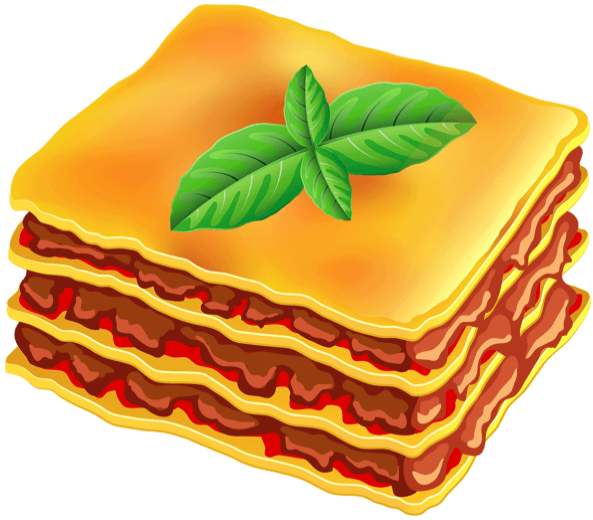
Tesseract LASAGNA v2.5



1. Rapid PWA Development

Tesseract LASAGNA is a high-performance, contemporary, and modular PHP object-oriented framework designed for the rapid prototyping of Progressive Web Applications (PWAs). It operates efficiently on an Apache 2.4 web server and provides comprehensive control through its command-line interface (CLI). The entire system is built with modularity in mind, ensuring that virtually no component of the framework is mandatory. This design allows for extensive flexibility, enabling parts to be easily replaced, modified, or even configured to utilize entirely different directory structures, adapting seamlessly to diverse project requirements.

1.1. Concept

Tesseract's core functionality revolves around its unique data input mechanism: it leverages **Google Sheets CSV exports** to construct its Model. This layered approach to data input is precisely why the framework is codenamed "LASAGNA."

Once the Model is built, **Presenters** take over, processing this data and transforming it into various output formats, including TEXT, JSON, XML, HTML5, or any other custom format required. The visual representation, or **View**, is flexibly constructed using a collection of **Mustache templates and partials**.

Fundamentally, Tesseract is built upon robust **Composer components**, enabling a sophisticated **RESTful API**. It also features a comprehensive **command-line interface (CLI)** for streamlined

operations and incorporates **Continuous Integration (CI) testing** to ensure stability and reliability.

A distinctive architectural choice for Tesseract is its **absence of traditional database structures**. This design simplifies the implementation of various scaling strategies and facilitates seamless integrations with other systems. For user authentication, Tesseract employs **OAuth 2.0**, securing access through an **encrypted "passport" cookie**.

2. Installation and Updating

This section outlines the process for installing and updating Tesseract LASAGNA from its PHP source, along with a note on Docker availability.

2.1. PHP Source

To get started with Tesseract LASAGNA from its PHP source, follow these steps:

Clone the Repository: Begin by cloning the official Tesseract LASAGNA repository from GitHub using

```
git clone https://github.com/GSCloud/lasagna.git
```

Initial Installation: Navigate into the newly created `lasagna` directory and run the `make install` command to set up the core components and dependencies:

```
make install
```

Updating Dependencies: To update your Tesseract LASAGNA installation and its dependencies, execute the `make update` command from within the `lasagna` directory:

```
make update
```

Explore Other Commands: For a comprehensive list of all available development and testing commands, simply run `make` without any arguments:

```
make
```

You will see a menu similar to this (note that port 9000 is reserved for Docker implementations):

```
👾 Tesseract LASAGNA

📦 TESSERACT: tesseract-lasagna 🔴 tesseract-lasagna port: 9000
```

```
» DEVELOPMENT
install    core installation
update     update dependencies
doctor     check installation

icons      update icons
base       download and build base CSV
refresh    refresh cloud CSV
clear      clear temporary files
sync       sync to the remote host
docs       convert documentation

» TESTING
stan       PHPStan test
unit       UNIT test
test       LOCAL integration test
prod       PRODUCTION integration test
```

2.2. Docker

The official Docker image for Tesseract LASAGNA is not yet publicly available. It is planned for release once the framework reaches its stable version 2.5.0.

