**Documentación de la Aplicación "Ticket Analyzer"**

**Descripción General**

La aplicación "Ticket Analyzer" es una herramienta basada en Symfony que permite analizar y listar las entradas disponibles para eventos en plataformas de venta de tickets como VividSeats y SeatGeek. El objetivo principal de la aplicación es extraer información de las páginas de estos proveedores y mostrar detalles relevantes sobre las entradas, como la sección, fila, cantidad disponible y precio.

**Estructura del Proyecto**

El proyecto se organiza siguiendo las mejores prácticas de Symfony, con una arquitectura basada en servicios y el uso de comandos de consola para interactuar con los proveedores de entradas.

La estructura clave incluye los siguientes componentes:

* **Providers**: Aquí se definen las clases que interactúan con los sitios web de los proveedores para extraer datos.
* **Command**: Un comando de Symfony que orquesta la ejecución de los proveedores y la presentación de los resultados en la consola.
* **Services**: Definidos en el archivo services.yaml, donde se configuran las dependencias y el autowiring de los servicios.

**Componentes Principales**

**1. AnalyzeTicketsCommand**

Esta clase se encarga de ejecutar el análisis de tickets desde la consola. El comando toma una URL de evento como argumento y determina a qué proveedor pertenece la URL para luego obtener las entradas correspondientes.

* **Archivo**: src/Command/AnalyzeTicketsCommand.php
* **Comando**: app:analyze-tickets <url>
* **Dependencias**:
  + VividSeatsProvider: Proveedor para extraer datos de VividSeats.
  + SeatGeekProvider: Proveedor para extraer datos de SeatGeek.
  + Logger: Registrador de eventos para la depuración y registro de errores.

**Flujo de trabajo**:

1. Se toma la URL del evento desde el input.
2. Se identifica el proveedor (VividSeats o SeatGeek).
3. Se utiliza el proveedor correspondiente para obtener las entradas disponibles.
4. Los resultados se imprimen en la consola, mostrando:
   * Sección.
   * Fila.
   * Precio por entrada.
   * Cantidad de entradas disponibles.
5. Si no se encuentran entradas o ocurre un error, se registran los mensajes apropiados en la consola.

**Ejemplo de uso**:

php bin/console app:analyze-tickets <https://www.vividseats.com/real-madrid-tickets-estadio-santiago-bernabeu-12-22-2024--sports-soccer/production/5045935>

**2. VividSeatsProvider**

Este proveedor es responsable de interactuar con la página de VividSeats. Utiliza el cliente HTTP de Symfony para realizar peticiones y extraer el HTML de la página, después lo analiza utilizando el componente DomCrawler de Symfony.

* **Archivo**: src/Provider/VividSeatsProvider.php
* **Métodos**:
  + getAvailableTickets(string $url): Recibe la URL de un evento y retorna una lista de objetos Ticket con los detalles de las entradas disponibles.

**Lógica de procesamiento**:

1. Realiza una petición GET a la URL del evento.
2. Verifica que la respuesta sea exitosa.
3. Utiliza el DomCrawler para identificar los elementos que contienen información de los tickets.
4. Extrae información relevante:
   * Sección (categoría del ticket).
   * Fila.
   * Precio.
   * Cantidad disponible.
5. Devuelve un array de objetos Ticket con esta información.

**3. SeatGeekProvider**

De manera similar a VividSeatsProvider, este proveedor extrae datos de SeatGeek. Actualmente implementa una lógica similar, pero su uso está enfocado a otra plataforma.

* **Archivo**: src/Provider/SeatGeekProvider.php

**4. Ticket Entity**

El objeto Ticket encapsula la información de una entrada específica, incluyendo la sección, fila y precio. Es la estructura de datos principal que se utiliza para retornar y procesar la información extraída de las páginas.

* **Archivo**: src/Entity/Ticket.php
* **Propiedades**:
  + section: Sección del estadio o lugar del evento.
  + row: Fila de la sección.
  + price: Precio del ticket.
  + quantity: Cantidad de tickets disponibles.

**Configuración de Servicios**

La configuración de los proveedores y otros servicios del proyecto se encuentra en el archivo services.yaml. Este archivo asegura que los servicios, como el cliente HTTP y el logger, sean inyectados correctamente en las clases que los necesitan.

* **Archivo**: config/services.yaml

Ejemplo de configuración para el proveedor de VividSeats:

yaml

Copiar código

App\Provider\VividSeatsProvider:

arguments:

$client: '@http\_client'

$logger: '@logger'

App\Provider\SeatGeekProvider:

arguments:

- '@http\_client'

**Monolog para Logging**

Monolog se utiliza para capturar logs del funcionamiento de la aplicación. Está configurado para diferentes niveles de severidad y puede registrar errores, advertencias, e información útil para el desarrollo y la depuración.

* **Archivo de configuración**: config/packages/monolog.yaml

En este archivo, se configura Monolog para manejar los logs en diferentes entornos (dev, prod, test) y permitir la depuración de cualquier problema con la extracción de datos.

**Ejecución del Proyecto**

1. **Instalar Dependencias**: Ejecutar composer install para instalar todas las dependencias necesarias del proyecto.
2. **Ejecutar el Comando**: Para analizar un evento, se debe ejecutar el comando de consola con la URL del evento correspondiente:

php bin/console app:analyze-tickets <URL\_DEL\_EVENTO>