General website documentation

Introduction

This project has been developed with the Laravel framework version 5.5 (LTS). Each country has its own repository branch hosted and deployed on its own server.

The administration panel has been developed with our own Laravel Backpack CRUD package fork. The repository for this forked package is hosted at Bitbucket: https://bitbucket.org/Hitachi-Aire-Acondicionado/laravel-backpack-crud.git

Requirements

Server requirements:

- PHP >= 7.3.27
- OpenSSL PHP Extension
- PDO PHP Extension
- Mbstring PHP Extension
- Tokenizer PHP Extension
- XML PHP Extension

And the following applications installed:
- composer 1.9.1
- Node.js v8.16.0 - TODO MARC: comprobar versión actual
- npm 6.1.0 - TODO MARC: comprobar versión actual
- git

Installation directions in the development environment

1. Clone the repository to a folder in the development server
2. Create a virtual host in the web server pointing to the public folder where the repository was cloned to. Example for Windows Wampserver:

```
<VirtualHost *:80>
    ServerName devel.hitachi-web.com
    DocumentRoot c:/wamp/www/hitachi-web/public/
    <Directory "c:/wamp/www/hitachi-web/public/">
        Options +Indexes +Includes +FollowSymLinks +MultiViews
        AllowOverride All
        Require local
    </Directory>
```

```
</VirtualHost>
```

3. Set up a DNS for the created vhost
4. Create a mysql database
5. Create an .env file from the .env.example file included in the repository (you can find the explanation for the variables in the .env file later in this document). Please don't change the .env.example file values.
6. From the command line, from the repository folder, run:

   composer config -g bitbucket-oauth.bitbucket.org TcwyUpPHeJnRA2tbtC matQb8CdEBEm85QYMjPJDLb78uL5LZYF
   (sets up composer with the keys to access the private Bitbucket repository to be able to install the Backpack fork)

   composer install
   (installs the necessary dependencies in the /vendor folder)

   php artisan migrate:refresh
   (creates the DB tables and fills them out with master data)

   npm install
   (installs the necessary javascript dependencies in the /node-modules folder)

   npm run dev
   (compiles css and js assets from the resources directory in the public directory)

   php artisan storage:link
   (creates a link in /public/storage pointing to /storage/app/public, which is the folder where files are stored via admin so they can be accessed via front-end)

The default user to access the administration panel is:

User: admin@test.com

Password: admin

Updates

After the project is installed, you will need to pull the remote when there are changes. After pulling changes from the remote, you should run these commands to make sure that your installation is updated:

```
composer update --lock
php artisan cache:clear
php artisan migrate
php artisan texts:load
php artisan images:load
php artisan pages:load
```

```
npm install
npm run dev
```

These scripts will update the dependencies, update the database structure, load new data and compile the assets.

IMPORTANT: If after executing these commands you observe any changes in the files composer.lock or package-lock.json, please discard them and don't commit them: you should only commit composer.lock changes if you have made changes to composer.json; you should only commit package-lock.json if you made changes to package.json. This is very important to prevent conflicts in future updates.

Configuration (.env file)

The .env file contains the application's configurable parameters. It's located in the root of the project and is not included in the repository because it contains sensitive data.

The values can be left empty (for example database password) and must be enclosed in double quotes if they contain spaces.

General configuration

`APP_NAME`: Application name. Ex: `Hitachi`

`APP_ENV`: Application environment: `local` for development environment, `production` for production environment

`APP_KEY`: key in base64 for data encryption. A new key can be generated through the command `php artisan key:generate`.

`APP_DEBUG`: show detailed error messages: `true` for development, `false` for production

`APP_LOG_LEVEL`: log level for messages in file storage/logs/laravel.log. Recommended `debug`.

`APP_URL`: full app domain url. Ex: `http://devel.hitachi-intranet.local` for development or `https://www.hitachi.com` for production. **Important: without '/' at the end.**

`DB_CONNECTION: mysql`

`DB_HOST`: IP of the database server. Ex: `127.0.0.1` or `localhost`

`DB_PORT`: port of the database server. Ex: `3306`

`DB_DATABASE`: database name. Ex: `hitachi`

DB_USERNAME: database username. Ex: root

DB_PASSWORD: database password. Ex: leave empty or root

MAIL_FROM_ADDRESS: e-mail address messages are sent from (ex. user@domain.com)

MAIL_FROM_NAME: sender's name. Ex: "Hitachi Website"

MAIL_DRIVER: smtp

MAIL_HOST: mail server address. Ex: smtp.domain.com. Mailtrap is recommended for development or testing environments.

MAIL_PORT: port of the mail server. Ex: 587

MAIL_USERNAME: e-mail user to authenticate with the email server

MAIL_PASSWORD: password to authenticate with the email server

MAIL_ENCRYPTION: encryption used by the mail server (ex. null or tls)

If the mail server has an invalid certificate (self-signed), the following values must be set up. If the certificate is valid, there is no need to set up these options:

MAIL_ALLOW_SELF_SIGNED: true

MAIL_VERIFY_PEER: false

MAIL_VERIFY_PEER_NAME: false

Storage configuration

IMAGES_STORAGE_DISK: public

IMAGES_STORAGE_WRITE_PATH: images/

IMAGES_STORAGE_READ_PATH: /storage/images/

FILES_STORAGE_DISK: private

FILES_STORAGE_WRITE_PATH: files/

FILES_STORAGE_READ_PATH: /private/files/

CUSTOM_RESOURCES_GUARD: if enabled, allows users to choose from the admin panel if resources can be downloadable without login. true to enable, false to disable.

Default language information

`DEFAULT_LOCALE`: default locale used to seed the database when the project is first created. If we load an existing database for development or testing purposes, **it's important that the language configured here is one of the languages in the database 'languages' table**. `Ex: en`

`DEFAULT_LANGUAGE_NAME`: Name of the default language as it will be shown in the language selectors. Only needed when installing the application and seeding the database for the first time. `Ex: English`

Maps and geolocation information

`GOOGLE_MAPS_API_KEY`: Gmaps API Key. It's needed to call the Google Maps API from the website. This key should have URL restrictions applied because it's visible in the public website code.

`GOOGLE_MAPS_COUNTRY`: name of the country for the Google Maps API. Ex: `"United Kingdom"`

`GOOGLE_MAPS_COUNTRY_CODE`: code of the country for the Google Maps API. Ex: `uk`

`GOOGLE_GEOLOCATION_API_KEY`: Gmaps API Key without URL restrictions. It's needed to use the Google Maps API to get the partners coordinates automatically from the admin panel. This key can't have URL restrictions applied, so it should be never made visible in public websites.

