Bored GAMES

Distributed systems,

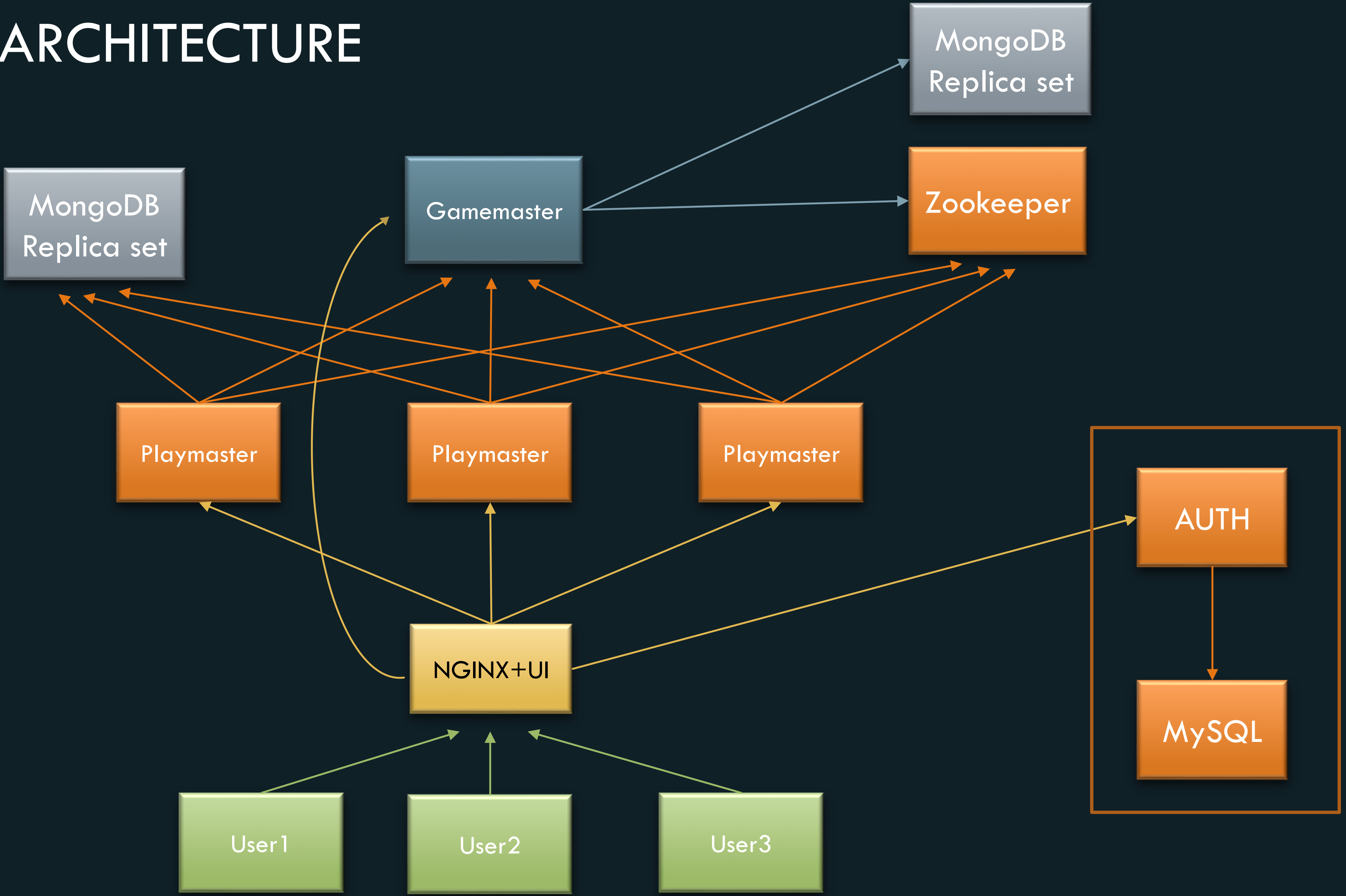
COMP414

KAPENEKAKIS ANTHEAS,

MARGARITIS GEORGIOS

INSTR. V. SAMOLADAS

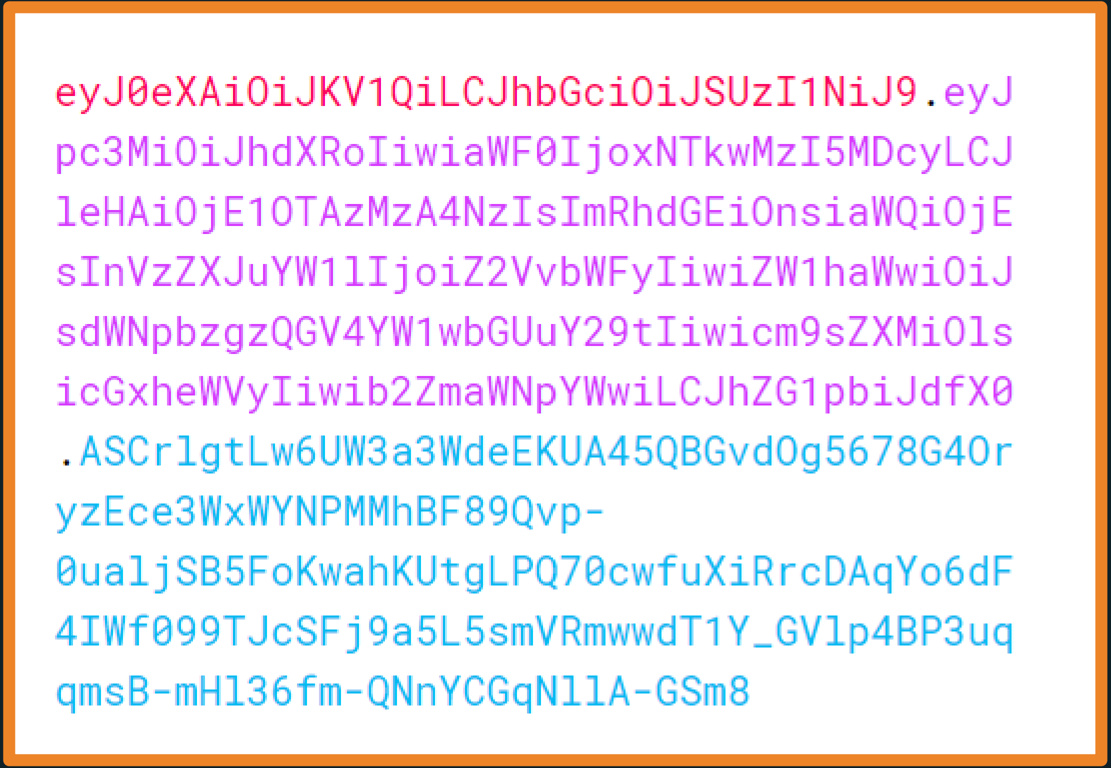
ARCHITECTURE



AUTHENTICATION

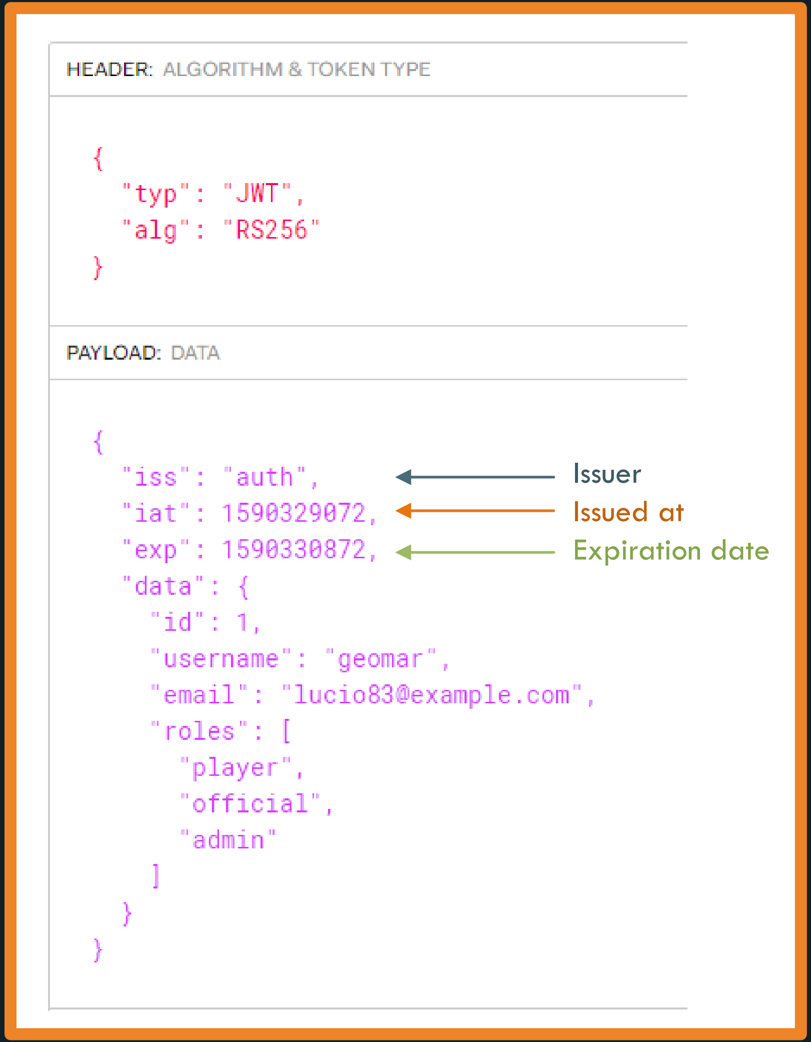
- Written in PHP with **external** MySQL DB
- Served by NGINX+php-fpm
- Uses JWT tokens with RS256 asymmetric encryption as AUTH bearer:
    1. UI submits user credentials (through REST)
    2. Auth validates user credentials
    3. Auth issues JWT signed token (with expiration) and refresh token
    4. Protected service validates JWT using public key (no API call to AUTH needed)
    5. If JWT expires (once every 20 minutes) the refresh token is used to generate a new one

AUTHENTICATION: JWT EXAMPLE



```
eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiJ9.eyJ
pc3MiOiJhdXRoIiwiaWF0IjoxNTkwMzI5MDcyLCJ
leHAiOjE1OTAzMzA4NzIsImRhdGEiOnsiaWQiOjE
sInVzZXJuYW1lIjoiZ2VvbWFyIiwiZW1haWwiOiJ
sdWNpbzgzQGV4YW1wbGUuY29tIiwicm9sZXMiOls
icGxheWVyIiwib2ZmaWNpYWwiLCJhZG1pbiJdfX0
.ASCrlgtLw6UW3a3WdeEKUA45QBGvdOg5678G4Or
yzEce3WxWYNPMMhBF89Qvp-
0ualjSB5FoKwahKUtgLPQ70cwfuXiRrcDAqYo6dF
4IWf099TJcSFj9a5L5smVRmwwdT1Y_GVlp4BP3uq
qmsB-mHl36fm-QNnYCGqNl1A-GSm8
```

Original



HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "RS256"
}
```

PAYLOAD: DATA

