Meilenstein Projektplan: «Schiffe versenken»

Autorin: Künsel Emchi

Repository: **LINK**

Commit-Hash für Abgabe: **HASH**

Datum: **XX.XX.XXXX**

## Ziel und Zweck

Der Umfang des Projekts umfasst die Entwicklung eines Multiplayerspiels «Schiffe versenken», bei dem der Server als Source of Truth dient und zwei Spieler:innen gegeneinander antreten.

Die Spieler:innen platzieren ihre Flotte auf einem 8×8-Spielfeld; anschliessend folgen abwechselnde Schüsse.

Pro Lobby steht ein Echtzeit-Chat zur Verfügung, der über WebSockets realisiert ist.

Für die Datensicherung und die Persistenz der Spielzüge, des Action-Logs und des Chats wird SQLite eingesetzt.

Der Fokus dieses Projekts liegt auf dem Aufbau einer stabilen Realtime-Architektur mit klar definierten Regeln und einer einfachen Bedienbarkeit.

## Regeln und Logiken

Das umgesetzte Spiel bilden eine leichte Adaption (Vereinfachung) der Grundregeln des Spiels «Schiffe versenken».[[1]](#footnote-1)

Die Regeln lauten wie folgt:

|  |  |
| --- | --- |
| Name der Regel: | Regelbeschrieb: |
| Flotten | Die eingesetzten Flotten besteht aus je einem *4er* und *3er* sowie je zwei *2er* und *1er* Schiffe. |
| Platzierung | Die Platzierung erfolgt via *Klick*, *horizontal/vertikal*, innerhalb der des *erlaubten Bereichs*, es sind *keine Überlappungen erlaubt* und es gibt *keinen Mindestabstand*. |
| Schussauswertung | Es gibt die Schussauswertungen «Hit», «Miss» und «Sunk». |
| Sichtschutz | Die Spieler:in sieht nur die eigene Schiffe vollständig. Bei der Gegner:in sind nur die eigenen Schüsse und Resultate sichtbar. |
| Sieg | Wer alle gegnerischen Schiffe trifft, gewinnt. |
| Spielfeld | Das umgesetzte Feld ist *8x8* |
| Züge | Die Züge sind alternierend |

## Mockups

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

KI-generierte Inhalte können fehlerhaft sein.

Abbildung : Android Mockup

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

KI-generierte Inhalte können fehlerhaft sein.

Abbildung : Desktop Mockup

## Anforderungen

|  |  |
| --- | --- |
| Name: | Anforderungsart: |
| Dokumentiertes Kommunikationsprotokoll  (Events und Payloads) | Muss |
| Nachvollziehbarkeit:  Jede Aktion muss mit «userId» und Zeitstempel im Action-Log ersichtlich sein. | Muss |
| Persistenz-Layer (SQLLite via Prisma)   * User:in * Lobby * Schiffe * Züge * Nachrichten * ActionLog. | Muss |
| Realtime-Chat via Websocket, pro Lobby | Muss |
| Rejoin mit Snapshot nach Reload | Kann |
| Responsives Frontend (Desktop und Mobil) | Muss |
| Server setzt die komplette Aktions-Logik durch (Platzierung, Zugrecht, Auswertung) | Muss |
| Statistik (Gewinne und Trefferquote) | Kann |
| Zug-Timer (Auto-Pass) | Kann |

## Abnahmekriterien (Prüfbare Punkte)

Folgende Punkte werden als Abnahmekriterium geprüft:

1. Alle Züge und Aktionen werden nach Spielende in der Datenbank gespeichert und sind dort nachvollziehbar.
2. Das Spiel lässt sich von zwei Benutzer:innen in separaten Browserfenstern vollständig spielen.
3. Nach einem Reload des Browsers wird der aktuelle Spielzustand korrekt wiederhergestellt.

## Technologien und Bibliotheken

Für das **Frontend** nutze ich *TypeScript* in Kombination mit *React* und *Vite*. Das Styling erfolgt mit *Tailwind CSS,* was ein responsives Layout ermöglicht. Die Realtime-Kommunikation zum Server wird über socket.io-client realisiert.

Im **Backend** nutze ich *Node.js* in *TypeScript* mit *Express* als Server. *Socket.IO* verwende ich für die Echtzeit-Kommunikation zwischen Server und Clients. Für die Valiedierung der eingehenden Daten und Events verwende ich *Zod.*

Die Persistenz übernimmt eine *SQLite***-**Datenbank. *Prisma ORM* wird genutzt, um einen direkten Zugriff auf die Datenbank aus *TypeScript* zu ermöglichen.

Zudem wird *Prettier* für die automatische Formatierung des Codes genutzt.

## Kommunikation zwischen Client und Server

Die Kommunikation zwischen Client und Server erfolgt in Echtzeit über WebSockets mithilfe von Socket.IO.

Zusätzlich kann HTTP/REST optional für Healthchecks und Debugging genutzt werden.

## Übertragene Daten

Folgende Daten werden zwischen Client und Server ausgetauscht:

|  |  |
| --- | --- |
| Daten: | Beschrieb: |
| Benutzeridentifikation | * name * userId |
| Lobby-Informationen | * erstellen * beitreten * Status |
| Platzierungsdaten | * Schiffe * Positionen |
| Spielzüge | * Koordinaten der Schüsse |
| Chatnachrichten | * Text * User * Zeitstempel |
| Snapshots des Spielzustands | * Boards * Spielerstatus * Chatverlauf |
| Fehlermeldungen | * Validierungsfehler * Berechtigungsfehler |

Alle eingehenden Event Daten werden serverseitig mit Zod-Schemas validiert. Der Server erzwingt ausserdem Turn-Ownership und Fog-of-War (Sichtschutz), sodass nur zulässige Daten übertragen werden.

