Bomb.io
Konzeptbericht

Auftraggeber	Tomas Fehr
Projektleiter	Mario Aeberhard
Autor	Mario Aeberhard, Loic Tobler, Nicolas Ammeter
Klassifizierung	*Nicht klassifiziert, ~~Intern, Vertraulich, GEHEIM~~*
Status	*In Arbeit, Genehmigt*

Änderungsverzeichnis

Datum	Version	Änderung	Autor
16.10.2025	V1.0	Initiale Erstellung	Mario Aeberhard, Loic Tobler, Nicolas Ammeter

Inhaltsverzeichnis

1 Zusammenfassung

Bomb.io ist ein browserbasiertes Mehrspieler-Spiel, das auf dem Prinzip von Hot Potato basiert. Mehrere Spieler bewegen sich gleichzeitig in einer 3D-Umgebung und versuchen, eine virtuelle Bombe durch Fangen an andere Spieler weiterzugeben, bevor der Timer abläuft. Ziel ist es, als letzter Spieler im Spiel zu bleiben.

Kapitel 2 beschreibt die Systemanforderungen mit den zentralen Entitäten Spieler, Lobby, Bombe und Runde. Diese bilden die Grundlage für die Anwendungsfälle Lobby erstellen, Lobby beitreten, Spiel starten und Bombe weitergeben durch Fangen.
Kapitel 3 stellt den logischen Aufbau der Benutzerschnittstelle dar. Sie umfasst eine Startseite, auf der Spieler ihren Namen und ihre Farbe wählen, eine Lobbyansicht zum Beitreten oder Erstellen von Spielen, die Spielansicht mit Bewegungssteuerung sowie eine Ergebnisseite nach jeder Runde.
Kapitel 4 erläutert die Systemarchitektur**.** Das System basiert auf Node.js und Express.js für den Server, Socket.IO für die Echtzeitkommunikation und Three.js für die 3D-Darstellung. Der Server verwaltet Lobbys, Spielrunden, Bombenübergaben und Kollisionsprüfungen.
Kapitel 5 beschreibt die Qualitätssicherung, welche Funktion, Stabilität, Reaktionszeit und Erweiterbarkeit testet.
Kapitel 6 verweist auf den Projektplan Version 2.0 vom 24.10.2025, in dem die bisherigen Aufwände und Risiken aktualisiert und Maßnahmen zur Stabilisierung der Echtzeitkommunikation definiert wurden.

2 Systemanforderungen

2.1 Fachliche Entitätstypen

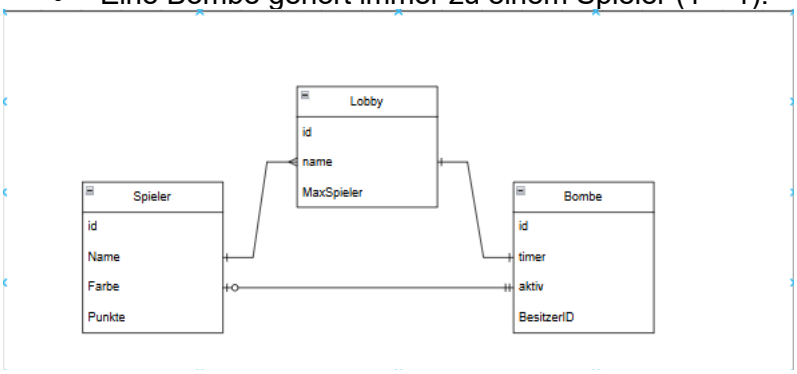
Im Problembereich von Bomb.io stehen die Spieler und deren Interaktion im Mittelpunkt. Das System verwaltet Sitzungen (Lobbys), Spieler, Runden und Bomben. Diese Entitäten bilden die fachliche Grundlage des Spiels.

