Projet G.S.B Web

Table des matières

I - Contexte	2
II - Analyse des besoins	2
A- Énoncer le besoin	2
B- Besoin de sécurité	
C- Besoin réglementaire	2
D- Objectif et Sprint	3
1- Sprint 1	3
a - Besoin 1	3
b - Besoin 2	
c – Besoin 3	7
d – Bilan du sprint	
2- Sprint 2	9
a - Besoin 1	9
b - Besoin 2	10
c – Bilan du sprint	
E – Bilan de la production	
III - Compétences validées	

I - Contexte

Le laboratoire Galaxy Swiss Bourdin (GSB) souhaite souhaite désormais entamer une transformation digitale efficace. Les visiteurs récupèrent une information directe sur le terrain. Ceci concerne aussi bien le niveau de la confiance qu'inspire le laboratoire que la lisibilité des notices d'utilisation des médicaments ou encore les éventuels problèmes rencontrés lors de leur utilisation, etc. Ces informations ne sont actuellement pas systématiquement remontées au siège, ou elles le sont dans des délais jugés trop longs. Le service rédaction qui produit les notices souhaite avoir des remontées plus régulières et directes. Ceci permettra également au service labo-recherche d'identifier et d'engager des évaluations complémentaires. Au moment de l'embauche d'un visiteur médicale, les ressources humaines doivent être capable de créer son compte utilisateur en renseignant ses informations (login, mot de passe, nom, prénom, adresse, ...).

II - Analyse des besoins

A- Énoncer le besoin

Afin de d'optimiser l'exploitation des données terrains produites par les visiteurs médicaux qu'ils remontent de leurs visites auprès des différents prescripteurs (médecins, pharmaciens, infirmières et autres métiers de santé), nous proposons de gérer les rapports des visites via une application web pour que les service rédaction et labo-recherche puissent analyser les données. De plus cela permettra aux responsables secteur, de pouvoir générer des indicateurs de performance de suivi de visite.

B- Besoin de sécurité

L'environnement doit être accessible aux seuls acteurs de l'entreprise. Une authentification préalable sera nécessaire pour l'accès au contenu. Une politique d'accès aux ressources suivant le rôle utilisateur sera définie. Les visiteurs médicaux pourront créer, modifier et visualiser leurs rapports. Les données transitant sur les réseaux étant sensible, l'application devra assurer la sécurité transmission des informations.

C- Besoin réglementaire

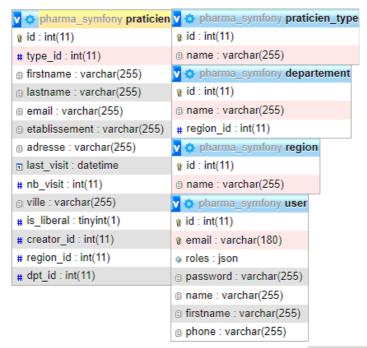
Les laboratoires achètent des fichiers à des organismes spécialisés qui donnent, outre les diverses informations d'état civil et d'origine, les données sur l'influence du praticien sur son entourage professionnel (est-il prescripteur, membre d'une association, relais de l'ordre des médecins...). Dans leurs spécialités, les praticiens possèdent des diplômes (on ne conserve que le plus haut niveau) et ont un coefficient de prescription (sont-ils reconnus par leurs collègues comme référents sur la spécialité, sont-ils dans un cabinet pointu sur le sujet, etc). Les données récupérées sur les praticiens sont confidentielles et doivent respecter le règlement général sur la protection des données (RGPD).

D- Objectif et Sprint

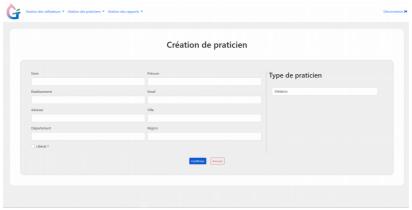
1- Sprint 1

a - Besoin 1

Le premier besoin consiste à gérer les praticiens, c'est à dire créer un profil de praticien, pouvoir lire ce profil, modifier les informations du profil et supprimer le profil. Un profil de praticien contient les informations suivantes : un nom, prénom, une adresse mail, une adresse postale, son métier, son établissement et/ou si il est libéral. Pour des raisons statistiques on enregistrera la personne qui a créé le profil, la date de la dernière visite du praticien ainsi que le nombre de visite qu'il a reçut. Dans notre base de donnée cela se présente comme tel :



Nous avons créé une entity praticien puis utilisé la commande : php bin/console make:crud et sélectionné l'entité praticien. La commande créé automatiquement un controller un répository ainsi que les vues pour faire les opérations Create Update Delete Edit. Nous les avons ensuite retouché pour correspondre à nos besoins.