## Zweck des Austauschs

Die Datenübertragung dient der Anmeldung und Identifikation von Benutzer:innen, der Verwaltung von Lobbys (Erstellen, Beitreten und Statusänderungen) sowie der Übermittlung der Flottenplatzierung inklusive Validierung und Speicherung. Sie ermöglicht ausserdem das Abfeuern und Auswerten von Schüssen in Echtzeit und die Synchronisierung des aktuellen Spielzustands zwischen allen Clients. Darüber hinaus unterstützt sie die Realtime-Kommunikation im Chat innerhalb einer Lobby und erlaubt die Übermittlung von Fehlermeldungen, wenn ungültige Aktionen erkannt werden.

## Übersicht der Events

### Client zu Server (Eingaben)

|  |  |  |
| --- | --- | --- |
| Event | Payload-Beispiel | Zweck |
| auth.login | {name} | Benutzer anlegen/identifizieren |
| Lobby.create | {name} | Lobby erstellen |
| Lobby.join | {lobbyId} | Lobby beitreten (Room) |
| Lobby.ready | {lobbyId, ready} | Bereitschaft in der Platzierphase melden |
| Board.place | { lobbyId, ships:[{type,cells:[{x,y}]}] } | Flotte senden (Validierung und Speicherung) |
| Shot.fire | |  | | --- | | { lobbyId, x, y } |  |  | | --- | |  | | Schuss auslösen (nur am Zug) |
| Chat.send | |  | | --- | | { lobbyId, text } |  |  | | --- | |  | | Chatnachricht senden |

### Server zu Client (Resultate/State)

|  |  |  |
| --- | --- | --- |
| Event | Payload-Beispiel | Zweck |
| Lobby.snapshot | { state, players, you:{seat}, yourBoardMask, enemyMask, chat } | Aktuellen Stand synchronisieren |
| Placing.updated | |  | | --- | | { seatIndex, done } |  |  | | --- | |  | | Platzierstatus je Spieler |
| Game.start | { firstTurnSeat } | Startsignal Spiel |
| Short.result | `{ bySeat,x,y,result:'MISS' | ‘Treffer’ |
| Turn.change | { seatIndex } | Zugwechsel anzeigen |
| Game.over | |  | | --- | | { winnerSeat } |  |  | | --- | |  | | Spielende kommunizieren |
| Chat.message | |  | | --- | | { id,user,text,createdAt } |  |  | | --- | |  | | Chatnachricht verteilen |
| error | { code,message } | Validierungs- oder Berechtigungsfehler melden |

## Datenmodell (Konzept)

Das Datenmodell umfasst die zentralen Entitäten User, Lobby, LobbyPlayer, Ship, Shot, Message und ActionLog.

Die Entität User repräsentiert die Nickname-Identität der Spieler:innen.

Lobby steht für eine einzelne Spielinstanz und speichert unter anderem den Status (waiting, placing, playing, finished) sowie die Gewinner-ID.

LobbyPlayer ordnet Benutzer:innen einer Lobby zu und enthält Angaben wie Sitz/Index, Bereitschaftsstatus und verbleibende Schiffszellen.

Ship beschreibt Typ und Zellen einer Flotte sowie bereits getroffene Segmente (z. B. in JSON-Form).

Shot hält Koordinaten der Schüsse, Schütze, Ziel-Sitz, Resultat und Zeitpunkt fest.

Message bildet die Chatnachrichten pro Lobby ab.

ActionLog speichert alle Aktionen chronologisch (JOIN, PLACE, SHOT, TURN, START, END, CHAT) zusammen mit Payload und Zeitstempel.

Arbeitsplan /Balkendiagramm

|  |  |
| --- | --- |
| Phase | Dokumente |
| M1 (Planung) | * Doku * Mockups * Protokoll * Datenmodell |
| M2 (Grundgerüst) | * Login * Lobby * Chat (Realtime, in-memory) * Snapshot-Skeleton |
| M3 (Spielbar + Persistenz) | * Platzieren (Validierung und DB) * Schüsse (Auswertung und DB) * Turn-Handling * Snapshot |
| M4 (Feature-Complete) | * Zod überall * Rejoin/Snapshot stabil * Action-Log-UI * Politur. |
| Finale Abgabe | * README/Install * Screenshots * Präsentation. |

Ein Bild, das Text, Diagramm, Reihe, Zahl enthält.

KI-generierte Inhalte können fehlerhaft sein.

Abbildung : GANTT Chart

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

KI-generierte Inhalte können fehlerhaft sein.

Ein Bild, das Text, Screenshot, Software, Multimedia-Software enthält.

KI-generierte Inhalte können fehlerhaft sein.

## Anhänge

## 10. Anhänge

- Mockup-Bilder (Desktop/Mobile)

- Event-Schemas (Zod) – Auszug/Beispiele

1. Gesellschaftsspiele.Spielen.de. Schiffe versenken [PDF]. <https://gesellschaftsspiele.spielen.de/uploads/files/3284/58e6634bd08c1.pdf> (abgerufen am 15.09.2025) [↑](#footnote-ref-1)