`COUNTRY_CODE`: optional. code of the country for the top menu country select, if it's different from the `GOOGLE_MAPS_COUNTRY_CODE`.

`PARTNER_LOCATOR_COUNTRY_CODE`: optional. Code of the country for the partner locator country select, if it's different from the `GOOGLE_MAPS_COUNTRY_CODE and COUNTRY_CODE`. It can be a list of codes, for the countries with multi country partner (the first code of the list will be used as the default for the country selector)

Recaptcha information

`GOOGLE_RECAPTCHA_KEY`: Google Recaptcha API Key. This key is visible in the public website code.

`GOOGLE_RECAPTCHA_SECRET`: Google Recaptcha API Secret. This key is not visible in the public website code, it's used in the backend code to check the recaptcha code generated by the Google Recaptcha API.

SASS font file to use

This code determines which of the different style options to use when compiling assets, it's used to show different fonts for the public website depending on the website primary language.

`MIX_FONT_FILE`: 1, 2 or 3.

CSV import options

Default values for CSV import (used for product technical data import)

`CSV_ENCODING`: `UTF-8` or `Windows-1252`

`CSV_DELIMITER`: "," or ";"

Distribution page

These options are needed to activate the distribution page feature. If they are not present, the option will not be available.

```
DISTRIBUTION_PAGE_FEATURE_ACTIVE: true

DISTRIBUTION_PAGE_AUTH_NAME: "GBDM Division"

DISTRIBUTION_PAGE_AUTH_MAILTO: global.brand@jci-hitachi.com

DISTRIBUTION_PAGE_DOMAIN:
```

distribution page domain url. Ex: `http://www.jci-hitachi.com`

```
DISTRIBUTION_PAGE_TITLE: "JCH Distribution Page"
```

Backup configuration

These options are needed to enable the periodic backup of the website database and assets. If they are not present, the backups will not be done. It's also needed to configure a cron job (see cron job section)

```
BACKUPS_ACTIVE: true

BACKUP_MAX_SIZE_MB: 20000

BACKUP_ONLY_DB: true

TEST_BACKUPS: false

DEVELOPER_SUPPORT_EMAIL: websupport@hitachiaircon-webresources.com
```

Production environment settings

`SESSION_SECURE_COOKIE: true` - needed in production for security reasons

`BACKPACK_LICENSE` - Laravel Backpack license key

Project file structure

The file structure is the standard for Laravel. The most important features are the following:

/app/Classes/:

It contains extensions of vendor libraries.

/app/Classes/CustomRedirector

Implementation of the class Spatie\MissingPageRedirector\Redirector\Redirector, which allows to make 301 redirects configured in the 'redirections' table.

/app/Classes/DatabaseLocalization

extension of the class Mcamara\LaravelLocalization\LaravelLocalization, which allows us to use the database as a source of the available/default languages to use, instead of the default LaraveLocalization config file.

/app/Console

It contains scripts that can be run from the command line, using the Laravel artisan command, and the scheduled tasks configuration.

/app/Console/Kernel

scheduled tasks configuration. Backup tasks are scheduled if the options are active in the .env file and the cron job is correctly configured (see Production - Scheduled tasks)

/app/Console/Commands/ImportRedirections

import a csv of 301 redirections to database.

/app/Console/Commands/LoadImagesToDB

load new images added to the file /resources/defaults/images.php to database. This command should be executed in the deploy process.

/app/Console/Commands/LoadPagesToDB

load new pages added to the file /resources/defaults/pages.php to database. This command should be executed in the deploy process.

/app/Console/Commands/LoadTextsToDB

load new texts added to the file /resources/defaults/texts.php to database. This command should be executed in the deploy process.

/app/Console/Commands/ValidateSlugs class and ValidateSlugsPages

commands used once to normalize the slugs in the database tables. Kept for reference and compatibility purposes, not needed anymore as currently the slugs are normalized when the models are saved.

/app/Exceptions/Handler

Logic to execute when there are errors during the execution of the code.

/app/Exports

implementations of Maatwebsite utilities used to export data from the database to Excel sheets.

/app/Helpers/helpers

Functions to be executed in the whole application without the need to instantiate a Class. Translate and url generation functions are here.

/app/Http/Controllers

This directory contains classes that contain the application routes endpoints as defined in the route files. These are the application entry points. See Controllers.

/app/Http/Controllers/Admin

controllers related with the admin panel. See “Administration panel: Controllers”.

/app/Http/Controllers/Auth

functions related to front-end user authentication: register, login, logout, profile update, password reset.

/app/Http/Controllers/Middleware

Code for access control, encryption and security.

- /app/Http/Controllers/Middleware/EncryptCookies class: cookies encryption.

- /app/Http/Controllers/Middleware/Frontend class: guards the routes that can only be accessed by logged in users
- /app/Http/Controllers/Middleware/LaravelLocalizationRedirectFilter class: redirects the non localized urls to the localized urls (for countries with multilanguage on)
- /app/Http/Controllers/Middleware/LocaleSessionRedirect class: saves the current locale in the user session cookie.
- /app/Http/Controllers/Middleware/Maintenance class: shows the maintenance page if the setting is enabled and the user is not logged as admin.
- /app/Http/Controllers/Middleware/RedirectIfAuthenticated class: redirects the user to the previous page they were navigating after a successful login.
- /app/Http/Controllers/Middleware/SecureHeaders class: injects headers for security purposes.
- /app/Http/Controllers/Middleware/TrimStrings class: takes care of removing the excess spaces in form introduced data.
- /app/Http/Controllers/Middleware/TrustProxies class: injects headers for security purposes.
- /app/Http/Controllers/Middleware/VerifyCsrfToken class: CSRF protection..

/app/Http/Controllers/Requests

see “Administration panel: Validation”

/app/Imports

Implementations of Maatwebsite utilities used to import data from Excel sheets to the database.

/app/Model

In this folder we can find the Model files reflecting the structure followed by the database (Page, Range, Product, News, etc). See Models section.

/app/Notifications

Messages sent by the application (notifications to administrators, welcome emails and password resets for front-end users). All notifications use the structure defined in the view resources/views/emails/email.blade.php. The HTML template and email style are located in /resources/views/vendor/mail/html

/app/Providers

It contains the service providers for the application

/app/Providers/AppServiceProvider

Common services provider for the application:

- Definition types of polymorphic relationship between the Seo Model and the "Seoable" Models.
- Customized validators (See Requests).
- Common data link for all views (view composer).
- Eloquent event capture to ensure database integrity.
- Cache purging when data is updated

/app/Providers/EventServiceProvider

Triggers the welcome email sending when a new frontend user is registered.