3. How It Works

Tesseract LASAGNA operates through a carefully orchestrated flow, starting from the web server's initial request and progressing through several core components to render the final output.

3.1. Index

The journey begins with the `www/index.php` file. This file serves as the primary entry point, directed by Apache's `.htaccess` configuration using `mod_rewrite`. The `index.php` file is responsible for establishing various constant overrides specific to the application's environment before it proceeds to load the essential `Bootstrap.php` core file, located in the root directory.

3.2. Bootstrap

The `Bootstrap.php` file takes charge of setting critical constants and configuring the application's environment. During this phase, the Nette Debugger is also dynamically instantiated, providing powerful debugging capabilities when needed. Once its initial setup is

complete, Bootstrap loads the `App.php` core file from the designated `app` folder, handing over control for further application logic.

3.3. App

The `App.php` component is central to Tesseract's operation. It handles the processing of both public and private application configuration files, sets up the caching mechanisms (with optional support for Redis database integration), and rigorously configures the URL routing. Furthermore, it takes responsibility for emitting Content Security Policy (CSP) headers to enhance security and constructs the foundational Model, which is represented as a multidimensional array.

Based on the actual URI route, App dynamically loads the corresponding Presenter. It can also adapt to a command-line interface (CLI) context, launching a CLI-specific Presenter if a CLI environment is detected. Once the Presenter completes its work and returns an updated Model, App echoes the final output and sets any remaining necessary HTTP headers, including optional debugging information. This marks the conclusion of the runtime process for the request.

3.4. Router

The Router functions as an integral part of the App script. Its routing logic is meticulously defined by combining multiple routing tables, formatted in NE-ON (Nette Object Notation), which are located within the `/app` folder. These tables are merged using an array recursive replacement strategy, allowing for flexible and powerful URL management.

3.5. Presenter

A Presenter in Tesseract is an instance of a subclass derived from the abstract class `APresenter.php`. Each Presenter must define at least a `process()` method, which is invoked by the App. The `process()` method offers dual capabilities: it can either directly output the resulting data, or it can return the processed data encapsulated within the Model back to the App for final rendering.

3.6. API

Tesseract's API is generated dynamically from the framework's internal routing tables, providing a flexible and extensible interface. For a live demonstration of the API in action, you can visit: https://lasagna.gscloud.cz/api

4. Command Line Interface

Tesseract LASAGNA offers comprehensive control and execution capabilities via its powerful Command Line Interface (CLI). Nearly every function within the framework can be managed and operated directly from the command line. For advanced testing scenarios, the CLI even includes a built-in PHP evaluator, allowing you to run PHP code inline (though this feature is strictly for development and should not be used in a production environment).

4.1. CLI Bootstrap

To interact with Tesseract via the CLI, you can use one of the following methods:

Using the `cli.sh` wrapper script:

```
./cli.sh <command> [<parameter> ...]
```

Directly via PHP:

```
php -f Bootstrap.php <command> [<parameter> ...]
```

To view the full list of available commands and their descriptions, simply execute the `cli.sh` script without any parameters:

```
./cli.sh
```

4.2. Examples

Here are a few common examples of CLI commands in action.

Clear the application cache:

```
./cli.sh clearcache
```

Run the main application CLI presenter (e.g., for general tasks):

```
./cli.sh app
```

Execute a demonstration script Hello World:

```
./cli.sh demo
```

5. Filesystem Hierarchy

Tesseract LASAGNA organizes its components within a clear and logical directory structure. This hierarchy is designed to keep related files together, making the system easy to navigate, understand, and maintain.

Below is an overview of the main directories and their contents:

- **`apache/`**: Contains example Apache configuration files, useful for setting up your web server environment.
- **`app/`**: This directory houses the core application logic, including all Presenters and various configuration files.
- **`bin/`**: Stores essential Bash scripts that are utilized by the `Makefile` for automated tasks.
- **`ci/`**: Dedicated to logs generated during the Continuous Integration (CI) testing processes.
- **`data/`**: A secure location for sensitive or private data, such as encryption keys, raw CSV imports (which form the basis of the Model), and other non-public assets.
- **`doc/`**: Contains documentation for the framework, typically generated automatically by tools like phpDocumentor.
- **`logs/`**: The designated location for all system-generated logs, crucial for monitoring and debugging.
- **`temp/`**: A temporary storage area for various ephemeral files, including compiled Mustache templates.
- **`vendor/`**: Manages all third-party libraries and dependencies, automatically generated and managed by Composer.
- **`www/`**: This is the web root directory, containing all static assets like images, stylesheets, and JavaScript files that are publicly accessible.