Les régions, départements, métiers du praticien sont déjà entré dans la base de données et se sélectionne sous forme de liste dans le formulaire.

b - Besoin 2

Le second besoin consiste à ajouter un module de recherche pour trier la liste de praticien afin que l'on puisse rechercher les praticien selon les critères suivant : la ville, le département, la région, si il a été visité récemment ou non, la date de la dernière visite ainsi que le métier du praticien.

Nus avons donc utilisé une carte interactive fourni par le bundle leaflet, ainsi qu'une liste affichant les noms, prénoms, l'établissement ou son statut libéral ainsi que son métier.

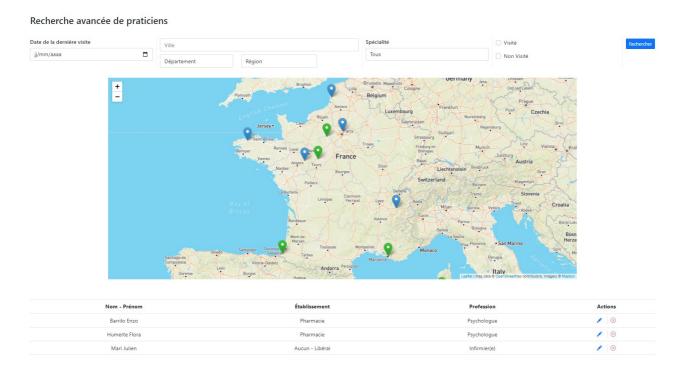
Pour cela nous avons utilisé des requête api en javascript sur la page html pour la carte et des requête SQL en PHP pour la recherche par critères.

```
datePickerId.max = new Date().toISOString().split("T")[0];
              window.addEventListener("load", function (event) {
   var mymap = L.map('map').setView([46.925539, 2.409112], 6);
                   L.tileLayer('https://api.mapbox.com/styles/vl/{id}/tiles/{z}/{x}/{y}?access_token={accessToken}', {
                       attribution: 'Map data © <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors, Imagery ⊕ <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
                       maxZoom: 18,
                       id: 'mapbox/streets-v11'.
                       tileSize: 512,
                       zoomOffset: -1
                        accessToken: 'pk.eyJlIjoieG9YY2VzIiwiYSI6ImNrdzZlMG9zYjJsZDAyb25vMWNidmdnb2qifQ.20frcp606JrWKD42dLzVeA
198
199
                  var adresses = document.getElementsBvClassName('data-of-praticien');
                  for (var i = 0: i < adresses.length: i++) {
                       var formatedAdress = adresses[i].dataset.adresse.split(' ').join('+');
                       console.log(formatedAdress);
                       window,fetch('https://api-adresse.data.gouv.fr/search/?g=' + formatedAdress).then(function (response) {
                            return response.json();
                       }).then(function (data) {
                            var adr = document.getElementsByClassName('data-of-praticien');
                            for (var j = 0; j < adr.length; j++) {
    if (adr[j].dataset.adresse === data.query) {</pre>
                                     var formatedName = adr[j].dataset.formatedname;
                                     var isLiberal = adr[j].dataset.liberal;
                            var coord = data.features[0].geometry.coordinates;
                            var pinLon = coord[0]:
                            var pinLat = coord[1];
                            var pinAdress = data.features[0].properties.label;
                            console.log(isLiberal):
                                var greenIcon = new L.Icon({
    iconUrl: 'https://raw.githubusercontent.com/pointhi/leaflet-color-markers/master/img/marker-icon-2x-green.png',
                                     shadowUrl: 'https://cdnjs.cloudflare.com/ajax/libs/leaflet/0.7.7/images/marker-shadow.png',
                                     iconSize: [25, 41],
                                     iconAnchor: [12, 41]
                                     popupAnchor: [1, -34],
                                     shadowSize: [41, 41]
                                }):
                                var marker = L.marker([pinLat, pinLon],{icon: greenIcon}).addTo(mymap);
                            }else {
                                var marker = L.marker([pinLat, pinLon]).addTo(mymap);
                            marker.bindPopup("<b>" + formatedName + "</b><br>" + pinAdress);
