

D A R K E N I N G  
A G E

MILESTONE FINAL



# ● Conteúdos



Equipa e Desenvolvimento



Arquitetura e Componentes



Serviços Web e Comunicação



Autenticação e Segurança



Detalhes de Implementação das Aplicações

*Game Server | Game Client | Jogo Unity*



# A Equipe e Responsabilidades (três entregas)

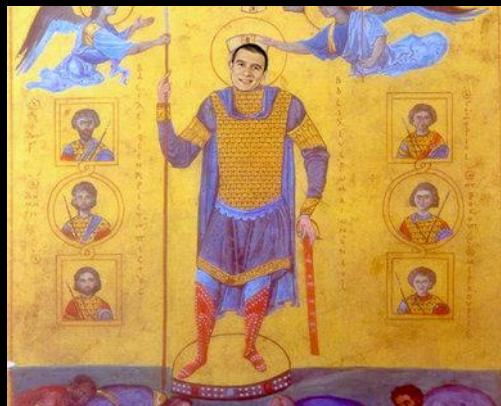
SCRUM MASTER / DEVELOPER



José Baltar

- Arquitetura e Configuração do Projeto
- Lógica de Comunicação e Salas
- Frontend Browser do Cliente
- Autenticação na API e Servidor Jogo

PRODUCT OWNER / DEVELOPER



Rodrigo Coelho

- Protocolo de Comunicação
- Lógica do Jogo no Servidor
- Sistema de Turnos
- Histórico Partidas na API

DEVELOPER



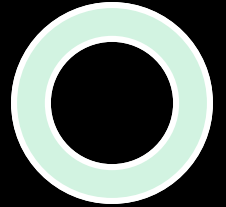
Fábio Mendes

- Frontend em Unity
- Lógica de Jogo no Cliente
- Sistema de Matchmaking
- Notificações, Amigos e Perfis na API



# Sprints do Projeto

- Sprint 3: Comunicação e API
  - Sistema de matchmaking (final)
  - Configuração REST API + Sistema de Autenticação
  - Processamento de turnos automáticos
- Sprint 4: Melhorias e *GameClient*
  - + Funcionalidades no Jogo
  - Sincronização Início do Jogo
  - Homepage + Auth pages
- Sprint 5: Implementações Finais
  - Atualizações no Sistema de Salas
  - Sistema de Notificações, Histórico de Partidas e Lista Amigos;
  - Testing Geral do Jogo



