AOUT 2016

DOSSIER DE PROJET

LA CARAVANE DU TOUR 2016

BAPTISTE FOUCHER
TROOPERS
Diplôme Développeur Logiciel session DL13 IMIE Nantes

Table des matières

Remerciements

Mon tuteur, Mr Paul ANDRIEUX, pour sa disponibilité et son aide précieuse à l'apprentissage du développement avec Victoire.

Mr Leny BERNARD, pour sa patience et l'encouragement à m'améliorer après chaque demande de revue de code.

Mr Loïc GOYET, pour son expertise du développement front, sa passion à faire découvrir et partager son univers et ses outils.

Toute l'équipe Troopers pour m'avoir accueilli et fait confiance en m'intégrant pleinement aux projets et en restant toujours à l'écoute à la moindre difficulté.

Mon formateur référent Benjamin POISSON, pour nous avoir partagé sa passion de la programmation et avoir rendu ce langage presque naturel.

Nicolas CHAUVET, formateur à l'IMIE, pour nous avoir fait découvrir toute la puissance et la simplicité de développer en Php avec Symfony.

Gabriel BOCK, second formateur référent, pour son accompagnement lors de la réalisation de nos dossiers et l'encouragement à nous faire découvrir de nouveaux framework autour du JavaScript.

Et toute l'équipe administrative de l'IMIE, pour leur accompagnement tout au long de notre formation, leur aide lors de nos recherches de stage et d'entreprise.

Competences mises en œuvre

Maquetter une application
- Utiliser un outil de maquettage p.11
- Connaissance du formalisme des cas d'utilisation et du diagramme d'état de la notation du langage de modélisation unifié UML p. 9

Concevoir une base de données
- Connaissance du modèle relationnel p.9
- Construire le schéma entité association p.9

Mettre en place une base de données
- Connaissance du langage de requête structurée SQL p.27
- Connaissance des différents types de codage des données p.22

Développer une interface utilisateur
- Connaissance des concepts de la programmation objet p.14
- Développer dans un langage objet p.14 –23
- Utiliser les normes de codage du langage et auto-documenter le code au moyen du nommage p.18-19
- Utiliser les bibliothèques de composants graphiques p.24

Développer des composants d'accès aux données
- Connaissance du langage de requête structurée SQL p.27
- Coder les accès aux données, la consultation, la création et la mise à jour, à partir de requêtes natives ou de procédures stockées p.26-27

Développer des pages web en lien avec une base de données
- Connaissance des langages du développement web, tels que langage de balise, feuilles de style et langage de script client p.24-25
- Connaissance des composants serveurs, pages web dynamiques p.14-23
- Développer la partie dynamique de l'application avec des composants serveur p.14-23
- Utiliser un cadre (framework) de persistance des données p.14-15
- Réaliser un jeu de tests de l'application web en priorisant les tests ou en appliquant une stratégie de test p.32

Utiliser l'anglais dans son activité professionnelle en informatique
- Identifier les différents types de documents techniques et leur structure
- Lire et exploiter différents documents techniques

Project Abstract

Caravane du Tour is an event occurring just before cycling race Tour de France. To promote it, it's organizator has decided to implement a website where fans can play and win many goodies. Another objective is to present the different commercial partners, with their custom brand universe, and to entertain fans with animations, games, and prizes.

A lot of fans are on the road to see cyclist runners and browse on internet when they are waiting for the race. Therefore, to offer a good experience to everyone, the website shall be responsive and respect good practices for mobile usage.

Besides spectators are sending a lot of messages and pictures on social networks. To animate and share with a maximum of people, the application offers a social wall for every brand. But to prevent abuse, a community manager must validate every publication before.

In order to update and propose new editorial content, a CMS (content management system) will be used and will also provide online tools for adding or updating the interface elements for the website.

Contexte

L'agence

L'agence Troopers est une agence qui accompagne ses clients sur l'ensemble de leurs projets numériques. Composée de 6 développeurs back, 1 développeur front, 1 UX/UI designer, 1 stratège digitale et 1 directeur général. Spécialisée dans la réalisation de projets sur mesure principalement portés par Symfony et son propre DCMS Victoire.

Les Différents acteurs :

Pour nous accompagner à la réussite du projet, nous sommes en étroite collaboration avec LeReuz, pour la partie gestion du projet et relation avec les équipes d'Amaury Sport Organisation (organisateur du Tour de France). La réalisation des jeux a été confiée à l'agence Casus Ludi spécialisée dans ce domaine.

En quelques mots :

Amaury Sport Organisation (A.S.O) est propriétaire et organisateur de 60 événements sportifs dans plus de 20 pays. Le Tour de France rassemble des millions de fans à travers le monde et se place aujourd'hui à la 3ème place dans le classement mondial des événements sportifs majeurs.

La Caravane Publicitaire contribue au succès populaire du Tour de France et depuis 2013 elle s'est digitalisée proposant un dispositif autour de ses marques partenaire. Chaque partenaire dispose d'une page responsive personnalisée, de jeux-concours et d'une promotion sur les plateformes du Tour de France.

Cahier des charges

Objectif du projet :

La Caravane Publicitaire consiste en un écosystème d'environ 35 000 fans et followers, un site à fort trafic sur la durée du Tour de France (environ 450k visites) et une communauté active avec entre autres plus de 140 000 photos postées par les fans. L'objectif est de générer une forte attraction sur le web en développant une plateforme innovante, interactive et addictive qui permette à n'importe quel internaute de découvrir la Caravane et ses partenaires

Fonctionnalités attendues :

- Développer une plateforme innovante de ludification

Proposer des jeux ludiques et interactifs avec un mécanisme de tranches horaires pour gagner les lots afin d'accroître la participation des fans

- Générer du trafic vers les sites partenaires

Proposer du contenu personnalisé aux couleurs et univers des marques. Offrir des portes d'entrées vers les différents écosystèmes digitaux de chaque marque

- Faire découvrir la caravane avec des contenus interactifs

Les fans publient des messages sur les réseaux sociaux avec des hashtags dédiés, il faut pouvoir agréger ces contenus, les modérer et les afficher sur un widget dédié.
Avoir la possibilité d'ajouter ou modifier rapidement du contenu éditorial, photos, vidéos ...

- Engager les fans au bord de la route

Le site doit pouvoir fournir une carte sur lequel les internautes peuvent géo-localiser en permanence les véhicules de la Caravane.

Caractéristiques Techniques :

- Adaptabilité : Le site devra être entièrement responsive mobile et tablette
- Compatibilité : Le site doit être pleinement compatible sur les principaux navigateurs : Chrome, Internet Explorer, Safari, Firefox
- Charte graphique : Les logos et chartes graphiques seront fournis par A.S.O.

Planification du projet

WBS

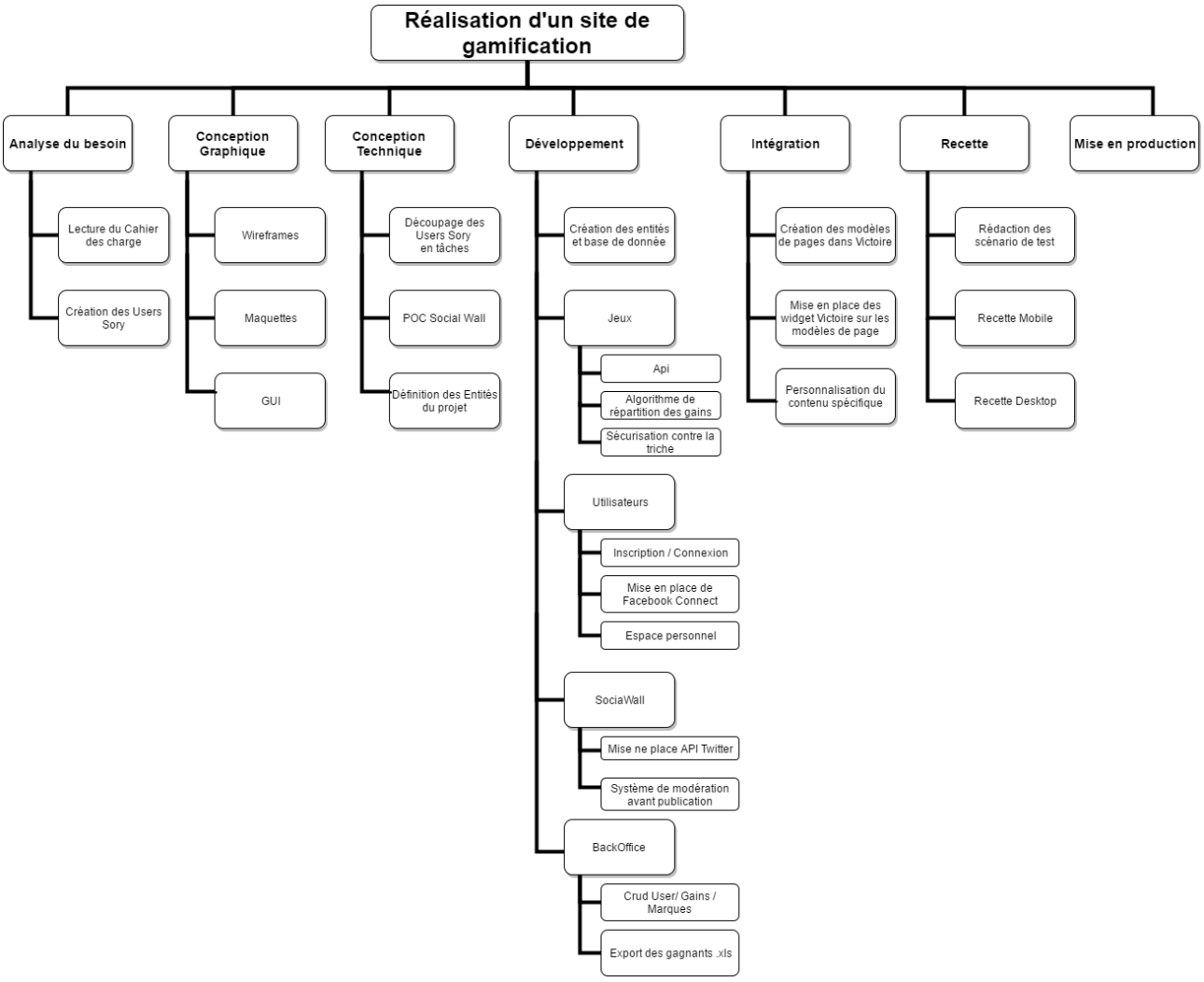
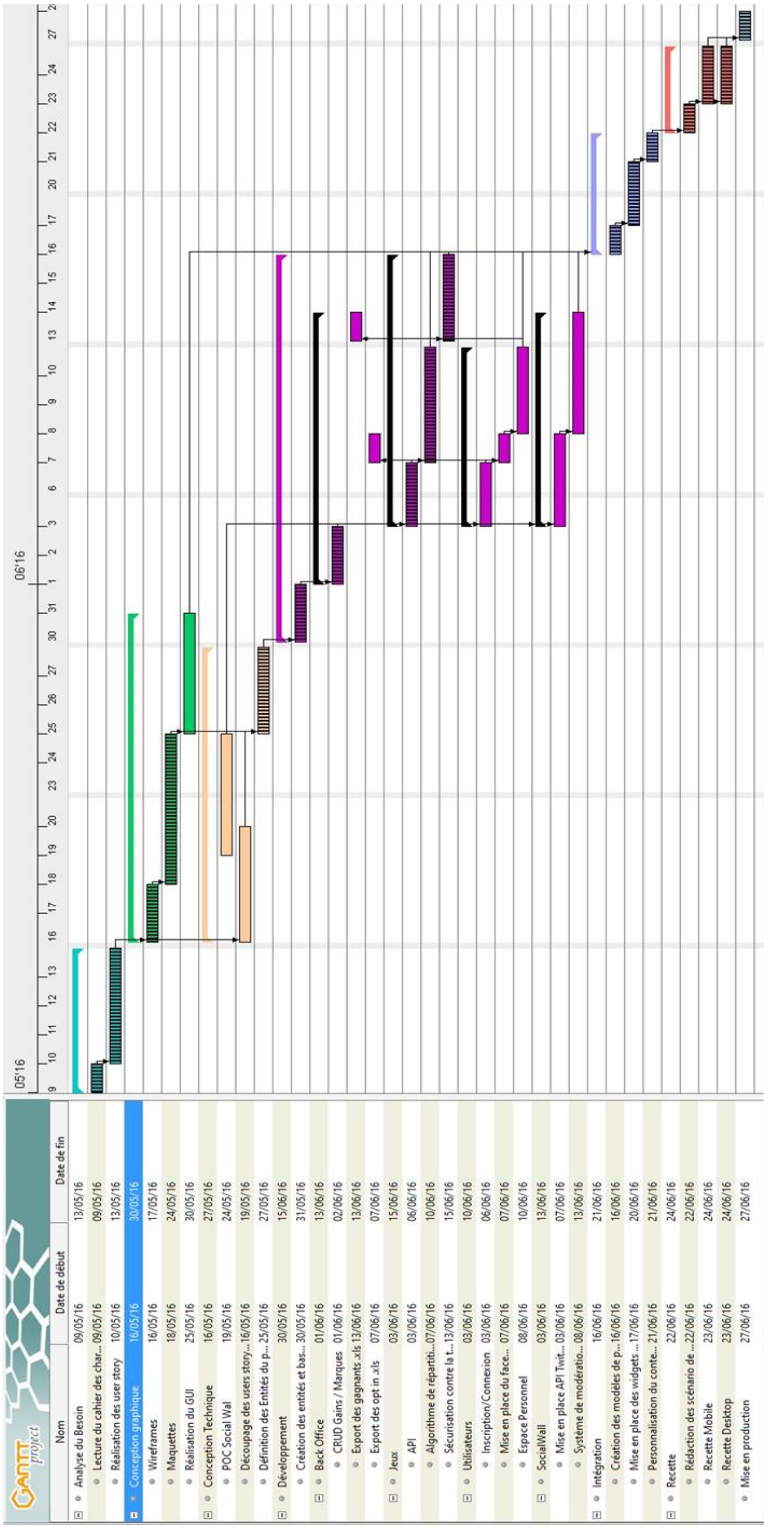


