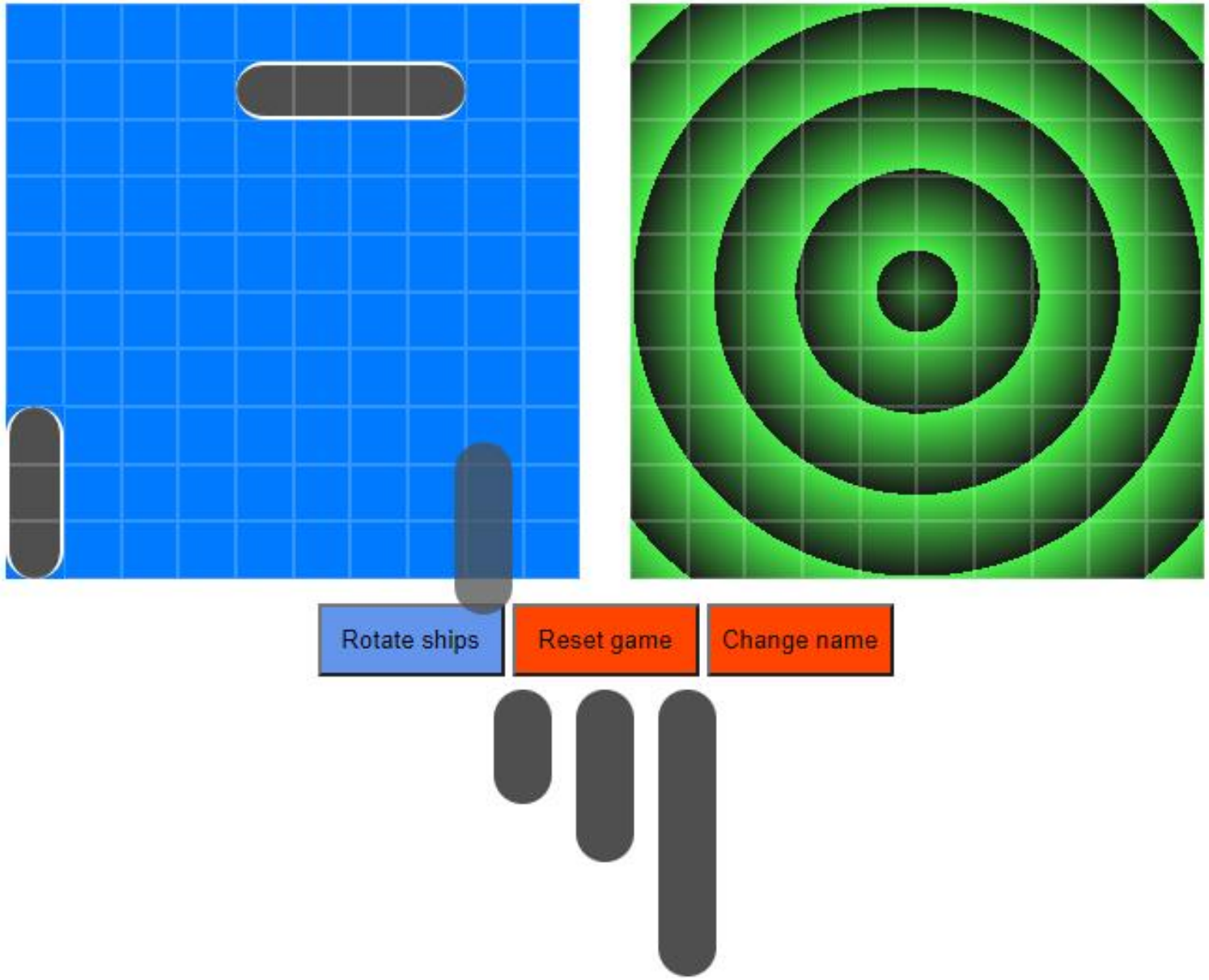
CARLO D.

DAMIAN D.

LAURENT D.