## Sprint-5

Dec 26, 2020–Jan 10, 2021

Closed

LDS Group 15 / Darkening Age

## Sprint-4

Dec 18, 2020–Dec 23, 2020

Closed

LDS Group 15 / Darkening Age

## Sprint-3

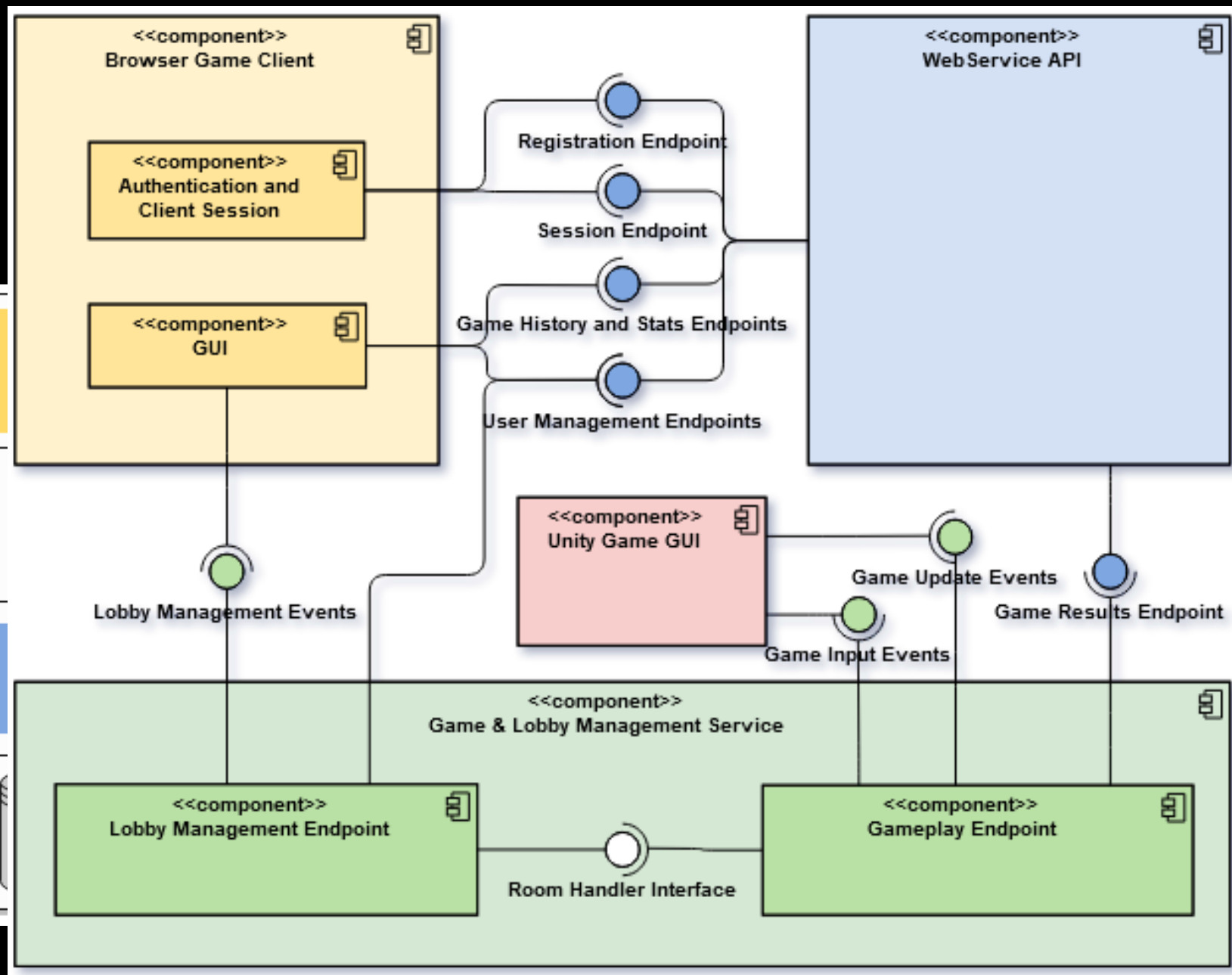
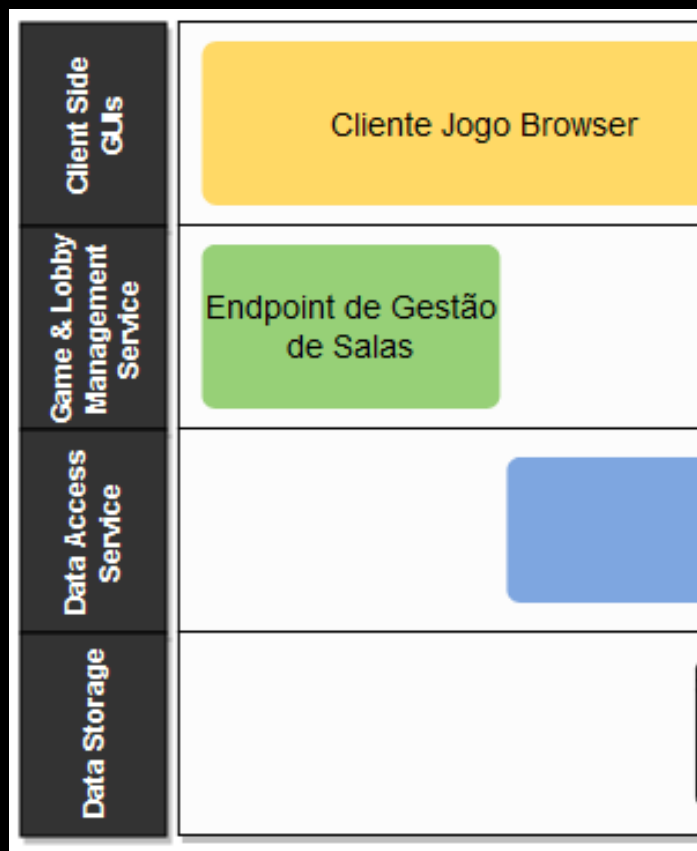
Dec 9, 2020–Dec 18, 2020

