Crowd2Fund

Tech Roadmap 2018

1st December 2017 Version 1.0.0
14th December 2017 Version 1.0.1

DRAFT

Contents

Introduction

This is the Tech Roadmap for 2018. This document contains the most important goals that Crowd2Fund should achieve this 2018, in order to provide a great service to all our customers – including both business owners and investors – but not limited to them.

Our tech team expects has designed this Tech Roadmap keeping in mind all needs on both cases, so Crowd2Fund team members will improve the user experience using our own system too, working faster and more efficiently, with all these upgrades.

This roadmap contains all requirements for all major areas that tech team needs to cover, including but not limited to:

- Our current Crowd2Fund's website and API
- Our Investors iOS App that reinvented the way to deal with the platform
- All third-party integrations, but specially focused on our new Crowd2Payments project, to provide a clean but powerful payment system
- Internationalisation of Crowd2Fund's services, to allow us to work on the new 2 markets that we have planned to expand
- New Investor Community to improve the user experience on Investor's side
- New features to improve both Business and Investor user experience
- New Android App available at Google Play store

Scope of work (projects)

Crowd2Fund's Tech Team roadmap for the 2018 will cover all these potential projects – please note more untracked features could be included as part of the Roadmap 2018:

Crowd2Fund project (C2F)

Current source code manages both Frontend and Backend environments in a single project that has both environments unified. Also, Backend code manages 2 projects – Website and API – causing issues and unnecessary delays to provide different data on both cases.

One of our goals for 2018 Roadmap is to split the C2F project into 2 isolated projects – Frontend (C2F-FE) and Backend (C2F-BE) – that will improve performance, maintainability and security.

Internationalisation

As per our plans to establish new markets on USA and Singapore, our platform should allow to manage some customised pages and content for each country. Most important changes are related with currencies, campaigns and payments.

New C2F project features

Several investors have requested new features for the Exchange. These changes could improve the user experience, and should be applied.

Messaging System to send (PM) Private Messages between 2 users was not fully built, at the current stage just the Business Owner can send a message to a group of Investors. We should work on this.

Investor Community pages should be built in order to enhance the user experience. Activity Feed should link to the new community pages.

Other minor enhancements and suggestions reported by our users could provide a better user experience for both Business Owners and Investors.

Crowd2Payments (C2P)

Now that we have been regulated to hold customer's money, we need to build the new payment system to replace the MangoPay integration.

This integration needs to allow to manage users, addresses, contact details, IDs and KYC approval processes, payments through card, bank and direct debit, anti-fraud features, etcetera.

The payment system should be split into 2 isolated projects – Frontend (C2P-FE) and Backend (C2P-BE) – in the same way we need to split the C2F project.

Mobile Apps

As part of the expected C2F-BE upgrades, the iOS App should be upgraded in order to use the new and simplified API endpoints that we will need to build. Current v1 API used by the App has already some deprecated code due all changes that we have needed to implement.

Objectives

The main goals for the scope of work described above are:

Crowd2Fund Frontend (C2F-FE)

Goals to achieve:

- Provide a new, clean, refreshed design for visitors, Businesses and Investors
- Upgrade from AngularJS to Angular v4 due SEO and performance reasons
- Improved performance due we can cache HTML content (Varnish)
- Improved performance due no need to run PHP code
- No downtimes on FE deployments

Crowd2Fund Backend (C2F-BE)

Goals to achieve:

- Fix deprecated code and upgrade to Symfony v3.4 LTS (long Term Support)
- Migrate (legacy) v1 API to an up-to-date v2 API
- Migrate website processed to the v2 API, in order to:
    - Decrease maintenance cost (time and money)
    - Remove duplicated code
    - Increase reusability
    - Increase performance
- Build more automated tests to avoid unexpected behaviours
- Improve deployment process to reduce the downtime
- Review and improve performance / cache content (when possible)
- Fix potential issues reported by the system

Internationalisation

Goals to achieve:

- Update our project to manage multiple countries, providing the expected content on every request, depending on each market.
- Integrate new payment system, managing multiple currencies.
- Other minor updates that we should apply for each country

New C2F project features

Goals to achieve:

- Exchange:
  - Redesign the Exchange filters to improve the user experience
  - Provide new Exchange filters to load just Exchange items that are between a given min/max values for Sale price, Original APR and Offered APR
  - Re-build the Exchange page to load content via AJAX calls, changing the Exchange URL on each new request (unique URLs)
- Messaging System:
  - Provide the new features to reply messages
  - Redesign Messaging System due now is too complex
- Investors Community:
  - Investors Community page for public profile investors
  - Build an Investor Forum to allow discussions among investors (new C2F Team Member – moderator – role might be required)
- Enhancements:
  - Send reminders to the Business Owner about unanswered questions

Crowd2Payments Frontend (C2P-FE)

Goals to achieve:

- Provide a clean & powerful Dashboard to interact with the C2P-BE project

Crowd2Payments Backend (C2P-BE)

Goals to achieve:

- Build this project based on Symfony v3.4 LTS or Symfony v4
- Provides a powerful up-to-date API to replace MangoPay integration
- Provide unique administration API endpoints to manage the system

Crowd2Payments PHP SDK (C2P-PHP-SDK)

Goals to achieve:

- Provides an easy way to interact with the C2F-BE API endpoints
- Allow third party developers to use the PHP SDK (Tech Team suggest to not convert Crowd2Fund into a payment platform like MangoPay, although this could be a potential case to get more revenue and to make sure the project itself is useful and profitable)

iOS Mobile App

Goals to achieve:

- Migrate to C2F API v2 and remove legacy code
- Enhance iOS App performance
- Provide new features, such:
    - Manage cards
    - Manage bank accounts
    - Manage Smart-Invest profiles
- Other potential changes based on upcoming changes

Android Mobile App

Goals to achieve:

- Provide a new Android App for all Android, aligned with the iOS App

Architecture design

Object-Oriented Design

Our project is based on a PHP Framework – Symfony – who deals with all functionalities through an OOP (Object-Orientated Programming). This allows us to create modular, flexible, and reusable software, by applying object-oriented design principles and guidelines.

Design Patterns

In software engineering, a design pattern is a general repeatable solution to a commonly occurring problem in software design. A design pattern isn't a finished design that can be transformed directly into code. It is a description or template for how to solve a problem that can be used in many different situations.

We already use several features included natively on Symfony or any of the 3rd party bundles that we use in order to deal with these processes, however we have not implemented yet the Design Patterns as part of our day-to-day development process.

Software Architecture

The way that software components (subroutines, classes, functions, etc.) are arranged, and the interactions between them, is called architecture.

Given our project is based on Symfony, our source code is already implementing the expected interactions between all of them through Symfony services, that allows us to reuse any piece of code we want.

As part of our roadmap, we have planned to extend our current OOP to implement Design Patterns and allow to reuse almost everything we have, resulting in a less time consuming as well as a more efficient way to develop new features and to improve our code quality.