```
{
  "iss": "auth",             ← Issuer
  "iat": 1590329072,         ← Issued at
  "exp": 1590330872,         ← Expiration date
  "data": {
    "id": 1,
    "username": "geomar",
    "email": "lucio83@example.com",
    "roles": [
      "player",
      "official",
      "admin"
    ]
  }
}
```

Base 64 decoded

AUTHENTICATION: API

/create?username=#&password=#&email=#&secret=#

- Creates a user with a username, password and a user secret

/login?username=#&password=#

- Logins the user and returns a signed JWT and a refresh token on success

/change_role?jwt=#&roles[]=admin&roles[]=official&username=#

- Changes the role of a specified user (can only be performed by an admin)

/forgot?username=#&password=#&secret=#

- Used to reset the user password given the secret answer

/refresh_token?refresh_token=#

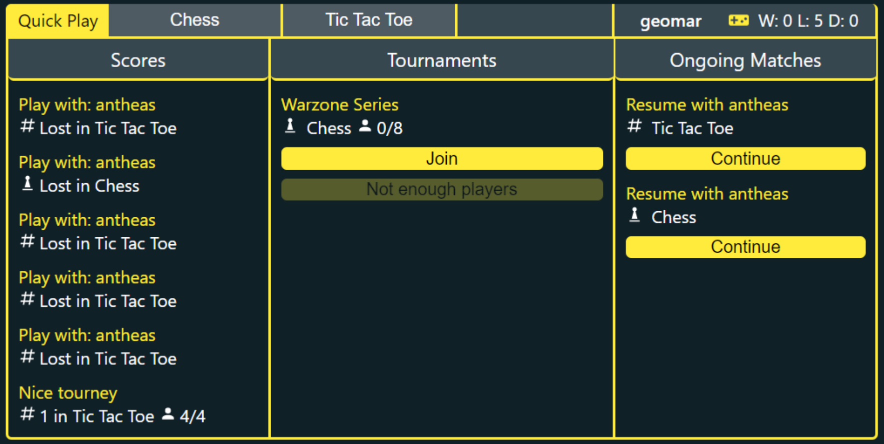
- Returns a new (valid) JWT token for the user with the specified refresh token

USER INTERFACE

- Simple Logo created in Illustrator
  - Comes in different styles
    - For Favicon, Header, and Logo
- App Written in React as a PWA
- Styled completely using SASS and CSS3 (flexbox)
  - without the use of a component library such as bootstrap
- Packaged using Webpack
  - Implements code bundling, chunking, minification and code splitting
  - Games are downloaded only when the user joins a play and cached
  - Can support hundreds of different games without slowdowns!
- Completely Static Resources with NO SSR!
  - Can be served by NGINX or CDN for pennies per month

Bored Games

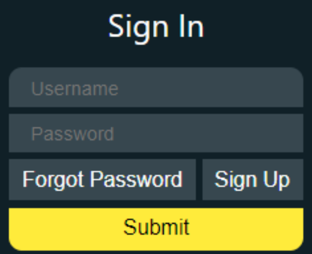
B G

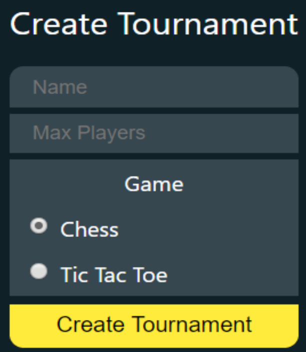


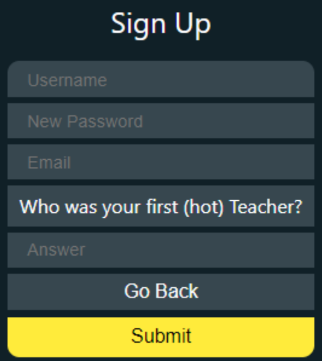
Scores	Tournaments	Ongoing Matches
Play with: antheas # Lost in Tic Tac Toe	Warzone Series Chess 0/8 Join Not enough players	Resume with antheas # Tic Tac Toe Continue
Play with: antheas Lost in Chess		Resume with antheas Chess Continue
Play with: antheas # Lost in Tic Tac Toe		