/app/Providers/RouteServiceProvider

Loads application routes from folder routes/ and assigns middlewares to different route groups.

/app/Traits

Common behaviours for models or controllers. See Traits Section.

/config

Configuration of different application aspects. When a value uses the env('VALUE') function, this means it can read this value from the .env file.

/database/migrations

Structure of data created when executing the artisan migrate php command. **All the database structure have to be created or updated using migrations**, if we make changes in the database structure directly in the server, is not possible to keep congruence between the development, test and production servers without having to import and export constantly big databases.

/database/seeds

This application doesn't use random content seeding or factories to load initial data. All the files in this folder are called from the migrations and introduce the "master" data that is loaded from the definitions in the folder /resources/defaults/ when installing the application.

/node_modules

Packages installed with `npm install` according to the configuration of package.json. **Important: the code in these files must not be modified**, since they are not included in the repository and they will be overwritten the next time npm install is executed.

/public

This is the website's root directory, the application's entry point and where the compiled assets, images, sources and publicly visible files are located. It also contains a symbolic link to the folder storage/app/public (created with the command `php artisan storage:link`). The compiled assets are not included in the repository because they are overwritten each time `npm run (dev/prod)` is executed.

/resources/assets

js and sass files that are compiled with Laravel Mix when `npm run (dev/prod)` is executed. The directions for Laravel Mix are in webpack.mix.js (in the project's root).

/resources/defaults

data loaded to the database when first installing the application, and which can be modified from the administration panel afterwards.

/resources/lang

Administration panel text strings. They could be translated to any language using different language files, but for now they are only in english, with the exception of the default backpack messages, that come in different languages.

/resources/views

The .blade.php files in this folder correspond to the different application views. These correspond to each of the pages shown on the front-end, except for the special view js-injection.blade, which is used to inject php variables in JavaScript, according to the setting in config/javascript.php. It contains the following folders and files:

- /resources/views/admin/sidebar_menu.blade.php: administration panel menu.
- /resources/views/admin/auth: views related to login, profile editing, and to front-end password resets.
- /resources/views/email/email.blade.php: email structure.
- /resources/views/email/email_xss_secured.blade.php: email structure protected against possible xss attacks. It must be used when sending data directly entered by a front user, to block the possibility of sending malicious code.
- /resources/views/errors: views shown for each error that can be generated by the application.
- /resources/views/exports: templates for exports from the admin panel (see exports)
- /resources/views/layouts/app.blade.php: template for pages in general
- /resources/views/layouts/landing_app.blade.php: template for distribution pages
- /resources/views/layouts/maintenance.blade.php: template for the maintenance page
- /resources/views/partials: code blocks used in different views
- /resources/views/vendor: views published from external packages overwriting those in said packages

- /resources/views/vendor/backpack: modifications in backpack views, see Administration panel custom views

/routes

Files that point the front urls to the application endpoints (controller methods).

- /routes/admin: Routes used in the administration panel. They require being logged in as admin user from the admin panel login. The url prefix is /admin. They point to methods in controllers in the /app/Http/Controllers/Admin folder.
- /routes/api: special routes used from certain controls in the administration panel to recover data swiftly from the database. They don't require authentication.
- /routes/web: routes used in the front-end. They require authentication, defined in each controller.

/storage

Location where all user data that is not storable in the database is stored. Also used for temporary files, cache, logs, etc. These files are not included in the version control.

- /storage/app/public: location where public accessible files are stored (images, videos, and wysiwyg media) loaded from the administration panel. This folder is soft linked from /public/storage.
- /storage/app/private: location where not public accessible files are stored (download files) loaded from the administration panel.
- /storage/framework: used by the system for cache, blade compilation and session storage.
- /storage/logs/laravel.log: log with all errors and debug messages generated by the application..

/vendor

Packages installed with composer install according to the configuration of composer.json. **The code in these files must not be modified**, since they are not included in the repository and they will be overwritten the next time composer install/update is executed.

Administration panel

The administration panel is built with the Laravel Backpack package, which lets us build panels based on CRUD robustly.

Backpack modifications

The reason for the Laravel Backpack fork is that we needed to develop some modifications to allow "child fields" (that is: allow us to create objects dependent on other objects directly in the admin panel editor) For example: page elements. The changes in the Backpack code are mainly to allow the "save and back" methods to go back to the parent, instead of to the crud

list. The rest of the changes to the default backpack behaviours are done with normal code overrides and extensions.

Administration controllers

Each Class in the folder /app/Http/Controllers/Admin extends the class Backpack\CRUD\app\Http\Controllers\CrudController. The administration panel controllers structure is as follows:

- App\Http\Requests\ModelNameStoreCrudRequest – request class containing validation rules for the creation of this model.
- App\Http\Requests\ModelNameUpdateCrudRequest – request class containing validation rules for the modification of this model. In most cases, it is the same as Store.
- method setup: with this method we configure the CRUD behaviour, calling the following methods of the CrudPanel object with the necessary values:
    - setModel – corresponding model.
    - setRoute – administration route it belongs to.
    - setEntityNameStrings – names to show the user.
    - setColumns – columns that will be shown in the list view.
    - addFields – fields that will be shown in the creation/edition view.
    - denyAccess – actions which are not allowed to be executed (for example, some models cannot be deleted or created, only modified).
    - removeAction – allows hiding the saving buttons if access to the corresponding actions has been denied with denyAccess.
    - addFilter – filter than can be applied to the record list.
    - enableExportButtons – show buttons to export the visible records in different formats.
    - removeButton – allows button hiding in the record list if access to the corresponding actions has been denied with denyAccess.
    - with – lets us specify relationships to load with eagerLoading, to reduce the number of requests to the database.
    - addClause – lets us add a predetermined filter to records shown in the list.
    - enableDownload: allow to download all the records of the table as excel.
    - downloadButtonText: what text to show in the download button
    - enableUpload: allow to upload new records from excel.
    - uploadButtonText: what text to show in the upload button.
- method update: called when a model is updated from the admin panel. Receives an UpdateRequest param to validate the received data. Calls the parent CrudController update method. Can have additional logic before and after the update. For example, the slug validation and child fields updating are done here.
- method store: called when a model is created from the admin panel. Receives a StoreRequest param to validate the received data.
- method create (optional): called when showing the create page. Used to add dynamic or variable fields. For example, in DownloadCrudController it's used to show or hide the optional field to make a Resource publicly downloadable, if the .env setting is set up.