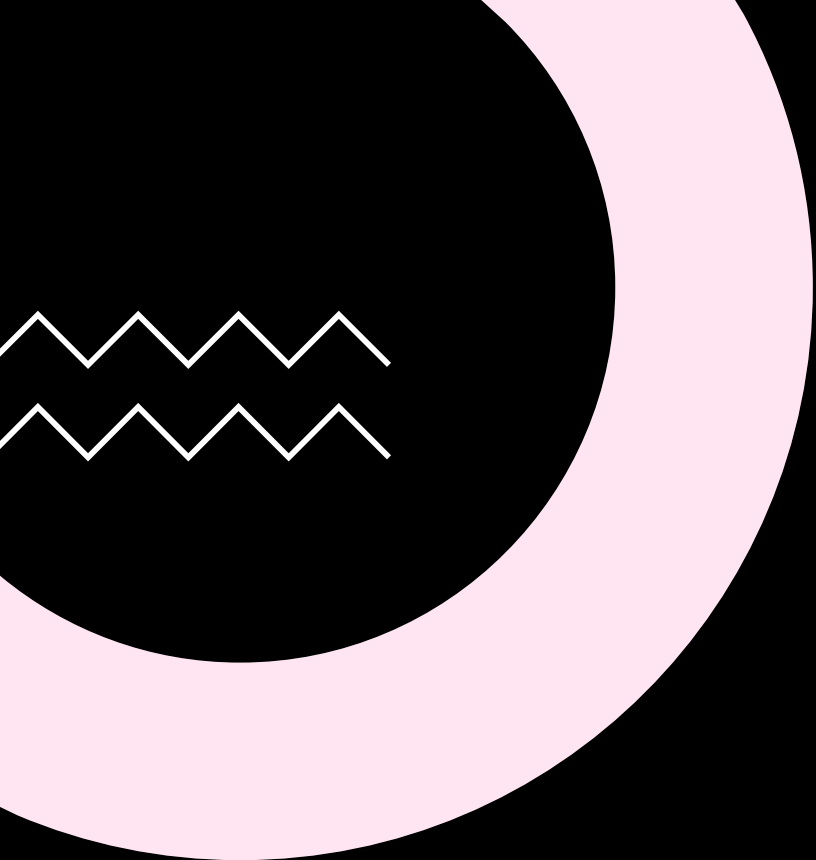
Closed

LDS Group 15 / Darkening Age



# Arquitetura





RESTful API |  
Configuração

MVC + .Net Design Patterns

HTTP + Json

Entity Framework

SQLite



# Game Server | Configuração Serviço Web

## Armazenamento e Processamento de Dados

- Configurações do jogo em ficheiros JSON
- Dados armazenados e processados em Memória

## OOP para organização das Entidades

- Jogadores
- Salas
- Elementos do Jogo

## Carregamento para Estruturas de Dados

- Grafo do mapa
  - carregado através dos Json de configuração
- Queue de ações
  - vindas dos cliente-side

## Dados Concorrentes


- Vários jogadores enviam pedidos simultâneos ao mesmo conjunto de dados

## Websockets – TCP

- Comunicação real-time
- Troca de mensagens através de um Protocolo definido – *event based*



# Autenticação e Segurança

- Autenticação e Sessão de Jogadores
    - JWT signed tokens
  - Dados sensíveis
    - Hash de password - HMACSHA512
  - Autenticação entre componentes
    - Endpoints “especiais” de comunicação entre o *Game Server* e a API
- 