Diagramme de Gantt



Le diagramme de Gantt nous permet de visualiser rapidement le chemin critique et donc les tâches nécessitant la plus grande vigilance sur le respect des délais.
Ici on s'aperçoit que l'intégration des jeux à notre application se trouve sur le chemin critique et pourrait retarder la réalisation du projet

Spécifications Fonctionnelles

Diagramme de cas d'utilisation – Social Wall

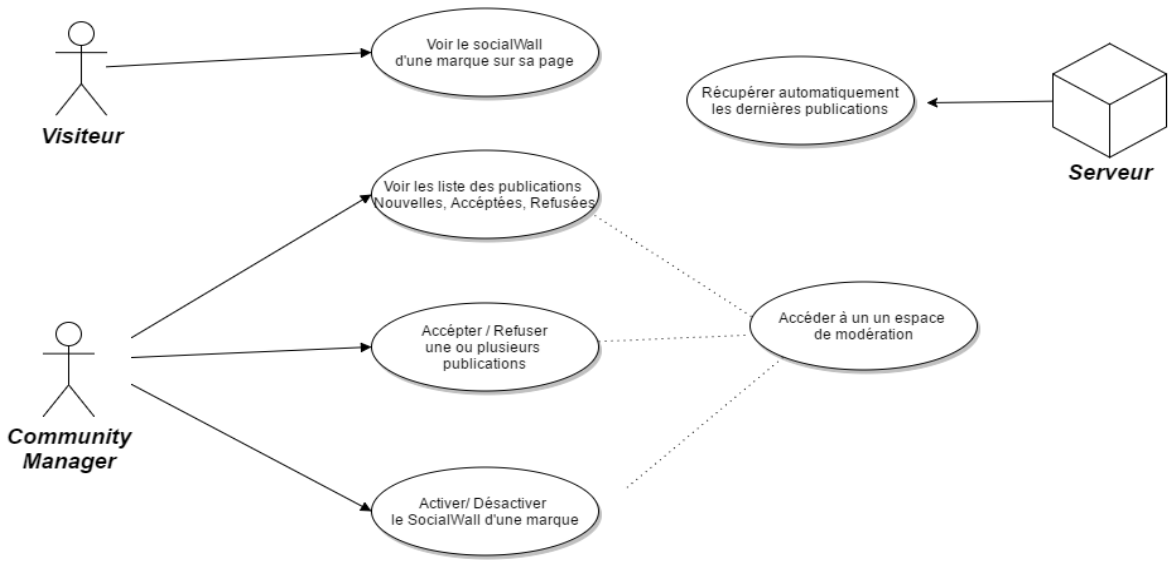
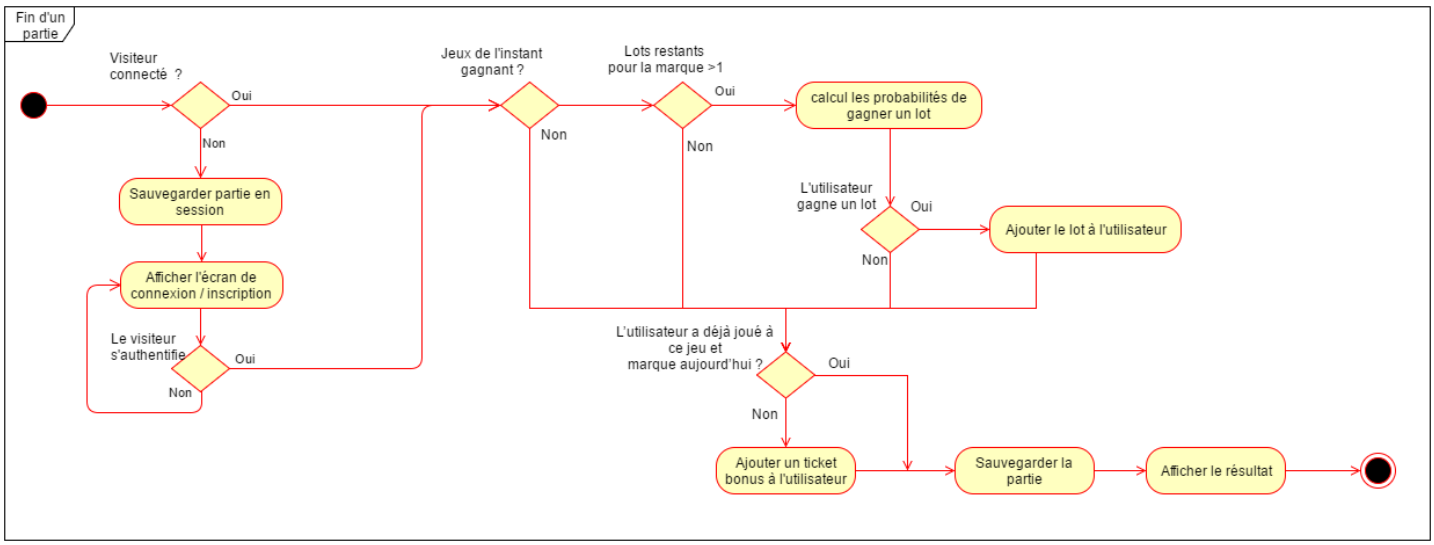
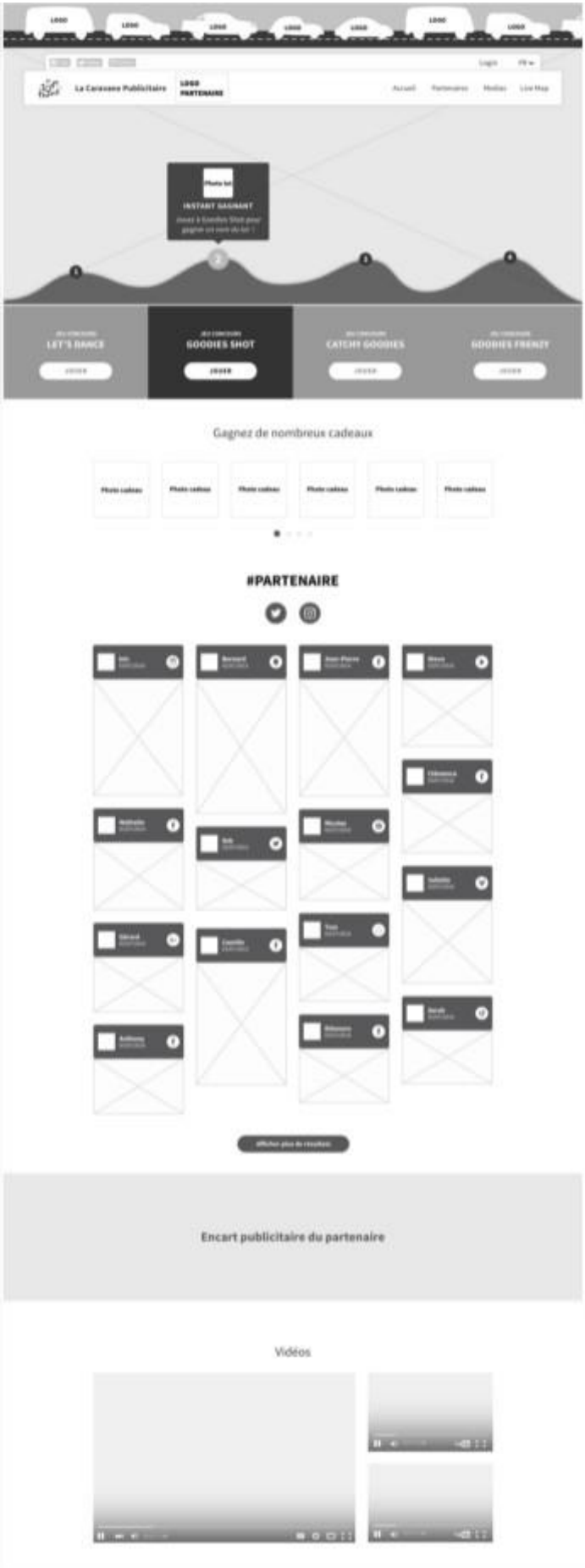


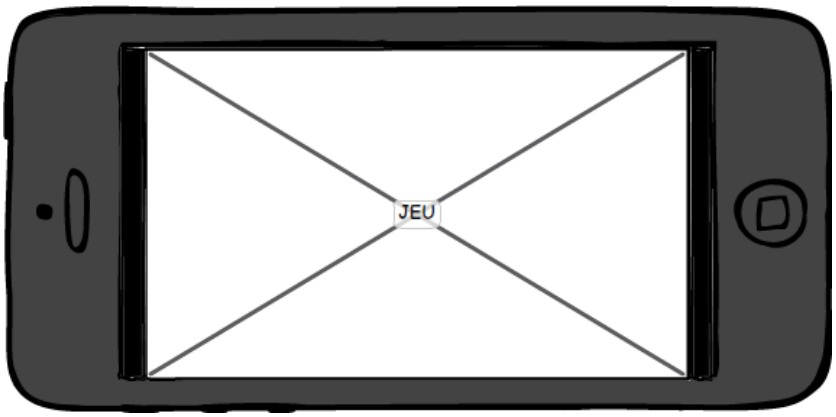
Diagramme global d'interaction – Fin d'une partie