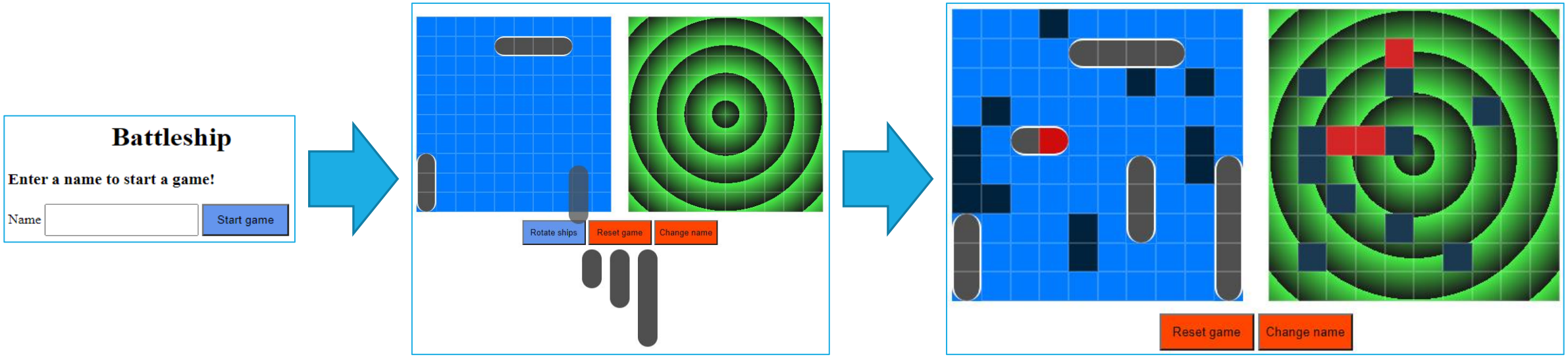
Gruppe 07
Battleship



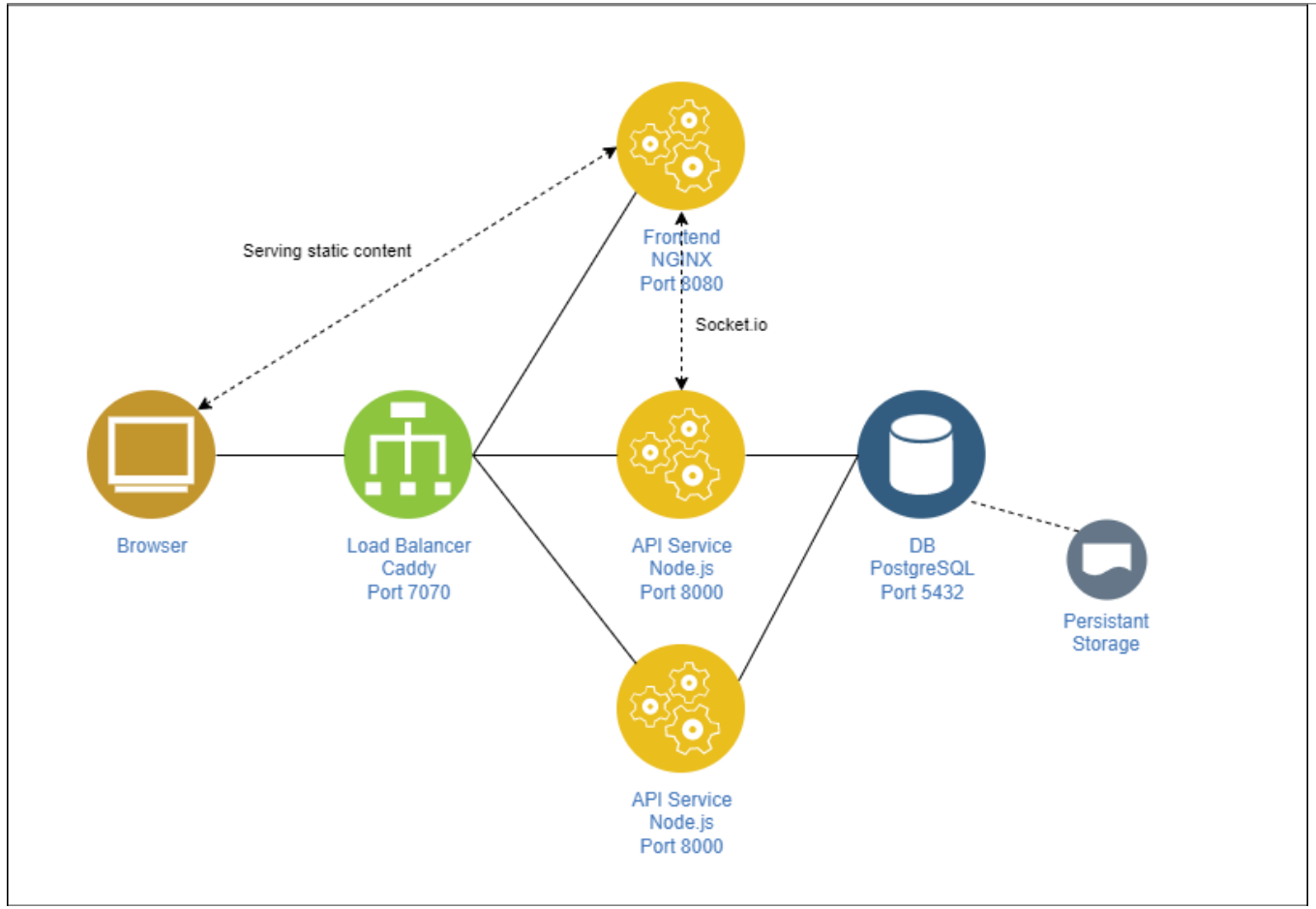
Inhalt

- Spielablauf
- Architektur
- Datenbank
- Backend
- Frontend
- Sockets
- Load Balancer