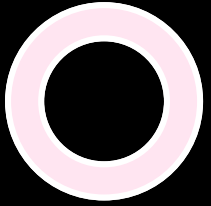
# Autenticação de Jogadores com o Servidor de Jogo

Autenticação Jogador com a API

Envio credenciais para o *Game Server*

*Game Server* verifica JWT com a API

WebSocket Aceite ou Recusado com base no anterior



# ● Conteúdos



Equipa e Desenvolvimento



Arquitetura e Componentes



Serviços Web e Comunicação



Autenticação e Segurança



Detalhes de Implementação das Aplicações

*Game Server | Game Client | Jogo Unity*





## Dois Endpoints

*ws-lobby*

Interação salas *lobby*  
normalmente acedido pelo  
*Game Client*

*ws-game*

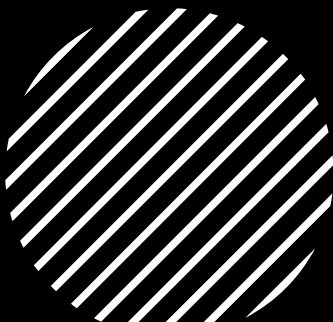
Interação salas *inGame*  
normalmente acedido pelo  
Unity



**Cada um dos endpoints  
abre uma conexão  
WebSocket gerida por um  
protocolo próprio.**

- O primeiro endpoint cria e gere as salas
- A sala pode encher e passar para o jogo ou entrar em *matchmaking*
- Após isto, a sala *transita* para um estado *in-game*
- O acesso a salas *in-game* é possível através do segundo endpoint

# Implementação *Game Server*





# Implementação *Game Server*



**Ou seja, cada um dos protocolos tem o seu próprio conjunto de mensagens ou eventos**

- O primeiro contém eventos para criação e gestão de salas
- O segundo contém eventos de processamento do jogo, relativamente à sala criada pelo endpoint *ws-lobby*

# ● Sistema de Salas e Ações de Jogo



## Dois tipos de salas

Sala *Lobby*

- Convites ou *Matchmaking*

Sala *InGame*

- Contém os jogadores da sala anterior



***Lobby* transita eventualmente para *InGame***

Sala passa a ser acedida por um endpoint diferente



**Cada sala *InGame* contém uma *GameInstance* própria**

Cada um dos jogadores executa ações concorrentes, através de uma *Queue* de ações


- First Come, First Served





# Sala *In Game*

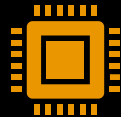
## Processamento dos Turnos em Jogo



Inicialmente, todos os Clientes recebem o estado do Jogo à data – no turno 0 equivale ao estado inicial



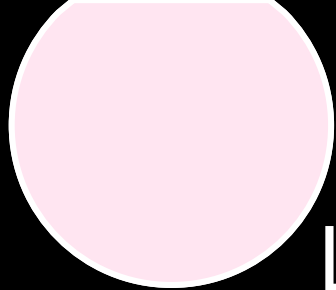
Os Clientes enviam ações para serem processadas no Servidor, durante um período de 45 segundos – organizadas na Queue



No final de cada turno, o Servidor envia o novo estado do Jogo – havendo já processado todas as ações recebidas



Os Clientes recebem as atualizações de tudo o que aconteceu no turno e têm 15 segundos até ao início do próximo turno



# Lógica e Eventos do Jogo

## Instância de jogo

- Mantém e atualiza o estado de jogo com base nos eventos, um em cada sala *in game*

## Gestão de Mapa

- Listagens de regiões
- Verificação de fronteiras
- Mudanças de Dono sobre regiões

