Certo, posso prepararti un documento che riassume l'implementazione che abbiamo sviluppato per lo scraping dei dati da FBRef. Questo documento includerà il codice completo per il servizio e per il comando Artisan, con spiegazioni dettagliate ed esempi su come utilizzarlo.

## **Documento di Implementazione: Web Scraper per FBRef con Laravel**

### **1. Obiettivo**

L'obiettivo di questa implementazione è estrarre dati statistici tabellari dalle pagine delle squadre del sito FBRef (es. https://fbref.com/it/squadre/...) utilizzando un approccio di web scraping. I dati estratti verranno salvati in formato JSON per una successiva analisi o importazione nel database.

### **2. Prerequisiti**

* Un progetto Laravel configurato.
* La libreria fabpot/goutte installata nel progetto. Se non ancora installata, eseguire:  
  Bash  
  composer require fabpot/goutte

### **3. Componenti dell'Implementazione**

L'implementazione si basa su due componenti principali:

* Un **Servizio** (FbrefScrapingService) responsabile della logica di scraping.
* Un **Comando Artisan** (ScrapeFbrefTeamStatsCommand) che utilizza il servizio per eseguire lo scraping e gestire l'output.

#### **3.1. Il Servizio: FbrefScrapingService**

Questo servizio incapsula tutta la logica per scaricare la pagina HTML, identificare le tabelle di statistiche e fare il parsing del loro contenuto.

**Percorso del file:** app/Services/FbrefScrapingService.php

**Codice Completo:**

PHP

<?php  
  
namespace App\Services;  
  
use Goutte\Client as GoutteClient;  
use Symfony\Component\DomCrawler\Crawler;  
use Illuminate\Support\Facades\Log;  
use Illuminate\Support\Facades\Storage; // Per il debug dell'HTML, se necessario  
use Illuminate\Support\Str; // Per generare chiavi "slug"  
  
class FbrefScrapingService  
{  
 private $goutteClient;  
 private $targetUrl;  
  
 public function \_\_construct()  
 {  
 $this->goutteClient = new GoutteClient();  
  
 // Imposta un User-Agent. Puoi usare quello personalizzato o uno più standard.  
 // $this->goutteClient->setServerParameter('HTTP\_USER\_AGENT', "FantaprojectBot/1.0 (contatto@tuasito.com)");  
 $this->goutteClient->setServerParameter('HTTP\_USER\_AGENT', "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/100.0.0.0 Safari/537.36");  
  
 // Esempio di configurazione Guzzle (opzionale, per timeout, SSL, ecc.)  
 // $guzzleClientOptions = [  
 // 'timeout' => 60, // Timeout in secondi  
 // 'verify' => false, // Disabilita la verifica SSL - sconsigliato in produzione  
 // ];  
 // $guzzleClient = new \GuzzleHttp\Client($guzzleClientOptions);  
 // $this->goutteClient->setClient($guzzleClient);  
 }  
  
 /\*\*  
 \* Imposta l'URL target per lo scraping.  
 \* @param string $url  
 \* @return self  
 \*/  
 public function setTargetUrl(string $url): self  
 {  
 $this->targetUrl = $url;  
 return $this;  
 }  
  
 /\*\*  
 \* Esegue lo scraping delle statistiche della squadra dall'URL impostato.  
 \* Cerca di estrarre tutte le tabelle il cui ID inizia con "stats\_".  
 \* @return array Dati estratti o array con chiave 'error'.  
 \*/  
 public function scrapeTeamStats(): array  
 {  
 if (!$this->targetUrl) {  
 Log::error('FbrefScrapingService: Target URL non impostato.');  
 return ['error' => 'Target URL non impostato.'];  
 }  
  
 try {  
 Log::info("FbrefScrapingService: Inizio scraping da {$this->targetUrl}");  
 $crawler = $this->goutteClient->request('GET', $this->targetUrl);  
  
 // DEBUG: Decommenta per salvare l'HTML grezzo ricevuto per analisi  
 /\*  
 $htmlContent = $crawler->html();  
 $debugFileName = 'fbref\_dump\_SERVICE\_' . Str::slug(basename($this->targetUrl)) . '.html';  
 Storage::disk('local')->put('scraping/' . $debugFileName, $htmlContent); // Salva nella sottocartella scraping  
 Log::info("FbrefScrapingService: HTML grezzo salvato in storage/app/scraping/{$debugFileName}. Lunghezza: " . strlen($htmlContent));  
 if (strlen($htmlContent) < 2000) {  
 Log::warning("FbrefScrapingService: ATTENZIONE - L'HTML ricevuto è molto corto.");  
 }  
 \*/  
  