Wireframe


La Caravane Publicitaire
LOGO PARTENAIRE
INSTANT GAGNANT
LET'S DANCE
GOODIES SHOT
CATCH GOODIES
GOODIES FRENZY
Gagnez de nombreux cadeaux
#PARTENAIRE
Encart publicitaire du partenaire
Vidéos

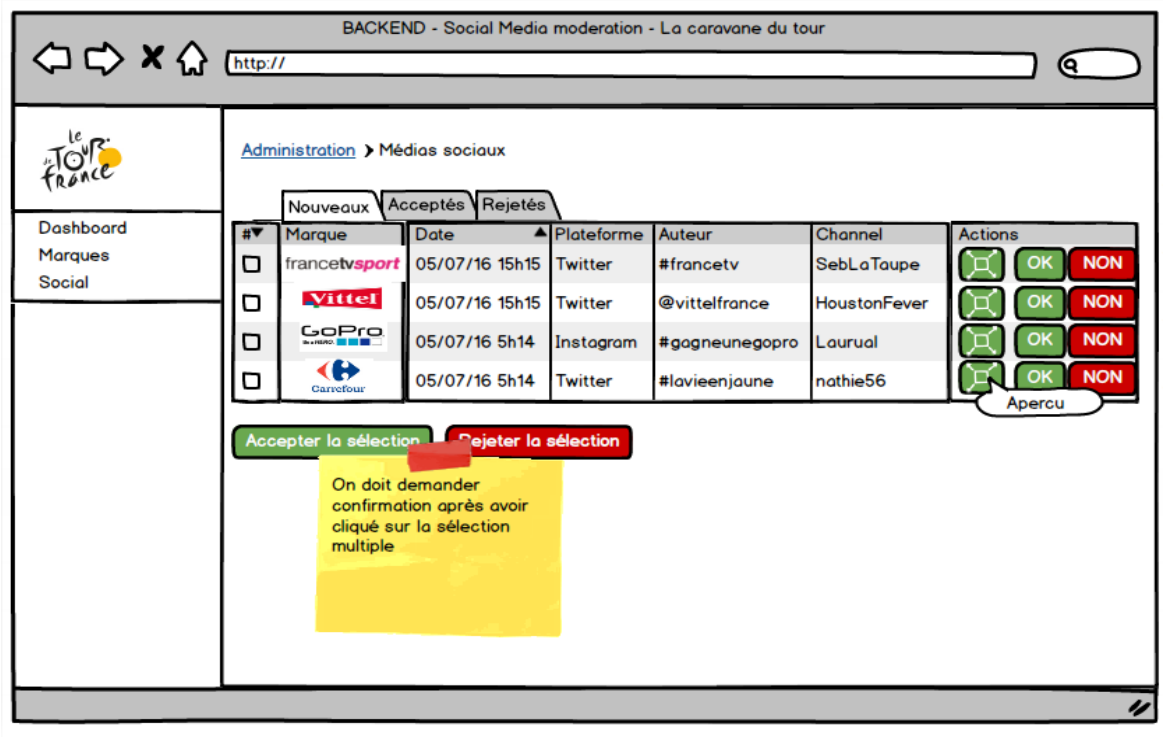
Wireframe des jeux sur mobile



Affichage de la même page une fois le mobile en mode paysage

Affichage de la page jeu en mode portrait

Mockup Back Office des publications :



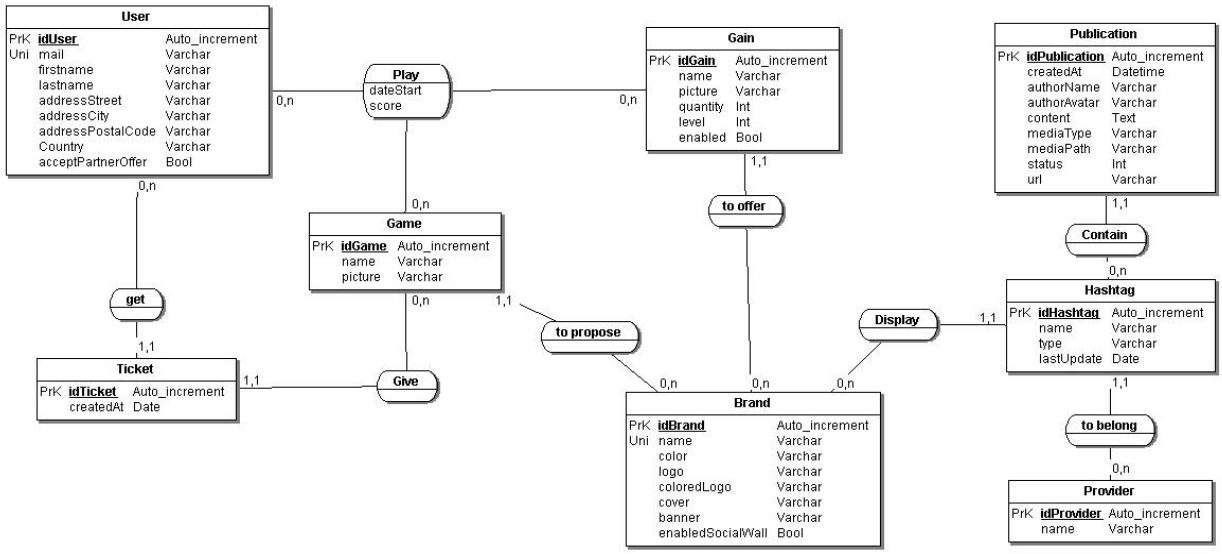
Spécifications Techniques

Choix des technologies :

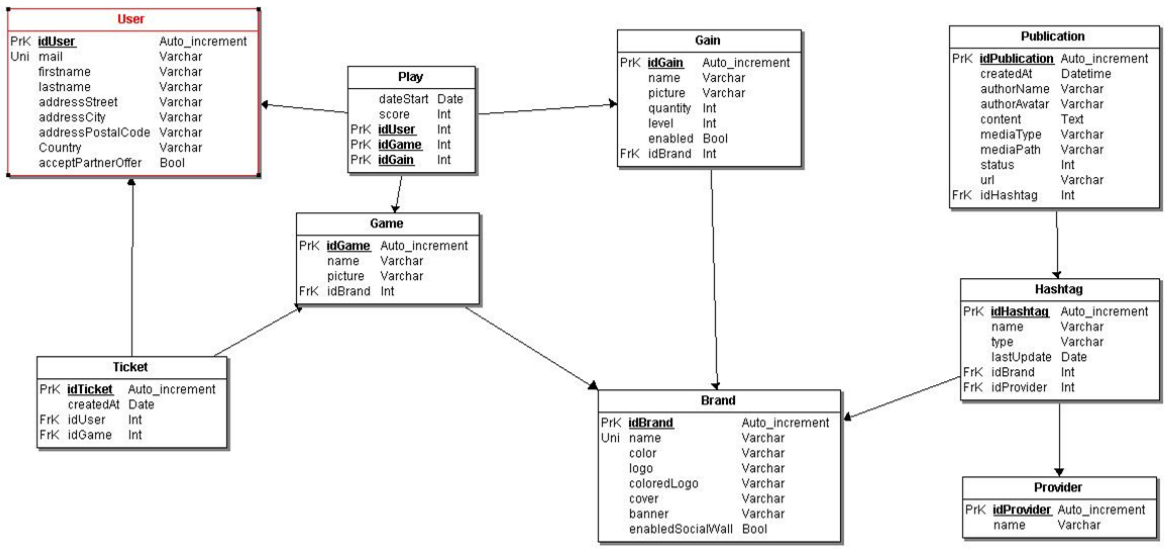
L'utilisation du framework php Symfony couplé au CMS Victoire ont été choisis afin de garantir un développement sur mesure tout en fournissant la souplesse du CMS pour permettre l'ajout ou la modification du contenu très rapidement une fois le site en lancé.

Nous avons aussi choisi d'utiliser Docker comme environnement en production pour garder un environnement identique entre le serveur de pré- production et de production.

Modèle conceptuel de données :

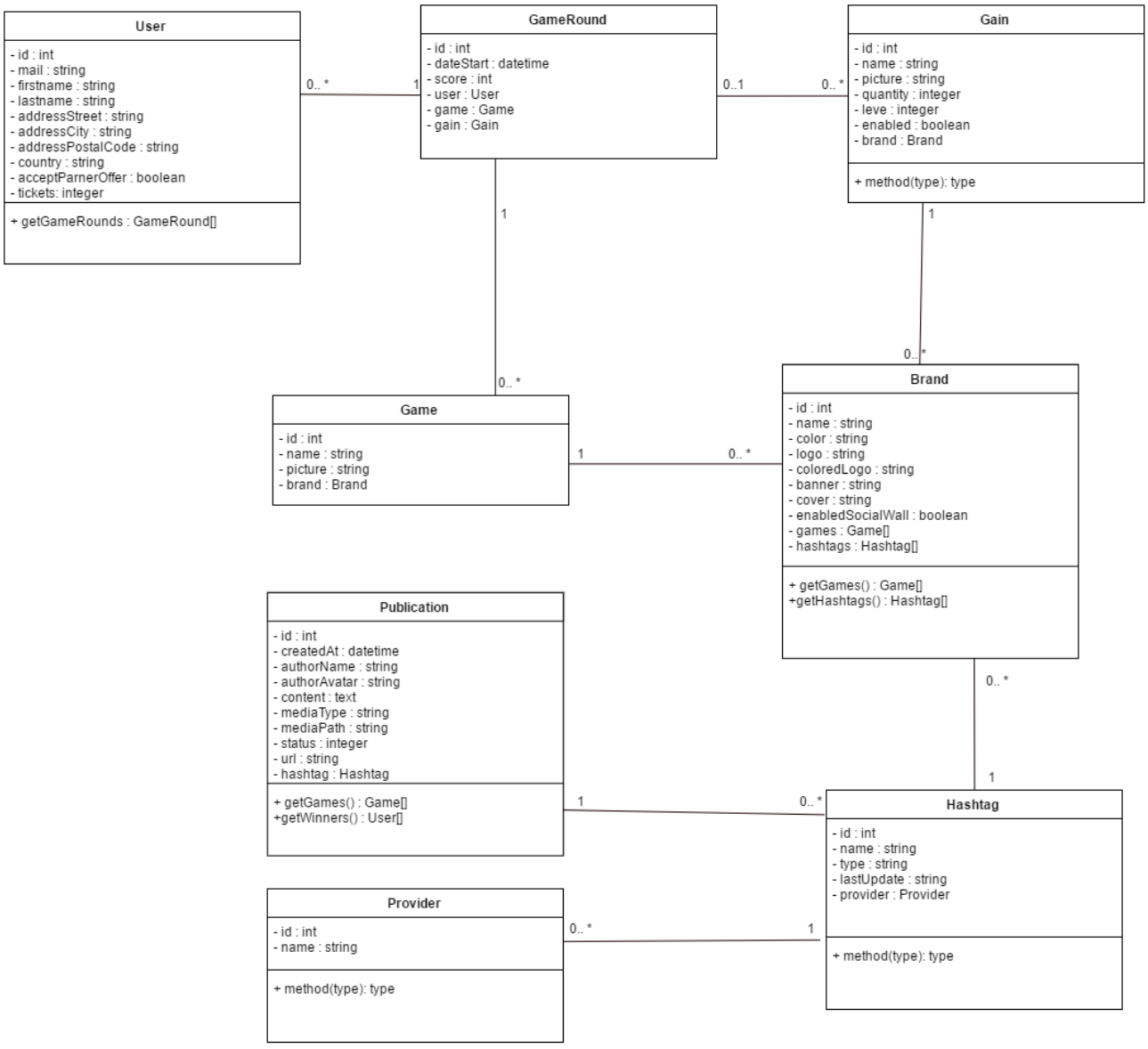


Modèle logique de données :



Comme on peut le voir la relation entre User, Game et Gain fait apparaître une nouvelle entité « Play ». Par la suite, pour la compréhension commune lors de la réalisation nous ferons le choix de renommer cette entité en GameRound

Diagramme de classes :



Réalisation

Symfony est l'un des frameworks php les plus populaires avec une très forte communauté active. Nous utilisons aujourd'hui la version lts 2.8 en appliquant dès à présent l'architecture de fichiers de la version 3

Le framework est basé sur le pattern Modèle-Vue-Contrôleur, notre CMS Victoire permet la création de pages à l'aide de widgets autonomes et indépendants qui peuvent s'interfacer directement sur les objets métiers grâce à quelques annotations sur les modèles.