Entitätstypen und Beziehungen (vereinfacht):

- **Spieler**
  Enthält die Identität eines Teilnehmers (Name, Farbe, Punktestand).
- **Lobby**
  Eine Spielinstanz, in welcher mehrere Spieler miteinander verbunden sind. Sie enthält die Spielregeln, den Timer und den aktuellen Spielzustand.
- **Bombe**
  Repräsentiert das zentrale Spielelement. Sie hat einen Timer und ist jeweils einem Spieler zugeordnet.

Diese Entitäten stehen zueinander in folgenden Beziehungen:

- Eine Lobby enthält mehrere Spieler (1 – n).
- Eine Lobby verwaltet eine aktive Bombe (1 – 1).
- Eine Bombe gehört immer zu einem Spieler (1 – 1).



2.2 Anwendungsfälle / Product-Backlog

2.2.1 Akteure

Der Hauptakteur ist der Spieler, der sich am System anmeldet, eine Lobby erstellt oder einer bestehenden beitritt und anschließend am Spiel teilnimmt. Eine besondere Rolle nimmt der Host ein, der beim Erstellen der Lobby automatisch als deren Besitzer festgelegt wird und zusätzliche Rechte besitzt, etwa das Starten oder Beenden des Spiels. Das System selbst fungiert als Serverkomponente, die Lobbys verwaltet, Spielzustände synchronisiert und den Bombentimer kontrolliert.

2.2.2 Überblick über die Anwendungsfälle

Das System Bomb.io umfasst in seiner ersten Ausbaustufe vier zentrale Anwendungsfälle: Lobby erstellen, Lobby beitreten, Spiel starten und Bombe weitergeben. Diese bilden die Kernfunktionen des Spielablaufs und definieren den grundlegenden Daten- und Interaktionsfluss zwischen Spieler und System. Sie können später um zusätzliche Funktionen wie „Runde auswerten“ oder „Statistik anzeigen“ erweitert werden.

2.2.3 Lobby erstellen

Ein Spieler kann eine neue Spielumgebung anlegen, um eine Partie zu starten. Beim Erstellen der Lobby legt der Spieler grundlegende Parameter fest, beispielsweise den Namen der Lobby, die maximale Anzahl an Mitspielern und die Dauer des Bombentimers. Nach der Bestätigung erzeugt das System eine eindeutige Lobby-ID und stellt die Benutzeroberfläche bereit, in der der Host auf weitere Spieler wartet. Damit wird die Grundlage für eine neue Spielsitzung geschaffen, die später von anderen Benutzern betreten werden kann.

2.2.4 Lobby beitreten

Sobald eine Lobby existiert, können andere Spieler über einen Code beitreten. Der Spieler gibt den Lobby-Code ein, woraufhin das System überprüft, ob die Lobby gültig und noch nicht voll ist. Nach erfolgreicher Prüfung wird der Spieler in die Lobby aufgenommen, sein Profil erscheint in der Teilnehmerliste, und er kann sich auf den Spielstart vorbereiten. Dieser Anwendungsfall ermöglicht es mehreren Personen, sich dynamisch zu einer Spielgruppe zusammenzuschließen.

2.2.5 Spiel starten

Der Host startet das Spiel, sobald genügend Spieler anwesend sind. Das System initialisiert die erste Runde, erstellt die Bombe und wählt zufällig einen Spieler aus, der sie zu Beginn hält. Ab diesem Moment läuft der Timer, und alle Spieler sehen den aktuellen Zustand in Echtzeit. Der Spielstart markiert den Übergang von der Vorbereitungs- in die aktive Phase und ist damit ein zentraler Steuerungspunkt des Systems.

2.2.6 Bombe weitergeben

Während des Spiels kann der Spieler, der die Bombe aktuell besitzt, sie an einen anderen Mitspieler weitergeben, bevor der Timer abläuft. Das System prüft diese Aktion, überträgt den Besitz der Bombe und setzt den Timer neu. Erfolgt die Weitergabe rechtzeitig, bleibt die Runde aktiv; läuft der Timer ab, verliert der betroffene Spieler und scheidet aus. Dieser Anwendungsfall bildet die dynamische Kerninteraktion des Spiels und erfordert eine präzise Synchronisation zwischen Client und Server.