Play with: antheas # Lost in Tic Tac Toe		
Play with: antheas # Lost in Tic Tac Toe		
Nice tourney # 1 in Tic Tac Toe 4/4		

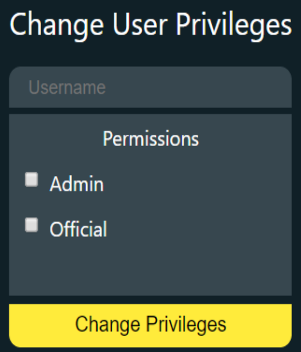
USER INTERFACE LOGIC

- Written in React as a PWA (Progressive Web App) with a Model-View-Controller Architecture
  - Uses redux to create a local Model with app data
  - Redux sagas implement a background thread (Controller) which mutates the Model
  - React binds the Model to the UI
- Architecture allows for complex logic such as refreshing the tokens, matchmaking and communicating with multiple services client-side
- Uses Websockets (socket.io) to communicate with Playmaster
  - Websockets provide instant response during the game, allowing fast moves, smooth gameplay, and access to Chat



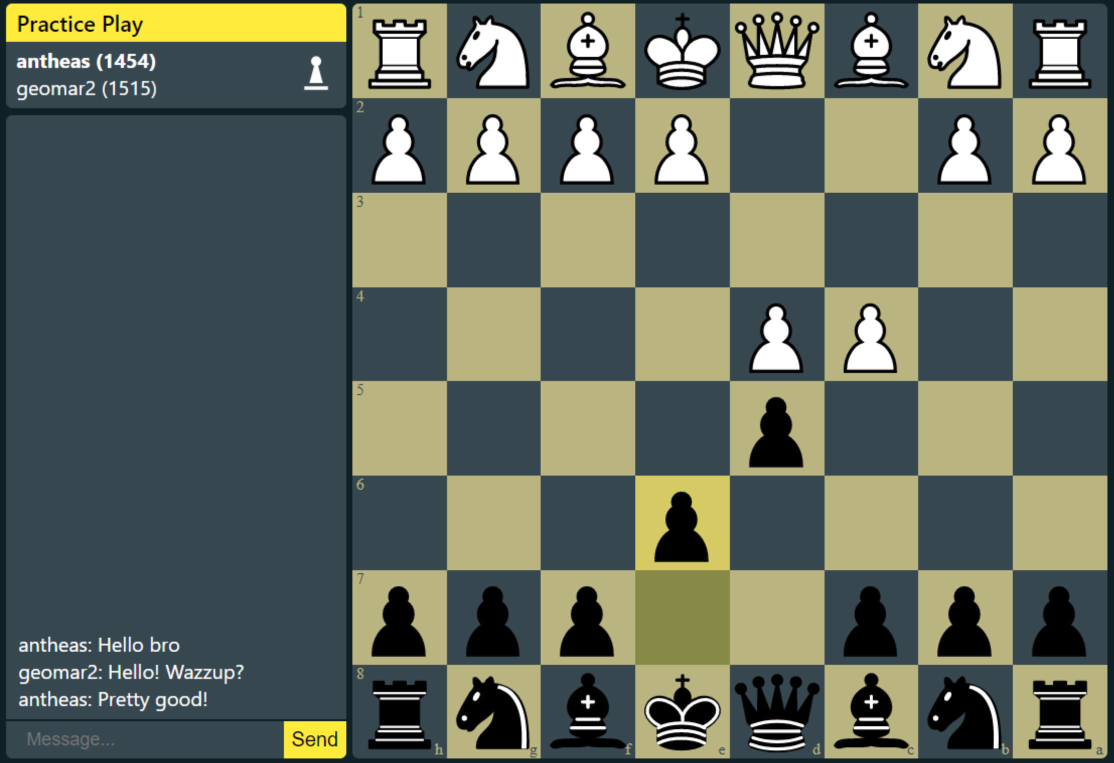




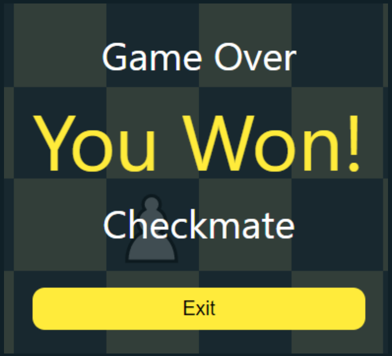


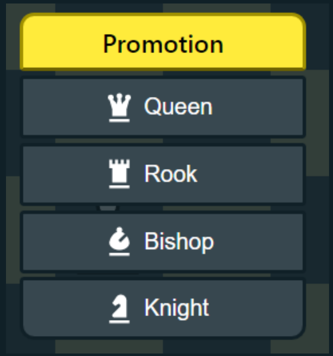
CHESS

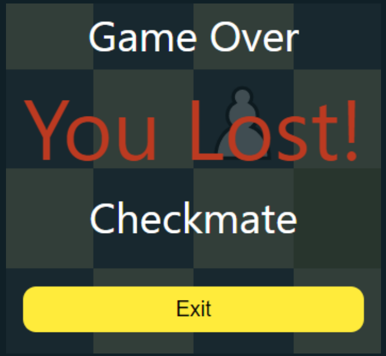
- Uses Chess.js library both on client and server for move validation
  - Possible moves are highlighted when hovering a piece
  - All Chess moves and game endings are supported, including promotion with any piece, en passant, and stalemate
  - Revalidation of moves on server prevents cheating
  - Source of truth of the play progress stays on the server and written to a replicated database every round
    - Even if a Playserver dies progress is online so no exploit is possible
- Backed by Chessboard.jsx, piece moves are animated



antheas: Hello bro
geomar2: Hello! Wazzup?
antheas: Pretty good!

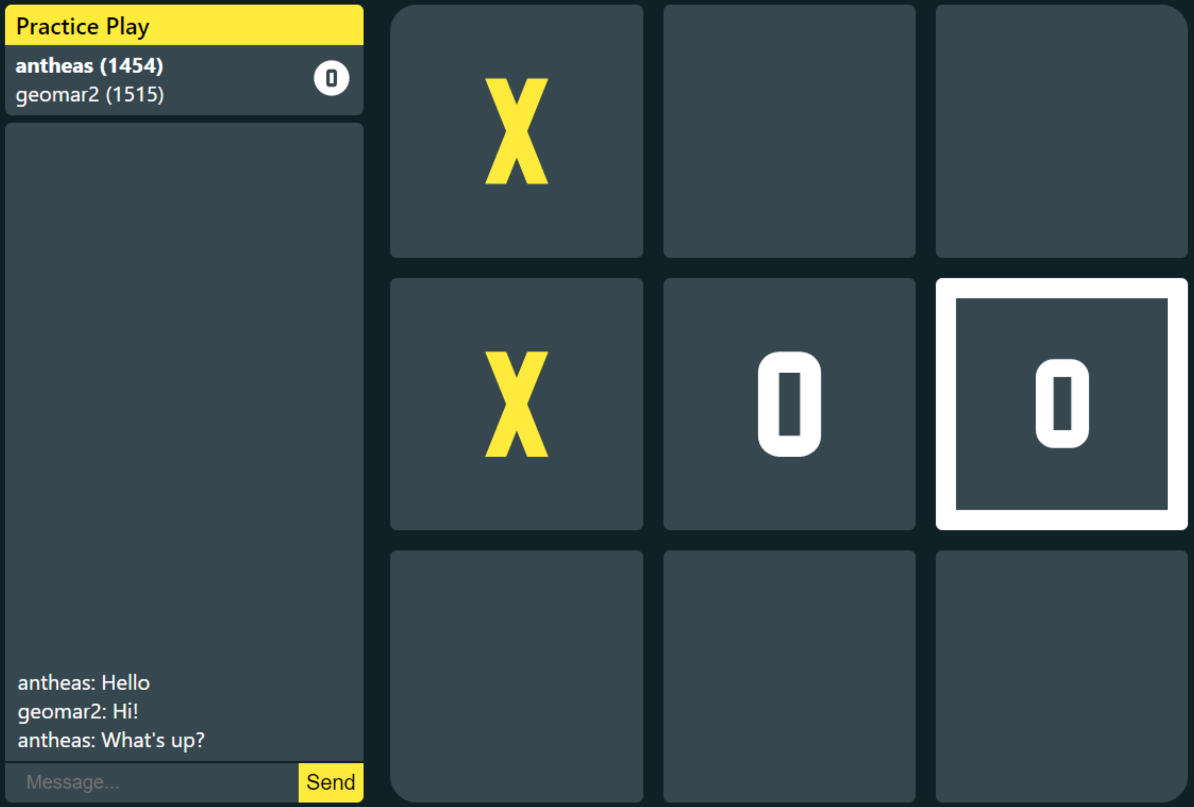






TIC-TAC-TOE

- Uses custom-made validator on both server and client
- Provides the same standard of security as Chess
- UI made from scratch

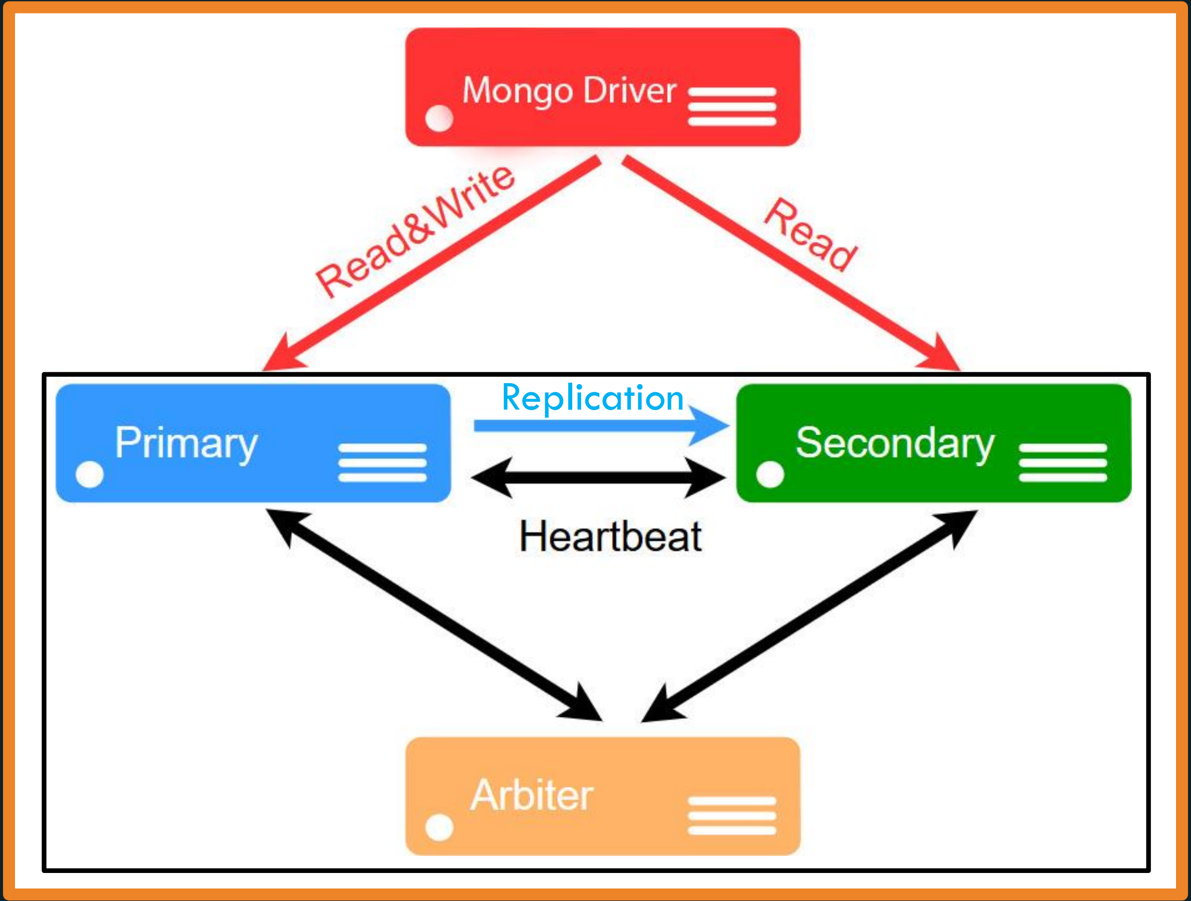


antheas (1454)
geomar2 (1515)

antheas: Hello
geomar2: Hi!
antheas: What's up?

MONGODB REPLICA SET

- Replica set consists of the following nodes:
  - Primary (Reads/Writes/Transactions)
  - Secondary (Reads)
