ARQUITETURA DE SOFTWARE

NÁDIO DIB – ENG. SOFTWARE

UNIPROJEÇÃO

SUMÁRIO

- Projeto 1: “LoESoft Games’ Official DevBlog”
  - Sobre
  - Padrões utilizados
- Projeto 2: “TK Games – Services Architecture”
  - Sobre
  - Aplicações de conceitos teóricos

PROJETO 1

LOESOFT GAMES' OFFICIAL DEVBLOG

SOBRE

- Este projeto é uma aplicação web que utiliza linguagens básicas como Php, JavaScript (junto com JQuery), CSS3, HTML5 e SQL;
- Foi meu projeto de apresentação na semana acadêmica em 2019/2 na disciplina de Aplicações Web;
- Duração de desenvolvimento: 1 mês (aprox.); e
- Contribuidores: Nádio “Devwarlt” (BRA) e Andrew “Shot” (USA)

Código-fonte disponível na GitHub

PADRÕES UTILIZADOS

- Singleton:
  - Principalmente utilizado em código Php ao chamar instâncias estáticas de funcionalidade genérica durante o processamento dos pacotes oriundos de requisições HTTP/POST via JS; e
  - O método nativo do Php “AutoLoader” utiliza concepções em Singleton para evitar call-backs desnecessários durante a execução da aplicação no servidor.

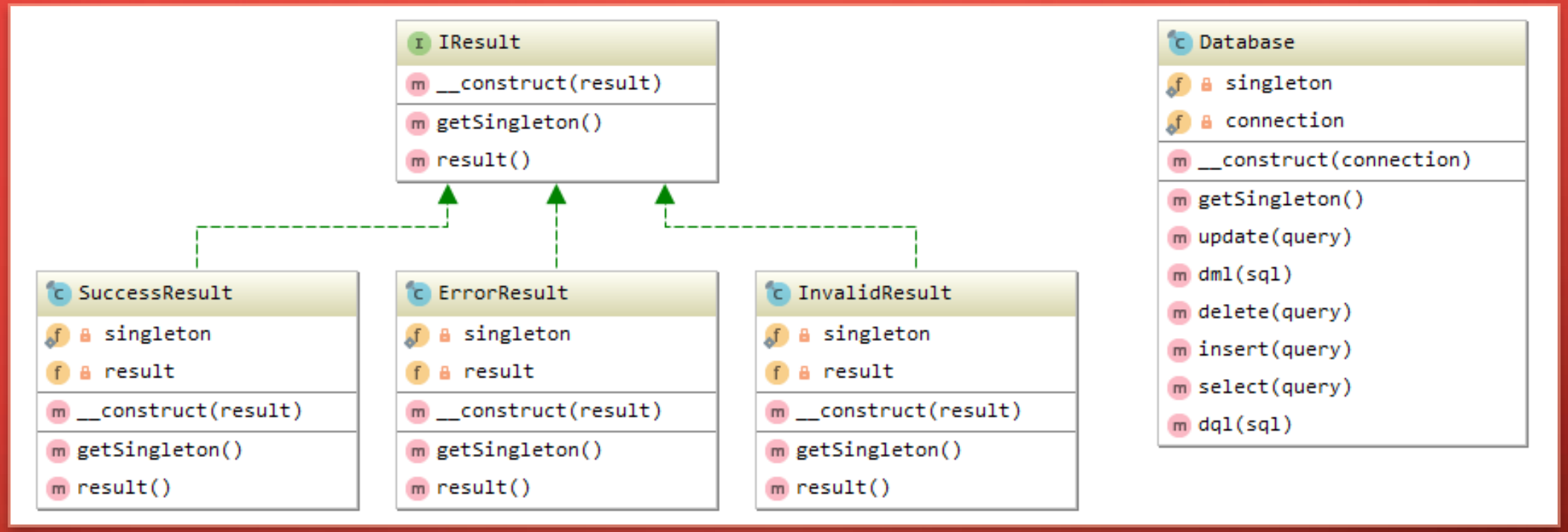

FIGURA 1

Aplicação do padrão Singleton nas classes de resultado e até dentro de uma ferramenta para interpretações das operações que serão replicadas para o banco de dados utilizando PDO.

