Web Assistant for Impaired People

Features

The WIA application assists individuals with impairments in entering a URL of a web page and subsequently retrieves descriptions of the image content on the page using AI Image Captioning.

Furthermore, WIA provides public access to an API for image description purposes.

Installation

1. Clone the WIA repository.
2. Configure the **wia-infrastructure/.env** file, and modify the **SOURCE_CODE_PATH** to point to the project directory. For instance, if you've cloned the project in /var/www/html, set SOURCE_CODE_PATH to that path.
3. Grant execution permissions to the setup file using the command **chmod +x setup && sed -i -e 's/\r$//' setup**.
4. Execute the installation file by running **./setup**.

Once the installation is complete, access the application by opening the following URL: https://wia.182.80.0.101.nip.io/.

For utilizing the public API, use the following endpoint and send a request with {url: image url} as application/json with POST method:
https://wia.182.80.0.101.nip.io/v1/webpages/images/describe

Demo url to test the system: https://www.france24.com/fr/

Technology Selection

To establish the development and testing environments, Docker is employed to create and initiate standalone services, fostering a conducive environment for application development and testing.

The frontend single page application is streamlined for ease of development using VueJS in conjunction with the Primevue UI ecosystem.

For the backend components, Laravel 10, PHP 8.1, and MySQL 8 are strategically chosen, aligning with the project's requirements, scope, time constraints, and development efficiency.

The web scraper module's architecture hinges on Symfony's browser-kit and http-client due to their established longevity, substantial GitHub stars, lightweight composition, easy integration capabilities, and suitability for the project's defined scope.

The decision to integrate Cloudinary AI Captioning is rooted in the service's auspicious potential, driven by its high precision in generating descriptive labels for image objects. Additionally, the service extends an accessible API, streamlining usage, and boasts a free trial plan accommodating up to 25,000 transformations—sufficient for comprehensive project testing. This service harnesses the power of Large Language Models to generate textual depictions of images in a natural language context.

Development Approach

The project follows the Shape Up methodology for analysis and management. Project scopes are defined and organized, with an initial focus on scrutinizing the most intricate, unfamiliar, and demanding aspects to proactively identify risks and limitations.

Supplementary requirements are designated as "Nice To Haves" and are addressed subsequent to fulfilling the essential project scopes.

The analysis phase proceeds by identifying and dissecting the core system components, offering an abstract view of the architecture.

Development commences with swift validation of the uncharted facets and subsequently adopts a Test-Driven Development (TDD) approach. This methodology allows for a gradual assembly of system components, alongside iterative code refinement and feature testing.

To enhance feature quality, a phase of manual testing is undertaken.

This structured approach assures that the project is systematically managed, development challenges are mitigated, and the final product is of commendable quality.

Architecture

The project comprises three core sub-systems:

- **wia-infrastructure:** This component orchestrates the creation of development and testing environments, in addition to managing the application's setup process.
- **wia-spa:** Responsible for presenting the application's user interface.
- **wia-api:** Manages backend services, catering to both the wia-spa and public API.

The following principles, patterns, and architectural styles are employed in wia-api and wia-spa:

- SRP
- Dependency Injection, Dependency Inversion and IoC container
- Acyclic dependency principle
- DRY
- Builder pattern
- Strategy pattern
- Active Record
- Component-based
- Separation of Concerns
- MVC
- Modular monolith
- Headless architecture
- Messaging Queue and Job processing for long processes
- Push notification to improve user experience
- TDD
- Code sniffer and linter
- Input Validations

Challenges Faced

The primary challenge emerged as the quest for an appropriate web scraping service or open-source library. To address this, a swift Proof of Concept (PoC) was executed, evaluating various options including Laravel Sanctum, Symfony browser-kit, and simple-html-dom. After careful consideration, Symfony browser-kit coupled with http-client was selected due to its promise in image extraction, good maintenance, lightweight design devoid of headless browser dependencies.

A second PoC was conducted to determine the most fitting AI Image Captioning library or service. Scrutinizing options like Google AI Vision, Cloudinary, Tensorflow, and HuggingFace transformers, HuggingFace transformers and Cloudinary underwent more extensive analysis. Although HuggingFace showcased commendable result accuracy, the architectural shift to gRPC usage from Python hindered its implementation within the defined time frame. Additionally, open-source libraries exhibited resource consumption and performance drawbacks. Consequently, Cloudinary emerged as the optimal choice—displaying promising results and seamless PHP integration through API calls, coupled with swift service processing.

Upon completing the initial iteration of API implementation and UI integration, a notable performance drop was observed when processing web pages containing over 10 images. Prior to optimization, users encountered delays exceeding 20 seconds while waiting for results after sending the page URL to the endpoint. To alleviate this performance challenge and elevate user experience, enhancements were enacted through messaging queues, job processing with 8 workers, and push notifications. These improvements tangibly enhanced performance and user satisfaction.

Further augmenting performance, a database caching system was incorporated. This initiative aimed to retrieve previously processed images directly from the repository, bypassing the image captioning process, and subsequently contributing to an optimized user experience and performance efficiency.

Areas for Improvement

Given the time constraints, certain areas that warrant enhancement were deferred and remain as pending tasks. These potential improvements encompass:

- **Token-Based Authentication:** Integrate token-based authentication into API services through Laravel Sanctum or equivalent libraries for fortified security.
- **CSRF Token:** Implement CSRF tokens to bolster URL input validation and overall security measures.
- **CAPTCHA Implementation:** Strengthen security defenses against bot attacks by incorporating CAPTCHA verification.
- **Third-Party Library Exception Handling:** Enhance resilience by implementing robust exception handling mechanisms for third-party libraries or services.
- **Extended Validation Layers**: Augment validations across separate layers such as services.
- **Rate Limitation Middleware:** Introduce middleware for API services to manage server load and quotas through rate limitation enforcement.
- **Failed Job Handling:** Develop process for managing failed jobs.
- **Decoupling AI Module:** Achieve scalability and efficient communication by transforming the AI module into an autonomous service via gRPC integration.
- **Enhanced Image ID Generation:** Enhance image ID generation by extracting data from image metadata, rather than relying solely on image source.
- **Webpage-Image Relationship Enhancement:** Consider either establishing a relation between the webpage and image tables or adopting NoSQL approaches to improve data management and performance.
- **UI Refinement:** Rectify UI errors, address image aspect ratio concerns, and mitigate flickering issues.
- **Code Quality Assurance:** Evaluate code quality through tools like SonarQube to identify areas for improvement.
- **Load Testing:** Assess system resilience under load by conducting load tests using tools like Locust.
- **Security Vulnerability Verification:** Mitigate vulnerabilities by validating the system for security weaknesses using tools like Snyk.

While the time constraints dictated certain prioritization, these potential improvements demonstrate an awareness of the project's potential for further refinement, scalability, security, and user experience.

Constraints

- The web scraper module has limitations in extracting images that employ deprecated xlink:href attributes (e.g., ocus.com) and filtered for those encoded in base64.
- Some websites limit access for non-ssl calls, to avoid SSL warning or limitation install mkcert and run the following command on the wia-infrastructure folder:
  `export CAROOT=$PWD/apache2/ssl/caRoot/ && mkcert -install`
- Additionally, the current implementation lacks push notification support for scenarios involving multiple user sessions