  - Arbiter (participates in master election)
- More Secondaries can be added
- Can use **sharding** for horizontal scalability
  - Schema was designed with it in mind
- When Primary dies:
  - New Primary is elected
  - Driver redirects reads/writes to new Primary



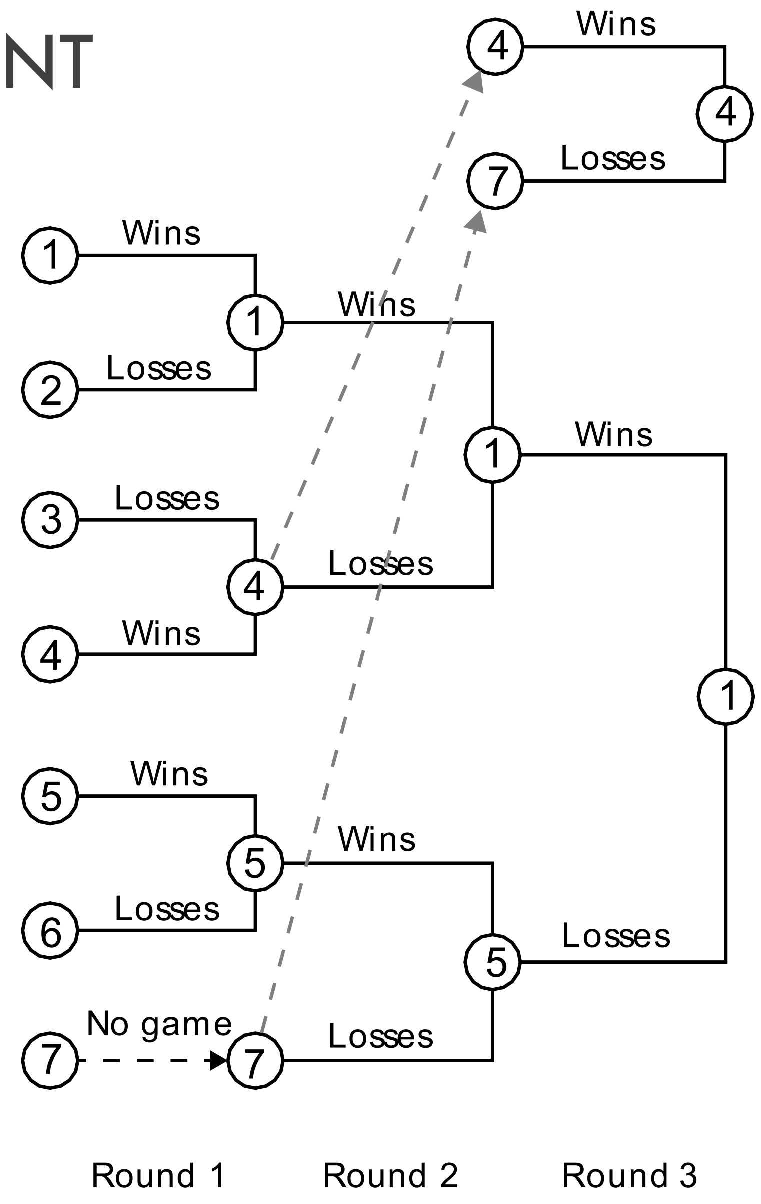
GAMEMASTER

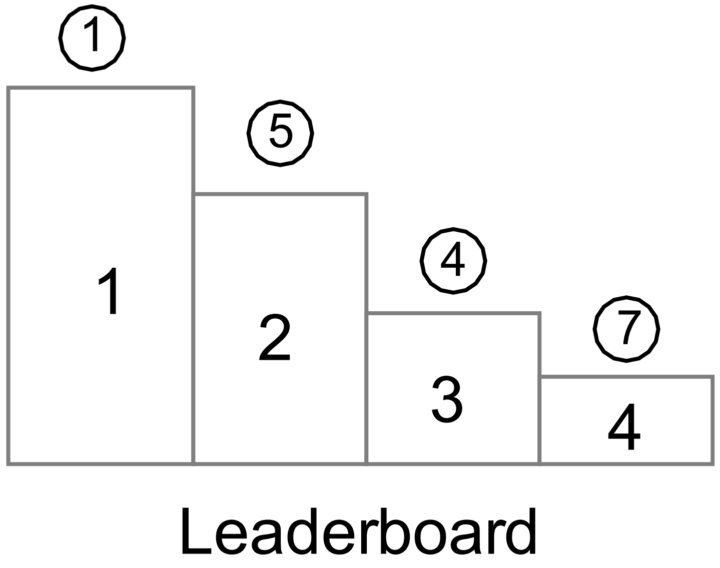
- Written in NodeJS
- Utilizes Express framework for REST API implementation
- Communicates with Zookeeper to assign plays to Playmasters
- Uses a MongoDB replicated cluster for persistent storage
- Responsibilities:
  - Tournament creation & Management
  - Tournament pairing & winner announcing
  - Matchmaking (practice plays, tournaments)
  - Player stats (game scores, tournament scores, number wins/losses/ties, ELO)
- Uses de-normalized schema for faster reads
- Writes use MongoDB Primary Node and frequent reads use MongoDB Secondary node
- Can be very **easily replicated** (stateless container)

GAMEMASTER: TOURNAMENTS

- An official creates a new tournament
- The tournament becomes visible to all connected users
- Users join the tournament
- When at least 4 users has joined, the official can **start** the tournament
- Let's say that there are n players. Then in the first round:
  - Only $\mathrm{A} = 2n - 2^{\lfloor \log_2 n \rfloor}$ players are paired.
    - Out of those, the $\frac{A}{2}$ winners pass to round 2 (draw leads to a rematch)
  - The rest $n - A$ players instantly pass to round 2
- This way next rounds will have power of 2 players $\longrightarrow$ Fairness
- Rounds continue until only 2 players are left
- In the last round:
  - The 2 remaining players play for 1st+2nd position
  - The last 2 eliminated players play for 3rd +4th position

TOURNAMENT EXAMPLE





GAMEMASTER: API (1)

/tournament/list?jwt=#&mode=active|future|past

- Lists the tournaments (active, future or past)

/tournament/info?jwt=#&id={tournament_id}

