# Crowdfunding Toolbox - developer guide

Currently in beta version.

## Package

Project (package) which is available on GitHub is a monorepo which consists of two apps.

- Backend app - folder **backend** (is developed in Laravel 5.8)
- Frontend app - folder **dashboard-app** (is developed in Angular 7.3)

## Configuration and installation

Please for running application on localhost follow these steps:

- make sure you have installed some local server (like XAMPP) with included PHP
- make sure you have installed composer
- make sure you have installed node.js
- clone project from github: https://github.com/cft-postoj/crowdfundingtoolbox.git
  - you can divide project to separate folders (backend – Laravel app and dashboard-app – Angular app).
- setup your database
  - it must be standard supported Laravel database
  - we recommend to use PostgreSQL (during the develop we are using PostgreSQL 10.0)
- setup your .env file (in root of backend folder)
  - please copy all content from .env.example and replace it with your variables (follow comments in .env.example)
- run **composer install** on backend folder
- run **php artisan migrate**
  - this will make 51 tables, which have relationships each other
  - if you want to add our dummy data to your database, please run this command: **php artisan db:seed --class=TestDashboardSeeder**
- run **npm install**

- in dashboard-app folder run **npm install**

Now you can run local servers

- for backend **php artisan serve**)
- for dashboard-app **ng serve**
- you can sign in with default login:
  - **username: admin**
  - **password: test123**

If you want to deploy application, please follow these steps:

- copy all content from backend app to your webhosting
- rename server.php to index.php in root directory
- use our .htaccess file

- in dashboard app, change apiUrl constant in environment.prod.ts file
- in dashboard-app run **npm build prod**
  - it'll create prod build which will be situated in dist folder (to replace dist from dashboard-app to root, please install `copyfiles` globally (`npm install -g copyfiles`))
- copy content of dist folder to your hosting (for example map it in some subdomain, it is up to you)
  - please, copy content include .htaccess file for correct Angular routing

**Connect application with your portal/website:**

Please update your .env file in root of backend app to set your portal URL here.

All you need you can find in dashboard-app (if you run it, you can find it in route*/dashboard/configuration/(right:portal-connections)*

Portal script and styles are consist of widgets requests and user section (login, register, my account).

You need to add Crowdfunding style to header of your page (before closing head tag) and script (before closing body tag).

Next you need to add widget placeholders to your portal templates and create place for register and my account pages.

Here is list of placeholders:

**Landing**

```
<div id="cr0wdfundingToolbox-landing"></div>
```

**Sidebar**

```
<div id="cr0wdfundingToolbox-sidebar"></div>
```

**Leaderboard**

```
<div id="cr0wdfundingToolbox-leaderboard"></div>
```

**Pop-up**

```
<div id="cr0wdfundingToolbox-popup"></div>
```

**Fixed**

```
<div id="cr0wdFundingToolbox-fixed"></div>
```

**Locked**

```
<div id="cr0wdfundingToolbox-locked"></div>
```

**Article**

```
<div id="cr0wdfundingToolbox-article"></div>
```

#### Custom

```
<div id="cr0wdfundingToolbox-custom"></div>
```

### Modules

Laravel app is consists of default settings and migration/seeders in app folder.

Whole logic are in modules (we used nwidart/laravel-modules in version 4.1).

Every module is consists of entities, controllers, services and repositories, so in entities there are definitions of models. Controllers are only for communicating between API and service.

In services there is only business logic and repository calls. Repositories manage DB layer.

We also developed same modules in Angular app, which are communicating with Laravel app.

#### UserManagement module

This module manage users in application. There is mutual table for all users - **users**. In users table there are portal and backoffice users who are authenticated with JWT.

Backoffice users are from portal users divided with group in API layer. Every backoffice user has record in **backoffice_users** table with one of role (administrator or manager).

Manager don't know delete campaigns, manage settings and create new backoffice user. Portal user has referrence in **portal_users** table.

Every backoffice user can be portal user too.

- **user_details** table manage basic user detail (first name, last name, street, city etc.)
- **user_gdpr** table manage if user agree with general conditions or newsletter sending
- **user_payment_options** table manage payment options and details of portal user

Every backoffice user has access to dashboard-app and every portal user has access to portal user section.

UserManagement module manage also emails.

Backoffice user can get one type of email (if some admin create new account).

Portal user can get 4 types of email templates (registration, forgotten password, donation, success payment paired).

These email templates you can find in Laravel app  resources/views/emails and email controller in UserManagement  Emails.

#### Campaigns

This module manage campaigns and all widgets. Every campaign is in creation initialized with 8 default types of widgets:

- Landing page widget (it is default monetization widget)
- Sidebar widget (default dimenstions set in 300x600)
- Leaderboard widget (full container width widget)
- Popup widget
- Fixed widget (like cookie bar)
- Article widget (some text or whatever...)
- Locked article widget (widget hide half of content of article. It come with button read more without donating)
- Custom HTML (custom html widget with inline css styles)

Except for landing page widget, monetization should have sidebar, leaderboard or popup widget.