                       })
                  }
              });
          </script>
```

```
* @Route("/", name="praticien_index", methods={"GET"})
                   public function index(PraticienRepository $praticienRepository, Request $request)
                           $praticiens = $praticienRepository->findAll();
$typeListe = $this->getDoctrine()->getRepository(\App\Entity\PraticienType::class)->findAll();
$dptListe = $this->getDoctrine()->getRepository(Departement::class)->findAll();
$regionListe = $this->getDoctrine()->getRepository(Region::class)->findAll();
                          $ville = $request->get('ville', null);
$dpt = $request->get('dpt', null);
$region = $request->get('region', null);
$type = $request->get('type', null);
$lastVisit = $request->get('stype', null);
$visited = $request->get('visited', false);
$notVisited = $request->get('notVisited', false);
                          if ($visited !== false) {
                                  $visited = true:
                          if ($notVisited !== false) {
                                 $notVisited = true;
                          $vIds = [];
$dIds = [];
$rIds = [];
$tIds = [];
                           $nvIds = [];
                           $result = $praticiens;
                          /** @var QueryBuilder $qb */
$qb = $this->entityManager->createQueryBuilder();
                           if (!empty($ville)) {
                                  rempry(syster) /
sargByVille = $qb->select('v')
->from(Praticien::class, 'v')
->orWhere('v.ville LIKE :search')
->setParameter('search', '%' . $ville . '%')
                                           ->getQuery()->getResult();
                                  $argByVille = $praticiens;
                          }
                          if ($argByVille != null) {
   foreach ($argByVille as $av) {
      $vIds[] = $av->getVille();
}
                         }
                         if (!empty($dpt)) {
                                  $dIds[] = $dpt;
                         } else {
                                 $argByDpt = $qb->select('dpt')
                                          ->from(Departement::class, 'dpt')
->getQuery()->getResult();
                                 if ($argByDpt != null) {
  foreach ($argByDpt as $ad) {
    $dIds[] = $ad->getId();
                                }
                      }
                        if (!empty($region)) {
    $rIds[] = $region;
                                 $argByRegion = $qb->select('r')
                                          ->from(Region::class, 'r')
->getQuery()->getResult();
                                 if ($argByRegion != null) {
  foreach ($argByRegion as $ar) {
    $rIds[] = $ar->getId();
                                }
                      }
                         if (!empty($type)) {
                                  $tIds[] = $type;
                         } else {
                                 $argByType = $qb->select('t')
->from(\App\Entity\PraticienType::class, 't')
                                 --getOuery()-->getResult();
if ($argByType != null) {
  foreach ($argByType as $at) {
    $tIds[] = $at->getId();
}
                                          }
                                }
                          if (!empty($lastVisit)) {
                                 (!empry($(astvisit)) {
    $argByLastVisit = $qb->select('lv')
    ->from(Praticien::class, 'lv')
    ->orWhere('lv.lastVisit LIKE :search')
    ->setParameter('search', '%' . $lastVisit . '%')
    ->getQuery()->getResult();
                         } else {
                                 $argByLastVisit = $praticiens;
```