- Retrieves information about a specific tournament (rounds, games, leaderboard e.t.c)

/tournament/create?jwt=#&name=#&game_type=chess|tic-tac-toe

- Creates a new tournament (requires official role)

/tournament/register?jwt=#&tournament_id=#

- Joins an existing tournament

GAMEMASTER: API (2)

- /me/match_history?jwt=#
  - Returns the match history of a specific user (total wins-losses-ties-ELO)
- /me/active_games?jwt=#
  - Returns the active games of a user
- /user/stats?jwt=#&username=geomar
  - Retrieves the stats of a user
- /practice/join_queue?jwt=#&game_type=chess|tic-tac-toe
  - Join matchmaking Queue
- /join_game?jwt=#&game_id=#
  - Joins an existing active game by id

PLAYMASTER

- Written in NodeJS
- Uses replicated mongoDB cluster to store last game positions
- Registers itself to Zookeeper during initialization
- Uses socket.io for communication with clients (web socket based) and includes chat
- Designed with **load-balancing** in mind:
  - Games are assigned to Playmasters with less games
- **Fault tolerant** (will be explained later)
  - If a Playmaster dies, the game is assigned to a new one

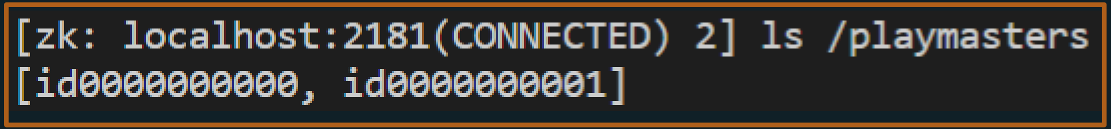
PLAYMASTER

- Playmaster creates a **sequential emphemeral** node in zookeeper containing:
  - Playmaster ID
  - Playmaster IP suffix -> 10.0.7.XX
  - Number of active games

- When the user wants to join a game the Gamemaster:
  - Checks if the game is assigned to a Playmaster
  - If it is assigned and the Playmaster is alive ⟶ the Playmaster is returned
  - If it is not assigned or the Playmaster is dead:
    - Gamemaster queries zookeeper
    - Gamemaster finds Playmaster with least plays and assigns the play to it

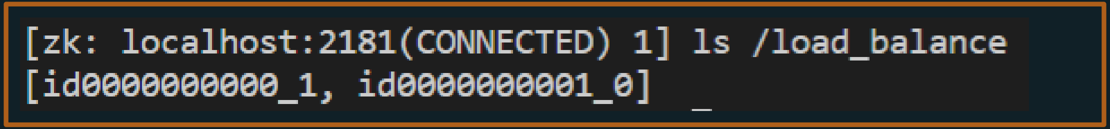
ZOOKEEPER

- Used for Playmaster service discovery & load balancing
- Contains 2 root Znodes:
  - /playmasters ⟶ Each playmaster creates a **sequential ephemeral** node:
    - Node **name:** Playmaster IDs (e.g. id0000002)
    - Node **value:** Last segment of their IPs