During the create/edit widgets there are created widget results (in HTML format) in three types (desktop, tablet and mobile).

This result is displayed on portal side.

#### Preview - used css selectors

After created campaign and also after edit single widget, their html response is updated. Results are store in table widget_results. Every single column contains HTML markup, sometimes even with CSS or javascripts. To correct behavior some widget (especially monetization widgets), there must be used some classes or id, that indicates elements - e.q - class `cft--monatization--only-monthly` identifies elements, that are show only if monthly frequency of donations is active. Please keep these classes to detect elements. During creating of new functionality, please create some  self-descriptive classes. To show what classes are used in this application, here is list of used classnames and its purpose:

**`cr0wdfundingToolbox-`** - All placeholders must start with `cr0wdfundingToolbox-` otherwise custom css won't work correctly. To create new placeholders, keep this naming convenction or prepare to change some functionalities in monetization widgets. All styles in all widgets have inline styles or place css style element, where every selector starts with [id^=cr0wdfundingToolbox]. Please be careful and when you use some new styles, don't forget that you want to modify only elements inside widget and don't override css of page, where is widget used. Our recommendation is to use inline style. To generate inline style, you can use Angular directive NgStyle and place function in .ts, that will return styles. You can see a lot of usage, e.q. in preview.component.html. Those style also can be modified by widget settings. This is recommended approach.

**`btn-cr0wdfunding--continueReading`** - class that has to be used in locked article widget to open hidden content

**`cr0wdWidgetContent--closeWidget`** - class that indicates, that inserted widget can be closed by clicking on this element

**`cft__redirect-to-my-account`** - class that indicates, that anchor tag should redirect to 'My profile'

**`cft--monetization–container`** - class that contains monetization in widgets. This classname is crucial to monetization functions due to javascripts, that are looking for this class to get parent of monetization

**`crowdWidgetContent-preview`** - class that indicates, that current widget is opened in Angular app (this placeholder is not in widget results, only in widget edit preview)

**`cft--monatization--donation-button–monthly`** - class that indicates, that element has input inside, that holds value, that can be used in monthly donation

**`cft--monatization--donation-button--one-time`** - class that indicates, that element has input inside, that holds value, that can be used in one time donation

**`cft--monatization--only-monthly`** - class that indicates, that element is visible only when monthly donation frequency is picked

**`cft--monatization--only-one-time`** -  class that indicates, that element is visible only when one time donation frequency is picked

**`cft--monatization–hidden`** - class used to hide content. Remove this class to show hidden element

**`cft--monetization--container-step-1`** - class, indicates that element in monetization widget is showed only in first step - when user is filling amount, email, donation frequency and in monetization-lite component payment method

**`cft--monetization--container-step-2`** - class, indicates that element in monetization widget is showed only in second step - when user obtain his payment data - IBAN and variable symbol or payBySquare

**`cft--monetization--container-step-3`** - class, indicates that element in monetization widget is showed only in second step - when user see thank you title and redirect to his account

**`active`** - class, indicates that element is active - change not only color but can also used to get value into request

**`cft--monatization--membership-checkbox`** - class, indicates that element should have also class active only if donation is more (or equal) than minimum donation to get benefit. Some other css styles are also binded to this class, so make sure to change them too if you want to modify this class

**`payment-option__show`** - wrapper of payment options. With data attribute help to hide hide/show content. Current list of data-id : `bank_transfer=1` , `pay_by_square=3`

**`bank-button__wrapper`** - wrapper of bank buttons (used in bank transfer to redirect to your bank)

**`pay-by-square__wrapper`** - wrapper of pay by square (used in one time donation)

All other classname should have enough self descriptive name and their modification should not be critical, but is not recommended to change them. Information about previously mentioned selectors should be used to better undestand their usage and should be inspiration to create new selectors that also will be well described.

**Payment**

This module manage all donations and payments. User can create one-time donation or monthly donation (like encashement).

*User story:*

User see some of widgets on website. He click on widget which it redirected him to monetization widget.

He makes donation via monetization widget. At first donation is created - it records data with portal_user_id, widget_id, referral_widget_id, amount and payment_method.

He also gets email about donation.

During the donation there are get payment data from backoffice and for one time payment method there is also generated QR code for Pay by square method.

For QR code we used Laravel library peterbodnar.com/bsqr in version 1.1.

We also made functionality for pairing payments. Actually you can do it manually, via import csv file in prescribed format.

Format of csv file you can find in Laravel application: PaymentService.php  method **importPayments()**

**Statistics**

This module manage analytics data of users, donations and campaigns.

Statistics are get on DB layer with joined selects.

You can check this logic in Statistics  Repositories.

**Targeting**

In campaign creation you can set targeting on who you want to target your campaign.

Targeting is calculating from all actual portal users. Portal script which get widgets call calculating via user cookies or portal user_id.

Here are categories of targeting:

- signed / not signed user
- registration before / after
- count of read articles
- last donation before / after
- targeting for spcifically urls

## Security

- standard Laravel/Angular apps security
- JWT authentication
- Make CORS secure (disable all requests from domain which is out of .env scope).

## License

MIT license