6. Data Model

The heart of Tesseract LASAGNA's data management lies in its **Model**, which is structured as a flexible multi-dimensional array. This design provides a highly adaptable way to store and access application data.

You can easily inspect the Model's structure and content directly from the command line:

List Model Keys: To get a quick overview of the top-level keys within the Model, use the following command:

```
./cli.sh app 'dump(array_keys($app->getData()));' | more
```

Dump the Entire Model: For a complete view of all data stored in the Model, including nested structures, you can dump its entire contents:

```
./cli.sh app 'dump($app->getData());' | more
```

The Model's data can be accessed and manipulated using two primary methods: `getData()` and `setData()`. Both methods support a convenient **dot notation**, allowing you to navigate deep within the multi-dimensional array structure with ease.

Examples:

Accessing a specific router view:

```
./cli.sh app 'echo $app->getData("router.home.view");'
```

This command would output:
home

Retrieving a project configuration value:

```
./cli.sh app 'echo $app->getData("cfg.project")'
```

This command would output:
LASAGNA

6.1. Constants

Tesseract LASAGNA utilizes a set of framework-specific constants that define crucial aspects of its operation. You can list all defined constants by running this command:

```
./cli.sh app '$app->showConst()'
```

These constants are primarily defined within the `Bootstrap.php` and `App.php` files. For advanced customization, Tesseract allows you to override these constants directly within the `www/index.php` file, providing a flexible way to tailor the framework to specific project needs.

7. Administration

Tesseract LASAGNA incorporates a streamlined administration system designed for security and flexibility, particularly in how it handles user authentication and remote operations.

7.1. Authentication

Currently, Tesseract's user access relies exclusively on **Google OAuth 2.0 client authentication**. This robust standard ensures a secure login process.

Upon successful user login, a highly secure "master key" is generated. This key is then used to create an **encrypted cookie** using the **Halite cryptography library**, which is set exclusively via the HTTPS protocol (strict). This cookie is meticulously protected against tampering, and its parameters can be managed either through the administration panel or remotely via authenticated API calls, offering significant control and flexibility.

Important Warning: Authentication functionality is only enabled and available when the necessary OAuth parameters have been properly configured within the `config_private.neon` file.

Significantly, Tesseract maintains **no database of connections or authenticated users**. Its stateless design contributes to its scalability and simplicity. The default URL for user login is `/login`, and for logging out, it's `/logout`.

Halite is a high-level cryptographic interface that relies on the robust `libsodium` library for all its underlying operations. Developed by Paragon Initiative Enterprises, Halite aims to make cryptographic implementations in PHP safer and more straightforward.

To inspect the structure of the unencrypted master key, you can run the following command via the CLI:

```
./cli.sh app 'dump($app->getIdentity())'
```

For more detailed information about the currently logged-in user's identity, use this command:

```
./cli.sh app 'dump($app->getCurrentUser())'
```

Note: These commands will always return "XX" for the country code, as this specific information is retrieved from the Cloudflare header, not stored internally by Tesseract.

7.2. Permissions

Tesseract LASAGNA comes with three fundamental built-in permission levels, which can be easily extended to suit more complex authorization needs. These predefined levels include:

- **admin** (also known as superuser): Possesses full administrative privileges.
- **manager:** Offers a broad set of management capabilities.
- **editor:** Authorized to refresh data and edit articles.
- **tester:** Has no elevated permissions, typically used for testing purposes without affecting core data.
- **authenticated user:** Any user successfully logged in via OAuth, with basic access.

- **unauthenticated user:** Represents an unknown identity, having guest-level access.

7.3. Remote Calls

Remote calls within Tesseract are managed by the `AdminPresenter`. Administrators have the capability to generate the necessary URIs for these calls directly within the administration panel.

