Aufgabe 3 – Tic-Tac-Toe Fronten-Implementierung

Einführung:

In dieser Aufgabe implementieren Sie ein **Frontend für ein Tic-Tac-Toe-Spiel** mit *Vue.js* oder *Angular.js*. Der Backend-Server, in der Programmiersprache GO, ist bereits für Sie implementiert und stellt REST- und WebSocket-APIs zur Verfügung (https://github.com/dgrewe-hse/tic-tac-go). Ihre Aufgabe ist es, eine benutzerfreundliche Web-Anwendung zu entwickeln, die mit diesem Server kommuniziert und es somit Spielern ermöglicht gegeneinander zu spielen.

Im zweiten Labor entwickeln Sie nun eine Spieloberfläche unter Einsatz moderner Webtechnologien aus dem Frontend-Umfeld:

- **Vue.js** als Framework zur komponentenbasierten Entwicklung
- **TypeScript** für typsichere, strukturierte JavaScript-Programmierung
- **Integration der Tic-Tac-Go API**, um dynamische Inhalte zwischen dem Frontend und dem Server auszutauschen.

Wichtige Hinweise für die Bearbeitung der Aufgabe:

- Verwenden Sie ausschließlich folgende Technologien: Vue.js (v3+) und Typescript, oder Angual.js (optional), HTML5, CSS (oder SCSS).
- Es geht nicht darum, möglichst schnell einen fertigen Spielprototypen zu haben, sondern den Zusammenhang der Bausteine und das Zusammenspiel der Frameworkkonzepte zu verstehen.
- Vermeiden Sie es, Lösungen direkt durch KI-Tools generieren zu lassen. Nutzen Sie KI höchstens um Fachbegriffe oder Syntaxdetails nachzuschlagen.
- Bitte reichen Sie das Projekt „tic-tac-toe-3“ **als ZIP-Datei** (ohne Abhängigkeiten) über Moodle ein. Stellen Sie dabei sicher, dass externe Abhängigkeiten und Dependency Ordner NICHT Teil Ihrer Einreichung sind. Nicht fristgerechte Einreichungen oder solche die ein anderes Format beinhalten (z.B.: einzelne Dateien), können nicht berücksichtigt werden.
- **Achten Sie bitte darauf**: Nach entpacken Ihrer Abgabe müssen die Befehle `npm install`, `npm run build` und/oder `npm start` fehlerfrei durchlaufen. Sie können davon ausgehen, dass eine lokale Serverinstanz bei den Dozenten bereits läuft.

Lernziele

Nach Abschluss dieser Aufgabe sind Sie in der Lage:

1. **REST-APIs verstehen und nutzen** können
    - HTTP-Requests (GET, POST) mit Headers und Body-Daten senden
    - JSON-Daten verarbeiten und anzeigen
    - Fehlerbehandlung bei API-Aufrufen implementieren

2. **WebSocket-Verbindungen** für Echtzeit-Updates implementieren können
    - WebSocket-Verbindungen aufbauen und verwalten
    - Echtzeit-Nachrichten empfangen und verarbeiten
    - Verbindungsfehler behandeln und Reconnect-Logik implementieren
3. **State Management** in modernen Frontend-Frameworks beherrschen
    - Komponenten-basierte Architektur nutzen
    - Lokalen State und Props verwenden
    - Event-Handling implementieren
4. **Benutzerfreundliche UI/UX** gestalten
    - Intuitive Spieloberfläche entwickeln
    - Feedback für Benutzeraktionen bereitstellen
    - Loading-States und Fehlermeldungen anzeigen
5. **Moderne Frontend-Entwicklungspraktiken** anwenden
    - Komponenten strukturiert aufbauen
    - Code wiederverwendbar gestalten
    - Responsive Design berücksichtigen

Weitere Informationen zum Server, dessen Schnittstellen und deren Nutzung finden Sie hinter folgenden Links:

- Spezifikation der Schnittstellen: https://github.com/dgrewe-hse/tic-tac-go/blob/main/docs/spec.md
- Erweiterte Aufgabenbeschreibung mit Beispielen: https://github.com/dgrewe-hse/tic-tac-go/blob/main/docs/aufgabe_frontend.md
- Helferskripte zum Testen und Debuggen: https://github.com/dgrewe-hse/tic-tac-go/tree/main/scripts
-

Teilaufgabe1: Grundfunktionalitäten implementieren

Ziel: Entwickeln Sie die Grundstruktur einer Tic-Tac-Toe-Frontend-Anwendung mit allen notwendigen Funktionalitäten zur Interaktion mit dem Backend-Server.