FIGURA 2

Código-fonte parcial da declaração Singleton da classe “Database”.

```php
/**
 * Gets the singleton-like instance of **Database**.
 * @return Database
 */
public static function getSingleton()
{
    if (self::$singleton === null)
        self::$singleton = new Database(new \PDO("mysql:host=" . DB_HOST . ";dbname=" . DB_SCHEMA, DB_USER, DB_PASSWORD));

    return self::$singleton;
}
```

PADRÕES UTILIZADOS

- Chain of Responsibility:
  - Os pacotes de dados oriundos do cliente são processados utilizando este padrão comportamental; e
  - Cada pacote de dados possui um atributo identificador, proveniente de uma requisição HTTP/POST, chamado de “PacketID” que é utilizado para chamar instâncias Singleton de Handlers dinamicamente através de um dicionário local pela classe “PacketManager”.

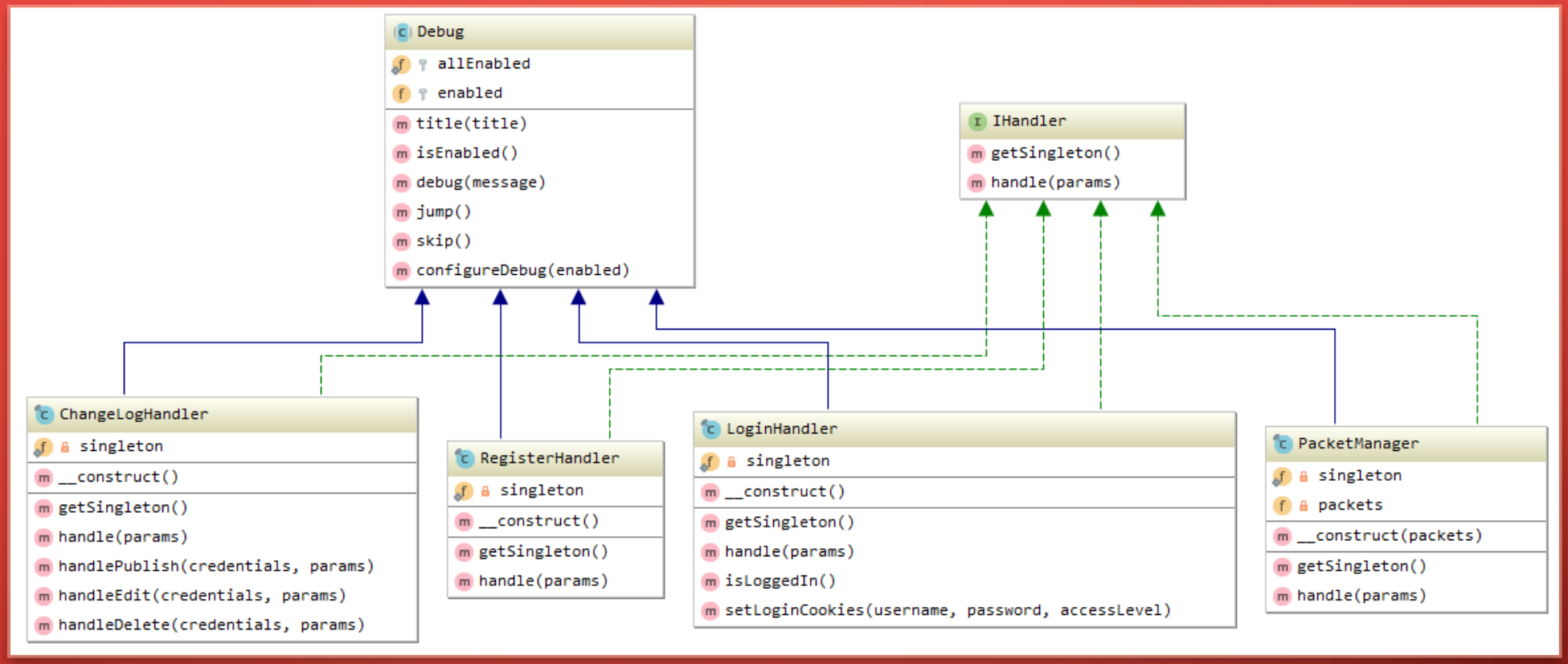

FIGURA 3

Aplicação do padrão Chain of Responsibility que é utilizado para realocar as obrigações através da classe “PacketManager” no qual atua como mediador de qual Handler deve atender aquele “PacketID” específico durante o runtime da aplicação.