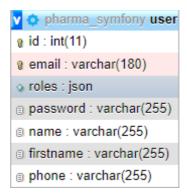
```
if ($argBvLastVisit != null) {
                                             foreach ($argByLastVisit as $lv) {
                                                     $\text{vIds[] = $\text{lv->getId();}
                                 if ($visited === true) {
                                             $argByNbVisit = $qb->select("nv")
                                                        ->from(Praticien::class, 'nv')
                                                        ->orWhere($qb->expr()->isNotNull("nv.nbVisit"))
                            ->getQuery()->getResult();
} elseif ($notVisited === true) {
                                             $argByNbVisit = $qb->select("nv")
                                                       ->from(Praticien::class, 'nv')
->orWhere($qb->expr()->isNull("nv.nbVisit"))
                                                        ->getQuery()->getResult();
                                 } else {
                                            $argByNbVisit = $praticiens;
                                  }
156
                                  if ($argByNbVisit != null) {
                                             foreach ($argByNbVisit as $nv) {
    $nvIds[] = $nv->getNbVisit();
                                  if (sizeof($vIds) == 0 or sizeof($dIds) == 0 or sizeof($rIds) == 0 or sizeof($tIds) == 0 or sizeof($vIds) == 0 or sizeof($nvIds) == 0 or 
                                             $result = [];
                                  } elseif ($visited === true) {
                                             $result = $qb->select('p')
    ->from(Praticien::class, 'p')
                                                       ->andwhere($qb->expr()->in('p.ville', $vIds))
->andwhere($qb->expr()->in('p.dpt', $dIds))
->andwhere($qb->expr()->in('p.region', $rIds))
                                                       ->andWhere($qb->expr()->in('p.type', $tIds))
->andWhere($qb->expr()->in('p.id', $lvIds))
                                                        ->andWhere($qb->expr()->isNotNull("p.nbVisit"))
                                ->getQuery()->getResult();
} elseif ($notVisited === true) {
                                             $result = $qb->select('p')
    ->from(Praticien::class, 'p')
                                                        ->andWhere($qb->expr()->in('p.ville', $vIds))
                                                       ->andwhere($qb->expr()->in('p.dpt', $dIds))
->andwhere($qb->expr()->in('p.region', $rIds))
                                                       ->andWhere($qb->expr()->in('p.type', $tIds))
->andWhere($qb->expr()->in('p.id', $lvIds))
                                                        ->andWhere($qb->expr()->isNull("p.nbVisit"))
                                                        ->getQuery()->getResult();
                               } else {
                                             $result = $qb->select('p')
                                                       ->from(Praticien::class. 'p')
                                                        ->andWhere($qb->expr()->in('p.ville', $vIds))
                                                       ->andwhere(sqb->expr()->in('p.dpt', $dIds))
->andwhere(sqb->expr()->in('p.region', $rIds))
->andwhere(sqb->expr()->in('p.type', $tIds))
->andwhere(sqb->expr()->in('p.id', $lvIds))
                                                        ->getQuery()->getResult();
                                  $praticiens = $result;
                                  return $this->render('praticien/index.html.twig', [
                                               'praticiens' => $praticiens,
'lastVisit' => $lastVisit,
                                               'ville' => $ville,
                                             'dpt' => $dpt,
                                               'region' => $region
                                               'type' => $type,
                                              'visited' => $visited,
'notVisited' => $notVisited,
                                               'typeListe' => $typeListe,
'dptListe' => $dptListe,
                                                'regionListe' => $regionListe,
                                1):
```

Nous avons ensuite modifier le design de la liste des praticiens pour le faire correspondre à la charte visuelle du site.



c - Besoin 3

Le troisième besoin consiste à gérer les comptes utilisateur des employés, c'est à dire créer un profil, pouvoir lire ce profil, modifier les informations du profil et supprimer le profil. Un profil d'employé contient les informations suivantes : un nom, prénom, une adresse mail, un mot de passe et un numéro de téléphone. Pour des raisons de sécurité nous attribuerons un rôle correspondant au département dans lequel est employé la personne afin qu'il n'est accès uniquement aux fonctionnalités qui lui sera utile pour son travail. Voici l'entité dans la base de donnée :



Nous avons créé une entity user grâce au security bundle avec la commande : php bin/console make:user qui génère une entité basique d'utilisateur et la définie comme entité qui servira lors de l'authentification. La commande créé automatiquement un controller un répository ainsi que les vues pour faire les opérations Create Update Delete Edit. Nous les avons ensuite retouché pour correspondre à nos besoins.