Server architecture

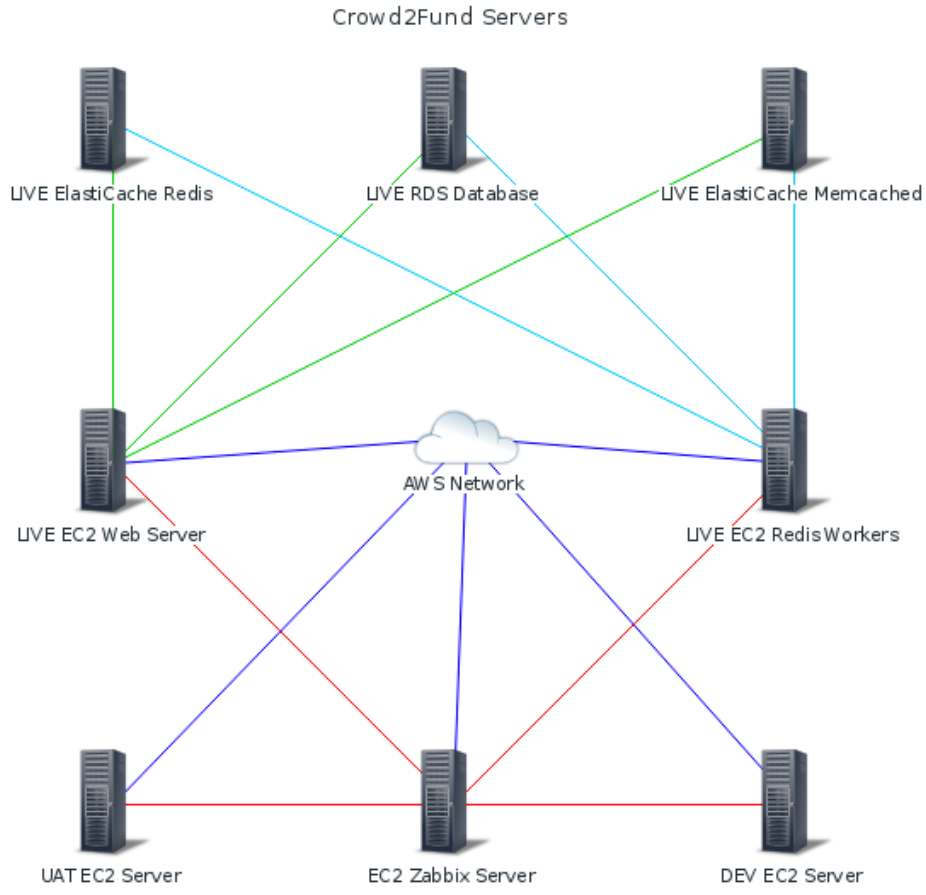
Server provider

Our servers are provided by and hosted on AWS (Amazon Web Services). They have been providing a great service so far, and there is no reason to change to another provider. Also, we are using other AWS services and we have some integrations too between other AWS products, like S3 Buckets.

David – our Tech Dev Director – has been in charge and taking care of Crowd2Fund's servers hosted on AWS for the last year, when we moved them from Ireland to London.

Server Map

This is the Server Map proposed to be implemented between Christmas 2017 and January 2018. This new server map will increase both security and performance. More information is provided on next section.



Production Environment

Our Live Environment should manage up to 5 servers to deal with all requests. There are several Security Policies applied on Amazon EC2 in order to allow incoming connections just from trusted IPv4 Addresses, to ensure the data is safe.

Live environment has an isolated RDS Server to manage the Live Database, so in case of a known issue on the Web Server then the database won't be compromised due is provided by an isolated environment that just allows connections to the database port.

We are introducing 2 isolated ElastiCache servers to provide both Redis Server and Memcached Server services. They are used in order to increase the server performance and enhance the security of potential known issues, to not compromise the other services.

Just the two EC2 instances will have public access via SSH. One of them is going to manage the Web Server (based on NGINX and PHP 7.0 FPM), to deal with Website & API requests.

The 2nd server will manage a copy of the same source that we have on the 1st server, and will run just Redis Workers to make sure heavy tasks are not causing a delay or degraded performance on Website and API requests.

We use other AWS Services too, like Route 53 to deal with all DNS requests (although the domains are not registered on Amazon) as well as several AWS S3 Buckets to upload user files as well as to keep some manual backups.

Non-Production environment

Currently we have other 3 EC2 instances to manage other servers.

Two of them are managing the DEV and UAT environments, used on the Development phase as well as the User Acceptance reviews. They both have all required services locally, like database, Redis or Memcached.

The last one is our own Zabbix Server, that is monitoring all our instances, in order to get alerts and manage reports in case there is some unexpected behaviour in any of our servers. Zabbix Server is supported by NIXStats as well as New Relic, as a complementary service, in case there is a problem with our own service (who watches the watcher?).

Agile Development

As part of our 2018 Roadmap, both Business and Tech Team should apply Agile methodologies from now on. In this way, the whole team should be able to work on a more efficient way, keeping tracked goals for short-term periods.

Please refer to "Quality Assurance Approach" to read more about how the Agile methodologies will be used and how it will improve our day-to-day developing new features and improving our project's quality.

Team Meetings

Team meetings should be every 15 days, and should not require more than 1 hour. The main goal is to discuss the main projects and achievements we got since the last team meeting as well as to plan the upcoming tasks for the next 2 weeks.

Team Catch-Ups

On a daily basis, all team members should have a quick catch-up (no more than 15 minutes in total) to describe what we did yesterday, what is today's plan and to report any potential blocker, to get help from another team member (when needed) or – at least – to keep everybody up-to-date about a potential delay on the scheduled tasks.

Sprints

Both Business and Tech teams should work on a 2-week Sprint basis, agreed as part of the team meetings.