 } catch (\Exception $e) {  
 Log::error("FbrefScrapingService: Impossibile raggiungere la pagina {$this->targetUrl}: " . $e->getMessage());  
 return ['error' => 'Impossibile raggiungere la pagina: ' . $e->getMessage()];  
 }  
  
 $scrapedData = [];  
  
 // Seleziona tutte le tabelle il cui ID inizia con "stats\_"  
 // Escludiamo alcune tabelle di servizio o navigazione per specificità  
 $allStatTableNodes = $crawler->filter('table[id^="stats\_"]:not([class\*="switcher\_"]):not([class\*="small\_text"])');  
   
 Log::info("FbrefScrapingService: Trovate {$allStatTableNodes->count()} tabelle potenziali con ID che inizia per 'stats\_'.");  
  
 $allStatTableNodes->each(function (Crawler $tableNode) use (&$scrapedData) {  
 $tableId = $tableNode->attr('id'); // Ottieni l'ID della tabella  
 $captionNode = $tableNode->filter('caption'); // Cerca la didascalia (caption)  
   
 $tableKey = $tableId; // Chiave di default nell'array dei risultati  
 $descriptionForLog = "Tabella ID: {$tableId}";  
  
 if ($captionNode->count() > 0) {  
 $captionText = trim($captionNode->text());  
 if (!empty($captionText)) {  
 $tableKey = Str::slug($captionText, '\_'); // Crea una chiave leggibile dalla didascalia  
 $descriptionForLog = "Tabella '{$captionText}' (ID: {$tableId})";  
 }  
 }  
   
 // Gestione semplice di collisioni di chiavi (se due didascalie diverse generano lo stesso slug)  
 if (array\_key\_exists($tableKey, $scrapedData)) {  
 $originalTableKey = $tableKey;  
 $tableKey = $tableKey . '\_' . $tableId; // Rendi la chiave unica aggiungendo l'ID  
 Log::warning("FbrefScrapingService: Collisione di chiavi per '{$originalTableKey}'. Uso nuova chiave '{$tableKey}' per ID '{$tableId}'.");  
 if (array\_key\_exists($tableKey, $scrapedData)) { // Se ancora collisione (improbabile)  
 Log::error("FbrefScrapingService: Collisione di chiavi irrisolvibile per ID '{$tableId}'. Salto.");  
 return; // Salta questa tabella  
 }  
 }  
  
 Log::info("FbrefScrapingService: Inizio parsing per {$descriptionForLog}.");  
 $parsedTable = $this->parseFbrefTable($tableNode, $descriptionForLog);  
  
 if (isset($parsedTable['error'])) {  
 Log::warning("FbrefScrapingService: Errore durante il parsing per {$descriptionForLog}: " . $parsedTable['error']);  
 $scrapedData[$tableKey] = ['error' => $parsedTable['error'], 'original\_id' => $tableId, 'description' => $descriptionForLog];  
 } else {  
 if (empty($parsedTable)) {  
 Log::info("FbrefScrapingService: Nessun dato estratto (o tabella vuota) per {$descriptionForLog}.");  
 }  
 $scrapedData[$tableKey] = $parsedTable; // Salva anche array vuoti per indicare che è stata processata  
 }  
 });  
  
 Log::info("FbrefScrapingService: Scraping completato per {$this->targetUrl}. Processate " . count($scrapedData) . " tabelle.");  
 return $scrapedData;  
 }  
  
 /\*\*  
 \* Fa il parsing di una tabella HTML standard di FBRef.  
 \* @param Crawler $tableNode Il nodo Crawler della tabella.  
 \* @param string $tableDebugInfo Un identificatore per il logging.  
 \* @return array Array di dati della tabella o array con chiave 'error'.  
 \*/  
 private function parseFbrefTable(Crawler $tableNode, string $tableDebugInfo = 'unknown\_table'): array  
 {  
 $tableData = [];  
 $headers = [];  
  
 $headerRowNode = $tableNode->filter('thead tr')->last();  
 if ($headerRowNode->count() > 0) {  
 $headerRowNode->filter('th')->each(function (Crawler $th) use (&$headers) {  
 $headers[] = trim($th->text());  
 });  
 }  
  
 if (empty($headers)) {  
 Log::error("FbrefScrapingService: Headers non trovati per la tabella '{$tableDebugInfo}'. La struttura potrebbe essere cambiata o la tabella non è standard.");  
 return ['error' => "Headers non trovati per tabella: {$tableDebugInfo}"];  
 }  
  