2.3 Anwendungsfall Beschreibungen

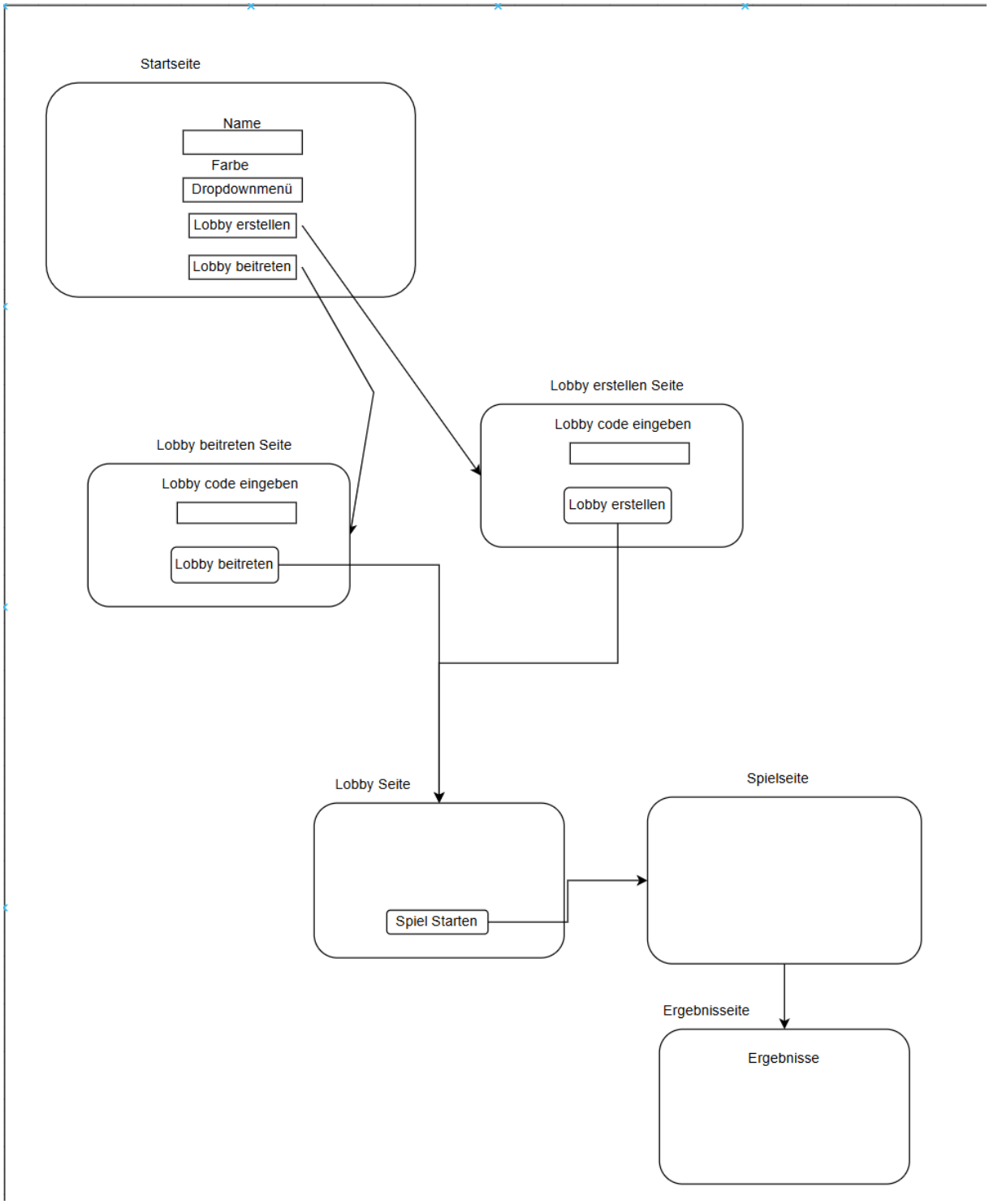
Anwendungsfall UC1 – Lobby erstellen	
Kurzbeschreibung	Ein Spieler erstellt eine neue Lobby, um ein Spiel zu starten und Mitspieler einzuladen.
Akteure	Spieler (Host), Systemserver
Vorbedingungen	Der Spieler ist im System angemeldet und besitzt keine andere aktive Lobby.
Ablauf	• Der Spieler wählt in der Benutzeroberfläche die Option „Neue Lobby erstellen“. • Das System zeigt eine Eingabemaske mit den Parametern Lobbyname, maximale Spielerzahl und Timerdauer. • Der Spieler bestätigt die Eingaben. • Das System prüft die Angaben und legt eine neue Lobby-Entität im Server an. • Eine eindeutige Lobby-ID wird generiert. • Der Spieler wird als Host eingetragen, und die Lobby-Ansicht öffnet sich.
Resultat	Eine neue Lobby ist erstellt, und der Host kann andere Spieler einladen.
Ausnahmen	• Ungültige Eingaben oder leere Felder verhindern die Erstellung. • Verbindungsprobleme unterbrechen den Vorgang.

