

Distributed Systems Projekt: Svindler

Baruyr Bousnaian & Ivo Cavelti

Frühlingssemester 2025

- Svindler ist ein soziales Deduktionsspiel.

- Svindler ist ein soziales Deduktionsspiel.
- Ziel: den “Impostor” anhand von Wortbeschreibungen identifizieren.

- Svindler ist ein soziales Deduktionsspiel.
- Ziel: den “Impostor” anhand von Wortbeschreibungen identifizieren.
- Verteiltes System mit Server Sent Events für Live-Spiel-Updates.

- ① Ein geheimer Begriff (z.B. eine Person oder Figur) wird an alle Spieler vergeben – ausser an eine Person: den **Impostor**.

- 1 Ein geheimer Begriff (z.B. eine Person oder Figur) wird an alle Spieler vergeben – ausser an eine Person: den **Impostor**.
- 2 Die Spieler beschreiben reihum den Begriff mit einem einzelnen Wort oder einer kurzen Phrase.

- ① Ein geheimer Begriff (z.B. eine Person oder Figur) wird an alle Spieler vergeben – ausser an eine Person: den **Impostor**.
- ② Die Spieler beschreiben reihum den Begriff mit einem einzelnen Wort oder einer kurzen Phrase.
- ③ Der Impostor kennt den Begriff nicht und muss glaubwürdig mitraten.

- ① Ein geheimer Begriff (z.B. eine Person oder Figur) wird an alle Spieler vergeben – ausser an eine Person: den **Impostor**.
- ② Die Spieler beschreiben reihum den Begriff mit einem einzelnen Wort oder einer kurzen Phrase.
- ③ Der Impostor kennt den Begriff nicht und muss glaubwürdig mitraten.
- ④ Nach drei Runden diskutieren die Spieler und stimmen ab, wer ihrer Meinung nach der Impostor ist.

- 1 Ein geheimer Begriff (z.B. eine Person oder Figur) wird an alle Spieler vergeben – ausser an eine Person: den **Impostor**.
- 2 Die Spieler beschreiben reihum den Begriff mit einem einzelnen Wort oder einer kurzen Phrase.
- 3 Der Impostor kennt den Begriff nicht und muss glaubwürdig mitraten.
- 4 Nach drei Runden diskutieren die Spieler und stimmen ab, wer ihrer Meinung nach der Impostor ist.
- 5 Wird der Impostor enttarnt, hat er eine letzte Chance zu gewinnen, indem er den geheimen Begriff errät.

- Webanwendung mit drei Hauptkomponenten:

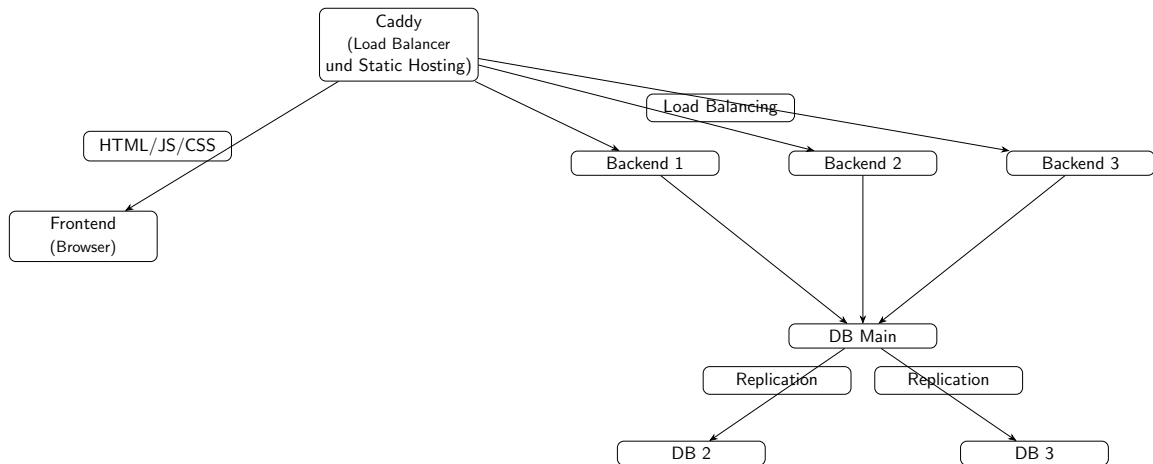
- Webanwendung mit drei Hauptkomponenten:
 - Frontend: Angular SPA (Single Page Application)

- Webanwendung mit drei Hauptkomponenten:
 - Frontend: Angular SPA (Single Page Application)
 - Backend: FastAPI (Python, REST-API)

- Webanwendung mit drei Hauptkomponenten:
 - Frontend: Angular SPA (Single Page Application)
 - Backend: FastAPI (Python, REST-API)
 - Datenbank: MongoDB (Replica Set)

- Webanwendung mit drei Hauptkomponenten:
 - Frontend: Angular SPA (Single Page Application)
 - Backend: FastAPI (Python, REST-API)
 - Datenbank: MongoDB (Replica Set)
- Load-Balancing und TLS durch Caddy

- Webanwendung mit drei Hauptkomponenten:
 - Frontend: Angular SPA (Single Page Application)
 - Backend: FastAPI (Python, REST-API)
 - Datenbank: MongoDB (Replica Set)
- Load-Balancing und TLS durch Caddy
- Deployment mittels Docker Compose



