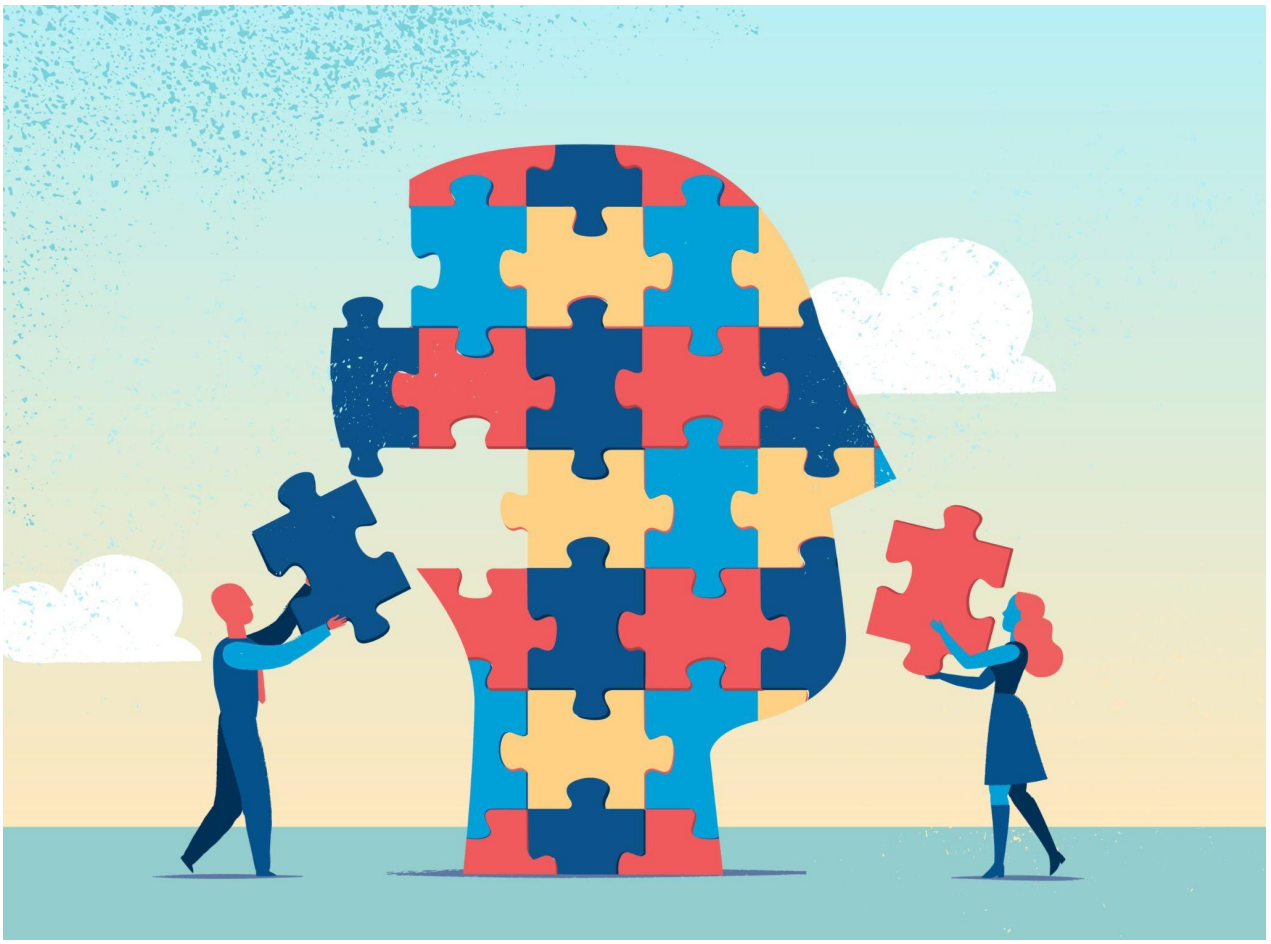
Projet tuteuré FIE 3 Semestre 2



Sujet : Présentation générale de l'application
Rapport technique

Wassim AIT YOUCEF
Johana DAHAN
Benjamin DA COSTA
Victoria SCHWINDENHAMMER
Séraphie MAURY

Résumé

Suite à nos recherches du premier semestre, nous savions à peu près ce que nous voulions faire. Notre projet a malheureusement pris du retard à cause d'un manque de communication entre nous et également du fait que nous étions tous les 5 plus à l'aise avec le front-end qu'avec le back-end. Nous avons donc tous commencé par faire les codes HTML et CSS. Le manque de communication a fait que nous avions trop de pages HTML et que rien n'était optimum. Au bout d'un moment, l'un d'entre nous a donc pris en main tout le front et a tout recommencé afin d'avoir des pages propres et bien agencées. Par ailleurs, nous avons quand même essayé de faire de notre mieux pour mener à bien ce projet en groupe que ce soit pour le front-end ou pour le back-end même si nous avons été un peu trop ambitieux quant à nos capacités pour le réaliser.