Hinweise: Ziehen Sie zur Umsetzung dieser Teilaufgabe folgende Leitfragen zu Hilfe:

- Welche Komponenten benötigen Sie für die verschiedenen Spielzustände? (z.B. `PlayerForm.vue`, `GameModeSelection.vue`, `GameBoard.vue`, `GameStatus.vue`)
- Wie organisieren Sie Ihr Projekt in einer übersichtlichen Ordnerstruktur?
- Wie gestalten Sie das Layout, sodass die Komponenten ein konsistentes Erscheinungsbild haben?
- Wie nutzen Sie Props und Emits (Vue) oder Props und Callbacks (React), um Daten zwischen Komponenten zu strukturieren?

Zu implementierende Funktionalitäten:

1. **Spieler-Registrierung**
   - Eingabefeld für Spielernamen
   - Erstellung eines Spielers über die REST API (`POST /players`)
   - Speicherung der `playerId` im `localStorage`
   - Automatische Erkennung bei erneutem Besuch (wenn `playerId` im `localStorage` vorhanden)

2. **Spielmodus-Auswahl**
   - Auswahl zwischen **PVP** (Player vs Player) und **PVC** (Player vs Computer)
   - Klare visuelle Unterscheidung der Modi
   - Navigation zu entsprechenden Spielabläufen

3. **PVP-Spiel-Fluss**
   - **Spiel erstellen:**
     - Button/Formular zum Erstellen eines neuen PVP-Spiels
     - API-Aufruf: `POST /games` mit `mode: "PVP"`
     - Anzeige des erstellten Spiels mit `gameId`
     - Status: `WAITING_FOR_PLAYER`
   - **Spiele auflisten:**
     - Liste aller verfügbaren PVP-Spiele mit Status `WAITING_FOR_PLAYER`
     - API-Aufruf: `GET /games?mode=PVP&status=WAITING_FOR_PLAYER`
     - Anzeige von Spielinformationen (Ersteller, Erstellungszeit)
     - Button zum Beitreten zu einem Spiel
   - **Spiel beitreten:**
     - API-Aufruf: `POST /games/{gameId}/join`
     - Automatischer Wechsel zum Spielbildschirm nach erfolgreichem Beitritt

4. **PVC-Spiel-Fluss**
   - **Spiel erstellen:**
     - Button/Formular zum Erstellen eines neuen PVC-Spiels
     - API-Aufruf: `POST /games` mit `mode: "PVC"`
     - Automatischer Start des Spiels (Status: `IN_PROGRESS`)
     - Spieler ist immer **X**, KI ist **O**

5. **Spieloberfläche**
   - **Spielfeld (3x3 Grid):**
     - Visuelle Darstellung des Spielfelds
     - Klickbare Zellen für leere Felder
     - Anzeige von **X** und **O** in belegten Feldern
     - Deaktivierung von Zellen, die nicht klickbar sein sollten

- **Spielstatus-Anzeige:**
  - Aktueller Spieler am Zug (`currentTurn`)
  - Spielstatus (`WAITING_FOR_PLAYER`, `IN_PROGRESS`, `FINISHED`)
  - Gewinner-Anzeige (**X**, **O**, oder **DRAW**)
  - Spieler-Informationen (wer ist **X**, wer ist **O**)

- **Zug machen:**
  - Klick auf leere Zelle
  - API-Aufruf: `POST /games/{gameId}/moves` mit `row` und `col`
  - Aktualisierung des Spielfelds nach erfolgreichem Zug
  - Bei PVC: Automatische Anzeige des KI-Zugs (kommt in der Response)

6. **Echtzeit-Updates mit WebSocket**
   - WebSocket-Verbindung zu `ws://localhost:8080/ws/games/{gameId}` aufbauen
   - Empfang von Spielzustands-Updates in Echtzeit
   - Automatische UI-Aktualisierung bei:
     - Gegnerzug (PVP)
     - Spielerbeitritt (PVP)
     - Spielende (Gewinner/Unentschieden)
   - Fallback zu Polling (`GET /games/{gameId}`) falls WebSocket fehlschlägt

7. **Fehlerbehandlung:**
   - Netzwerkfehler anzeigen
   - Ungültige Züge verhindern/erklären
   - Server-Fehler (400, 403, 404, 500) benutzerfreundlich anzeigen
   - WebSocket-Verbindungsfehler behandeln