On Tech side, Business team should describe all specs for upcoming changes and business team should build just the requested steps, no more (to avoid unnecessary delays or processes) no less (to avoid a complaint due the requirements were not covered).

On Business side, Business team should plan their tasks aligned with the Tech team sprints, so Business Development and Risk teams will be able to use new features as scheduled, and Marketing team will know exactly what to announce on Newsletters and Social Media posts.

Performance

Current C2F project is managing a mixed & extremely complex project, due it has evolved to a mix of 4 environments:

- Frontend Website (with several JavaScript and Twig processes)
- Frontend AngularJS website for Business Registration journey
- Backend Website (with multiple non-reused pieces of code)
- Backend API (with multiple non-reused pieces of code)

A set of upcoming updates should be applied in order to improve the code quality as well as the performance (both are related here).

Database queries refactor

We have several duplicated database queries due we have too many methods inside the Repositories. This makes impossible to reuse the code, and increases the risk to apply human mistakes, that will return wrong data now or in a near future.

All database queries should be reviewed and refactored, in order to ensure we can return the expected data reusing as much code as possible. Also, the returned content should be cached by Memcached in order to increase the performance, however it means we will need to include multiple events everywhere to make sure the cached content is removed in case something changes.

Refactor of existing processes

We have several duplicated codes on both Website and API, including multiple checks on both cases that could be simplified and unified.

These processes should be redesigned as part of the migration from mixed website to a single API project.

Blackfire (performance as a requirement)

A common mistake is to build features without taking care of the performance. Blackfire has helped us already, improving performance on multiple cases. However current performance is not good enough yet.

All new features and updates should pass a Blackfire analysis and response time should not take more than 2 seconds to be served.

Poor performance on Local environments (blocker)

A common issue that all tech team members have been reporting is the poor performance on local environments. This problem is related to 2 main issues, and we should fix them as soon as possible:

Live database (about 200MB plain text content) on local environment:

Our laptops are laptops, not Live servers, and dev environment provides several additions in order to debug and track any potential issue on Live before it's merged and deployed.

The only fix we can do here is just not use Live databases on local environments for both reasons, security and performance.

Please refer to "Security" section for more details.

Mixed code for Frontend and Backend

Every time we need to apply a frontend update, tech team spend more time waiting for a reload of any page than the time used to apply the fix.

This is happening because due Symfony is a PHP Framework all requests are managed by the PHP code through the framework, so we need to load all PHP resources and wait for a full PHP process and render to HTML to check anything.

As described on the "Project scope" section, our Frontend and Backend code should be split, in order to create 2 isolated projects. In this way, any Frontend code will be managed away from PHP code, allowing to take care of any Frontend issue with just 10% of the current time.

Local environment will simulate API calls using local files that will provide the same response format that we do through the Live API. In this way, all Frontend updates won't depend on Backend features to build the implementation, and we will just need to switch between Local or Live environment to use the Live API or the local files. Applying this methodology, all Frontend-related updates could be provided faster.