Notre développement se déroule en 3 principales étapes :

1. Développement de briques fonctionnelles

   Les développeurs back construisent les modèles de données, les contrôleurs et services nécessaires au bon fonctionnement de la logique métier propre à l'application.

   Notre développeur front crée un GUI (Graphical User Interface) et des briques d'interface prêtes à être utilisées dans les vues ou comme templates à nos widgets.

2. Intégration de Victoire

   Les développeurs back relient les modèles et rendus plus complexes aux différents widgets du CMS.

3. Création et assemblage des pages grâce aux outils de Victoire

   Cette étape peut être réalisé par l'ensemble de l'équipe, c'est le moment où nous assemblons chacune des briques, widgets créés précédemment et en ajoutant le contenu éditorial.

Modèles

Pour faciliter et accélérer la création des modèles, Symfony intègre Doctrine, un ORM (Object Relationnal Mapper) qui nous permet à l'aide d'annotations d'interfacer nos classes PHP à la base de données MySQL. Il permet également de définir très simplement les différentes jointures entres les classes en respectant les cardinalités définies sur le modèle conceptuel de données.

Classe Publication

```php
<?php

namespace AppBundle\Entity\SocialWall;

use Doctrine\ORM\Mapping as ORM;
use Victoire\Bundle\BusinessEntityBundle\Entity\Traits\BusinessEntityTrait;
use Victoire\Bundle\CoreBundle\Annotations as VIC;

/**
 * Publication
 *
 * @ORM\Table()
 * @ORM\Entity(repositoryClass="AppBundle\Entity\Repository\PublicationRepository")
 */
class Publication
```

Figure 1: extrait de la classe Publication, déclaration

Les annotations, commentaires en gris, indique à Doctrine que la classe doit être suivie et persistée en base de données. La déclaration « repositoryClass » indique la classe où seront écrites les différentes requêtes spécifiques au format DQL (Doctrine Query Language).

```php
    const STATUS_NEW = 0;
    const STATUS_ACCEPTED = 1;
    const STATUS_REFUSED = 2;
    /**
     * @var integer
     *
     * @ORM\Column(name="id", type="integer")
     * @ORM\Id
     * @ORM\GeneratedValue(strategy="AUTO")
     */
    private $id;

    /**
     * @var \DateTime
     *
     * @ORM\Column(name="createdAt", type="datetime")
     */
    private $createdAt;

    /**
     * @var string
     *
     * @ORM\Column(name="content", type="text")
     */
    private $content;
```

Figure 2 : Extrait de la classe Publication, constantes et attributs

Ici nous déclarons 3 constantes représentants les différents statuts de l'objet. Nous utiliserons ensuite ces constantes dans notre développement pour simplifier la compréhension ainsi garder un code plus aisément maintenable.

Nous voyons également comment nous indiquons à Doctrine comment doivent être stockés et manipulés les attributs. $createdAt sera stocké dans la colonne created_at et sera de type datetime.

```php
/**
 * @var integer
 *
 * @ORM\Column(name="status", type="integer")
 */
private $status = self::STATUS_NEW;

/**
 * @ORM\ManyToOne(targetEntity="AppBundle\Entity\SocialWall\Hashtag")
 */
private $tag;
```

Figure 3 : extrait de la classe Publication, relation

Lors de l'instanciation d'un objet Publication, l'attribut $status est automatiquement renseigné.

L'attribut $tag est une relation unidirectionnelle avec l'entité Hashtag.

```php
    /**
     * Set tag
     *
     * @param \AppBundle\Entity\SocialWall\Hashtag $tag
     *
     * @return Publication
     */
    public function setTag(\AppBundle\Entity\SocialWall\Hashtag $tag)
    {
        $this->tag = $tag;

        return $this;
    }

    /**
     * Get tag
     *
     * @return \AppBundle\Entity\SocialWall\Hashtag
     */
    public function getTag()
    {
        return $this->tag;
    }

    /**
     * Get Brand (For EasyAdmin)
     * @return mixed
     */
    public function getBrand(){
        return $this->getTag()->getBrand();
    }

    /**
     * Hydrate with Array
     * @param array $data
     */
    public function fromArray(array $data)
    {
        foreach ($data as $key => $value) {
            if (property_exists($this, $key)) {
                $this->{$key} = $value;
            }
        }
    }
}
```

Figure 4: extrait de la classe Publication, Getters/Setters

Viennent ensuite les getters et setters qui nous permettent d'accéder aux attributs privés ou protégés de la classe. Ici nous rajoutons 2 méthodes spécifiques :

- getBrand(), nous permet de suivre la relation avec Hashtag pour accéder à la marque liée à la classe Hashtag. Cet appel direct nous sera utile dans l'interface d'administration comme nous le précisons dans le commentaire.
- fromArray(),nous créons ici une méthode nous servant à peupler un objet Publication à partir d'un tableau portant les attributs comme clé. Cette méthode sera utilisée lors de la récupération des publications à partir de différentes API.

Social Wall

Pour récupérer les publications à partir des réseaux sociaux, nous utiliserons des librairies propres à chaque API. Pour normaliser et uniformiser l'ensemble, nous faisons le choix de créer une classe qui servira d'interface avec la librairie, et une autre classe qui s'occupera uniquement de transformer les données reçues à notre format de Publication.

Twitter API

```php
<?php
namespace AppBundle\SocialWall\Twitter;

use AppBundle\Entity\SocialWall\Hashtag;
use TwitterOAuth\Auth\ApplicationOnlyAuth;
use TwitterOAuth\Serializer\ObjectSerializer;

class TwitterApi
{
    private $api;
    private $dataTransformer;

    /**
     * @param type $customerKey
     * @param type $customerSecrect
     */
    public function __construct($customerKey,$customerSecrect)
    {
        $credentials = [
            'consumer_key' => $customerKey,
            'consumer_secret' => $customerSecrect,
        ];
        $serializer = new ObjectSerializer();
        $this->api = new ApplicationOnlyAuth($credentials,$serializer);
        $this->dataTransformer = new TwitterDataTransformer();
    }
```

Figure 5: extrait de TwitterAPI, déclaration et construct

Lors de déclaration de la classe TwitterAPI, à la construction de l'objet, nous lui passons les clés privées et secrètes du client d'api Twitter, ces clés sont stockées dans le fichier parameters.yml qui rassemble les variables d'environnement de l'application.

A l'instanciation de l'objet, nous instancions également la librairie de connexion à l'api et la classe dataTransformer qui nous servira à formater les données selon notre classe Publication.

```php
/**
 * Get last tweets (max100) from twitter username
 * @param $username
 * @param null $sinceId
 * @return mixed
 */
public function getUserTimeLine($username,$sinceId = null)
{
    $params = [
        'screen_name' => $username,
        'count' => 100
    ];
    if ($sinceId) {
        $params['since_id'] = $sinceId;
    }
    $response = $this->api->get('statuses/user_timeline',$params);

    return $response;
}

/**
 * Get last tweets (max 100) from hashtag
 * @param $hashtag
 * @param null $sinceId
 * @return mixed
 */
public function getHashTagRecent($hashtag,$sinceId = null)
{
    $params = [
        'q' => $hashtag,
        'result_type' => 'recent',
        'count' => 100
    ];
    if ($sinceId) {
        $params['since_id'] = $sinceId;
    }
    $response = $this->api->get('search/tweets',$params);

    return $response->statuses;
}
```

Figure 6: extrait de TwitterAPI, methodes interfaces

Nous créons ensuite 2 méthodes qui servent d'interfaces avec la librairie pour récupérer les dernières publications par compte utilisateur ou hashtag. Nous en profitons pour passer nos propres paramètres notamment $sinceId qui est le dernier message récupéré lors du précédent appel, ceci afin d'éviter les doublons.

```php
    /**
     * Get Recent tweets from AppBundle:SocialWall\Hashtag
     * @return int Nb New Tweet
     */
    public function getRecent(Hashtag $tag){
        $result= '';
        if ($tag->getType()=='user'){
            $result = $this->getUserTimeLine($tag->getName(), $tag->getLastUpdateRef());
        } elseif ($tag->getType()=='hashtag') {
            $result = $this->getHashTagRecent($tag->getName(), $tag->getLastUpdateRef());
        }

        return $this->parseResults($result,$tag);
    }

    /**
     *  Parse Twitter Api Result and return array of Publication
     *
     * @param $results
     * @param Hashtag $tag
     * @return array
     */
    protected function parseResults($results,Hashtag $tag){
        $publicationsArray = [];
        foreach ($results as $tweet)
        {
            //Ignore RT on hashtag
            if ($tag->getType()==='hashtag' && isset ($tweet->retweeted_status))
            {
                continue;
            }

            $publication = $this->dataTransformer->tweetToPublication($tweet,$tag);
            $publicationsArray[]=$publication;
        }

        return $publicationsArray;
    }

}
```

Figure 7: extrait de TwitterAPI

Pour nous simplifier l'utilisation du service nous créons la méthode getRecent qui prend en paramètre un objet de type Hashtag. La méthode fait ensuite l'appel à l'api suivant le type de tag (utilisateur ou hashtag) et renvoie le résultat à la méthode parseResult.

Jusque-là nous avons une réponse formatée en JSON, cette méthode élimine dans un premier temps les retweets issue d'un hashtag (toujours pour éviter des doublons d'affichage) et passe ensuite chaque Tweet formaté en JSON à notre classe dataTransformer qui nous renverra un objet Publication.

Nous voyons que c'est l'enchaînement de ces 2 méthodes qui nous serviront dans le reste de notre application. Il nous suffit de passer un objet Hashtag à la fonction getRecent et celle-ci nous renverra un tableau d'objets Publication.