## Gestão de Exército

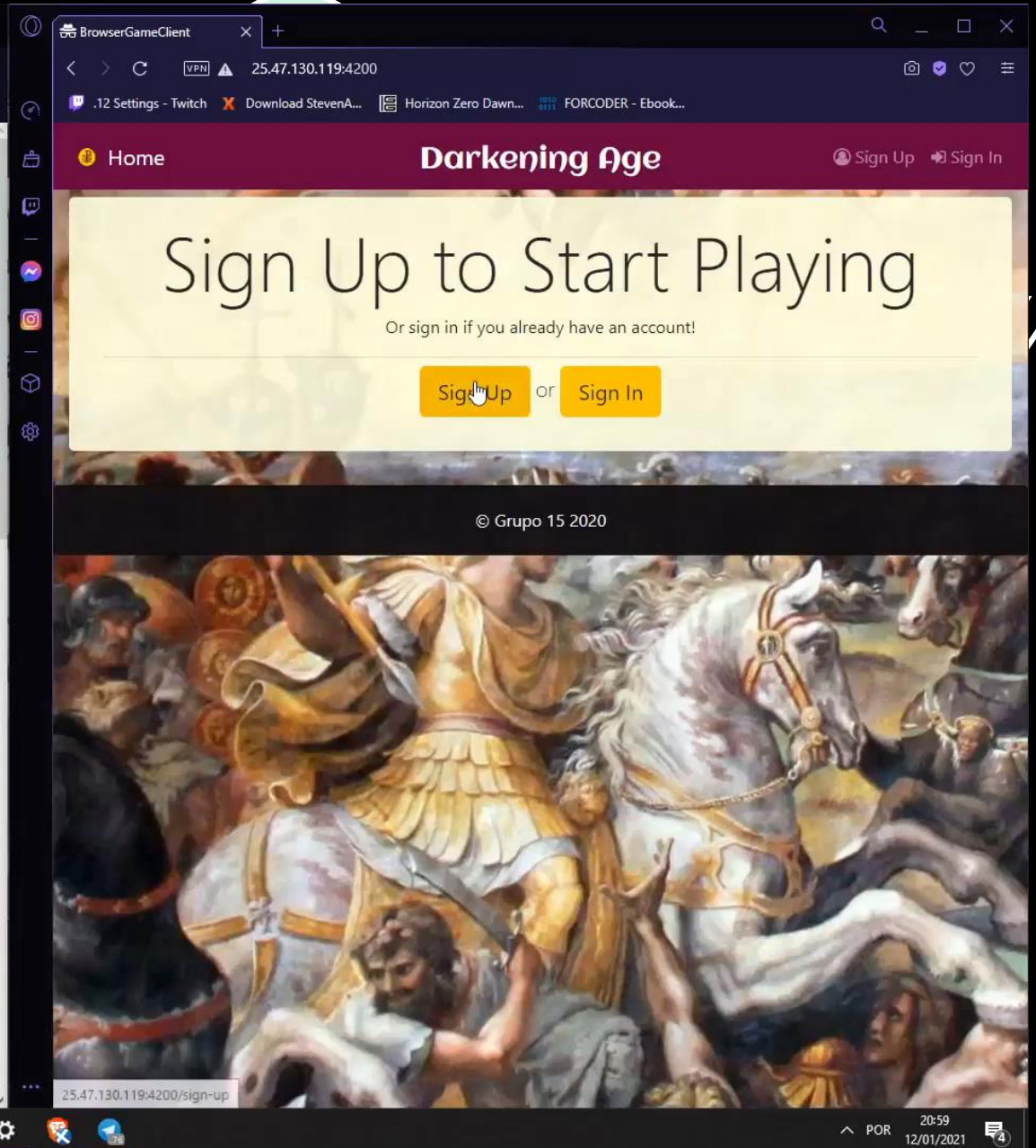
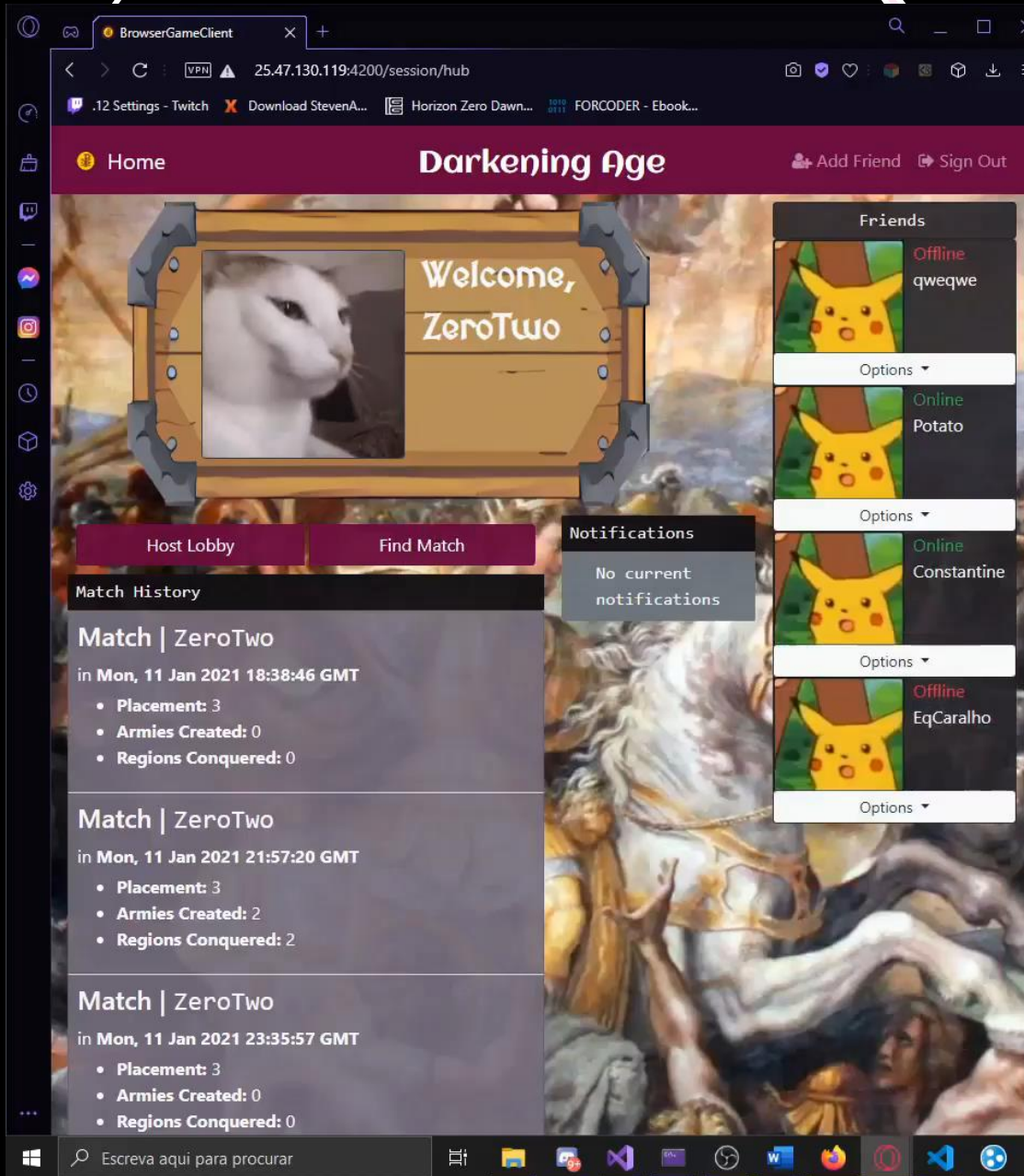
- Criação de exércitos e unidades
- Reforço de unidades e manutenção
- Combate e Destruição