- method edit(optional): called when showing the update page. Used to add dynamic or variable fields. For example, in ProductCrudController it's used to show the product options depending on the related Range configured filter options.
- method download (optional): called in the admin panels that allow the export of the data in an excel file. Needed when the option enableDownload is enabled in the crud setup.
- method upload(optional): called in the admin panels that allow the import of excel data. Needed when the option enableUpload is enabled in the crud setup.

Validation

Files in the folder /app/Http/Controllers/Requests are used to validate the data introduced in the admin panel forms, following the rules of the business. This is the structure of a validation Request:

- method rules: basic validation rules (fields that are always required, and some basic field restrictions). Custom validation rules are defined in AppServiceProvider:
    - password_complexity: passwords must contain at least 3 of the following: lowercase, uppercase, number, special character
    - image_required: in the admin panel, when there is no image in the model, a default image is shown, we need to take this into account when validating if the user uploaded a new image.
- method getValidatorInstance: allows to add dynamic validators that depend on other values of the request. For example, if a model has a switch that allows to attach a video and the user activates the switch, the video url is required.
- method withValidator: allow to add additional advanced validations that depends on other model characteristics. For example, in product validation, here we can find advanced validation of product features depending on the related range filter options.

Custom views, fields and columns

In the folder /resources/views/vendor/backpack we can find files that override the default backpack views. Most of them just override some default html styling or apply minor changes. These are the ones that change the original fields behaviour, or that add new behaviours for the admin panels to use:

- /resources/views/vendor/backpack/crud/columns/boolean_en.blade.php - show a Yes/No value instead of the localized value.
- /resources/views/vendor/backpack/crud/fields/button_download_file.blade.php - download a file (files are stored in private storage, so they can't be downloaded using the url)
- /resources/views/vendor/backpack/crud/fields/button_gmap_lookup.blade.php - find coordinates from an address or city name.
- /resources/views/vendor/backpack/crud/fields/checkbox_disabled.blade.php - field of type "checkbox" that is disabled (not updateable). This special field is needed because if we use the normal checkbox field with a disabled attribute, the database value would save as unchecked when saving the record.

- /resources/views/vendor/backpack/crud/fields/checklist_array.blade.php - used to show a list of checkboxes from a json field in the database (used for Settings - hidden sections)
- /resources/views/vendor/backpack/crud/fields/child_check.blade.php - show a checked/unchecked value in a child list.
- /resources/views/vendor/backpack/crud/fields/child_list.blade.php - child list field, with add/delete/edit/reorder buttons as defined in the field configuration.
- /resources/views/vendor/backpack/crud/fields/image.blade.php - adds the following functionalities to the default backpack image field:
    - no-image.png default, for empty images.
    - accept only png, gif, jpeg
    - allows to add an option to restrict the image max file size
- /resources/views/vendor/backpack/crud/fields/select2_from_ajax_no_null.blade.php - select2_from_ajax that does not allow to delete the selected value. Needed because the standard select2_from_ajax allows nulls if the field is nullable in the database, and in some cases we need to allow/disallow empty fields depending on other model conditions. Used in Datatable Elements.
- /resources/views/vendor/backpack/crud/fields/select2_link_models.blade.php - special select that shows linkable models, for link type fields. When the value is changed, it triggers a refresh on the field select_2_link_values.
- /resources/views/vendor/backpack/crud/fields/select2_link_values.blade.php - special select that shows the linkable data, dynamically related to the field select2_link_models selected model.
- /resources/views/vendor/backpack/crud/fields/select2_link_models.blade.php - special select that shows linkable models, for link type fields.
- /resources/views/vendor/backpack/crud/fields/select2_multiple_create.blade.php - used for tag fields, it allows to enter new values, and will create them on the database when saving.
- /resources/views/vendor/backpack/crud/fields/select2_multiple_from_array.blade.php - combination of the fields select2_multiple and select_form_array, that allow a tag-type field using values from an array.
- /resources/views/vendor/backpack/crud/fields/select2_multiple_max_6.blade.php - used for a unique need of a select 2 multiple with maximum of selectable values (used in Product, for the downloads relation)
- /resources/views/vendor/backpack/crud/fields/select2_range.blade.php - special select that shows the ranges in the database. en the value is changed, it triggers a refresh on the field select2_subrange
- /resources/views/vendor/backpack/crud/fields/select2_subrange.blade.php - special select that shows the subranges in the database, dynamically related to the field select2_subrange selected range.
- /resources/views/vendor/backpack/crud/fields/table_product_options_number.blade.php - special table to introduce the product options of numeric type, that shows a number type html field.
- /resources/views/vendor/backpack/crud/fields/tinymce_multiple.blade.php - allows possible to add different tinymce fields in the same edit panel, with different initialization options for each one.
- /resources/views/vendor/backpack/crud/fields/toggle.blade.php - radio type field that allows to hide other fields depending on the selected option.

- /resources/views/vendor/backpack/crud/fields/upload.blade.php - shows the file name (files are stored in the storage folder with random filenames, the file name that will be used for downloading is stored in another field).

Controllers

Controllers contain the methods that give an answer to the routes defined in the application.

PageController

index() method where the main logic of the application is located. Almost all routes go to this controller, which finds the corresponding page to show, loads its related data and shows it.

The most important methods are:

- frontendRestriction method – used to restrict a page to only logged in front-end users. If the user is not logged in they will be redirected to the login page.
- guestRestriction – used to restrict a page only to not logged in front-end users. If the user is logged in they will be redirected to the page that they were trying to see before logging in.
- showOneMapRestriction method – used to show a page only if the setting show_one_map is true. If it's not true it will redirect to the map selection page. Kept for compatibility purposes with the old "Where to buy" pages, deprecated in the new Partner Locator design.
- showTwoMapsRestriction method – used to show a page only if the setting show_one_map is false. If it's true it will redirect to the single map page. Kept for compatibility purposes with the old "Where to buy" pages, deprecated in the new Partner Locator design.
- getPageFromExactUrl method – find if a page exists with the exact url. It was used when slugs could have the character "/" in it (for example search/pages), it should not be needed anymore now because this is not possible with the current slug restrictions. Kept for compatibility purposes.
- getPageFromUrl method – find a page in the "tree" of url slugs. For example, with /equipment/heating/yutaki the method will follow these steps to show the product page:
    - equipment = first level page, which has in its config that its children are of type range.
    - heating = 'range' page which has in its config that its children are of type product.
    - yutaki = 'product' page.
- loadPageData method - loads the data to show on the page using the page config information.
- getSearchData method - on the search page, load search results using the passed filters.
- getResourceData method - on the resources page, load search results using the passed filters.