Data Transformer

```php
class TwitterDataTransformer
{
    private $ignoredTerms;

    /**
     * Transform a tweet to Publication object
     * @param $tweet
     * @return Publication
     */
    public function tweetToPublication($tweet,Hashtag $tag)
    {
        $this->ignoredTerms = [];
        $publication = new Publication();

        //Group all Medias
        $tweetMedias = [];
        isset($tweet->entities->media) ? $tweetMedias = array_merge($tweetMedias, $tweet->entities->media) : null;
        isset($tweet->extended_entities->media) ? $tweetMedias = array_merge($tweetMedias,$tweet->extended_entities->media) : null;

        $tweetMedias = $this->parseMedia($tweetMedias);

        // Array representation
        $tweetArray = array (
            'createdAt'       =>  new \DateTime($tweet->created_at),
            'authorName'      =>  $tweet->user->name,
            'authorPicture'   =>  $tweet->user->profile_image_url_https,
            'providerId'      =>  $tweet->id,
            'providerPath'    =>  'https://twitter.com/'.$tweet->user->screen_name.'/status/'.$tweet->id,
            'provider'        =>  $tag->getProvider(),
            'tag'             =>  $tag,
            'mediaType'       =>  $tweetMedias['type'],
            'mediaPath'       =>  $tweetMedias['path'],
            'content'         =>  isset($tweet->retweeted_status) ? $tweet->retweeted_status->text : $tweet->text
        );

        //clean Content (medias url, UTF8mb4 chars...)
        $this->cleanContent($tweetArray['content'], $this->ignoredTerms);

        //hydrate Object
        $publication->fromArray($tweetArray);

        return $publication;
    }
```

Figure 8: extrait de TwitterDataTransformer

tweetToPublication est la seule méthode publique de cette classe, sa seule fonction est de transformer un tweet au format Json en un objet Publication utilisable dans notre application.

```php
/**
 * Find Media in Medias Array
 *
 * @param array $medias
 * @return array [type, path]
 */
protected function parseMedia(array $medias){
    $mediaPath = null;
    $mediaType = null;
    foreach ($medias as $media)
    {
        if ($media->type ==='photo')
        {
            $mediaType = 'photo';
            $mediaPath = $media->media_url_https;
            $this->ignoredTerms[] = $media->url;
        }
        if (in_array($media->type,['animated_gif','video']))
        {
            foreach ($media->video_info->variants as $variant)
            {
                if ($variant->content_type==="video/mp4")
                {
                    $mediaType = 'video';
                    $mediaPath = $variant->url;
                }
            }
        }
    }
    return [ 'type'=>$mediaType, 'path'=>$mediaPath ];
}
```

Figure 9: extrait de twitterDataTransformer, traitement des médias

L'API de Twitter renvoie énormément d'informations notamment quand le tweet comporte des médias de type photo ou vidéo. Cette méthode s'occupe du traitement des médias pour ne conserver que les médias que nous utiliserons dans notre application.

```
    /**
     * Delete ignoredTerms & 4 Octets chars
     * @param $string
     * @param array
     */
    protected function cleanContent(&$string,array $ignoredTerms)
    {
        $string = str_replace($ignoredTerms,'',$string);
        //delete 4octets chars
        $string = preg_replace('%(?:
          \xF0[\x90-\xBF][\x80-\xBF]{2}      # planes 1-3
        | [\xF1-\xF3][\x80-\xBF]{3}          # planes 4-15
        | \xF4[\x80-\x8F][\x80-\xBF]{2}      # plane 16
        )%xs', '', $string);
    }
}
```

Avant de remplir l'objet Publication nous nettoyons le contenu du message grâce à cette méthode, lors de l'analyse du tweet nous stockons des portions de textes (url d'un média, 'RT' lors d'un retweet …) que nous venons supprimer avec la fonction str_replace. Enfin nous supprimons également tous les caractères sur 4 octets (norme UTF8_mb4) qui représente principalement des émojis et caractères asiatiques. Le choix de supprimer ces caractères a été fait car nous n'étions pas en mesure de garantir un affichage correct pour tous nos utilisateurs.

Création d'un service :

Symfony nous permet de déclarer des services, un service est un objet PHP qui remplit une fonction, associé à une configuration. Dans notre cas nous allons déclarer TwitterApi comme service, ainsi dans notre application je pourrais facilement appeler ce service, sans connaitre ni son fonctionnement propre, ni sa configuration, un service prêt à l'emploi !
Nous déclarons ces services dans un fichier yaml dans le répertoire app/config/services.yml

```
## SOCIALWALL ##
app.socialwall.twitter:
    class:        AppBundle\SocialWall\Twitter\TwitterApi
    arguments:    ['%twitter_cutomer_id%','%twitter_customer_secret%']
```

Figure 10: extrait de services.yml, Twitter Api

Nous déclarons notre service : app.socialwall.twitter et nous lui donnons les arguments nécessaires à l'instanciation de l'objet : les clés publique/privée de l'api twitter.

Création d'une commande :

Maintenant que nous avons récupéré des publications à partir d'un Hashtag de notre application, et qu'un service nous permet de le faire simplement n'importe où dans notre application, il ne nous reste plus qu'à mettre en place un système automatique pour exécuter cette action et sauvegarder ces publications en base de données. Nous créons donc une commande qui expose une fonction de l'application à la console serveur.

```php
<?php
namespace AppBundle\Command;

use Symfony\Bundle\FrameworkBundle\Command\ContainerAwareCommand;
use Symfony\Component\Console\Input\InputInterface;
use Symfony\Component\Console\Output\OutputInterface;

class SocialWallCommand extends ContainerAwareCommand
{
    protected function configure()
    {
        $this
            ->setName('socialWall:twitter:update')
            ->setDescription('Get last publications from Twitter')
        ;
    }

    protected function execute(InputInterface $input, OutputInterface $output)
    {
        $twitter = $this->getContainer()->get('app.socialwall.twitter');

        //Get all Twitter Hashtag
        $tags = $this->getContainer()->get('app.hashtag_repository')->getByProviderName('Twitter');

        $newPublications = [];

        $em = $this->getContainer()->get('doctrine.orm.default_entity_manager');

        //Get last tweet for each Hashtag
        foreach ($tags as $tag)
        {
            $results = $twitter->getRecent($tag);

            if($results)
            {
                //Save lastId for next update
                $tag->setLastUpdateRef($results[0]->getProviderId());
                $em->persist($tag);

                $newPublications = array_merge($newPublications,$results);
            }
        }

        //Persist all new Publication
        foreach ($newPublications as $publication)
        {
            $em->persist($publication);
        }

        $em->flush();
    }
}
```

Figure 11: SocialWallCommand

Symfony nous met à disposition une classe ContainerAwareCommand qu'il suffit d'hériter pour créer facilement des commandes accessibles ensuite en console. Il nous suffit de créer au moins 2 méthodes, la première qui configure la commande, notamment en définissant son nom, ici socialWall:twitter:update et la seconde qui décrit le code à exécuter à l'appel de la commande.

Cette commande récupère d'abord notre service TwitterApi, les différents Hashtag de notre application, et pour chacun d'entre eux la méthode getRecent est appelée. Nous stockons dans l'objet Hashtag l'id du dernier tweet récupéré qui servira lors du prochain appel. Enfin nous enregistrons en base de données tous les tweets récupérés.

Cron :

Cron est un programme disponible sur les systèmes Unix qui permet d'exécuter automatiquement des scripts, commandes à une date et heure précise ou selon un cycle réguliers définis à l'avance.
Dans notre cas nous faisons le choix de récupérer les nouvelles publications toutes les 10min, l'API de Twitter autorise 180 appels par tranche de 15min et notre application ne comportant pas plus d'une trentaine de hashtags, nous n'aurons pas à nous préoccuper des limites d'utilisation.

Une commande Cron s'écrit de la manière suivante :
Minute, heure, jour du mois, mois, jour de la semaine, commande à exécuter
Chaque valeur est séparée par un espace, ' * ' permet de spécifier toutes les valeurs. Notre commande cron à exécuter à la racine de notre projet sera donc :

***/10 * * * * php bin/console socialWall:twitter:update**

Forcer le mode paysage pour les jeux sur mobile :

Les jeux réalisés en Javascript sont conçus pour être affichés sur un format paysage, de ce fait nous sommes obligés de forcer nos utilisateurs mobile à tourner leur téléphone. Pour arriver à cela nous utiliserons uniquement du style css qui viendra occulter le jeu lorsque le téléphone sera en mode portrait et disparaitra en mode paysage.

HTML

```
<div id="game-container" class="iframe-responsive-wrapper" >
    <div class="mobile-portrait">
        <div class="mobile-message">
            <img class="picto" src="{{ asset('bundles/app/images/game/rotate.png') }}" /><br/>
            <p>{{ 'app.game.mobile.rotate'|trans() }}</p>
        </div>
    </div>
    <div id="captcha-container" style="display: none;"></div>
    <div id="game" ></div>
</div>
```

Figure 12: extrait de showGame.html.twig

Notre structure html se compose d'un conteneur principal : game-container, qui possède lui-même 3 conteneurs, le premier sera le message affiché sur mobile en mode portrait, le second sera la zone où sera affichée le captcha en fin de partie et enfin le dernier contiendra le jeu.

Nous pouvons voir au passage que ce fichier est écrit en Twig. Twig est un moteur de template utilisé par défaut dans Symfony. Il s'écrit en html classique et apporte des fonctionnalités écrites entre accolades. On peut apercevoir ici asset() qui sert à appeler des fichiers de ressources (images, css, js…) pour lesquels Twig lors du rendu de la page renverra le chemin relatif pour trouver ce fichier. La fonction trans() permet d'aller rechercher la traduction

d'une chaine de caractère dans les fichiers de traduction du site. Nous n'utilisons que des « variables » pour représenter les textes dans nos fichiers Twig, ceci afin de rassembler ensemble toutes les traductions liées à une langue. Ainsi notre application n'est pas directement dépendante d'une langue, il sera très facile de changer celle-ci ou d'apporter d'autres versions pour l'internationalisation du site.

Style Css

```
.mobile-portrait{
    display: none;
}

.picto {
    max-width: 80px;
}

.mobile-message{
    text-align: center;
    position: absolute;
    left: 50%;
    top:50%;
    transform:translate(-50%,-50%);
}

@media (orientation: portrait) and (max-width: 768px) {
    .mobile-portrait {
        display: block;
        position: absolute;
        top:0;
        left:0;
        width: 100%;
        height: 100%;
        background-color: #fbba07;
    }

    #game {
        display: none;
    }

}

@media (orientation: landscape) and (max-width: 768px) {
    #game {
        top:0;
        left:0;
        bottom:0;
        right: 0;
        background-color: black;
        z-index:2000;
        position: fixed;
    }
}
```

Voici la feuille de style nous servant à obliger ce mode paysage sur mobile. Par défaut seul notre conteneur du jeu est visible. Les sélecteurs @media nous permettent d'appliquer du style lié aux spécificités de l'appareil sur lequel s'affiche la page. Quand l'appareil est un mobile (max-width : 768px) et orienté en mode portrait (orientation : portrait), nous affichons notre conteneur mobile-portrait. Ce message vient s'afficher au-dessus du jeu grâce à sa position : absolute.

Quand le mobile est passé en mode paysage, le containeur n'est plus visible et nous modifions le conteneur du jeu #game, pour qu'il occupe tout l'espace disponible passant au-dessus du reste du contenu grâce à position :fixed et z-index. Z-index est une position de calque en css. Sa valeur semble haute ici mais elle est liée à une pratique répandue/courante notamment dans Bootstrap de placer le z-index par défaut à 1000 et d'incrémenter ensuite.

Administration

La partie administration de l'application sera réalisée à l'aide du bundle open source EasyAdmin. Ce bundle nous permet grâce à un fichier de configuration écrit en yaml de mettre en place très simple et rapidement des CRUD sur nos entités. Il a aussi l'avantage d'être très facilement surchargeable pour apporter un traitement particulier aux objets le nécessitant.