Spielablauf



Architektur



Datenbank

- Speichert den Spielstand der Spieler von Battleship
- PostgreSQL Image von Docker Hub initialisiert mit Umgebungvariablen
- Tabelle als SQL-Datei erstellt, die mit docker-compose als docker-entrypoint übergeben wird
- Dateisystem des Localhosts als Ablageort

```
volumes:
    - ./service/init_sql/:/docker-entrypoint-initdb.d
    - ./battleship-data:/var/lib/postgresql/data
```

Backend

- Node.js (JavaScript) mit Express als HTTP-Server

- Verfügt über einen Healthcheck, der Status Code 200 zurückgibt

- Stateless API und Failover möglich

- Spiellogik: Das Spielfeld wird als eindimensionaler String Array implementiert, auf dem die vertikalen Schiffspositionen entsprechend umgerechnet werden. Im Array wird der Status der Felder gespeichert, damit man weiss, ob es frei, besetzt, beschossen oder gar getroffen ist.

- Socket.IO zur Kommunikation mit dem WebClient

- Mehr dazu bei den Sockets

Frontend

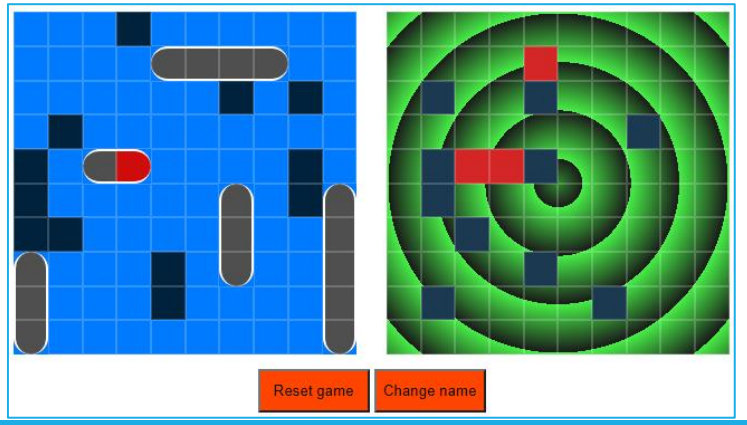
- HTML/CSS/JavaScript WebClient, Kommunikation über Socket.IO zum API-Service

- NGINX Container liefert Dateien

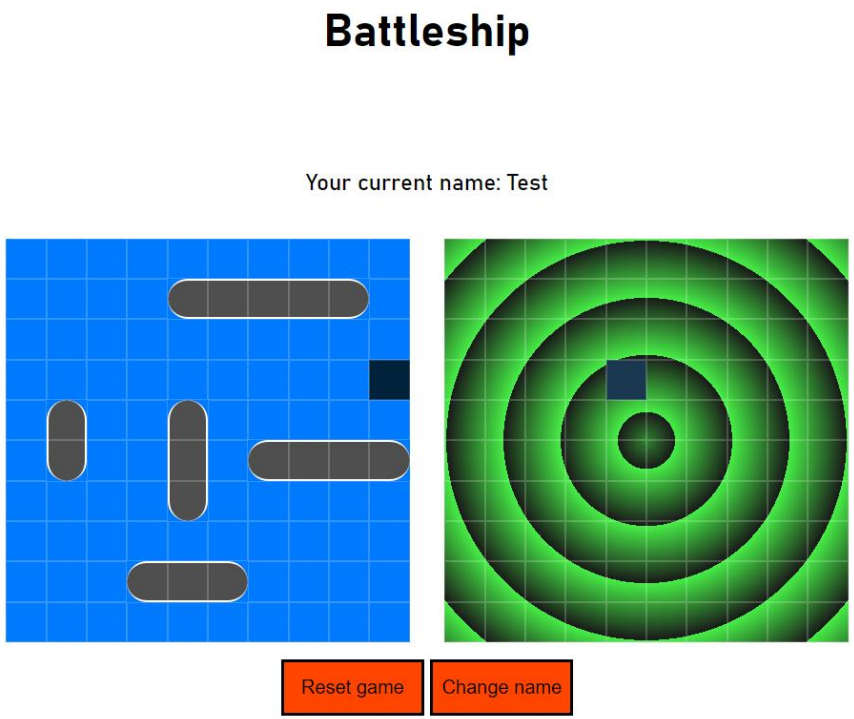
- Ein Benutzer kommt dabei als erstes ein Eingabefeld für seinen Spielernamen zu sehen. Unter diesem Namen wird sein Spiel anschliessend in der Datenbank gespeichert.