For automation tasks on a localhost environment, remote calls can be authenticated by including the `admin.key` content as a `?key=` parameter in a `curl` call. The `admin.key` file contains the required key and is readable by users belonging to the `root` or `www-data` system groups.

You can view the key's content using:

```
cat data/admin.key
```

8. Core Features

Tesseract LASAGNA is built with a strong emphasis on performance, maintainability, and, crucially, security. This section details some of the key features that underpin the framework's robust operation.

8.1. Security

Security is a paramount concern within Tesseract, with several built-in mechanisms designed to protect the application and its users.

8.1.1. Robots Blocking

The framework incorporates measures to identify and block malicious or unwanted web crawlers and bots, safeguarding resources and ensuring fair access for legitimate traffic. (Details on implementation, e.g., using `robots.txt` or specific HTTP checks, can be added here later).

8.1.2. GEO Blocking

Leveraging Cloudflare's capabilities, Tesseract can implement geographical blocking, restricting access to the application from specific regions to enhance security or comply with regional regulations. (More details on how this is configured, e.g., Cloudflare Workers or settings, could be added).

8.1.3. CSP Headers

To mitigate cross-site scripting (XSS) and other content injection attacks, Tesseract allows for the definition of Content Security Policy (CSP) headers. These headers can be configured within the `app/csp.neon` file, providing granular control over resource loading. The system also supports Nonce (Number Used Once) for further strengthening CSP by ensuring that only whitelisted scripts can execute.

8.1.4. Rate Limiter

To prevent abuse and brute-force attacks, Tesseract includes a rate limiting mechanism directly within its Presenters. This feature helps control the frequency of requests from individual users or IP addresses, enhancing the stability and security of the application.

8.2. DevOps

Tesseract LASAGNA is designed with DevOps practices in mind, offering tools for code quality, performance analysis, and debugging.

8.2.1. PHPStan

Code quality is ensured through integration with **PHPStan**, a static analysis tool. Developers can run comprehensive checks on the codebase using the `make stan` command:

```
make stan
```

8.2.2. Profiler

For performance monitoring and optimization, Tesseract provides profiling capabilities. Developers can inspect various runtime metrics by examining custom HTTP headers that expose execution times for different stages of the request, or by utilizing the integrated Nette Debugger.

Example custom headers that can be used for profiling:

```
X-Country: $country
X-Time-Data: $time1 ms
X-Time-Process: $time2 ms
X-Time-Run: $time3 ms
```

8.2.3. Debugging

Comprehensive debugging features are available for logged-in administrators, provided they are enabled in the `config.neon` file. Tesseract integrates **Nette Tracy**, a powerful and highly visual

debugging tool, to assist in identifying and resolving issues efficiently.

8.3. Caching

Tesseract LASAGNA employs a robust caching strategy to enhance application performance. It utilizes the flexible **CakePHP cache infrastructure**, with a preference for **Redis** when available. In scenarios where Redis is not configured or accessible, the system gracefully falls back to a **file-based caching mechanism** to ensure continued optimization.

8.4. Assets Versioning

To ensure that users always receive the latest versions of static assets (like images, CSS, and JavaScript) and to prevent browser caching issues during deployments, Tesseract automatically implements **asset versioning**. This is achieved by using a unique virtual hash that is mapped through an Apache directive.

The generated symbolic link for versioned assets can be seen by running:

```
./cli.sh app 'echo $app->getData("cdn")'
```

This command would output something similar to:

```
/cdn-assets/4790592b350262b8e1960a96a097de0af1828532
```

This versioned path can then be seamlessly integrated into Mustache templates, ensuring correct asset delivery:

```
<image src="{{ cdn }}/img/logo.png">
```

9. Extras

Tesseract LASAGNA includes several convenient extra features to simplify common development tasks.

9.1. QR Images

The framework offers a built-in capability to generate QR code images dynamically via a dedicated route. This functionality allows for easy creation of QR codes of various sizes directly from URLs.

The route structure for generating QR codes is:

`qr/[s|m|l|x:size]/[**:trailing]`

For instance, to generate a QR code for "Hello World" in a small size, you can use the following URL:
https://lasagna.gscloud.cz/qr/s/Hello%20World