- getRangeProducts method - on range pages, filter the range products using the filters that the user selected.
- getRangeFilters method - on range pages, load the filters to show in the front end.
- getPartnerLocatorSearchData method - on partner locator search pages, load the data that corresponds to the current partner type.
- getPartnerLocatorData method - on partner locator results pages, load the data that corresponds to the current search.
- getPageElementsDataFromDB method - load other page elements as defined in page config.
- getPageTranslatedData method - load page translations.
- getSeoData method method - load the SEO related data.

ContactController

send method – receives contact forms, validates the data and sends it to the e-mail configured in settings contact_form_mailto, and saves it in the database.

CookiesLogController

accept method - saves the user cookies consent in the cookies log table.

DownloadController

- download method – returns a download file related to the specified resource if the user is logged in.
- download_datatable method – returns a download file related to a product technical data (datatable).
- download_innovation_brochure method – returns a brochure file related to an innovation record.
- download_innovation_resource method - returns a resource file related to an innovation record.

ImageController

thumb method - used to generate a cropped image from an image path. Deprecated in favour of ImageFields Trait getThumb method. Kept for compatibility purposes.

InnovationController

getQr method - generate a personalized QR code for an innovation Url and Google Analytics Client Id

LandingContactController

- index method – shows the distribution contact page.

- send method – receives contact forms, validates the data and sends it to the e-mail configured in the distribution page, and saves it in the database.

LandingController

- index method – shows the distribution page.
- preview method – shows the preview of the distribution page before publishing it.

NewsController

- paginate method – loads 4 more news items for the magazine page.
- newsAndProjects method - loads news and projects to show on the home page news module.

PartnerContactController

send method – receives partner contact forms, validates the data and sends it to the e-mail configured in the related state, or in settings -> partner_default_mailto, and saves it in the database.

PartnerController

- index method - returns technical services partenrs in json format for the technical service page.
- cities method - returns all the cities in json format for the partner locator pages.

ProfileController

setDefaultProfile method - used to change the default profile and save it in the session cookie when the user selects a diferent profile in the menu.

ProjectController

- paginate method – loads 4 more projects (of a category if specified in the request).

SearchController

- onSearch method – shows a list of results for quick searches in the search bar.

SitemapController

- partners method - generates a sitemap xml file with the current list of partners in the database.

SubscriptionController

subscribe method - creates a new subscription register with the user entered data.

Models

Models extend the Illuminate\Database\Eloquent\Model class, reflecting the data structure – the way they are stored in the database.
Also, models using Trait Backpack\CRUD\CrudTrait are manageable from the administration panel.
Models using Trait App\Traits\Translatable have some translatable fields, and are stored in another Model with the name ModelNameTranslation (for example: Product & ProductTranslation).

A model structure structure is as follows:

Properties

- $fillable: fields that can be batch written in the database. It is required to specify all the fields which must be editable from the administration panel.
- $translatedAttributes: translatable fields that are stored in the Translation.
- $appends: fields not stored in the database, that are calculated at the time of loading the model and must be appended to all representations of this model.
- $visible: in some models that can be transferred via Ajax, we do not want to append all the fields to that model out of security or privacy concerns. In such a case, we specify the fields we do need, and the rest will remain hidden.
- $timestamps: true by default, indicates that this table has the created_at and updated_at columns, which will be updated automatically whenever the model is created or modified. When false, it indicates that these fields do not exist in this table and, therefore, Eloquent must not try to update them.
- $rememberTokenName: name of the field which stores the token to remember the user in models using the Authenticatable trait. Since this application does not use this feature, in the user models this property is ‘’ (empty string).
- $table: by default, the table corresponding to a model has the same name of the model converted to snake_case. If this is not the case for some model, it is necessary to specify it here.
- $dates: this indicates which table fields use the date format. By default, it is created_at and updated_at. If there are more fields of this date type, they must be specified here.
- $casts: Laravel automatically converts these fields to the type specified here, which is useful for json fields.
- $fakeColumns: the CRUD package uses this property to save virtual fields as json values in a database field.

Relationships

The relationship methods specify how this model relates to other models in the database. They are Laravel standard and can be of the following types:

- hasMany: one-to-many.
- belongsTo: inverse of hasMany.
- belongsToMany: many-to-many. This type of relationship means that there is an intermediate pivot table which, by default, is named model1_model2 (in alphabetical order). If it is not this way, the table name must be specified.
- morphOne: polymorphic relationship one-to-one. Used for storing SEO data.
- morphTo: inverse of morphOne.

Accessors

Accessors allow us to add calculated fields to a model, and access them as if they were part of the corresponding database record. If they have the same name as an existing field on a table, they allow for modification of the presentation of said field before showing it to the user. They follow the nomenclature:

public function getFieldNameAttribute()

and are accessed with:

$model->field_name;

Mutators

Mutators allow field modification before saving it to the database. The nomenclature is:

public function setFieldNameAttribute($value)

and are accessed with:

$model->field_name = ‘value’;

Existing models in the application and correspondence to administration panel controllers

- App (tables: apps, app_translations; admin: AppCategoryCrudController child field): Information shown on “Apps” page. Managed from the App Category admin panel.
- AppCategory (table: app_categories, app_category_translations; admin: AppCategoryCrudController): Categories in which apps are organized. Fixed, non-manageable values: home, business, professional.
- BusinessRange (table: business_ranges, business_range_translations; admin: BusinessRangeCrudController): Ranges shown on “In your business” page.
- Contact (table: contacts; admin: ContactCrudController): Contact requests sent by front-end users.
- ContactMailto (table: contact_mailtos, contact_mailto_translations; admin: ContactMailtoCrudController): Mailto links list on Contact page.

