diese können in der `tsconfig.json` aktiviert werden.

4.3 Wortvorhersage

Für die Wortvorhersage wird die Library *markov* bereitgestellt. Da diese jedoch in der Programmiersprache *Rust* geschrieben ist, muss sie zunächst zu WebAssembly-Bytecode kompiliert werden, um im Browser ausgeführt werden zu können. Passen Sie dafür zunächst die Datei `lib/markov/src/lib.rs` an, sodass die Funktionen `initialize_chain` und `generate_prediction` als WebAssembly-Funktionen aufgerufen werden können.

- Importieren Sie nun die beiden WebAssembly-Funktionen, sowie die `init` Funktion aus der Datei `markov/pkg/index.js` mithilfe der `import`-Anweisung. Danach sollte ein erneutes Bundling mit Webpack das Kompilieren des WebAssembly-Codes starten. Wenn alles funktioniert hat, sollte dies eine Reihe von Dateien im Verzeichnis `lib/markov/pkg` erzeugen.
- Erweitern Sie den Callback zum *DOMContentLoaded*-Event (Ende der `check.ts`-Datei), sodass dieser die `init`-Funktion aufruft, um das WebAssembly-Modul zu instanziieren. Der Parameter sollte dabei die generierte WebAssembly-Datei `index_bg.wasm` sein.
- Auf dem instanziierten Modul soll nun die `initialize_chain`-Funktion aufgerufen werden, welche ein trainiertes Modell als Parameter erwartet. Dieses kann als `train_data.yaml` vom Server angefragt werden.

[4] `https://developer.mozilla.org/en-US/docs/Web/API/HTMLTextAreaElement`
[5] `https://www.typescriptlang.org/tsconfig#Strict_Type_Checking_Options_6173`

- Implementieren Sie nun die `getPredictionFor`-Funktion, welche für ein gegebenes Wort bis zu drei Wörter vorschlägt. Dafür soll drei Mal die `generate_prediction`-Funktion des WebAssembly-Moduls aufgerufen werden. Diese gibt jedoch einen Satz zurück, von dem nur das nächste Wort, also das zweite Wort des Satzes, verwendet werden soll (oder eine leere Antwort, wenn das System keine Vorhersage treffen kann). Zusätzlich sollen doppelte Vorschläge nur ein Mal angezeigt werden.
- Nun soll die `handleKeyUp`-Funktion erweitert werden, sodass die generierten Vorschläge angezeigt werden. Diese sollen angezeigt werden, wenn ein Wort durch ein Leerzeichen beendet, oder wenn durch Bewegen des Text-Cursors durch die Pfeiltasten ein Wortende erreicht wird. Zum Anzeigen der Vorschläge kann die `showWordPredictions`-Funktion genutzt werden.
- Zuletzt implementieren Sie die `handleClickWordPrediction`-Funktion, um ausgewählte Vorschläge an der aktuellen Textposition einzufügen. Wenn ein Vorschlag durch Klicken eingefügt wird, soll basierend auf diesem zudem erneut die Vorhersage durchgeführt werden.

Sehen Sie sich zudem die Funktionen `init` und `load` in der Datei `lib/markov/pkg/index.js` genauer an und versuchen Sie nachzuvollziehen, welche Schritte diese beim Initialisieren des WebAssembly-Moduls durchführen.

Letzter Commit bis zum 13. Januar.
Termine für die Vorstellungen in StudOn buchbar.

Präsentation der fertigen Lösung spätestens am Tag der Abgabefrist in der Rechnerübung!