Anwendungsfall UC2 – Lobby beitreten	
Kurzbeschreibung	Ein Spieler tritt einer bestehenden Lobby bei, um an einem Spiel teilzunehmen.
Akteure	Spieler, Systemserver
Vorbedingungen	Eine aktive Lobby existiert, und der Spieler besitzt einen gültigen Lobby-Code.
Ablauf	• Der Spieler wählt in der Benutzeroberfläche die Option „Lobby beitreten“. • Das System fordert den Code an. • Der Server prüft, ob die Lobby existiert und noch Plätze frei sind. • Der Spieler wird der Lobby hinzugefügt. • Die Lobby-Ansicht aktualisiert sich und zeigt den neuen Teilnehmer an.
Resultat	Der Spieler befindet sich erfolgreich in der Lobby und kann am Spiel teilnehmen.
Ausnahmen	• Die Lobby existiert nicht oder ist voll. • Der Code ist ungültig. • Netzwerkfehler verhindern den Beitritt.

Anwendungsfall UC3 – Spiel starten	
Kurzbeschreibung	Der Host startet das Spiel, sobald genügend Spieler in der Lobby sind.
Akteure	Host, Systemserver
Vorbedingungen	Mindestens zwei Spieler befinden sich in der Lobby, und alle sind verbunden.
Ablauf	• Der Host klickt auf „Spiel starten“. • Das System prüft, ob die Voraussetzungen erfüllt sind. • Der Server erstellt eine neue Runde und generiert die Bombe. • Ein Spieler wird zufällig als Bombenbesitzer ausgewählt. • Das System startet den Timer und sendet den Status an alle Teilnehmer.
Resultat	Das Spiel wird gestartet, die erste Runde beginnt, und ein Spieler hält die Bombe.
Ausnahmen	• Zu wenige Spieler oder Verbindungsprobleme verhindern den Start. • Ein Serverfehler unterbricht die Initialisierung.

Anwendungsfall UC4 – Bombe weitergeben	
Kurzbeschreibung	Der Spieler, der die Bombe besitzt, gibt sie während der laufenden Runde an einen anderen Spieler weiter.
Akteure	Spieler (aktueller Bombenbesitzer), Systemserver
Vorbedingungen	Das Spiel läuft, und der Timer ist aktiv.
Ablauf	Der Spieler versucht einen anderen Spieler zu fangen und ihm die Bombe weiterzugeben.
Resultat	Die Bombe wurde erfolgreich weitergegeben, und das Spiel läuft weiter.
Ausnahmen	• Der Timer läuft ab, bevor die Übergabe abgeschlossen wurde. • Zielspieler ist inaktiv. • Verbindungsprobleme verhindern die Aktion.