Les rôles sont déclarés dans le fichier security.yaml et empêche les utilisateurs d'accéder au contenue qui ne leurs est pas utile.

```
security:
          role_hierarchy:
            ROLE_VISITEUR: [ROLE_USER]
ROLE_RH: [ROLE_USER]
            ROLE_RO: [ROLE_USER]
ROLE_REDACTOR: [ROLE_USER]
ROLE_ADMIN: [ROLE_USER, ROLE_RH, ROLE_VISITEUR, ROLE_RD]
            ROLE_SUPER_ADMIN: [ROLE_USER, ROLE_RH, ROLE_ADMIN, ROLE_VISITEUR, ROLE_RD]
            App\Entity\User:
               algorithm: auto
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
          # https://symfony.com/doc/current/security.html#where-do-users-come-from-user-providers
             # used to reload user from session & other features (e.g. switch_user)
            app_user_provider:
               entity:
                  class: App\Entity\User
                  property: email
          firewalls:
               pattern: ^/(_(profiler|wdt)|css|images|js)/
                security: false
               lazy: true
provider: app_user_provider
                  authenticators:
                      - App\Security\AppCustomAuthenticator
               logout:
                 path: app_logout
# where to redirect after logout
                 # target: app_any_route
               # activate different ways to authenticate
               # https://symfony.com/doc/current/security.html#firewalls-authentication
               # https://symfony.com/doc/current/security/impersonating user.html
               # switch user: true
         # Easy way to control access for large sections of your site
# Note: Only the *first* access control that matches will be used
          access_control:
            ccess_control:
    { path: ^/admin, roles: ROLE_ADMIN }
    { path: ^/rh, roles: ROLE_RH }
    { path: ^/risiteur, roles: ROLE_VISITEUR }
    { path: ^/rapport/visiteur, roles: ROLE_VISITEUR }
    { path: ^/rapport, roles: [ROLE_VISITEUR, ROLE_RD, ROLE_REDACTOR] }
    { path: ^/secured, roles: ROLE_USER }
    { path: ^/secured, roles: ROLE_USER }
}
                                                                  Création d'un nouvel utilisateur
                   Prénon
                 Nom
                  Numéro de téléphone
                 Email
                   Email
                 Mot de passe (8 caracteres)
                   Mot de passe
```

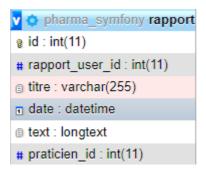
d – Bilan du sprint

Nous avons atteint les objectif attendu, dans le temps qu'il nous a été accordé. Il a fallut plus de temps que prévu avant de commencé la phase de développement. La phase de débogage c'est quand à elle déroulé sans accros.

2- Sprint 2

a - Besoin 1

Le premier besoin consiste à gérer les rapports des visiteurs, c'est à dire créer un rapport, pouvoir lire ce rapport, modifier les informations du rapport et supprimer le rapport. Un rapport contient les informations suivantes : un titre, un créateur, un concerné, un corps ainsi qu'une date d'écriture. Voici l'entité dans la base de donnée :



Nous avons créé une entity praticien puis utilisé la commande : php bin/console make:crud et sélectionné l'entité rapport. La commande créé automatiquement un controller un répository ainsi que les vues pour faire les opérations Create Update Delete Edit. Nous les avons ensuite retouché pour correspondre à nos besoins.

Création d'un rapport

Praticien concerné Confirmer Annuler Liste des rapports Nouveau rapport Date Visiteur Titre Praticien Profession actions 08/05/2022 Infirmier(e) rapport 1 Jaures Roger 10/04/2022 Rapport 1 vermont michel Pneumologue

b - Besoin 2

Le second besoin consiste à pouvoir lire de façon intuitive un rapport nous avons donc refait le design de la page de présentation d'un rapport :



c – Bilan du sprint

Nous avons atteint l'objectif attendu avec quelques jours d'avances. La phase de développement et de débogage se sont déroulé sans accros.

E - Bilan de la production

Les versions on était fournis à temps et tous les objectifs ont été rempli. Néanmoins il reste encore du débogage de problèmes non bloquant. Durant cette réalisation nous avons pu mettre en œuvre nos compétences en PHP, HTML, SCSS, JS, SQL. Nous avons aussi pu nous familiarisé avec le framework Symfony. Cette réalisation a pu consolider la cohésion de cette équipe de développement naissante.

III - Compétences validées

J'ai pu validé au travers de cette réalisation 4 compétences.

- Gérer le patrimoine informatique
- Répondre aux incidents et aux demandes d'assistance et d'évolution
- Travailler en mode projet
- Mettre à disposition des utilisateurs un service informatique

Pour plus d'informations veuillez consulter ce document