## Gestão de Fação

- Atualização de Recursos
- Histórico de Eventos

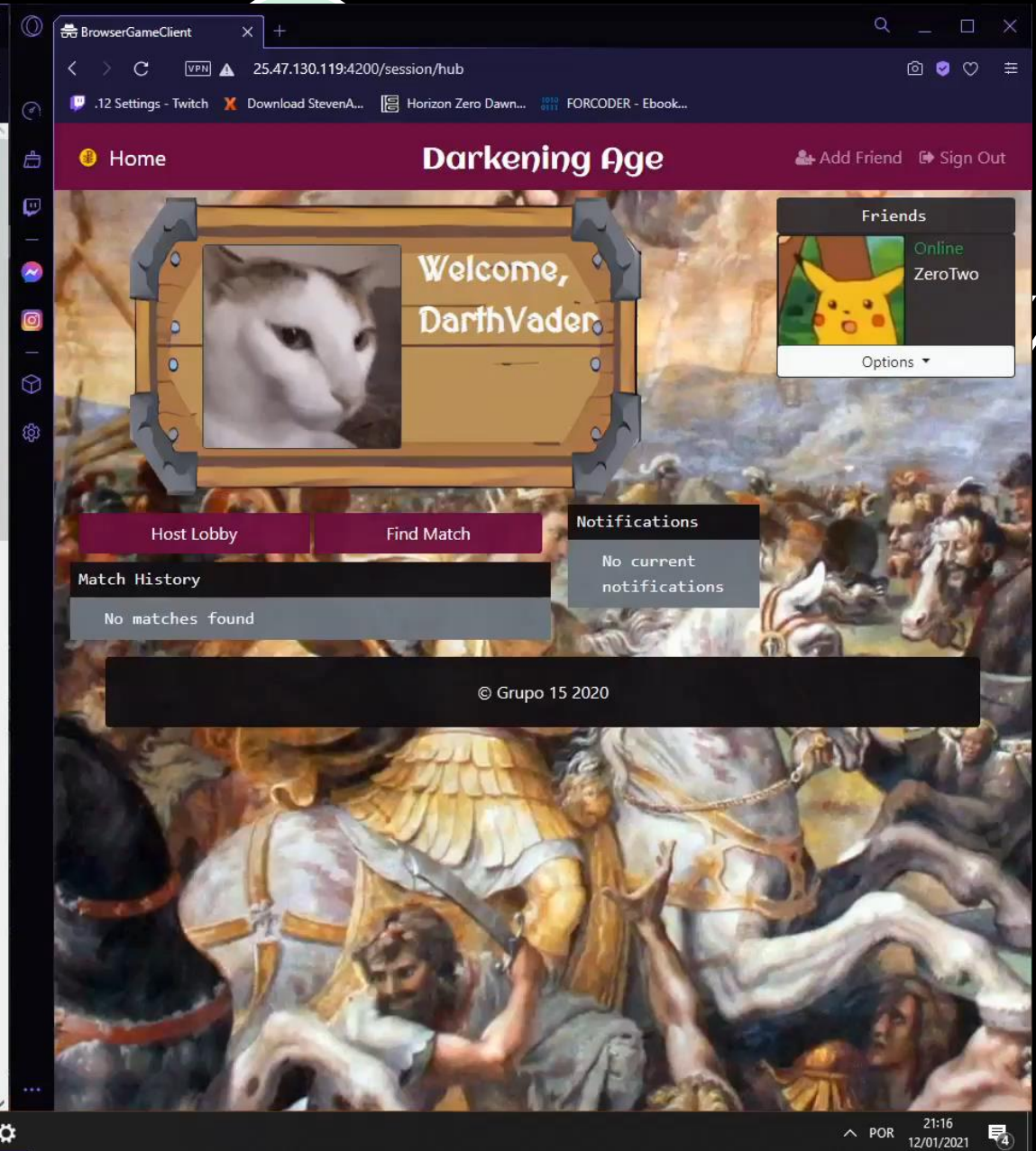
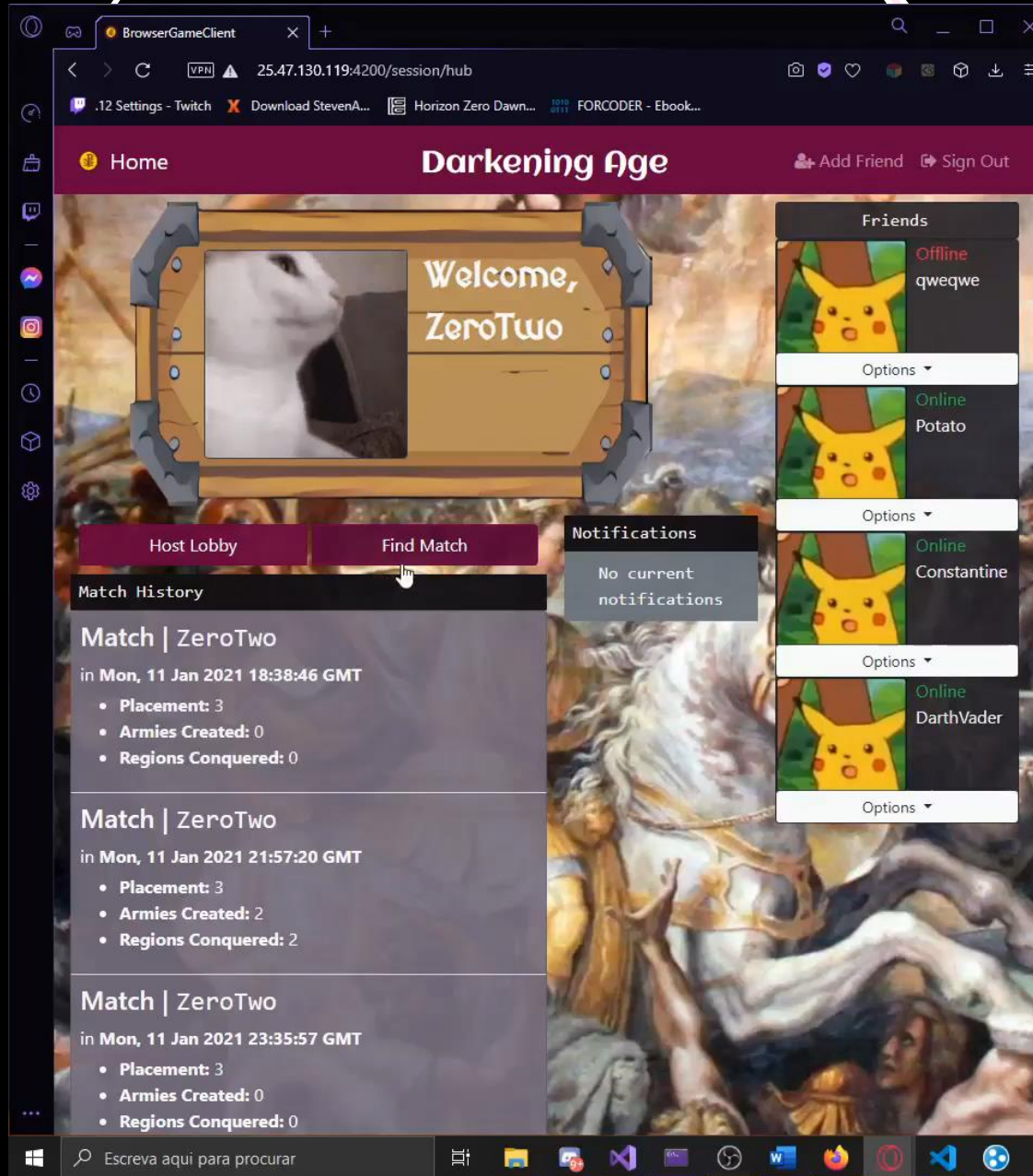


# Implementação *Game Client*





# Implementação Game Client





# Implementação Jogo *Unity*

Integração dos Eventos do Servidor

Comunicação com o servidor

- Particular atenção ao uso de Programação Concorrente

Verificações lógicas com vista à otimização de eventos e envios de mensagem

- Bloqueio de certas ações tanto no servidor como no cliente

Utilização de “Dummies” para representar informação futura



- Implementação Jogo *Unity*

**LOADING**

PLAYING AS REMNANT