3 Benutzerschnittstelle



4 Systemarchitektur

Die gewählte Lösung basiert auf einer browserbasierten Multiplayer-Web-Applikation mit Node.js + Express.js als Server, Socket.IO für die Echtzeitkommunikation und Three.js zur 3D-Darstellung im Browser. Ziel ist ein leichtgewichtiges Spiel ohne Installation, das direkt per URL startet.

4.1 Gliederung der Lösung

Präsentationsschicht (Client, Browser)

- **View / Rendering**
  - ThreeScene (Initialisierung der Three.js-Szene)
  - CameraController, Lighting, MapRenderer
  - HUDView (Timer, Bombenanzeige, Spielerstatus)
- **Input & UI**
  - InputController (Tastatur/Maus)
  - Menu/LobbyUI (Name eingeben, Lobby beitreten/erstellen)
- **Netzwerk-Client**
  - SocketClient (Socket.IO-Client, Event-Handling)
- **Client-seitige Logik (thin)**
  - ClientGameController (Statusdarstellung, minimale Vorhersage)
  - **Anti-Cheat (Client-seitig leichtgewichtig):** nur UI-Beschränkungen, Logik bleibt server-autoritätsbasiert

Applikationsschicht (Server, Node.js)

- **WebSocket-Gateway**
  - SocketGateway (Annahme von Verbindungen, Namespaces/Rooms)
- **Spiel-Domäne (server-autorität)**
  - MatchmakerService (Lobbys/Rooms)
  - PlayerService (Spieleranmeldung, Zustände)
  - GameLoopService (Tick/Timer, Rundensteuerung)
  - BombService (Bombe zuweisen, Übergaben validieren, Explosion)
  - CollisionService (Berührungsprüfung Server-seitig)
  - RoundService (Rundenstart/-ende, Schwierigkeitseskalation)
  - StateSyncService (Diff/State an Clients senden)
- **Infrastruktur**
  - ExpressApi (statisches Hosting der Client-Assets, Health-Check)
  - Logger
  - (optional später) Metrics

- **Persistenz**: initial **keine Datenbank** vorgesehen (nur flüchtiger Spielzustand im RAM).

4.2 Schnittstellen

Gemäss Studie/Varianten sind keine externen System-Schnittstellen vorgesehen (keine Konten, keine Fremdsysteme, kein Import/Export). Fokus ist die interne Client↔Server-Echtzeitschnittstelle über Socket.IO (OSI-Layer 7, Nachrichten-basiert).

Systeminterne Schnittstelle (WebSocket/Socket.IO)

Transport: WebSocket (Socket.IO)
Rollen: Client (Browser) ↔ Server (Node.js)
Grundsatz: *Server-Autorität* für alle spielrelevanten Zustände (Positionsvalidierung, Bombenbesitz, Timer).

Ereignisse (Client → Server)

1. **join_lobby**

```
{
  "nickname": "Loic",
  "lobbyId": "ABC123"
}
```

Wirkung: Server legt Spieler an, ordnet Room zu, antwortet mit lobby_state/joined.
Validierung: Nickname non-empty, Lobby existent/erstellt.

2. **player_input**

```
{
  "seq": 42,
  "dt": 16,
  "input": {"up":true,"down":false,"left":false,"right":true}
}
```

Wirkung: Server aktualisiert gewünschte Richtung, speichert seq für ACK/Glättung.

3. **request_bomb_pass**

```
{
  "targetPlayerId": "p2"
}
```

Wirkung: Server prüft Besitz, Kollisions-/Distanzbedingung und Timer, setzt Bombenbesitz um, emittiert bomb_state.

4. **request_start**

```
{ "lobbyId": "ABC123" }
```

Wirkung: Startet Runde, initialisiert Timer/Bombe(n).