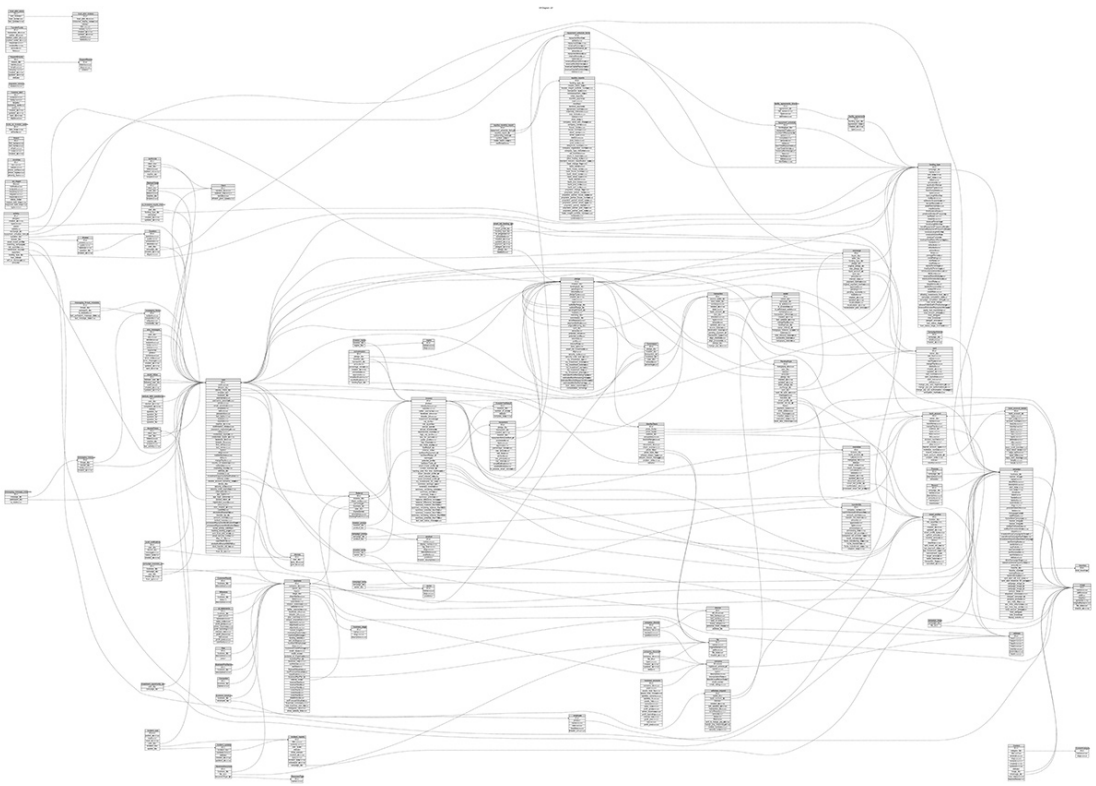
Deployment time

Although in the last year we improved the deployment time from 30 minutes to less than 4 minutes, this is still too much time.

Our deployments should be completed in less than 30 seconds, in order to make sure we don't have downtimes and everything is always up-to-date.

Database scheme refactor

Our current database scheme is inappropriate, probably due a bad design at the beginning of the project, and now it is not allowing us to simplify some processes or reduce the high volume of SQL queries that we need to run in order to retrieve all data that we need.



This scheme should be reviewed and refactored, and several processes should be redesigned and optimised due performance reasons as well as to make sure we don't get unexpected content due the complexity caused by this design, that contains too many conditionals to be tracked.

Code Quality

Automated tests

We are providing a Fintech service, that means our features must be developed in a bullet-proof way always. Automated tests on Backend side will allow us to make sure anything related with user's details will be managed in the way we expect, and we won't face any other unexpected behaviours.

Please read more about automated tests on "Quality Assurance Approach" section below.

Continuous Integration

We should implement a Continuous Integration (CI) service, in order to build our project and run all automated tests on an isolated environment before to deploy new code on Live servers.

Potential CI tools to be considered are: Scrutinizer CI, Travis CI or GitLab. If we opt for GitLab then we could think about our need to pay for GitHub, due GitLab provides Git hosting services too as well as a fully integrated CI services via GitLab settings.

SensioLabs Insight

Currently we have a paid subscription to SensioLabs Insight, and we fix the critical issues reported by this service, before to merge and deploy new features. However, we have around 500 non-critical issues that should be fixed in order to warranty the expected code quality that a finance platform should provide.

Security

As a Fintech company, Security is one of the most important goals for us. Our projects will manage all expected details in a secure way, as described below for Website, API, Payment System and local environments.

Crowd2Fund (C2F) project

Our current web project should remove in an upcoming update any kind of content that was stored due our business requirements before or after they were sent to MangoPay, specially cards details, bank accounts and ID check documents. The new Payment system should take care of all these sensitive details from now on.

Crowd2Payments (C2P) project

We have planned to have our own secure system, that will encrypt all details provided by our customers, so in case someone steals our database then they won't be able to get access to these details.

The cypher method will use a unique master key to encrypt and decrypt the unique user's key, and then all details in the system will be encrypted based on this unique key for each user. In this way, in case of an attack, the master key won't be the only security layer to decrypt the content.

Given all details will be encrypted, the only way to retrieve data will be through entity IDs, so the payment system will provide the expected security level that should provide any payment platform.

Also, this project won't be built without automated tests, due we need to make sure we won't have unexpected behaviours. BeHat Scenarios and PHP Spec examples will validate the API endpoints and PHP Unit will cover the external integrations and adapters.

