CREATIVEHOUSE HANDOVER

by Nicolas Gominet

INTRO

Creativehouse (CH) is an in-house project management application I re-developed, starting in February 2022 (technically v4).

Because Rob had a plan to make a native mobile app, I decided to split the web version into:

- api using Laravel framework (v10)
- app (web) using Vue framework (v3) as SPA

Indeed, doing so then would allow me to have an api almost completely ready to be used for the mobile native app.

DEV SETUP

In order to improve efficiency when working on creativehouse, I developed a suite of tools in **ch-dev** that manage all required docker containers (8 in total) and a few shortcuts for repetitive complex commands run against containers.

⚠️ Do not forget to set up your hosts file to make vh.test and its all subdomains to point to localhost as follows:
127.0.0.1	vh.test creativehouse.vh.test api.creativehouse.vh.test ws.creativehouse.vh.test s3.creativehouse.vh.test

CODE

Pull the code from https://github.com/visualhouse/ch-dev and read the readme file.

LOCAL ENVIRONMENT

Once setup this will set up 8 containers:

- api: laravel
- app: vue SPA
- nginx: web server
- mysql: MySQL database
- redis: cache and queue jobs storage
- meili: meilisearch
- mailhog: email inbox
- ws: websockets using soketi

API

The api has been developed using Laravel PHP framework.

If you know Laravel, you will quickly understand how everything works as I tried to stick as close to the framework as possible.

Production url is: https://creativehouse-api.visualhouse.com (I wanted to use *api* as subdomain but CloudFlare free plan only allows one level subdomains).

LOCAL DEVELOPMENT

In local development you have access to Telescope to easily inspect requests at: https://api.creativehouse.vh.test/telescope

CODE

Before diving into specific parts of the api architecture, there is a linting and formatting tool used called PHP CS Fixer, with a lot of rules that I defined in the *.php-cs-fixer.php* config file.

So before pushing to your pull request (PR), run **ch api check** command to make sure that your code is following the formatting rules. If you don't dependbot on github will do it for you and cause a retrigger of the test workflow on your PR.

APP SPA AUTH

Authentication to the SPA is handled using Laravel Sanctum.

IMPORTANT ENTITIES

The single most important database structure you need to understand is regarding deliverables:

1. Deliverables have stages with types: wireframe, draft, …
2. Deliverable stages have a status: in_progress, approved, cancelled, …
3. A Deliverable belongs to a Job

All the way to a Company:

4. A Job belongs to a Proposal
5. A Proposal belongs to a Project
6. A Project belongs to a Company
7. A Company has Users

Now regarding Projects:

8. Users have many Projects with a UserRole
9. All visualhouse Users have their Project UserRole based on their actual Roles

Once you understand this structure then the rest is not as important and less"deep" in terms of relationships between entities.

TYPICAL REQUEST FLOW

Usually this is how a typical incoming request is processed:

1. Match defined **route** in *routes/api.php*
2. Call relevant **Controller**
3. Request is validated
    a. using **Policy** for authorisation (at the controller level)
    b. then **FormRequest** for data validation
4. Controller method calls relevant app **service**
5. App service reach out to the database using relevant **repository**
6. Controller wraps app service result into relevant **Api Resource**
7. Controller returns response in a json format.

QUEUES

Some features do not need to happen asynchronously or are very slow to process and in this case we use one of the 2 queues we have: default and cli.

default

This is the general queue we use for sending notifications, generating proposals, generating contracts, ...

In production it is run on the same **ch-prod** server as api/app.

cli

The **cli** queue is a specific one that I create to deal with cli commands related tasks, in particular for generating images from uploaded pdf files.

I had to externalise this cli queue because this process is very resource intensive, but also can take a long time (minutes), mainly depending on the number of pages in the pdf. So I took the decision to create a worker only server in production: **ch-worker** that currently only does this.

On a side note, this process (images from pdf pages) is handled by calling **gs** (ghostscript) tool on the command line, and once complete the Laravel job pushes a notification to the frontend user that initiated it (using a private channel on websocket via redis).

DOMAINS

Some parts of the api were so specific that I decided to use the **Domain** namespace for them.

Accounting

This is basically handling all communication with **Xero**:

- Connect CH to creativehouse Xero app
- Create an invoice on Xero
- Update an invoice on CH from Xero: this is a job because Xero wants a 200 response straight away, so we do the actual work on a dispatched job

Migration

There used to be a lot more files here as this is where I had all the code to migrate data from the previous creativehouse database (aws) to the current database and structure.

In that regard there is still one use case for this domain: users password.

Indeed, in order to avoid having all users to reset their password I imported the previous hashed password and what this service does is depending on the user password field encryption detected will use the previous method or the new one.

If it detects that it's the old password and that the given password matches, the user will be logged in as expected but on top we'll reencrypt their password using the new method.

EXTERNAL SERVICES

There are a couple of external services that the api relies on to function correctly regarding prices and accounting.

Xero

Xero is the accounting software used by visualhouse to bill clients.

Visualhouse users with an accounting role can access a page where they can create new invoices based on jobs (with the same currencies).

Once filled, the api sends the info to Xero via the creativehouse app I created that accounting (Joe Romeo) then connected to visualhouse Xero account.

Exchange Rates

All default prices used on proposals are stored in USD only so when project managers create proposals in a different currency, the api uses a cache with exchange rates based on USD.

There is a command **ch:refresh-rates** that is automatically run weekly by Laravel scheduler to keep the rates up to date.

The service used for these rates is CurrencyApi on their free tier plan.

TESTING

Tests are automatically run on pull requests and *main* branch.