- ContactType (table: contact_types, contact_type_translations; admin: ContactTypeCrudController): Contact types for Contact page form dropdown.
- CookiesLog (table: cookies_logs; admin: CookiesLogCrudController): cookies consents sent by front-end users.
- Country (table: countries, country_translations; admin: CountryCrudController): Countries for Contact page form dropdown
- DataTable (table: data_tables, data_table_translations; admin: ProductCrudController child field): Technical data for products. Managed from the Product admin panel.
- DataTableElement (table: data_table_elements, data_table_element_translations; admin: ProductCrudController child field): Technical data for products loaded from a csv or xls file. Managed from the Product/Datatable admin panel.
- DataTableCategory (table: data_table_categories, data_table_category_translations; admin: ProductCrudController child field): Categories technical data is organized in data tables of type “multiple”.
- Download (table: downloads, download_translations; admin: DownloadCrudController): Resources available to download.
- DownloadCategory (table: download_categories, download_category_translations; admin: DownloadCategoryCrudController): Categories downloads are organized in.
- DownloadSubCategory (table: download_subcategories, download_subcategory_translations; admin: DownloadCategoryCrudController child field): Subcategories in which downloads are organized. Managed from the Download Category admin panel.
- DownloadType (table: download_types, download_type_translations; admin: DownloadTypeCrudController): Types of downloads in which downloads are organized.
- DownloadFrontendUser (table: download_frontend_user; admin: DownloadFrontendUserCrudController): Register of which users downloaded which resources
- FrontendUser (table: frontend_users; admin: FrontendUserCrudController): Users can download resources when they register and sign in with their email and password.
- Image (table: images; admin: ImageCrudController): Modifiable front-end images.
- Innovation (table: innovations, innovation_translations; admin: InnovationCrudController): Data shown in “Innovations” page.
- InnovationAward (table: innovation_awards; admin: InnovationCrudController child field): Innovation awards related data. Managed from the Innovations admin panel.
- InnovationCompatibility (table: innovation_compatibilities, innovation_compatibility_translations; admin: InnovationCrudController child field): Innovation compatibility information. Managed from the Innovations admin panel.
- InnovationDownload (table: innovation_downloads, innovation_download_translations; admin: InnovationCrudController child field): Innovation downloads. Managed from the Innovations admin panel.
- InnovationElement (table: innovation_elements, innovation_element_translations; admin: InnovationCrudController child field): Innovation elements. Managed from the Innovations admin panel.
- InnovationElementImage (table: innovation_element_images; admin: InnovationCrudController child field): Innovation elements related images. Managed from the Innovations/Elements admin panel.

- InnovationHeaderImage (table: innovation_header_images, innovation_header_image_translations; admin: InnovationCrudController child field): Innovation header images (gallery). Managed from the Innovations admin panel.
- Landing (table: landings; admin: LandingCrudController):
  Distribution page configuration. Distribution page publishing authorizations are managed with LandingRequestAuthController
- LandingLink (table: landing_links; admin: LandingCrudController child field): Distribution page links. Managed from the Distribution page admin panel.
- LandingContact (table: landing_contacts; admin: LandingContacCrudController child field): Contact requests sent by distribution page front-end users.
- Language (table: languages; admin: LanguageCrudController): Available languages for translations.
- Legal (tables: legals, legal_translations; admin: LegalCrudController): Legal links shown in the footer.
- News (table: news, news_translations; admin: NewsCrudController): Magazine news.
- NewsCategory (table: news_categories, news_category_translations; admin: NewsCategoryCrudController): Categories in which news are organized.
- Office (table: offices; admin: OfficeCrudController): Offices listed in the “About” section.
- Page (table: pages, pages_translations; admin: PageCrudController): Every page shown in the front-end.

  This application works as a CMS. Each page has a config json field that stores:
  - From which Model it loads its main data (optional).
  - What view file it uses.
  - Information about secondary elements to load (optional).
  - What page is its “child” page (optional).
  - Optional access restrictions (for example only logged in users can download resources, and only NOT logged in users can login or register).
  - Information about other data used in this page, saved in the data and t_data fields.

  The PageTranslation Model has a t_data field that stores translatable texts used in this page.

- PageElement (table: page_elements, page_element_translations: admin PageCrudController child field): Child elements managed from the page admin panel. Only some pages can have elements: home (can have the elements: header, products, business_solutions, lets_talk, news, features), apps (header), innovations (header).
- PageElementImage (table: page_element_images, page_element_image_translations; admin PageCrudController child field): Page elements related images.
- Partner (table: partners, partner_translations; admin PartnerCrudController): Partners listed in the partner results pages.
- PartnerCity (table: partner_cities; admin PartnerCityCrudController): Partners cities listed in the partner search and result pages.

- PartnerContact (table: partner_contacts; admin: PartnerContactCrudController): Contact requests sent by front-end users using the partner contact form.
- PartnerState (table: partner_states; admin PartnerStateCrudController): Partners states listed in the partner search and result pages.
- PartnerTechservice (table: partner_techservices; admin PartnerTechserviceCrudController): Partners shown in the technical service map.
- PartnerType (table: partner_types, partner_type_translations; admin: PartnerTypeCrudController): Partner types (fixed values)
- PartnerSubType (table: partner_subtypes, partner_subtype_translations; admin: PartnerSubtypeCrudController): Partner subtypes (fixed values)
- Product (table: products, product_translations; admin: ProductCrudController): Products. They can be linked to several business ranges. They can have related downloads.
- ProductElement (table: product_elements, product_element_translations; admin: ProductCrudController child field): Product elements. Managed from the Products admin panel.
- ProductElementImage (table: product_element_images, product_element_image_translations; admin: ProductCrudController child field): Product elements related images. Managed from the Product/Elements admin panel.
- ProductOption (table: product_options; admin: ProductCrudController field): product options as defined by the product related Range. Managed from the Product admin panel.
- ProfileType (table: profile_types, profile_type_translations; admin: ProfileTypeCrudController): Profile types for Register Professional page form dropdown.
- Project (table: projects, project_translations; admin: ProjectCrudController): Projects listed on Project page.
- ProjectCategory (table: project_categories, project_category_translations; admin: ProjectCategoryCrudController): Categories in which projects are organized.
- Range (table: ranges, range_translations; admin: RangeCrudController): Ranges shown on “Equipment” page.
- RangeFilter (table: range_filters, range_filter_translations; admin: RangeCrudController child field): filters configured for a range and its products. Managed from the Range admin panel.
- RangeFilterOption (table: range_filter_options, range_filter_option_translations; admin: RangeCrudController child field): possible options for the range filters of type checkbox. Managed from the Range admin panel.
- Redirection (table: redirections; not manageable, only configurable from database)
- Seo (table: seos, seo_translations): SEO data is managed from each Seoable Model Seo Tab.
- Setting (table: settings; admin: SettingCrudController): Site settings.
- Social (tables: socials, social_translations; admin: SocialCrudController): Social networks links.
- State (table: states, state_translations; admin: StateCrudController): States/provinces for Register page form dropdown.
- Subrange (table: subranges, subrange_translations; admin: RangeCrudController child field): Range subranges. Managed from the Range admin panel.