EasyAdmin configuration

```yaml
Publication:
    class: AppBundle\Entity\SocialWall\Publication
    label: app.menu.publication
    controller: AppBundle\Controller\Back\PublicationController
    disabled_actions: ['new','delete','edit','search']
    templates:
                list: '::back/publication/list.html.twig'
    list:
        actions:
            - { name: 'preview', icon: 'arrows-alt', css_class: 'btn btn-info', label: '' }
            - { name: 'acceptTweet', icon: 'check', css_class: 'btn btn-success', label: '' }
            - { name: 'refuseTweet', icon: 'ban', css_class: 'btn btn-danger', label: '' }
        fields:
            - { property: 'brand', template: '::back/publication/field_brand.html.twig', label: 'admin.publication.brand' }
            - { property: 'tag', label: 'admin.publication.tag' }
            - { property: 'provider', label: 'admin.publication.provider' }
            - { property: 'provider', label: 'admin.publication.provider' }
            - { property: 'authorName', label: 'admin.publication.authorname' }
            - { property: 'createdAt', label: 'admin.publication.createdat' }
            - { property: 'createdAt', label: 'admin.publication.createdat' }
            - { property: 'content', label: 'admin.publication.content' }
```

Figure 13: extrait de easy_admin.yml, Publication

Voici un extrait de configuration du bundle. Nous déclarons une entité et la classe à laquelle elle fait référence. Dans notre cas nous appelons aussi un contrôleur spécifique pour ajouter de nouvelles actions 'preview', 'acceptTweet', 'refuseTweet'. Nous désactivons aussi la possibilité d'ajouter, modifier ou supprimer une Publication. En effet nos publications sont récupérées automatiquement et ne doivent pas être modifiées par le community manager. Celui-ci doit seulement pouvoir pré-visualiser, accepter ou refuser la publication pour que celle-ci s'affiche ou non sur le mur social de la marque concernée. Nous déclarons également des labels afin de toujours utiliser nos fichiers de traduction.

Surcharge pour l'affichage des publications

```php
/**
 * Override Publication List in Back Office
 * @return Response
 */
public function listPublicationAction()
{
    $this->dispatch(EasyAdminEvents::PRE_LIST);

    $fields = $this->entity['list']['fields'];

    $repo = $this->getDoctrine()->getRepository('AppBundle:SocialWall\Publication');
    $queryBuilder = $repo->queryByBrandAndStatus(
            $this->request->query->get('sortField'),
            $this->request->query->get('sortDirection'),
            $this->request->get('status',0),
            $this->request->get('brand',-1)
            );

    $this->dispatch(EasyAdminEvents::POST_LIST_QUERY_BUILDER, [
```

Figure 14: extrait de PublicationController, affichage des publications

Voici un extrait du contrôleur qui surcharge easy admin pour l'affichage de la liste des publications. Comme sur le wireframe affiché plus précédemment, nous affichons les publications dans 3 onglets en fonction de leur statut. Pour ce faire nous créons une requête spécifique dans le Repository que nous appelons ici avec les paramètres en fonction de l'onglet actuel de l'utilisateur.

Repository

Les classes repository dans Symfony nous servent à déclarer des requêtes DQL (Doctrine Query Language), ces requêtes sont interprétées par Doctrine pour les traduire ensuite en requête SQL à notre base de données.

```php
public function queryByBrandAndStatus($sortField=null, $sortDirection='ASC', $status=null, $brand=-1)
{
    $query = $this->getInstance('publication');
    if (null !== $status){
        $query->where('publication.status = :status');
        $query->setParameter(':status',$status);
    }

    if (-1 != $brand){
        $query->join('publication.tag','tag');
        if ($brand == 0){
            $query->andWhere('tag.brand IS NULL');
        }else{
            $query->join('tag.brand','brand');
            $query->andWhere('brand.id = :brand');
            $query->setParameter(':brand',$brand);
        }
    }
    if (null !== $sortField) {
        $query->orderBy('publication.'.$sortField, $sortDirection);
    }

    return $query;
}
```

Figure 15: extrait de PublicationRepository, requête par marque et statut

Cette fonction nous permet de construire la requête récupérant la liste des publications, triée sur la colonne $sortField, dans le sens $sortDirection. Nous pouvons également lui passer un $status particulier ainsi qu'une $brand. Si aucun attribut n'est passé à la fonction, elle renvoie la liste de toutes publications quelques soient le statut et la marque.

Exemple d'appel à cette fonction avec les valeurs suivantes :
queryByBrandAndStatus('createdAt', 'DESC', 1, 2)

Requête SQL exécutée :

```
SELECT p.id, p.created, p.content, p.mediaPath, p.mediaType, p.authorName,
p.authorPicture, p.providerPath, p.status, p.tag_id
FROM publication p
INNER JOIN hashtag h ON p.tag_id = h.id
INNER JOIN brand b ON h.brand_id = b.id
WHERE p.status = 1
AND b.id = 2
ORDER BY p.createdAt DESC
```

Intégration de Victoire

Déclarer les entités comme objets métier

La première étape dans l'intégration du CMS est de spécifier dans chaque entité utilisée par Victoire quels sont les arguments manipulables dans l'interface.

```
/**
 * Brand
 *
 * @ORM\Table()
 * @ORM\Entity
 * @VIC\BusinessEntity({"Listing", "Cover", "Image"})
 * @Vich\Uploadable
 */
class Brand
{
    use BusinessEntityTrait;
```

Figure 16: extrait de l'entité Brand, BusinessEntity

Il nous suffit d'ajouter l'annotation à la déclaration de la classe ainsi que la liste des widgets qui auront accès aux paramètres de l'objet métier en question.

L'utilisation du Trait, BusinessEntityTrait, apporte les attributs et méthodes nécessaires au bon fonctionnement de Victoire.

```php
/**
 * @var string
 * @Assert\NotBlank
 * @ORM\Column(name="name", type="string", length=30)
 * @VIC\BusinessProperty({"textable", "businessParameter", "seoable"})
 */
private $name;

/**
 * @Vich\UploadableField(mapping="brand_cover", fileNameProperty="cover")
 * @Assert\Image()
 * @VIC\BusinessProperty("imageable")
 * @var File $cover
 */
private $coverFile;
```

Figure 17: extrait de l'entité Brand, BusinessProperty

BusinessProperty nous sert à définir le type des attributs de la classe, l'ajout de 'businessParameter' permet ensuite de créer des représentations métier à partir de ce paramètre. Dans notre cas de Brand, nous créerons une page par marque.

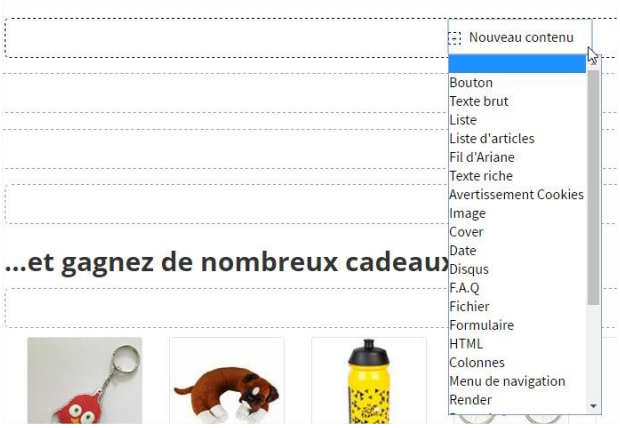
Construction des pages avec victoire

Une fois arrivé à cette étape, la construction des pages et l'assemblage des différents widgets, se font directement via l'interface de Victoire en front. Nous installons le projet sur le serveur de pré-production afin que toute l'équipe puisse travailler ensemble.

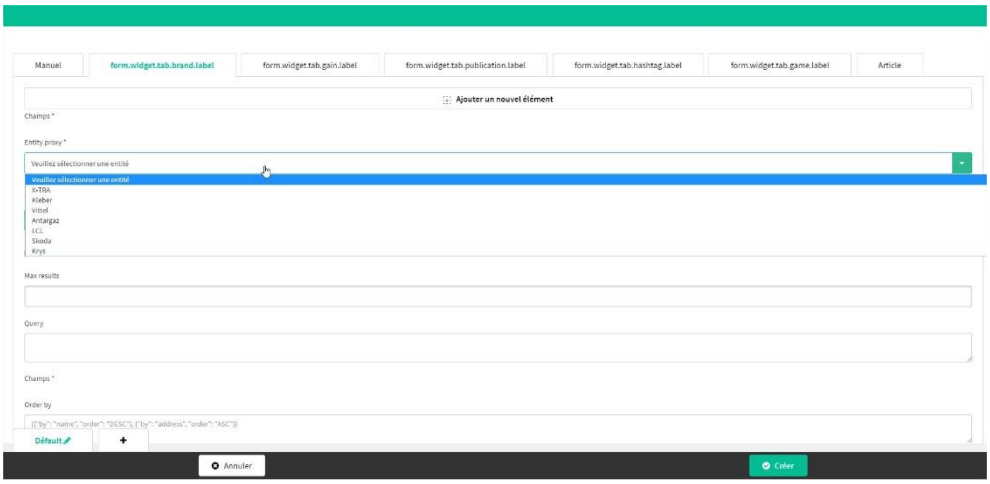
Figure 18: Victoire, mode de visualisation

Victoire nous met à disposition 4 modes de visualisation de la page :

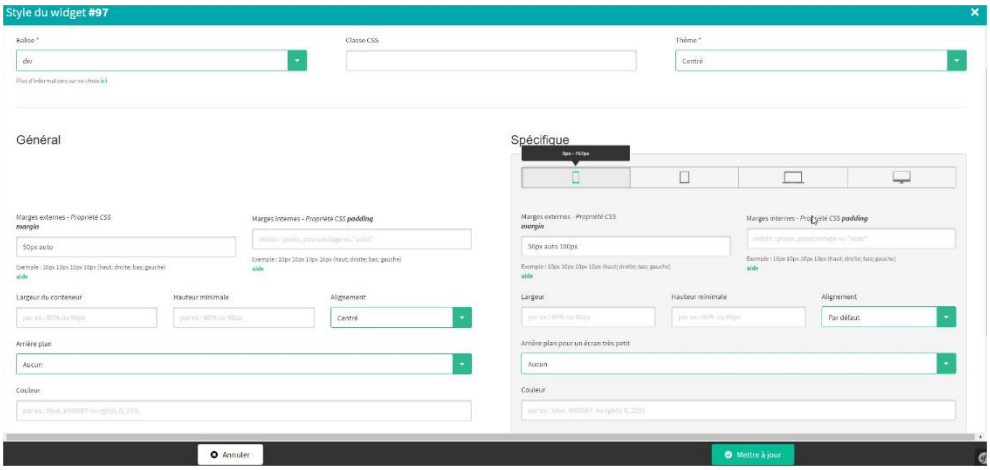
- Aperçu, pour naviguer normalement dans le site
- Création, pour ajouter du contenu
- Edition, pour modifier le contenu existant
- Style, modifier/personnaliser le style



L'outil création nous fait apparaitre des emplacements disponibles où il suffit de cliquer dessus pour ajouter le widget souhaité

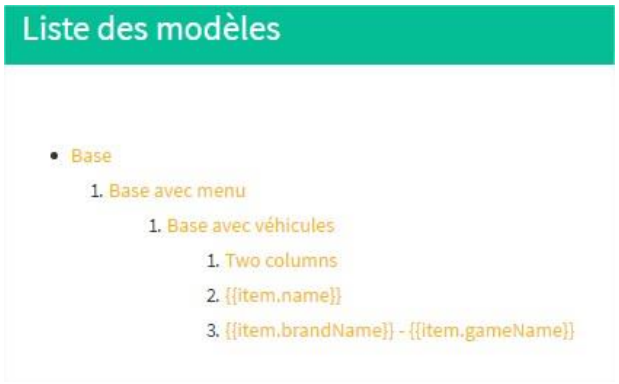


Quand un widget propose du contenu lié aux objets métiers, des onglets pour chaque classe apparaissent. Il est alors possible d'utiliser 1 objet en particulier (Entity proxy) ou une requête personnalisable en DQL.