To run them locally, simply run:

- **ch api test** - you can add any PHPUnit arguments to it like --filter
- **ch api paratest** - running tests in parallel
- **ch api coverage** - to generate an html code coverage report, stored in *tests/_coverage*

VIEWS

Although this is purely an api, there are some views that are required to generate contracts, proposal pdfs and notification emails.

APP

The webapp is an SPA developed using Vue v3 and Typescript.

For styling we use tailwindcss with custom configuration in *tailwind.config.mjs*.

Production url is: https://creativehouse.visualhouse.com

CODE

Everything is what you would expect on a Vue SPA (router, pinia, i18n, ...) except that there is a separation in folder structure between guest portal (guest), visualhouse portal (vh) and client portal (client).

To avoid having to manage multiple repositories with a lot of shared code, this is one codebase that handles 3 different portals: *guest*, *client* and *vh*. All related code is under *src/{portal}*.

For everything that is shared between at least 2 views, it ends up under *src/shared*.

TESTING

There is no unit testing setup for the application but there are some e2e tests using Cypress when running **ch cypress** command.

Also on each PR and merge on *main* branch there are Github Actions that will build the app, so testing it can sort of “compile” without issue.

DEPLOYMENT

SERVERS

All servers are run on Akamai/Linode using Forge for easy management.

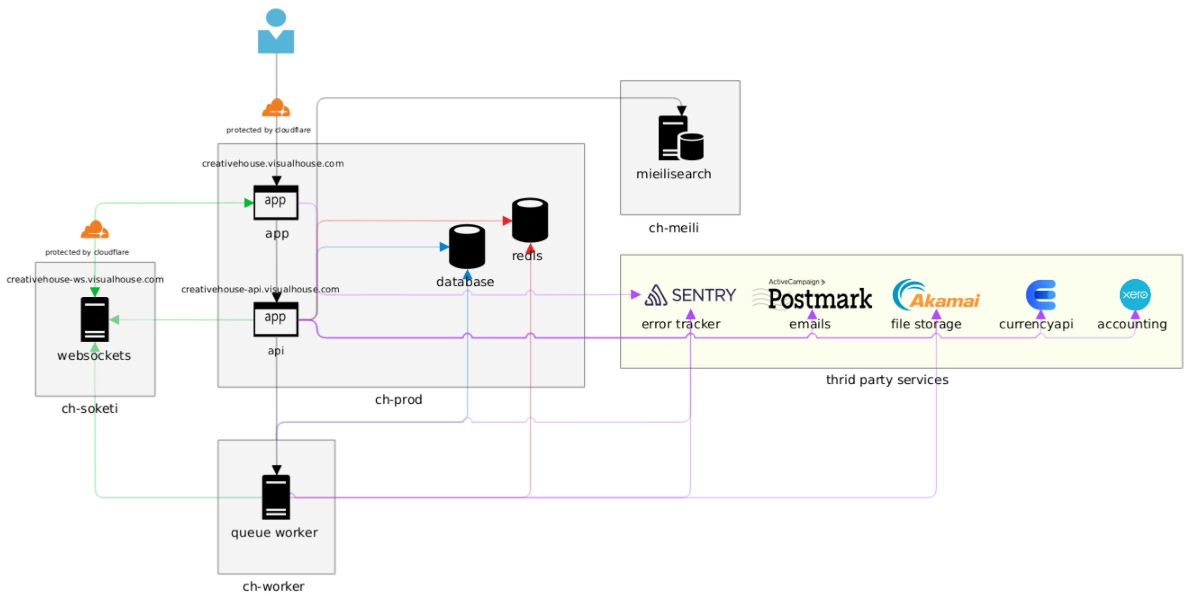
CLOUDFLARE

I set up cloudflare for efficient DNS management but also as a (free) protection layer against DDoS attacks.

In addition to CloudFlare's standard solution, I manually blocked some countries using their WAF firewall product that were the ones trying to attack visualhouse domain the most.

⚠️ Always remember that cloudflare sits between all creativehouse servers and the Internet.
I had an issue with Xero webhooks that suddenly stopped working and they then deactivated the prod endpoint. They assured me it was sent and I could see nothing in the logs until I remembered that Cloudflare is between the two and sure enough Cloudflare was blacklisting Xero after I activated their “Bot Fight Mode”.

PRODUCTION ARCHITECTURE



STAGING ARCHITECTURE

There’s also a staging server **ch-staging** that is doing what all ch production servers do but just in one server instead.

It's using all the same third-party services as **ch-prod** but on different instances (e.g. file storage) or credentials (e.g. postmark).

I struggled for a few things when setting up the staging server, so here are some of them:

- Data:
  - Need a rob user with signature for contracts
  - Need a project director for contracts: configurable in *.env* file now
- Config:
  - Symlink */home/{user}/.cache/puppeteer* to */home/{user}/{site}/.cache/puppeteer* to generate contracts

STRATEGY

PRE-DEPLOYMENT

In **ch-api** if the PR passes it's generally very safe to deploy to servers before the GitHub Action on main branch finishes as it's doing exactly the same as in the PR plus code coverage (Code Climate).

In **ch-app** on the other hand you have to wait for the GitHub Action on main to complete as this is the building step. Once it's finished, it pushes all built assets to the dist branch.

STAGING

Go to https://forge.laravel.com/servers/748145/sites/2208543/application and hit *Deploy Now* button.

PRODUCTION

- **ch-worker:** https://forge.laravel.com/servers/722991/sites/2120646/application (ch-api repo, *main* branch)
- **ch-api:** https://forge.laravel.com/servers/600674/sites/1780589/application (ch-api repo, *main* branch)
- **ch-app:** https://forge.laravel.com/servers/600674/sites/1781843/application (ch-app repo, *dist* branch)