- Subscription (table: subscriptions; admin: SubscriptionCrudController): newsletter subscription request by front-end users.
- Tag (table: tags; admin: TagCrudController): Tags used for news articles and downloads.
- Text (table: texts, text_translations; admin: TextCrudController): Modifiable front-end texts.
- User (table: users; admin: UserCrudController): Admin users with access to the administration panel with their email and password.
- VisitorCountry (table: visitor_countries, visitor_country_translations; admin: VisitorCountryCrudController): countries shown in the header country dropdown selector. Only translatable, can't create or delete from the admin panel.
- VisitorProfile (table: visitor_profiles; admin: VisitorProfileCrudController): possible profiles used to show different menus and home pages. Fixed values: residentials, business, professionals, manufacturers. Not manageable.

Traits

Traits contain functionalities that can be used in several Models or Controllers.

- /app/Traits/AdminChildFields: Trait for Crud Controllers. Methods for updating child fields. Can be called from Crud Controllers, after creating or updating the model, so the child vales are correctly updated after creation/reorder/deleting/updating (depending on the child field options)
- /app/Traits/AdminFileFields: Trait for Crud Controllers. Method to visualize/download a file attached to a model from the admin panel (files are stored in private storage, so they can't be download using the url)
- /app/Traits/AdminLinkFields: Trait for Crud Controllers. Methods to manage special "link" fields in admin panels. Link fields are composed fields that can be internal or external.
- /app/Traits/AdminOverrideNoindexFields: Trait for Crud Controllers. Method to add dynamic "override noindex" fields in admin panels. These fields depend on variable partner types and subtypes.
- /app/Traits/AdminSeoFields: Trait for Crud Controllers. Method to add a tab with SEO fields to any admin panel.
- /app/Traits/AdminSluggable: Trait for Crud Controllers. Methods to validate and normalize slugs when creating or updating a model from the admin panel.
- /app/Traits/AdminVideoField: Trait for Crud Controllers. Methods to manage different types of video fields in admin panels.
- /app/Traits/FileField: Trait for Models. Methods that take care of saving, deleting and retrieving files from the file system.
- /app/Traits/ImageFields: Trait for Models. Methods that take care of saving, deleting and retrieving images and videos from the file system, and creating and retrieving thumbnails from images. A max height or max width can be passed to the saveImage method and the image will be resized to that value, maintaining the original image aspect ratio.
- /app/Traits/Linkable: Trait for Models. Retrieve link fields and convert them to easy to use attributes for the frontend.

- /app/Traits/Seoable: Trait for Models. Update and retrieve related SEO data.
- /app/Traits/Sluggable: Trait for Models. Extension for library CviebrockSluggable, creates unique slugs from the Model title.
- /app/Traits/Sortable: Trait for Models. Allows to use a single field for sortable models instead of the 4 needed fields in standard Laravel Sortable Trait.
- /app/Traits/Translatable: Trait for Models. Extension for library DimsavTranslatable that allows the models to have translatable fields.

Authentication

Authentication configuration is in /config/auth.php.

Authentication for the administration panel uses the Laravel default authentication (guard web, provider users, table users), and the Admin middleware which comes with the administration package from Laravel Backpack.
To let front-end users have independent access, a guard front-end has been set up, as well as a front-end provider, frontend_users table and applying the Frontend middleware.
The authentication logic for front-end users is located at /app/Http/Controllers/Auth.

Front-end

On top of the blade templates, the front-end logic is supported by the JavaScript code, using the jQuery and Vue.js frameworks. All JavaScript code used by the application is located at /resources/assets/js. The JavaScript application entry is app.js, which takes care of loading all necessary libraries and building the application structure. There is a Vue component:

SearchBar.vue – quick search bar, called from the partial header.blade.

TODO MARC: document the new vue components

The jQuery code is in main.js.

The assets for the distribution page are located at /resources/dist_assets

Production

Deployment

To deploy changes in the code, the changes must be committed and pushed to the server master branch (generally called countrycode-master). The servers are configured so the changes in this master branch are automatically deployed to the site httpdocs folder.

The script deployment must execute the following commands after checking out the repository code (this script is optimized for the current servers, in a Plesk environment, it's possible that it have to be adapted for different environments). This script generates a

deploy.log file, in the httpdocs folder of the server, with the output of all the commands executed, that can be used to solve deployment problems.

```
echo "Executing additional deploy actions" > deploy.log 2>&1
/usr/bin/composer -V >> deploy.log 2>&1
/usr/bin/php -v >> deploy.log 2>&1
/usr/bin/npm -v >> deploy.log 2>&1
/usr/bin/node -v >> deploy.log 2>&1
echo "Running composer install..." >> deploy.log 2>&1
/usr/bin/composer config -g bitbucket-oauth.bitbucket.org
TcwyUpPHeJnRA2tbtC matQb8CdEBEm85QYMjPJDLb78uL5LZYF
/usr/bin/composer install --no-interaction --prefer-dist
--optimize-autoloader >> deploy.log 2>&1
echo "Clearing cache..." >> deploy.log 2>&1
/usr/bin/php artisan cache:clear >> deploy.log 2>&1
echo "Running migrations..." >> deploy.log 2>&1
/usr/bin/php artisan migrate --force >> deploy.log 2>&1
echo "Loading new texts..." >> deploy.log 2>&1
/usr/bin/php artisan texts:load >> deploy.log 2>&1
echo "Loading new images..." >> deploy.log 2>&1
/usr/bin/php artisan images:load >> deploy.log 2>&1
echo "Loading new pages..." >> deploy.log 2>&1
/usr/bin/php artisan pages:load >> deploy.log 2>&1
echo "running npm prune..." >> deploy.log 2>&1
/usr/bin/npm prune --no-progress >> deploy.log 2>&1
echo "running npm install..." >> deploy.log 2>&1
/usr/bin/npm install --no-progress >> deploy.log 2>&1
echo "running npm run prod in background..." >> deploy.log 2>&1
/usr/bin/npm run --quiet prod >> deploy.log 2>&1
echo "Deploy finished " >> deploy.log 2>&1
```

This script:

- installs libraries using composer, according to the configuration of composer.json
- clears the server cache
- runs new migrations that will update the database structure
- loads new texts, images and pages into the database
- installs libraries using npm, according to the configuration of package.json
- compiles and minifies the css and js assets

Scheduled tasks (cron)

A cron job must be configured in the server so the scheduled tasks run:

```
* * * * * php /path-to-your-project/artisan schedule:run >>
/dev/null 2>&1
```

See Laravel documentation https://laravel.com/docs/5.5/scheduling for more information.

Developer mode

Developer mode is deprecated and will be removed in a future update. Developers must directly modify the database using migrations (recommended) or phpmyadmin to make modifications that are restricted in the admin panel (for example: creating or deleting pages, or manipulating a Page or Page elements config).