5. **pong (Latenz-Messung)**

```
{ "ts": 1731324000000 }
```

Ereignisse (Server → Client)

1. **joined**

```
{
  "playerId": "p1",
  "lobbyId": "ABC123"
}
```

2. **lobby_state**

```
{
  "players": [{"id":"p1","name":"Loic"},{"id":"p2","name":"Nico"}],
  "status": "waiting|running"
}
```

3. **game_state** ***(periodisch, z. B. 15–30 Hz)***

```
{
  "tick": 988,
  "players": [
    {"id":"p1","x":1.2,"y":0.0,"z":-3.4,"dir":[0.8,0,0.6]},
    {"id":"p2","x":-0.3,"y":0.0,"z":2.1}
  ],
  "bombs": [
    {"owner":"p1","timeLeftMs":1800}
  ],
  "round": {"number":2,"difficulty":"+bombs|-time"}
}
```

4. **bomb_state**

```
{
  "owner":"p2",
  "prevOwner":"p1",
  "timeLeftMs":2500
}
```

5. **explosion**

```
{
  "playerId":"p2",
  "reason":"timer_zero"
}
```

6. **round_end**

```
{
  "winner":"p3",
  "stats":{"durationSec":125,"passes":14}
}
```

7. **error**

```
{
  "code":"INVALID_PASS",
  "message":"Target not in range"
}
```

8. **ping**

```
{ "ts": 1731324000000 }
```

5 Qualitätssicherung

Nr.	***Abgedeckter Anwendungsfall oder Use story Beschreibung***	***Beschreibung***
TC01	*B1: Funktionserfüllung (Spielrunden und Mehrspieler)*	*Ein Systemtest mit mehreren Spielern wird durchgeführt, um sicherzustellen, dass alle geplanten Spielfunktionen (z. B. Timer, Bomben-Übergabe, Ablauf mehrerer Runden) korrekt funktionieren. Mehrere Spieler spielen gleichzeitig mehrere Runden.*
TC02	*B2: Effizienz (Spielstart)*	*Ein Benutzer ruft die Spiel-URL auf und startet das Spiel. Der Test überprüft, ob das Spiel mit maximal 5 Klicks spielbereit ist.*
TC03	*B3: Zuverlässigkeit (Stabilität)*	*Ein Belastungstest wird durchgeführt, bei dem 50 Spielrunden mit mehreren Spielern gespielt werden, um sicherzustellen, dass das Spiel höchstens einmal abstürzt.*
TC04	*B4: Benutzbarkeit (Reaktionszeit)*	*Ein Benutzer führt verschiedene Eingaben (z. B. Bewegungen, Aktionen) aus, und die Reaktionszeit des Spiels wird gemessen, um sicherzustellen, dass sie unter 100 ms pro Eingabe liegt.*
TC05	*B5: Sicherheit (Zugriffskontrolle)*	*Ein Testbenutzer versucht, ohne Berechtigung auf eine geschützte Spielsession zuzugreifen. Der Test überprüft, ob der Zugriff durch Lobby-Passwörter oder andere Schutzmechanismen verhindert wird.*
TC06	*E1: Erweiterbarkeit (Neue Maps)*	*Ein Entwickler fügt eine neue Spiel-Map hinzu, und der Test überprüft, ob die Integration innerhalb von 2 Stunden erfolgreich abgeschlossen werden kann.*
TC07	*E3: Übertragbarkeit (Cross-Browser-Kompatibilität)*	*Das Spiel wird in den Browsern Chrome, Firefox und Edge gestartet und getestet, um sicherzustellen, dass es in allen drei Browsern stabil läuft.*
TC08	*E4: Wiederverwendbarkeit (Module)*	*Zwei Module (z. B. Timer, Spieler-Management) werden isoliert getestet, um sicherzustellen, dass sie in einem anderen Projekt ohne Anpassungen verwendet werden können.*

6 Projektplanung

Siehe Projektplan, Version 2.0 vom 24.10.2025