```
[zk: localhost:2181(CONNECTED) 2] ls /playmasters
[id0000000000, id0000000001]
```

  - /load_balancing ⟶ Each playmaster creates an **ephemeral** node:
    - Node **name:** playmaster ID + Playmaster number of games
    - Find playmaster with least games **in a single zookeeper call**



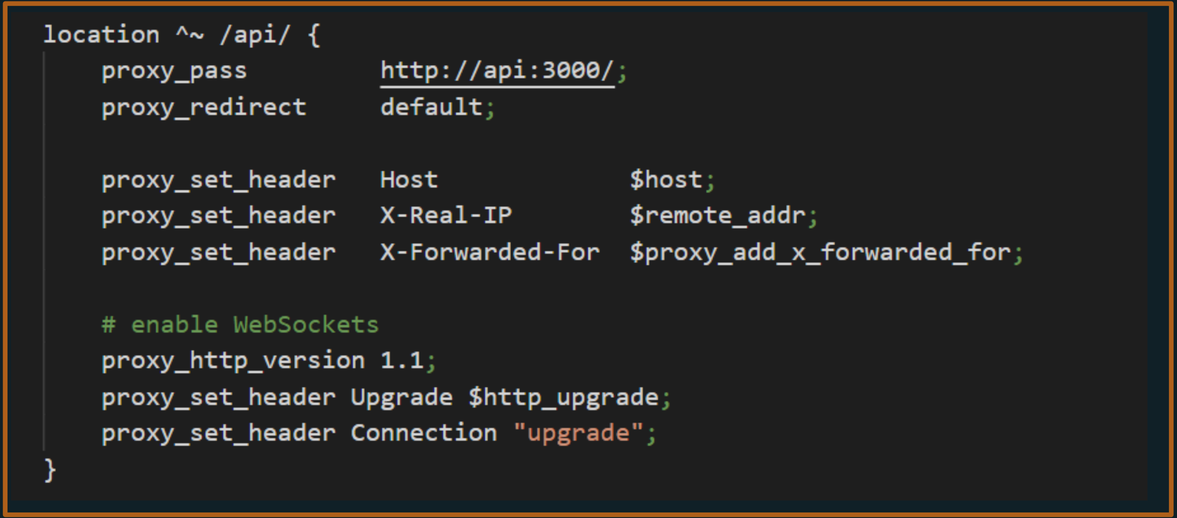
```
[zk: localhost:2181(CONNECTED) 1] ls /load_balance
[id0000000000_1, id0000000001_0] _
```

FAULT TOLERANCE: PLAYMASTER IS DOWN

- Consider the scenario: "Playmaster of an active game dies"
- The followings steps are followed:
  - UI socket connection times out ⟶ UI detects Playmaster is down
  - UI waits for Zookeeper session timeout (3 sec)
  - UI requests from Gamemaster to re-join the game
  - Gamemaster checks zookeeper to see if Playmaster is dead
  - Gamemaster queries zookeeper to get a new Playmaster (with the least active games)
  - Gamemaster returns the new Playmaster
  - Users join the new Playmaster
  - The new Playmaster loads the game progress from the shared MongoDB cluster
- If the Playmaster loses connection with Zookeeper it severs all of its connections and restarts with a new ID, acting like it died. That way, consistency is maintained.

NGINX

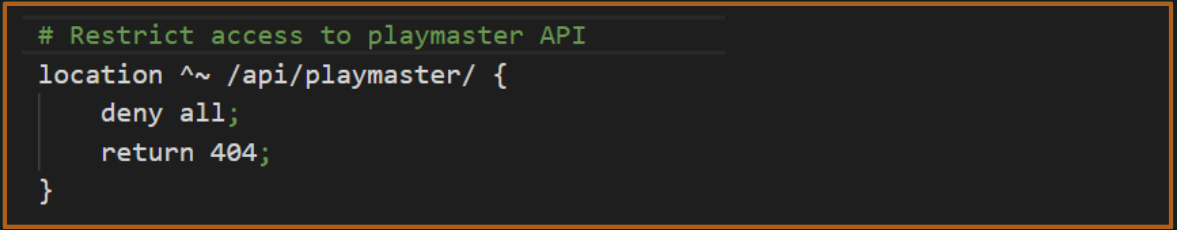
- Serves the complete Frontend UI
- All traffic from the outside world passes through NGINX
- Solves CORS issues
- Limits access to internal services to a specified port and protocol
- Used as reverse proxy for:
  - Connection with Playmaster (sockets + REST)
  - Connection with Gamemaster
  - Authentication service
- Proxy-redirect rules:
  - /games/g# ⟶ 192.170.0.#:3000
  - /auth ⟶ http://auth
  - /api ⟶ http://api:3000
- One is enough per Datacenter (can saturate uplink connection)
  - If the Node containing it fails, it can be moved to another one



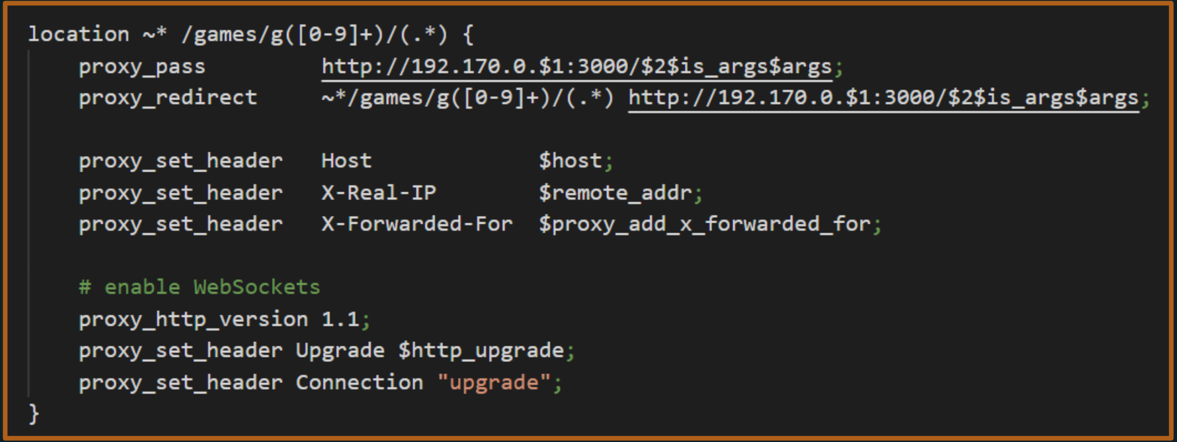
```
location ^~ /api/ {
    proxy_pass          http://api:3000/;
    proxy_redirect      default;