 $tableNode->filter('tbody tr')->each(function (Crawler $rowNode, $rowIndex) use (&$tableData, $headers, $tableDebugInfo) {  
 $rowData = [];  
 $cells = $rowNode->filter('th, td');   
  
 if ($rowNode->matches('.spacer') || $rowNode->filter('td[colspan]')->count() > 0 || $cells->count() == 0) {  
 // Log::debug("FbrefScrapingService: Riga {$rowIndex} saltata (spacer/colspan/vuota) nella tabella {$tableDebugInfo}.");  
 return;   
 }  
  
 if ($cells->count() === count($headers)) {  
 $cells->each(function (Crawler $cellNode, $index) use (&$rowData, $headers, $rowIndex, $tableDebugInfo) {  
 if (isset($headers[$index])) {  
 $rowData[$headers[$index]] = trim($cellNode->text());  
 } else {  
 $rowData['col\_imprevista\_' . $index] = trim($cellNode->text());  
 Log::warning("FbrefScrapingService: Header mancante per indice {$index}, riga {$rowIndex}, tabella {$tableDebugInfo}.");  
 }  
 });  
 if (!empty($rowData)) {  
 $tableData[] = $rowData;  
 }  
 } else {  
 Log::warning("FbrefScrapingService: Discrepanza numero celle/header ({$cells->count()}/".count($headers).") per riga {$rowIndex}, tabella {$tableDebugInfo}. Contenuto (parziale): " . substr(preg\_replace('/\s+/', ' ', trim($rowNode->text())), 0, 100));  
 }  
 });  
 return $tableData;  
 }  
}

**Spiegazione del Servizio:**

* **\_\_construct()**: Inizializza il client Goutte e imposta uno User-Agent per le richieste HTTP.
* **setTargetUrl(string $url)**: Imposta l'URL della pagina da cui fare lo scraping.
* **scrapeTeamStats()**:
  + Effettua la richiesta HTTP GET all'URL specificato.
  + Seleziona tutte le tabelle il cui ID inizia con stats\_ (escludendo alcune classi comuni per elementi non di dati).
  + Per ogni tabella trovata:
    - Estrae l'ID e la didascalia (<caption>).
    - Genera una chiave univoca per l'array dei risultati (preferibilmente dalla didascalia, altrimenti dall'ID).
    - Chiama parseFbrefTable() per estrarre i dati.
    - Registra informazioni e potenziali errori nei log di Laravel.
  + Restituisce un array associativo dove le chiavi sono gli identificatori delle tabelle e i valori sono gli array dei dati estratti.
* **parseFbrefTable(Crawler $tableNode, string $tableDebugInfo)**:
  + Prende un oggetto Crawler che rappresenta una singola tabella.
  + Estrae gli header delle colonne dall'ultima riga di <thead>.
  + Itera sulle righe (<tr>) in <tbody>.
  + Per ogni riga, estrae il contenuto delle celle (<th> e <td>) e le mappa agli header.
  + Salta righe "spacer" o con colspan che spesso non contengono dati di giocatori.
  + Restituisce un array di array associativi (ogni array interno rappresenta una riga della tabella).

#### **3.2. Il Comando Artisan: ScrapeFbrefTeamStatsCommand**

Questo comando fornisce un'interfaccia da riga di comando per utilizzare FbrefScrapingService. Prende un URL come argomento, esegue lo scraping e salva i risultati in un file JSON.

**Percorso del file:** app/Console/Commands/ScrapeFbrefTeamStatsCommand.php

**Codice Completo:**

PHP

<?php  
  
namespace App\Console\Commands;  
  
use Illuminate\Console\Command;  
use App\Services\FbrefScrapingService;  
use Illuminate\Support\Facades\Log;  
use Illuminate\Support\Facades\Storage;  
use Illuminate\Support\Str;  
  
class ScrapeFbrefTeamStatsCommand extends Command  
{  
 /\*\*  
 \* The name and signature of the console command.  
 \* Esempio: php artisan fbref:scrape-team "URL\_SQUADRA"  
 \*/  
 protected $signature = 'fbref:scrape-team   
 {url : L\'URL completo della pagina della squadra su FBRef (es. https://fbref.com/it/squadre/id/Statistiche-NomeSquadra)}   
 {--team\_id= : ID opzionale della squadra nel tuo database per associare i dati (non usato attivamente in questa fase)}';  
  