FIGURA 4

Código-fonte parcial da classe “PacketManager” responsável em delegar as obrigações para os Handlers dinamicamente.

```php
/**
 * Handle all incoming packets from client-side (jQuery asynchronous integration).
 * @param array $params
 * @return null|mixed
 */
public function handle(array $params)
{
    if ($this->isEnabled()) {
        $this->title("Packet Manager Handler");
        $this->debug("Fetching data from request:");
        $this->debug(utils::arrayToJSON($params));
        $this->jump();
    }

    if (!array_key_exists("packetId", $params)) {
        if ($this->isEnabled()) $this->debug("Packet ID doesn't contains in params!");
        return null;
    }

    $id = $params["packetId"];
    $response = null;

    if (array_key_exists($id, $this->packets)) {
        $packetSingleton = $this->packets[$id];

        if (is_subclass_of($packetSingleton, "Debug"))
            if ($packetSingleton->isEnabled())
                $packetSingleton->skip();

        $response = $packetSingleton->handle($params);
    } else {
        if ($this->isEnabled())
            $this->debug("Packet ID <strong>" . $params["packetId"] . "</strong> isn't implemented!");
    }

    if ($this->isEnabled()) {
        $this->jump();
        $this->skip();
        $this->jump();
    }
    return $response;
}
```

FIGURA 5

Código-fonte da interface “IHandler” no qual é implementada nas classes que utilizam o padrão Chain of Responsibility.

```php
interface IHandler
{
    /**
     * Gets a singleton-like instance of class that implements **IHandler**.
     * @return mixed
     */
    public static function getSingleton();

    /**
     * A method to handle all required processing tasks of class that implements **IHandler**.
     * @param array $params
     * @return mixed
     */
    public function handle(array $params);
}
```

PROJETO 2

TK GAMES – SERVICES ARCHITECTURE

SOBRE

- Este projeto representa a estruturação arquitetural dos serviços que são implementados e constantemente utilizados em diferentes ambientes;
- Esta arquitetura aborda a vinculação de softwares que utilizam os frameworks .NET Standard 4.7.2 e .NET Core 3.1, além do toolkit Adobe Flex SDK 4.9.1;
- Utilizado as linguagens de programação nos sistemas: C++, C# 8.0, ActionScript 3.0, Flex, Php 7, JavaScript, HTML5 e CSS3;
- Fui um dos engenheiros a propor, reorganizar e apresentar a nova arquitetura dos serviços para melhor atender as necessidades das aplicações de forma eficiente e independente;
- Duração do planejamento: 2 semanas (aprox.); e
- Contribuidores: Nádio “Devwarlt” (BRA), Jay “Slendergo” (UK), Sharath “Devin” (USA), Richard “GlassBQ” (USA), Joe “Orb” (USA) e demais integrantes da TK Games.

APLICAÇÕES E CONCEITOS TEÓRICOS

- Principais conceitos arquiteturais abrangidos:
  - Comunicação inter-serviços através do design RESTful API (principalmente no serviço fornecido pela API App Engine); e
  - Comunicação intra-serviços através de protocolos TCP/IP entre processos via IPC (abrangidos pelos serviços da API Network Multiplexer e os processos de Game World Server).