    proxy_set_header    Host                $host;
    proxy_set_header    X-Real-IP           $remote_addr;
    proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;

    # enable WebSockets
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```



```
# Restrict access to playmaster API
location ^~ /api/playmaster/ {
    deny all;
    return 404;
}
```



```
location ~* /games/g([0-9]+)/(.*) {
    proxy_pass          http://192.170.0.$1:3000/$2$is_args$args;
    proxy_redirect      ~*/games/g([0-9]+)/(.*) http://192.170.0.$1:3000/$2$is_args$args;

    proxy_set_header    Host                $host;
    proxy_set_header    X-Real-IP           $remote_addr;
    proxy_set_header    X-Forwarded-For     $proxy_add_x_forwarded_for;

    # enable WebSockets
    proxy_http_version 1.1;
    proxy_set_header Upgrade $http_upgrade;
    proxy_set_header Connection "upgrade";
}
```

BENCHMARKS, UI

- NGINX Capacity: 2GB/s
  - Can fully saturate a 1gbps uplink 16 times
- Asset Size: 400kB + 200kB Chess + 20kB Tic Tac Toe
  - Asset name contains hash -> can cache until asset update or 30 days
  - At 1gbps we can serve assets to 700k unique users per hour
  - A CDN plan of 400gB/month for 5$ can serve 400k users
  - Estimated Cost: 0.00002$ per user per month for bandwidth