 /\*\*  
 \* The console command description.  
 \*/  
 protected $description = 'Esegue lo scraping delle statistiche di una squadra da un URL FBRef e salva i dati grezzi in un file JSON nella sottocartella /scraping.';  
  
 private $scrapingService;  
  
 /\*\*  
 \* Create a new command instance.  
 \*/  
 public function \_\_construct(FbrefScrapingService $scrapingService)  
 {  
 parent::\_\_construct();  
 $this->scrapingService = $scrapingService;  
 }  
  
 /\*\*  
 \* Execute the console command.  
 \*/  
 public function handle(): int  
 {  
 $url = $this->argument('url');  
 $teamId = $this->option('team\_id');   
  
 $this->info("Avvio scraping per l'URL: {$url}");  
 if ($teamId) {  
 $this->line("> ID Squadra (opzionale) per riferimento futuro: {$teamId}");  
 }  
  
 $data = $this->scrapingService->setTargetUrl($url)->scrapeTeamStats();  
  
 if (isset($data['error'])) {  
 $this->error("Errore critico durante lo scraping: " . $data['error']);  
 Log::channel('stderr')->error("Errore scraping FBRef per URL {$url}: " . $data['error']);  
 return Command::FAILURE;   
 }  
  
 $this->info('Scraping dei dati grezzi completato dal servizio.');  
 $this->line('--- Riepilogo Tabelle Processate ---');  
  
 $foundDataCount = 0;  
 if (is\_array($data) && !empty($data)) {  
 foreach ($data as $tableKey => $tableContent) {  
 if ($tableContent === null) {  
 $this->warn("- Tabella '{$tableKey}': Non trovata o nessun dato (valore null).");  
 continue;  
 }  
 if (isset($tableContent['error'])) {  
 $this->error("- Tabella '{$tableKey}': Errore durante il parsing - " . $tableContent['error']);  
 continue;  
 }  
   
 $rowCount = is\_array($tableContent) ? count($tableContent) : 0;  
 if ($rowCount > 0) {  
 $this->line("[OK] Tabella '{$tableKey}': {$rowCount} righe estratte.");  
 $foundDataCount++;  
 } else {  
 $this->line("[INFO] Tabella '{$tableKey}': 0 righe estratte (vuota o solo header).");  
 }  
 }  
 } else {  
 $this->warn("Nessun dato o struttura dati inattesa ricevuta dal servizio di scraping.");  
 }  
  
 if ($foundDataCount > 0) {  
 $this->info("\nAlmeno una tabella con dati è stata elaborata con successo.");  
 } else {  
 $this->warn("\nNessun dato tabellare significativo è stato estratto (o tutte le tabelle erano vuote/hanno avuto errori di parsing).");  
 }  
  
 // Salva l'intero output $data in un file JSON nella sottocartella "scraping"  
 $urlPath = parse\_url($url, PHP\_URL\_PATH);  
 $baseFileName = Str::slug(basename($urlPath ?: 'unknown\_url'));  
 $urlHash = substr(md5($url), 0, 6); // Hash breve dell'URL per unicità  
   
 $fileName = 'fbref\_data\_' . $baseFileName . '\_' . $urlHash . '\_' . date('Ymd\_His') . '.json';  
 $subfolder = 'scraping'; // Sottocartella definita  
 $filePath = $subfolder . '/' . $fileName;   
  
 try {  
 Storage::disk('local')->put($filePath, json\_encode($data, JSON\_PRETTY\_PRINT | JSON\_UNESCAPED\_UNICODE));  
 $this->info("\nOutput completo dei dati grezzi salvato in: storage/app/{$filePath}");   
 } catch (\Exception $e) {  
 $this->error("Impossibile salvare il file JSON di debug: " . $e->getMessage());  
 }  
  
 Log::info("Comando fbref:scrape-team completato per URL {$url}.");  
 $this->info("\nComando terminato.");  
 return Command::SUCCESS;  
 }  
}

**Spiegazione del Comando:**

* **$signature**: Definisce come chiamare il comando (fbref:scrape-team), l'argomento obbligatorio {url} e l'opzione facoltativa {--team\_id=}.
* **$description**: Descrizione del comando che appare con php artisan list.
* **\_\_construct(FbrefScrapingService $scrapingService)**: Inietta il servizio FbrefScrapingService tramite dependency injection.
* **handle()**:
  + Ottiene l'URL e l'eventuale team\_id dai parametri del comando.
  + Chiama il servizio per eseguire lo scraping.
  + Stampa un riepilogo sintetico in console del numero di righe estratte per ogni tabella.
  + Salva l'intero array $data (contenente tutte le tabelle estratte) in un file JSON. Il file viene salvato in storage/app/scraping/ con un nome univoco che include parte dell'URL e un timestamp.