L'onglet Style nous propose de personnaliser entièrement l'apparence graphique du widget. Il est possible d'apporter un comportement particulier suivant la résolution d'écran. Nous pouvons également modifier la balise html utilisé pour respecter au mieux les standards Html5.

Les pages sont construites à partir de modèles dont elles héritent les widgets. Il est possible de modifier, surcharger ou supprimer un widget hérité du modèle et ce pour chaque page de l'application. Cette fonctionnalité nous permet sur le modèle de page des marques d'avoir un contenu alimenté par l'objet Brand (Cover, Titre, SocialWall …) mais d'apporter un contenu unique à celle-ci simplement en ajoutant des widgets au milieux de ceux existants.



Recette

Recette interne

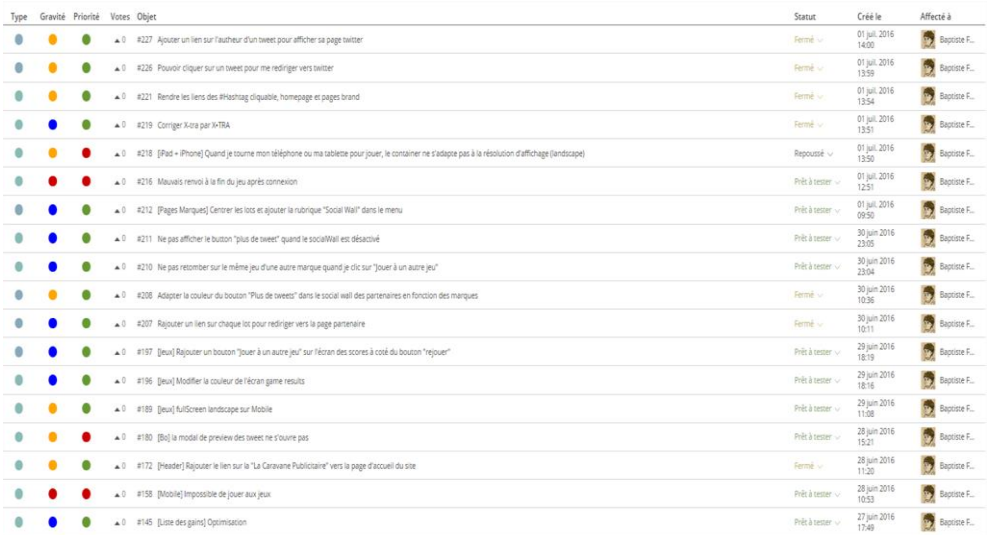
Une fois l'intégration terminée, nous attaquons la phase de recette interne. Sur la base d'un document reprenant l'ensemble des fonctionnalités de l'application, page par page, nous venons les tester une à une, sur différents navigateurs et mobiles.

PAGE MARQUE ANTARGAZ				
Je vois		La cover est bonne	OK	
		Je vois bien le logo sur le header	OK	
		Je vois le bouton Tweets et j'accède aux tweets de la marque	OK	
		L'instant gagnant apparait bien au bon timing	OK	
Je vois		Le hashtag twitter est cliquable	NOK	
		Je vois les icones sociaux et je peux cliquer vers les bons liens	OK	
		Je vois le social wall et plus de tweets	OK	
Je vois		Je vois la bannière et je peux cliquer, elle renvoit vers le site antargaz	OK	
		Médias, je vois les 2 vidéos, je peux les lire, les arrêter	OK	
		Je vois la photo Antargaz dans médias	OK	
COMPTE				
CRÉATION DE COMPTE				
Je peux		me connecter au site via le bouton se connecter dans le header	OK	
	et	je suis renvoyé vers la page de connexion	OK	
	et	si je rentre les mauvais identifiants, une notification me prévient	OK	
Je peux		me connecter avec facebook	OK	
Je peux		utiliser l'oubli de mot de passe	NOK	ne fonctionne pas
Je peux	cliquer sur	créer un compte	OK	
	et	je suis renvoyé vers la page de créationd e compte	OK	
Je ne peux pas		valider la création si tous les champs obligatoires ne sont pas remplis	OK	
Je peux		créer mon compte une fois toutes les infos obligatoires indiquées	OK	
Je peux		m'inscrire à la newsletter des offres commerciales	OK	
	et	les infos sont indiquées dans la base de données	OK	
Je peux		me déconnecter	OK	
ESPACE PERSONNEL				
Je VOIS		3 onglets sur la page - informations / gains / tickets	OK	
	et	je peux naviguer sur chacun des espaces	OK	
Je vois	dans l'onglet mes informations	celles que j'ai rentré à l'inscription	NOK	J'ai coché la case pour les offres commerciales à l'inscription et là elle n'est pas cochée.
	et je peux	modifier/compléter mes informations	OK	
Je vois	dans l'onglet mes gains	je vois aucun gain quand je n'ai rien gagné encore	OK	
	et	je vois mes gains lorsque j'en ai gagné	NT	
Je vois	dans l'onglet mes tickets	la grille qui permet de voir le nombre de jeux auquels j'ai joué	OK	
	et	je vois une indication lorsque j'ai joué à un jeu	OK	

Figure 19: Extrait du document de recette interne

Chaque testeur vient noter OK ou Non OK pour la fonctionnalité ou l'élément vérifié. Une fois l'ensemble de l'application testé, nous faisons une synthèse des problèmes rencontrés et nous créons des tickets décrivant le problème accompagné d'impressions d'écran si besoin. Les tickets sont priorisés en fonction de la gravité du problème rencontré et de l'urgence à régler ce ticket.

Résolutions des tickets



Type	Gravité	Priorité	Votes	Objet	Statut	Créé le	Affecté à
			▲0	#227 Ajouter un lien sur l'auteur d'un tweet pour afficher sa page twitter	Fermé	01 juil. 2016 14:00	Baptiste F...
			▲0	#226 Pouvoir cliquer sur un tweet pour me rediriger vers twitter	Fermé	01 juil. 2016 13:59	Baptiste F...
			▲0	#221 Rendre les liens des #Hashtag cliquable, homepage et pages brand	Fermé	01 juil. 2016 13:54	Baptiste F...
			▲0	#219 Corriger X-tra par X+TRA	Fermé	01 juil. 2016 13:51	Baptiste F...
			▲0	#218 [IPad + iPhone] Quand je tourne mon téléphone ou ma tablette pour jouer, le container ne s'adapte pas à la résolution d'affichage (landscape)	Repoussé	01 juil. 2016 13:50	Baptiste F...
			▲0	#216 Mauvais renvoi à la fin du jeu après connexion	Prêt à tester	01 juil. 2016 12:51	Baptiste F...
			▲0	#212 [Pages Marques] Centrer les logs et ajouter la rubrique "Social Wall" dans le menu	Prêt à tester	01 juil. 2016 09:50	Baptiste F...
			▲0	#211 Ne pas afficher le button "plus de tweet" quand le socialWall est désactivé	Prêt à tester	30 juin 2016 23:05	Baptiste F...
			▲0	#210 Ne pas retomber sur le même jeu d'une autre marque quand je clic sur "Jouer à un autre jeu"	Prêt à tester	30 juin 2016 23:04	Baptiste F...
			▲0	#208 Adapter la couleur du bouton "Plus de tweets" dans le social wall des partenaires en fonction des marques	Fermé	30 juin 2016 10:36	Baptiste F...
			▲0	#207 Rajouter un lien sur chaque lot pour rediriger vers la page partenaire	Fermé	30 juin 2016 10:11	Baptiste F...
			▲0	#197 [Jeux] Rajouter un bouton "Jouer à un autre jeu" sur l'écran des scores à coté du bouton "rejouer"	Prêt à tester	29 juin 2016 18:19	Baptiste F...
			▲0	#196 [Jeux] Modifier la couleur de l'écran game results	Prêt à tester	29 juin 2016 18:16	Baptiste F...
			▲0	#189 [Jeux] fullScreen landscape sur Mobile	Prêt à tester	29 juin 2016 11:08	Baptiste F...
			▲0	#180 [Bo] la modal de preview des tweet ne s'ouvre pas	Prêt à tester	28 juin 2016 15:21	Baptiste F...
			▲0	#172 [Header] Rajouter le lien sur la "La Caravane Publicitaire" vers la page d'accueil du site	Fermé	28 juin 2016 11:20	Baptiste F...
			▲0	#158 [Mobile] Impossible de jouer aux jeux	Prêt à tester	28 juin 2016 10:53	Baptiste F...
			▲0	#145 [Liste des gains] Optimisation	Prêt à tester	27 juin 2016 17:49	Baptiste F...

Figure 20: extrait des tickets créés lors de la phase de recette

Chaque développeur vient ensuite traiter les tickets un par un selon l'ordre de priorité indiqué. Une fois le problème résolu, nous mettons le statut du ticket à « prêt à tester » afin qu'une autre personne vienne tester une nouvelle fois l'élément pour valider la correction.

Recette client

Une fois cette première phase de recette interne réalisée, nous invitons notre client à réaliser à son tour une recette de l'application sur la même base de documents. Lorsque l'ensemble des fonctionnalités attendues répond aux exigences souhaitées, nous déployons l'application sur son serveur de production.

Une fois en ligne, l'application reste disponible sur le serveur de pré-production et le système de ticket continue d'être utilisé pour remonter d'éventuelles anomalies constatées en production. Dans ce cas un correctif est appliqué à la pré-production, testé et validé, puis ensuite appliqué sur l'application en production.

Bilan

Bilan du projet La caravane du tour

Le site a été mis en ligne comme prévu pour le lancement du Tour de France le 2 juillet. En 3 semaines nous avons comptabilisé plus de 230 000 visiteurs uniques et 500 000 parties jouées dont 50% des utilisateurs étaient sur des supports mobiles (téléphones et tablettes).

Délais

Comparé au planning prévisionnel nous avons accusé un léger retard sur la partie réalisation principalement dû au lancement plus tardif que prévu des développements. Initialement prévu du 30 mai au 15 juin, 2 développeurs ont commencé le projet avec 1 semaine de retard et l'intégration s'est finie le 25 juin nous laissant 1 semaine pour la phase de recette et de mise en ligne.

Périmètre

Le périmètre de l'application a été légèrement réduit, notamment la partie SocialWall qui prévoyait de récupérer les publications de plusieurs réseaux sociaux. Les difficultés techniques liées à la mise à jour de certaines API (notamment Instagram) et également le retour des marques fournissant uniquement des comptes et hashtag Twitter, nous avons décidé avec A.S.O de limiter la récupération de messages uniquement depuis Twitter.

Difficultés rencontrées

Intégration des jeux

Les jeux, développés par Casus Ludi, ont d'abord été intégrés dans des balises iframe occasionnant de gros soucis de compatibilité notamment sur mobile. Nous avons rassemblé 1 développeur de chaque équipe pendant 1 journée pour réussir à optimiser et intégrer les jeux directement dans les différentes pages.