BENCHMARKS, AUTHENTICATION

- Authentication server can handle 1800 login/refresh requests per second
- Each user makes a request only during initial load and every 20 minutes afterwards
  - Worst case: 10 Requests per hour (User reloads website often)
  - Can be limited to 4 by storing access token as a cookie as well
- Performance: 650k users per hour
- PHP container is stateless, can be replicated for horizontal scaling
- MYSQL Database cannot be replicated for scalability
  - All requests write to it
  - Refresh token changes and is written to database on every access token request

BENCHMARKS, GAMEMASTER

- Gamemaster runs on Node.JS -> Single Threaded
  - Each instance uses one vCPU
- Performance: 500-1500 requests per second (depending on request) for one instance
- By caching lobby polling requests we can reduce requests to worst case 100 per user per hour
  - User only makes REST requests on lobby load, joining games, and using the admin form
- Total Performance: 18k Active Users per vCPU/instance
  - Horizontal Scaling is an option since Gamemaster is stateless
  - Database can be horizontally scaled using Sharding

BENCHMARKS, PLAYMASTER

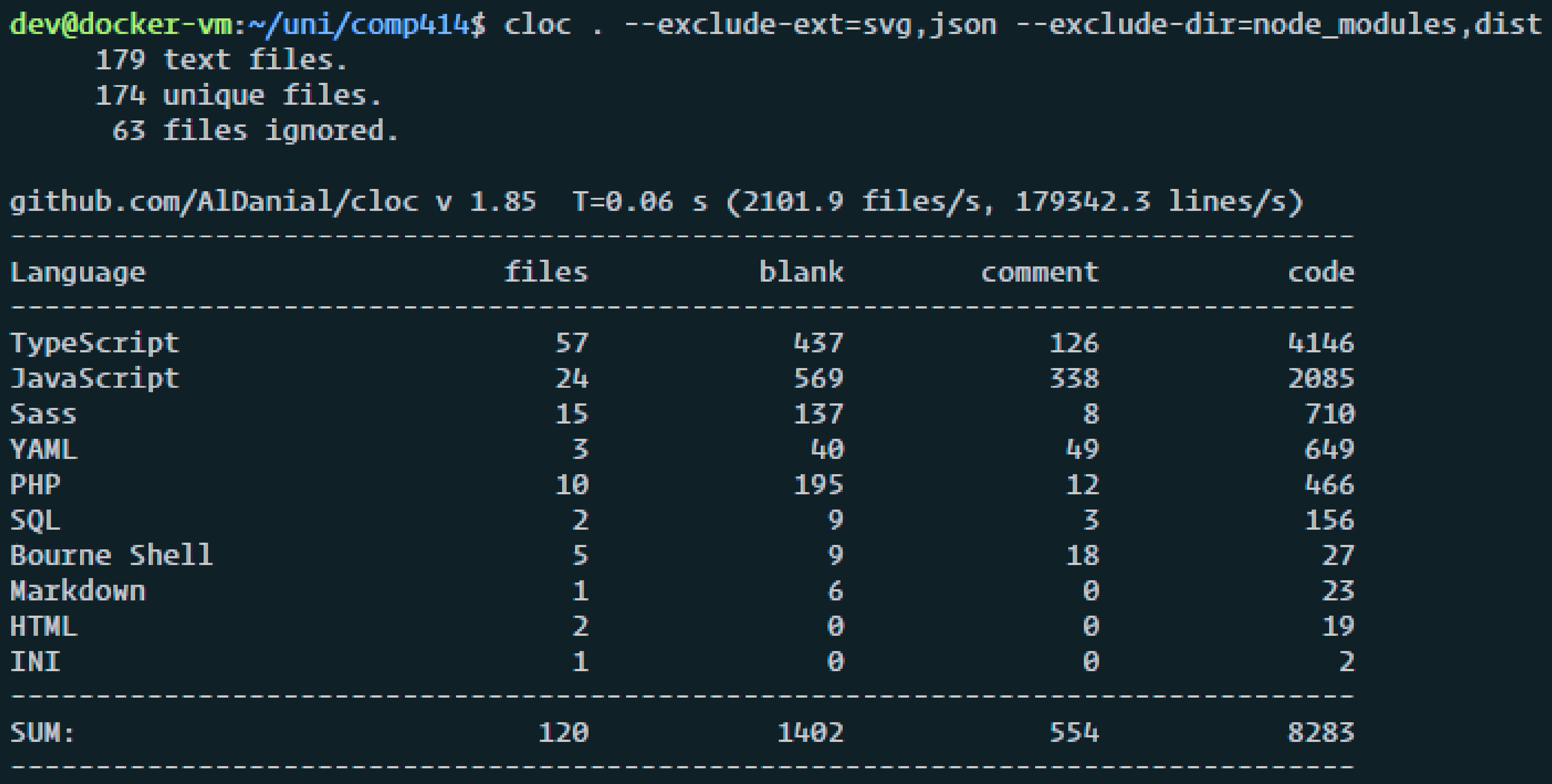
- Playmaster uses sockets and cannot be benchmarked using Apache Bench
- Hard Limit: 65k sockets over all Playmasters due to sharing a single NGINX interface
- Ongoing plays are limited to 20k ongoing simultaneously
  - TCP Requests use sockets as well
- Solution: a domain name, HTTPS, proper CORS headers, and reserving multiple IPv4 IPs
  - reverse proxies can then be set up for multiple banks of Playmasters in subdomains

PROJECT DIRECTORY STRUCTURE

- Authentication: ./auth
- Gamemaster: ./api
- Playmaster: ./game
- Web Interface: ./web

To run consult README.md for instructions

CODE LINES OVERVIEW



```
dev@docker-vm:~/uni/comp414$ cloc . --exclude-ext=svg,json --exclude-dir=node_modules,dist
     179 text files.
     174 unique files.
      63 files ignored.

github.com/AlDanial/cloc v 1.85  T=0.06 s (2101.9 files/s, 179342.3 lines/s)
-------------------------------------------------------------------------------
Language                     files          blank        comment           code
-------------------------------------------------------------------------------
TypeScript                      57            437            126           4146
JavaScript                      24            569            338           2085
Sass                            15            137              8            710
YAML                             3             40             49            649
PHP                             10            195             12            466
SQL                              2              9              3            156
Bourne Shell                     5              9             18             27
Markdown                         1              6              0             23
HTML                             2              0              0             19
INI                              1              0              0              2
-------------------------------------------------------------------------------
SUM:                           120           1402            554           8283
-------------------------------------------------------------------------------
```

THANKS FOR YOUR ATTENTION

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable” — **Leslie Lamport**