### **4. Registrazione del Comando**

Assicurati che il comando sia registrato nel kernel della console di Laravel. Apri app/Console/Kernel.php e aggiungi il tuo comando all'array $commands (se non è già presente o se la tua versione di Laravel non lo scopre automaticamente):

PHP

// in app/Console/Kernel.php  
  
protected $commands = [  
 // ... altri comandi esistenti ...  
 \App\Console\Commands\ScrapeFbrefTeamStatsCommand::class,  
];

Dopo aver modificato i file, è buona pratica eseguire:

Bash

composer dump-autoload  
php artisan optimize:clear

### **5. Come Eseguire lo Scraping**

Una volta che tutto è configurato, puoi eseguire lo scraping da terminale con il seguente comando:

Bash

php artisan fbref:scrape-team "URL\_DELLA\_PAGINA\_SQUADRA\_SU\_FBREF"

**Esempio:**

Bash

php artisan fbref:scrape-team "https://fbref.com/it/squadre/4cceedfc/Statistiche-Pisa"

### **6. Output Atteso**

* **Console:** Vedrai messaggi informativi sull'avvio e il completamento dello scraping, e un riepilogo per ogni tabella processata (es. [OK] Tabella 'statistiche\_ordinarie\_serie\_b': 25 righe estratte.).
* **File JSON:** Un file JSON verrà creato nella cartella storage/app/scraping/. Il nome del file sarà simile a fbref\_data\_statistiche-pisa\_abcdef\_20250603\_103000.json. Questo file conterrà la struttura completa dei dati estratti, con un array per ogni tabella.

**Esempio (molto semplificato) della struttura del JSON:**

JSON

{  
 "statistiche\_ordinarie\_serie\_b": [  
 {  
 "Giocatore": "Nome Giocatore 1",  
 "Naz": "IT",  
 "Ruolo": "CC,ATT",  
 "Età": "25",  
 // ... altre colonne ...  
 },  
 {  
 "Giocatore": "Nome Giocatore 2",  
 // ...  
 }  
 ],  
 "statistiche\_portiere\_serie\_b": [  
 {  
 "Giocatore": "Nome Portiere 1",  
 "Naz": "FR",  
 // ... altre colonne ...  
 }  
 ],  
 // ... altre tabelle ...  
}

### **7. Passi Successivi Suggeriti**

1. **Analisi del JSON:** Apri il file JSON generato per ispezionare nel dettaglio i dati estratti. Verifica la correttezza degli header e dei valori per tutte le tabelle di tuo interesse.
2. **Affinamento del Parsing:** Se noti che alcune tabelle non sono parsate correttamente (es. header sbagliati, dati mancanti, righe indesiderate), dovrai affinare la logica nel metodo parseFbrefTable() di FbrefScrapingService.php. Potrebbe essere necessario gestire strutture di tabella diverse o escludere righe specifiche.
3. **Implementazione del Salvataggio nel Database:** Una volta che sei soddisfatto dei dati grezzi, modifica il metodo handle() del comando ScrapeFbrefTeamStatsCommand.php per iterare sull'array $data e salvare le informazioni nel tuo database utilizzando i modelli Eloquent.

### **8. Considerazioni Importanti**

* **Robustezza:** Il web scraping è intrinsecamente fragile. Se FBRef cambia la struttura HTML delle sue pagine (ID, classi, layout delle tabelle), lo scraper potrebbe smettere di funzionare o estrarre dati errati. Sarà necessario monitorare e aggiornare lo scraper periodicamente.
* **Rispetto per il Sito Target:**
  + Controlla sempre il file robots.txt del sito (es. https://fbref.com/robots.txt) per le direttive sui bot.
  + Evita di fare troppe richieste in un breve lasso di tempo (crawl delay). Se devi fare scraping di molte pagine, introduci delle pause tra le richieste.
* **Gestione degli Errori:** L'implementazione attuale include logging di base. Per un uso in produzione, potresti voler migliorare la gestione degli errori e delle notifiche.

Spero che questo documento ti sia utile!