Scheduled Backups

Introduction

To ensure control over the backups, and independently of the backups provided by the hosting provider, each server has been configured to run the package Spatie/laravel-backup (source https://github.com/spatie/laravel-backup).

Features

This package has been configured to create a backup of the website content. The backup is a Zip file that contains all files in storage/app/public along with a dump of the database. The backup is stored in a folder named backups - automatically created, if it doesn't exist - at the same level than the website installation folder (usually named httpdocs or www). The package will notify via email to DEVELOPER_SUPPORT_EMAIL if something goes wrong while performing a backup.

In addition to making the backup, the package also cleans up old backups, monitors the health of the backups, and can show an overview of all backups.

Every day at 3:00 AM local time the following actions are performed:

- Cleanup old backups
- Create new backup
- Send an alert if the backup finished with errors.

Cleanup policy

Backups will be deleted according to this set of rules:

- Number of days for which backups must be kept: 3
- Number of days for which daily backups must be kept: 3
- Number of weeks for which one weekly backup must be kept: 1
- Number of months for which one monthly backup must be kept: 1
- Number of years for which one yearly backup must be kept: 1

Configuration

These parameters are set in the .env file located in the root public web folder. Here default values are shown.

BACKUPS_ACTIVE=true Whether scheduled backups should be activated or not.
BACKUP_MAX_SIZE_MB=2000 Maximum size (MB) assigned to all the backups.
BACKUP_ONLY_DB=true true: db+storage. false: everything (including code)
TEST_BACKUPS=false true: will change the default frequency of backups to every minute!
DEVELOPER_SUPPORT_EMAIL=email@example.com Support email where alerts will be sent to.

Other configuration variables used:
MAIL_FROM_NAME Name and email used to send alerts.
MAIL_FROM_ADDRESS

The server must run the following crontab process:
* * * * * php WEBROOT/artisan schedule:run >> /dev/null 2>&1

Server Backups

Currently, complete, incremental server backups are made for every country.

Incremental backups are made daily, and complete backups are made three times a week.

In addition this, server snapshots are made twice a day and kept for 48 hours. A snapshot is an image of the server state at a particular moment. It is not a full backup.

JCH Distribution Page

Introduction

Some Hitachi websites may require an alternate domain name, which will show a page with a distributors grid to choose from. This feature has been integrated in the general CMS.

Features

This feature will be only available for authorized websites, and it will be activated or deactivated only at server level (not via CMS).

- Activation is done via the DISTRIBUTION_PAGE_FEATURE_ACTIVE parameter (inside the .env file located in the website root folder, usually httpdocs).
- Integrated with the CMS, thus sharing the DB and the backend.
- The user will be able to create or modify a page.
- The user will not be able to directly publish a page.

- The user will be able to preview the page and send a publishing request to an established authorizer email (see parameter DISTRIBUTION_PAGE_AUTH_MAILTO in .env file).
- This page's frontend will be accessible only from an alternate domain name (DISTRIBUTION_PAGE_DOMAIN in .env file).
- All texts are configurable in the main language (not in multi-language).
- The home page will admit 2 to 6 elements, including: image, text, link, and a hidden title for the alt-image html field.
- Contact form.
- Optional menu item.
- Editable SEO data and Google Analytics tracking code.

Resources

Database

The migrations create_landing_table and create_landing_contact create and seed the necessary tables to manage this feature. To keep this functionality isolated from the general CMS, none of the existing PagesController logic (see the general code guide) has been reused. All fields are stored in, and only in, the tables landings, landing_links, and landing_contacts.

Project files

- Frontend: custom blade views and layouts have been used.
- Frontend: separate stylesheets and JS have been used (webpack.mix.js).
- Backend: custom backpack component has been created to show the current authorization and publishing status, preview button, and request button (button_landing_auth_request.blade.php).
- Backend: the class defined in LandingRequestAuthNotification.php is responsible for sending the email with the request to publish.
- Front and backend: usual files under Controllers and Controllers/Admin.

Other resources

- CME administrative texts are stored in /resources/lang/en/admin_distpage. These texts are editable only by developers. They include, among others, the text of the authorization email.
- Non-sensitive texts are stored in /resources/defaults/texts.php and they are accessible via the CMS-Texts menu, under the categories section_dist_* . These texts will be editable with no effect over the page publishing status. These texts can be translated to multiple languages, although this feature was not requested. The reason is that they use the CMS translatable feature included in this section (CMS-Texts).
- Any other text is stored in the DB tables. Any modification to these texts will require a new publishing request submission and will be published after approval.

Authorization and preview logic

When a user submits a request to publish a page a unique token is generated. This token is encrypted and stored into the DB. The authorizer will receive an email with a link including this token. The link will lead to a preview page with the buttons Publish and Reject. This preview page passes through the CMS to ensure the user is authenticated, even if they have the correct token.

See the model class Landing.php to view the specific authorization process as well as the publishing cases and logic. Also, web.php and admin.php contain important information about routes definitions.

Server configuration

.env file located in the root public web folder will include these parameters:

DISTRIBUTION_PAGE_FEATURE_ACTIVE=true
DISTRIBUTION_PAGE_AUTH_MAILTO=mailto@authorizer.example
DISTRIBUTION_PAGE_DOMAIN=distribution.domain.example
DISTRIBUTION_PAGE_TITLE=”JCH Distribution Page”

The DISTRIBUTION_PAGE_DOMAIN parameter must not include the http protocol (http:// or https://).
The DISTRIBUTION_PAGE_TITLE contains the page title to be shown in emails.

The distribution domain name has to be configured at server level as a domain alias pointing to the same IP than the main domain. This feature has been tested with domains inside the same server (DNS from main.domain and distribution.domain are the same). Tests with different DNS have not been performed.

Best practices and recommendations

This code is modified by different developing teams in different countries. The code base is constantly updated and merges from the master branch to your country-master branch can be very difficult. So please follow this conventions to avoid conflicts:

- Comments, commit messages, variable names, etc should be in english
- Indentation for new code should be 4 spaces (no tabs)
- Try to modify just the minimal lines of code that you need, don’t do refactors of others code.
- Disable automatic code linters. Don’t change indentations or code style, specially if it’s code that you don’t need to modify
- Don’t commit package.lock and composer.lock changes if you didn’t intend to modify it. Rule of thumb is: if you didn’t modify composer.json, don’t commit composer.lock changes (and the same is valid for package.json and package-lock.json)

- All database changes must be done using migrations, don’t update the production database directly!