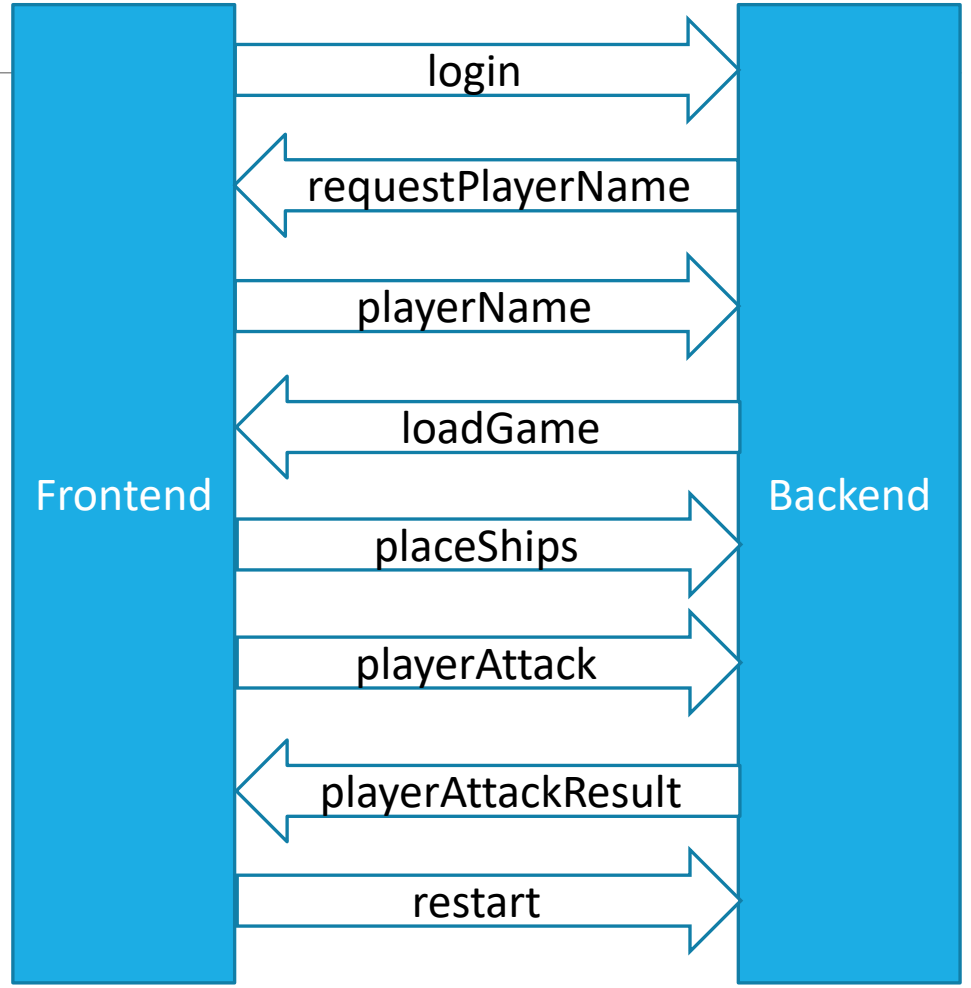
- Nach der Namenseingabe kann der Spieler seine Schiffchen platzieren und das Spiel beginnen.

- Während dem Spiel kann der Spieler im rechten Spielfeld seine Angriffe ausführen, während ihm im linken Feld die Treffer des Gegners angezeigt werden.



Sockets


Battleship
Your current name: Test
Reset game
Change name


Frontend
Backend
login
requestPlayerName
playerName
loadGame
placeShips
playerAttack
playerAttackResult
restart

Load Balancer

- Caddy v2 Reverse Proxy

- Load Balancing zwischen mehreren Services

- Failover, wenn aktive Healthchecks erkennen, dass ein Host nicht vefügbar ist

- Es gab zuerst Probleme mit den Sockets → Interaktionen und Failover dauerten lange und waren spontan

- Grund: Socket.io führt aus Performance-Gründen standardmässig HTTP long-polling aus, diese landeten aber bei verschiedenen Service-Instanzen

- Lösung: Cookies setzen, um Sticky Sessions zu gewährleisten

- Befehl zum Starten von mehreren Service-Container ohne Angabe von Replicas: *docker compose up –scale api=2*

```
#Caddyfile
:7070
rewrite /api /api/
handle /api/* {
  uri strip_prefix /api
  #rewrite * /socket.io{path}
  reverse_proxy * {
    header_up Host {host}
    header_up X-Real-IP {remote}

  to http://battleship_api_1:8000
  to http://battleship_api_2:8000

  lb_policy cookie mycookie
  health_uri /healthcheck
    health_interval 2s
    health_timeout 5s
    health_status 200
  }
}

reverse_proxy * {
  to http://battleship_web_1
}

```

Demo