APLICAÇÕES E CONCEITOS TEÓRICOS

- Principais aplicações arquiteturas:
  - Estrutura Componente-Conector: foi abordado a estrutura **Cliente-Servidor** durante o desenvolvimento das funcionalidades que seriam utilizadas pelo serviço Network Multiplexer API, cuja interação é atuar como disparador de pacotes entre os serviços internos do ambiente (“componentes”);
  - Padrão Transaction-Processing: implementados dentro do serviço Network Multiplexer API no qual também atua como gateway de comunicação e disparador de mensagens para os subprocessos; e
  - Padrão Orientado a Mensagens: este padrão é utilizado entre a comunicação IPC dos processos Network Multiplexer API e os subprocessos Game World Server (criados dinamicamente), nos quais utilizam os clusters de servidores Redis como mediadores para operações de “publish” e “subscribe” dentro de canais específicos.

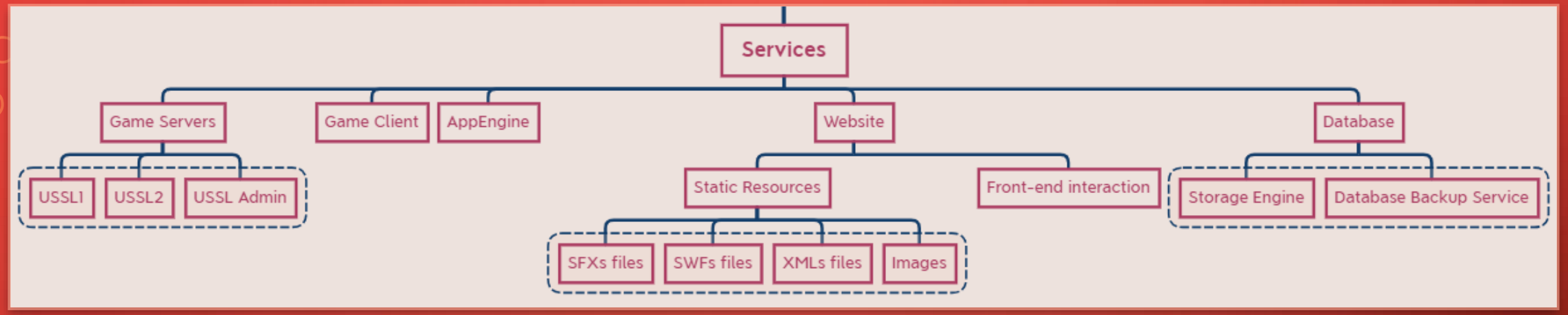

FIGURA 6

Principais ambientes abrangidos pelos serviços.

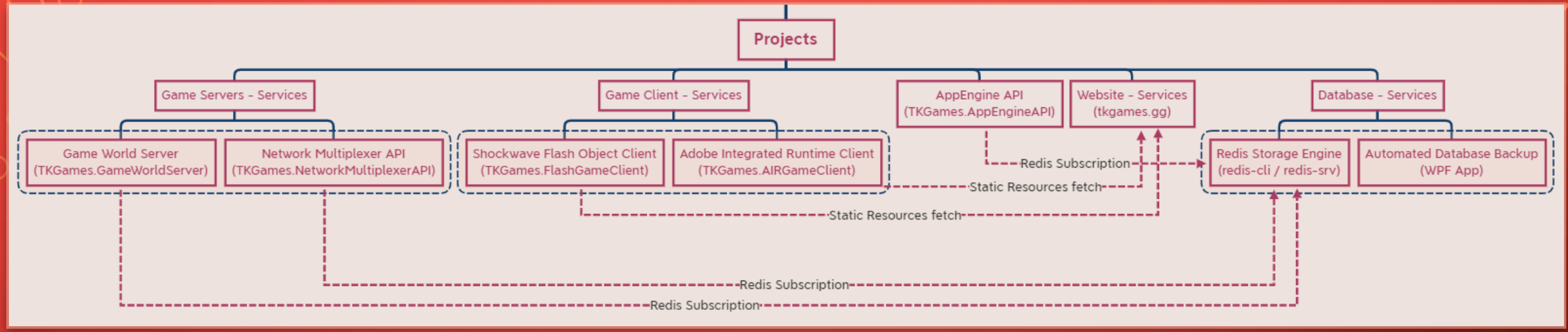

FIGURA 7

Visão geral dos principais e fundamentais serviços a serem desenvolvidos.

DÚVIDAS?

NÁDIO DIB

Contatos

OBRIGADO!