Local environments

Local environments could be the most vulnerable access points, due we will never know if someone checks what we have or what we do with our laptops. Because of this, Live data on local environments is strictly forbidden as a company policy.

If there is a Live issue, then UAT environment should have a way to interact with a copy of Live database, in order to replicate the same environment on a secure server, and all updates should be deployed on DEV and UAT servers to validate the expected behaviour.

Once we have the isolated projects for Frontend and Backend code, Frontend project could target Live API endpoint easily, in case we need to review and fix an existing issue, but never hold Live databases on local.

AWS Servers

All live data will be accessible just from trusted sources, as described on the Servers section (please check previous pages for more details).

Mobile Apps

Current metrics

Since the day that we recorded our 1st Pledge through the API, we have got more than 3300 investments through the iOS App. Some pledges were not flagged as API pledges in the 1st phase due the feature to track them was added lately, so we believe there are at least 3500 API pledges.

Mobile Apps have been re-inventing the way all investors can interact with our system and invest on great British businesses, and this should become the most important asset for us.

iOS App

Our iOS App should grow, allowing all features we currently have on our website, and making sure it provides a great Mobile-First access point and experience for all new investors.

Please refer to the "Objectives" section to read more about scheduled changes on the iOS App for 2018 Roadmap.

Android App

A new Android App should be developed in 2018, to provide a new way to interact with the system for all Android users. All features should match with the iOS App.

Quality Assurance Approach

As a financial company, we cannot allow any issue neither on our website nor on our API. Because of this, the Quality Assurance approach requires new efforts from a Tech Team point of view, but a QA role should review any potential change before to go to Live too.

QA Role

A new team member to review and validate any potential new feature or update should work close to both tech team and business team, due the business team will describe all requirements and tech team will need to implement them.

QA Role must check that all business team requirements are covered, by the tech team, and then approve the changes. All Jira tickets must remain open until the QA role approves the changes.

Please refer to "Agile" section in order to get more details about the expected way to schedule upcoming tasks and features.

API with BeHat scenarios (BDD)

As a financial company, we cannot expect

As part of the migration to the only-API web project, we should

- Add multiple BeHat scenarios and features to make sure we have all expected API endpoints and we return the expected content on each case.
- Build the File System repositories that we will need to simulate existing data in the database, so we can confirm the expected content on each case

Source code with PHP Spec examples (TDD)

PHP Spec provides a clean and powerful way to make sure all content does exactly what we need, due it uses mocked objects to validate the expected behaviour on each case.

There is no sense on build PHP Spec examples for existing code, however we should not allow new code without verified behaviour.

Third party integrations with PHP Unit tests (TDD)

PHP Spec uses local mocked objects to validate the expected behaviour, however when we use any third-party library or bundle, or when we integrate an external API then we need PHP Unit to validate the expected integration.

There is no sense on build PHP Unit tests for existing code, however we should not allow new code without verified behaviour.

Code Quality

Please refer to the “Code Quality” section above on previous pages to read more about our Code Quality approach.

Project Plan

(At the current stage I am unable to propose an accurate project plan for all these changes due all tech team members might need to schedule their tasks and we need some external collaboration like Geoff or Matt for the redesigns as well as Lee or another mobile developer for the Mobile Apps)

Tech Team

Tech Team Leaders

Our current tech team is leaded by 3 members:

David Garcia
Tech Dev Director
London, UK

Liviu Dragulin
Tech Ops Director
Bucharest, RO

Marilena Beliciu
Frontend Developer
Bucharest, RO

David is our Backend expert, who improves our system providing security upgrades, improving our code quality as well as performance, and build new features to be used on both website and API.

Liviu is the data expert, and he takes care of all operational processes, related with the data integrity as well as to provide enhancements to improve the current steps that we already have.

Mary is our Frontend Lean, and she provides and reviews any update related with the user experience and usability, based on the wireframes and designs provided by our external collaborators.

New Tech Team incorporations

At least one new full-stack developer should join our tech team early on 2018, in order to assist David as part of the London's team.

This new developer will work on maintenance tasks, supporting both frontend and backend developers, so David will be able to focus on new features and, in the meantime, Mary will be able to work on the new C2F and C2P projects, to provide isolated frontend environments.