8. **Spielende und Neustart**
   - Anzeige des Gewinners oder Unentschiedens
   - Button zum Starten eines neuen Spiels
   - Navigation zurück zur Spielmodus-Auswahl

Für die Umsetzung haben wir Ihnen laufende Serverinstanzen zur Verfügung gestellt. Diese erreichen Sie unter folgende Adressen:

- http://193.197.231.20:8080 (prüfen: http://193.197.231.20:8080/health -> OK)
- http://193.197.231.20:8081 (prüfen: http://193.197.231.20:8081/health -> OK)
- http://193.197.231.20:8082 (prüfen: http://193.197.231.20:8082/health -> OK)

Nutzen Sie diese Server um Ihre Anfragen und Funktionalitäten zu realisieren.

Teilaufgabe2 (optional): Erweiterte Funktionalitäten

Optionale Erweiterungen, die die Bewertung positiv beeinflussen können:

9. **Spielhistorie / Statistik**
   - Lokale Speicherung von beendeten Spielen
   - Anzeige einer Historie der letzten Spiele
   - Statistik (Gewinne, Niederlagen, Unentschieden). Hinweis: Der Server speichert keine Historie, daher müssen Sie diese im Frontend (z.B. `` `localStorage` ``) verwalten
10. **Verbesserte UI/UX**
    - Animierte Züge
    - Sound-Effekte (optional)
    - Responsive Design für mobile Geräte
    - Dark Mode / Theme-Switching

11. **Erweiterte WebSocket-Funktionalität**
    - Automatische Reconnect-Logik mit exponentieller Backoff-Strategie
    - Verbindungsstatus-Anzeige (verbunden/getrennt)
    - Ping/Pong für Keep-Alive

12. **Spiel-Features**
    - Countdown-Timer für Züge (Frontend-seitig)
    - Highlighting der Gewinnlinie
    - Zug-Historie (Undo-Funktion nur im Frontend, nicht im Backend)

Wichtiger Hinweis: Lokale Entwicklungsumgebung

Wenn Sie lokal auf ihrem Rechner entwickeln wollen können Sie den Server und Ihr Frontend auch lokal laufen lassen. Sie müssen sich in diesem Fall nicht mit einem Server im Internet verbinden. Folgendes gilt zu beachten:

- Tic-Tac-Toe Backend-Server läuft lokal auf `` `http://localhost:8080` ``
- Frontend-Anwendung läuft lokal (z.B. auf `` `http://localhost:5173` `` für Vite/Vue oder `` `http://localhost:3000` `` für React)
- Vollständige Kontrolle über die Entwicklungsumgebung

Prüfen Sie folgende Voraussetzungen auf Ihrem System und installieren Sie gegebenfalls Komponenten nach:

- Go 1.22+ installiert (`go version`) siehe: https://go.dev/doc/install
- Node.js und npm installiert (`node --version`, `npm --version`)
- Backend-Server-Repository verfügbar/kopiert (https://github.com/dgrewe-hse/tic-tac-go)

Setup-Schritte:

1. Der Backend-Server wird als **separates Repository** bereitgestellt und ist in der Programmiersprache GO implementiert. Sie müssen dieses Repository klonen oder herunterladen, um den Server lokal ausführen zu können. Folgen Sie dem Link (https://github.com/dgrewe-hse/tic-tac-go) und laden oder klonen Sie sich eine Kopie des Servers.
2. Backend-Server starten (siehe Abschnitt "Server-Beschreibung" oder folgen Sie den Anweisungen in der `README.md` Datei im Code Repository)
   - `cd <server-verzeichnis>`
   - `go mod download`
   - `go build -o tic-tac-go-server ./cmd/server`
   - `go run ./cmd/server`
3. Nach dem Starten des Servers, testen sie die Server-Verfügbarkeit mittels Terminal und curl:
   - `curl http://localhost:8080/health`
   - `# Erwartet: {"status":"ok"}`
4. Frontend-Projekt erstellen (Vue.js oder React.js)
5. Frontend mit lokalem Server verbinde (API-Base-URL: `http://localhost:8080`)

Wichtig:

- Der Backend-Server **muss laufen**, bevor Sie das Frontend starten
- Das Frontend kommuniziert **ausschließlich** mit dem lokalen Server auf Port 8080
- Beide laufen in **separaten Terminal-Fenstern** parallel