- **Home:** Startseite zur Eingabe oder Erstellung eines Spielraums

- **Home:** Startseite zur Eingabe oder Erstellung eines Spielraums
- **Room:** Lobby-Ansicht, in der sich die Spieler vor Spielstart versammeln

- **Home:** Startseite zur Eingabe oder Erstellung eines Spielraums
- **Room:** Lobby-Ansicht, in der sich die Spieler vor Spielstart versammeln
- **Game:** Hauptspielansicht mit Rundenanzeige, Beschreibungseingabe und Abstimmungsfunktion

- **Home:** Startseite zur Eingabe oder Erstellung eines Spielraums
- **Room:** Lobby-Ansicht, in der sich die Spieler vor Spielstart versammeln
- **Game:** Hauptspielansicht mit Rundenanzeige, Beschreibungseingabe und Abstimmungsfunktion
- Alle Komponenten reagieren auf Datenänderungen per Change Streams (MongoDB) und halten das UI synchron

- Synchronisierung in Echtzeit zwischen mehreren Clients

- Synchronisierung in Echtzeit zwischen mehreren Clients
- Zustandsverwaltung in einem zustandslosen Backend

- Der gesamte Spielzustand wird in MongoDB gespeichert

- Der gesamte Spielzustand wird in MongoDB gespeichert
- Pro Spiel ein MongoDB-Dokument

- Der gesamte Spielzustand wird in MongoDB gespeichert
- Pro Spiel ein MongoDB-Dokument
- MongoDB Change Streams “beobachten” Game-Dokumente

- Der gesamte Spielzustand wird in MongoDB gespeichert
- Pro Spiel ein MongoDB-Dokument
- MongoDB Change Streams “beobachten” Game-Dokumente
- Jede Game-Änderung wird via Server-Sent Events (SSE) an alle verbundenen Clients geschickt

- SSE-Verbindung stabil am Leben halten

- SSE-Verbindung stabil am Leben halten
 - Heartbeat-Events alle 10 Sekunden

- SSE-Verbindung stabil am Leben halten
 - Heartbeat-Events alle 10 Sekunden
- Probleme mit fehlschlagenden automatischen SSE-Reconnects

- SSE-Verbindung stabil am Leben halten
 - Heartbeat-Events alle 10 Sekunden
- Probleme mit fehlschlagenden automatischen SSE-Reconnects
 - GZIP-Encoding durch Caddy deaktiviert

Codebeispiel: SSE mit Change Stream

```
def game_event_generator(self, game_id: str):
    q = queue.Queue()

    def watch_changes():
        try:
            for change in self.game_repository.game_update_listener(game_id):
                q.put(f"data: {change}\n\n")
        except Exception as e:
            q.put(f"event: error\ndata: {str(e)}\n\n")

    def send_heartbeat():
        while True:
            time.sleep(10)
            q.put(": heartbeat\n\n")

    threading.Thread(target=watch_changes, daemon=True).start()
    threading.Thread(target=send_heartbeat, daemon=True).start()

    while True:
        yield q.get()
```

- Voll funktionsfähiges Multiplayer-Spiel

- Voll funktionsfähiges Multiplayer-Spiel
- Stabile Performance bei mehreren gleichzeitigen Sessions

- Voll funktionsfähiges Multiplayer-Spiel
- Stabile Performance bei mehreren gleichzeitigen Sessions
- Einfaches Deployment dank Docker und Infrastrukturautomatisierung

- Voll funktionsfähiges Multiplayer-Spiel
- Stabile Performance bei mehreren gleichzeitigen Sessions
- Einfaches Deployment dank Docker und Infrastrukturautomatisierung
- Spielverlauf persistent gespeichert

- Svindler demonstriert die Anwendung verteilter Systeme in einem Spielkontext.

- Svindler demonstriert die Anwendung verteilter Systeme in einem Spielkontext.
- Reaktive, fehlertolerante Architektur mit modernen Tools umgesetzt.

- Svindler demonstriert die Anwendung verteilter Systeme in einem Spielkontext.
- Reaktive, fehlertolerante Architektur mit modernen Tools umgesetzt.
- Erweiterungspotenzial:

- Svindler demonstriert die Anwendung verteilter Systeme in einem Spielkontext.
- Reaktive, fehlertolerante Architektur mit modernen Tools umgesetzt.
- Erweiterungspotenzial:
 - Sicherheit

- Svindler demonstriert die Anwendung verteilter Systeme in einem Spielkontext.
- Reaktive, fehlertolerante Architektur mit modernen Tools umgesetzt.
- Erweiterungspotenzial:
 - Sicherheit
 - Lokaler Spielmodus

- Svindler demonstriert die Anwendung verteilter Systeme in einem Spielkontext.
- Reaktive, fehlertolerante Architektur mit modernen Tools umgesetzt.
- Erweiterungspotenzial:
 - Sicherheit
 - Lokaler Spielmodus
 - Eigene Wörterlisten

- Svindler demonstriert die Anwendung verteilter Systeme in einem Spielkontext.
- Reaktive, fehlertolerante Architektur mit modernen Tools umgesetzt.
- Erweiterungspotenzial:
 - Sicherheit
 - Lokaler Spielmodus
 - Eigene Wörterlisten
 - Spiel-Statistiken