Serveur et version des logiciels

Pour supporter la charge prévue, nous avons installé un serveur dédié entièrement affecté à ce site. Certains services, tels que MySQL 5.7, n'étaient pas encore supportés par Symfony et Doctrine au moment du déploiement. Utilisant Docker il a été très simple et rapide pour nous de repasser MySQL en version 5.6 pour éviter ce conflit.

Annexes

Docker

```
caravane-memcached:
  image: memcached:1.4.25
  container_name: caravane-memcached

caravane-redis:
  image: redis:3.0.7
  container_name: caravane-redis

caravane-mysql:
  image: mysql:latest
  container_name: caravane-mysql
  environment:
    - MYSQL_ROOT_PASSWORD=root-password
    - MYSQL_DATABASE=caravane
    - MYSQL_USER=caravane
    - MYSQL_PASSWORD=pass
  ports:
    - "3306:3306"

caravane-webserver:
  build: .
  dockerfile: docker/Dockerfile.nginx.conf
  container_name: caravane-webserver
  volumes:
      - .:/var/www/caravane
  ports:
   - "2000:80"
  links:
   - caravane-php-fpm

caravane-php-fpm:
  build: .
  dockerfile: docker/Dockerfile.php-fpm.conf
  container_name: caravane-php-fpm
  volumes:
    - .:/var/www/caravane
  links:
    - caravane-memcached
    - caravane-mysql
    - caravane-redis
  environment:
    - SYMFONY__ASSETIC__SASS__BIN=/usr/local/bin/sass
    - SYMFONY__BOWER_BIN='/usr/local/bin/bower --allow-root'
    - SYMFONY__DATABASE_HOST=caravane-mysql
    - SYMFONY__DATABASE_NAME=caravane
    - SYMFONY__DATABASE_USER=caravane
    - SYMFONY__DATABASE_PASSWORD=pass
    - SYMFONY__LOG_DIR=/tmp/caravane/logs
    - SYMFONY__MEMCACHED_HOST=caravane-memcached
    - SYMFONY__MEMCACHED_PORT=11211
    - SYMFONY__NODE_PATH=/usr/bin/node
    - SYMFONY__SESSION_REDIS_PATH=redis://caravane-redis/2
    - SYMFONY__SESSION_REDIS_PATH_TEST=redis://caravane-redis/4
    - SYMFONY__CACHE_DIR=/tmp/caravane/cache
    - SYMFONY__VICTOIRE_REDIS_PATH=redis://caravane-redis
    - SYMFONY__VICTOIRE_REDIS_PATH_TEST=redis://caravane-redis/3
```

Fichier docker-compose en environnement de développement

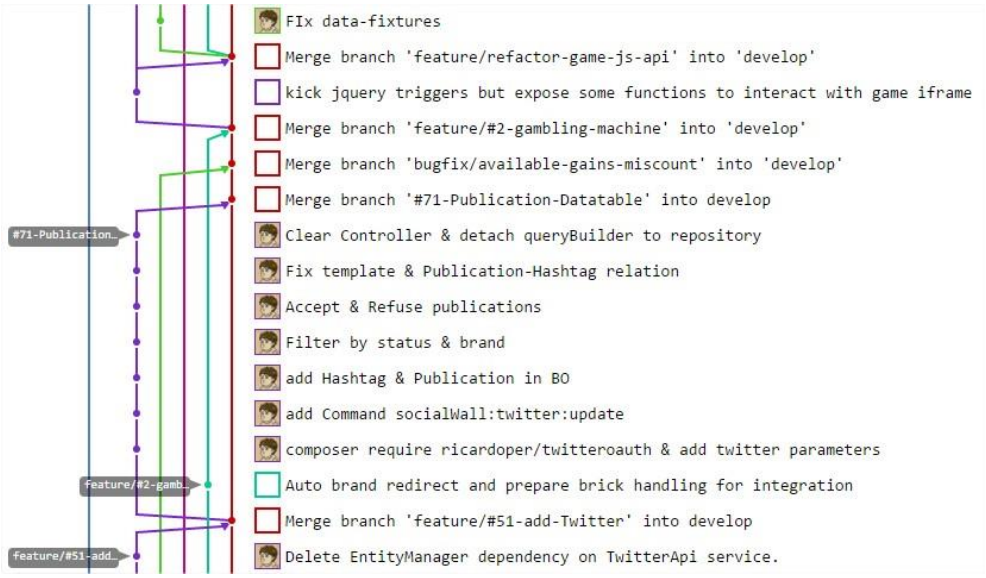
Fixtures

```
AppBundle\Entity\Brand:
    brand1:
        name: Xtra
        color: '#D42D08'
        updatedAt: <dateTime('now')>
        logo: <image("brand/logo", 300, 150, 'sports')>
        coloredLogo: <image("brand/logo", 150, 150, 'sports')>
        vehicleImage: <image("brand/vehicle", 200, 100, 'transport')>
        banner: <image("brand/banner", 1000, 300, 'sports')>
        url: http://www.x-tra.fr/
        socialWallEnabled: true
    brand2:
        name: Kleber
        color: '#E20026'
        updatedAt: <dateTime('now')>
        logo: <image("brand/logo", 300, 150, 'sports')>
        coloredLogo: <image("brand/logo", 150, 150, 'sports')>
        vehicleImage: <image("brand/vehicle", 200, 100, 'transport')>
        banner: <image("brand/banner", 1000, 300, 'sports')>
        url: http://www.kleber.fr/
        socialWallEnabled: true
    brand3:
        name: Vittel
        color: '#FF0017'
        updatedAt: <dateTime('now')>
        logo: <image("brand/logo", 300, 150, 'sports')>
        coloredLogo: <image("brand/logo", 150, 150, 'sports')>
        vehicleImage: <image("brand/vehicle", 200, 100, 'transport')>
        banner: <image("brand/banner", 1000, 300, 'sports')>
        url: https://www.vittel.com/fr/
        socialWallEnabled: true
    brand4:
        name: Antargaz
        color: '#70BC1E'
        updatedAt: <dateTime('now')>
        logo: <image("brand/logo", 300, 150, 'sports')>
        coloredLogo: <image("brand/logo", 150, 150, 'sports')>
        vehicleImage: <image("brand/vehicle", 200, 100, 'transport')>
        banner: <image("brand/banner", 1000, 300, 'sports')>
        url: https://www.antargaz.fr/
        socialWallEnabled: true
```

Extrait d'un fichier des fixtures

Différents fichiers de fixtures sont créés pour peupler la base de données lors des déploiements en test, pré-production. Certains sont classés dans un répertoire 'seeds' et servent également au déploiement sur le serveur de production pour apporter un jeu de données vitale au bon fonctionnement de l'application.

Git

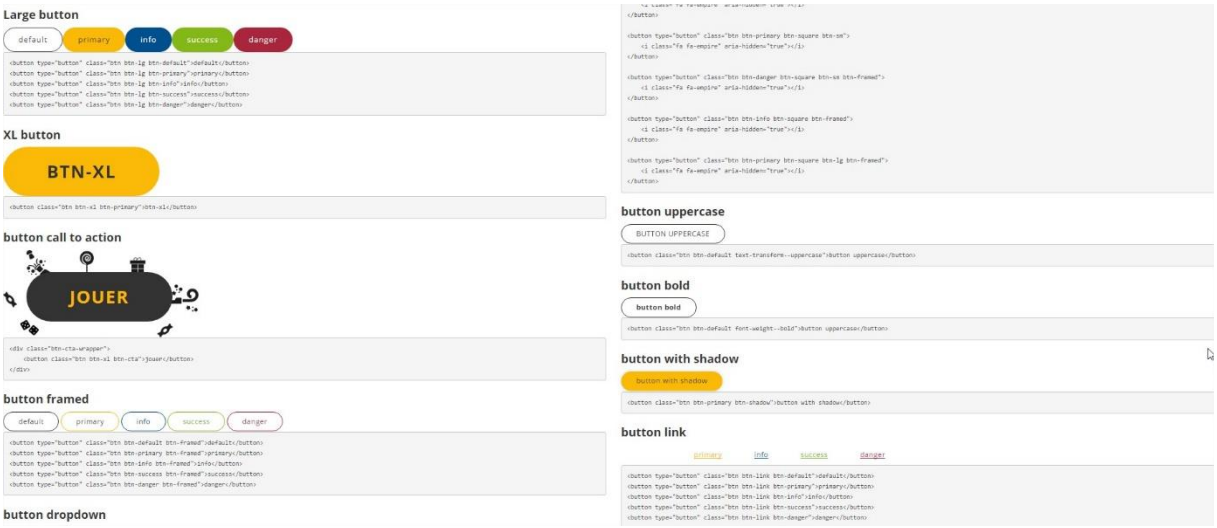


Extrait du dépôt git

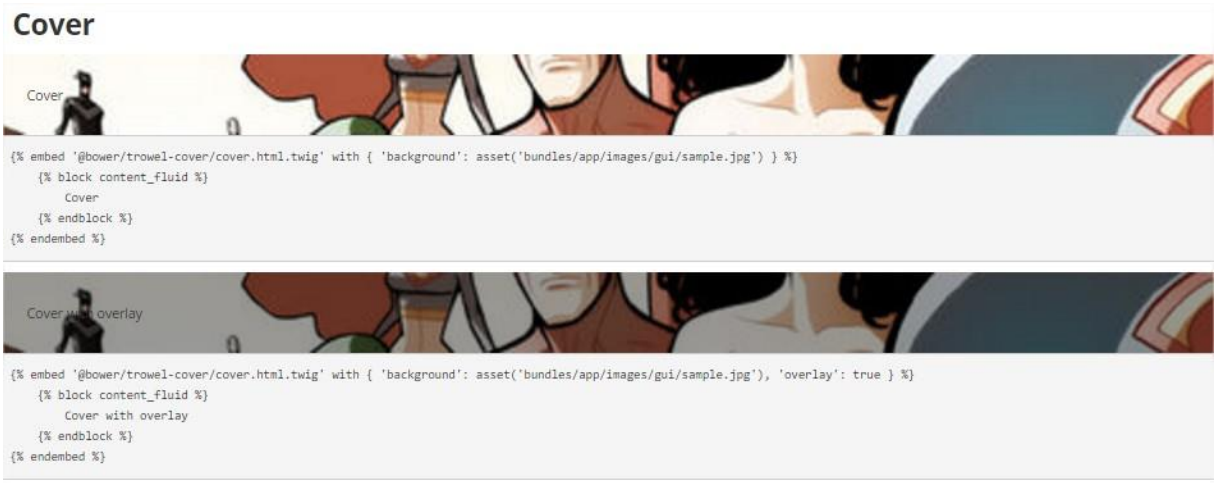
Nous utilisons Git pour le développement et versionning de l'application. Nous appliquons un pattern proposé par Git Flow :

- 1 branche master de production
- 1 branche develop (reflet de la pré-production)
- Les développeurs créent une nouvelle branche 'feature/', 'hotfix', 'bugfix' pour chaque nouvelle tâche.
- Une fois la tâche terminée, nous faisons une demande de pull request sur 'develop' ou 'master' qui doit être relue et validée par un autre développeur, si possible plus expérimenté.

GUI



Extrait de GUI



Extrait de GUI

Liens

Victoire DCMS

https://github.com/Victoire

Widgets disponibles pour **Victoire**

https://github.com/FriendsOfVictoire

Trowel framework front-end

https://github.com/Trowel

Invision, logiciel de maquettage, prototypage collaboratif

https://www.invisionapp.com/

Taiga, logiciel de gestion de projets agiles

https://taiga.io/

Slack, logiciel de communication, partage de fichiers avec une forte inter connexion possible (git, taiga, Google analytics …)

https://slack.com/