Mots-clés : Github, CodeSandbox, front-end, back-end, diagramme UML, fréquence cardiaque, médecin, patient, stéréotypies

Tables des matières

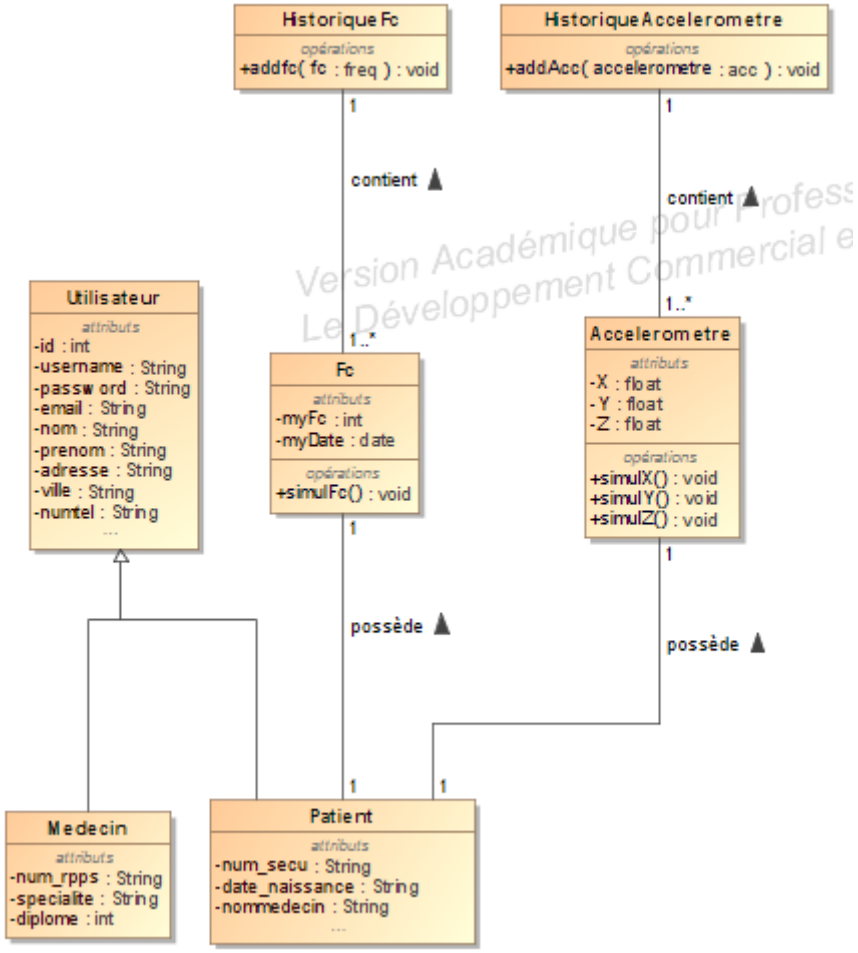
1. Présentation générale de l’application

Dans le cadre de leurs thérapies, les personnes autistes apprennent à gérer leurs crises de stress qui se manifestent par des stéréotypies. Nous nous sommes alors attardés sur la signification que pourraient avoir ces stéréotypies, ainsi que leur fréquence et leur durée sur l’état du patient et la potentielle évolution et efficacité de la thérapie suivie. Nous souhaitons donc trouver un moyen d’avoir un suivi des stéréotypies du patient et permettre au psychiatre chargé de la thérapie de voir si le patient arrive à mieux les gérer. Notre projet viserait en fait à être un indicateur pour le médecin, le patient (et son tuteur si le patient est un enfant) de l’avancée du traitement, de son bon fonctionnement et le cas échéant il permettrait en amont de le réajuster pour l’adapter aux besoins du patient.

En clair, ce projet est destiné à des spécialistes, s’occupant d’enfants ou d’adultes autistes, qui souhaiteraient effectuer un suivi de l’évolution de la fréquence des stéréotypies de leurs patients dans leur vie de tous les jours. Pour coller au mieux à ces contraintes, nous avons envisagé un bracelet connecté qui pourrait détecter les mouvements des différentes stéréotypies possibles et qui les transfèreraient sur un site que nous aurons créé. Sur ce site, le psychiatre pourra ainsi consulter en temps réel l’état de son patient et voir l’évolution de la fréquence et de la durée des stéréotypies afin d’adapter les solutions pour son patient.

Nous avons donc créé une application capable d’afficher, du côté du patient, ses données personnelles et l’évolution de ses stéréotypies et du côté du médecin, ses données personnelles et les données personnelles et l’évolution des stéréotypies de ses patients. Nous aimerions également ajouter une messagerie instantanée qui permettrait au patient et au médecin de communiquer.

2. Modèle de données



Comme vous pouvez le voir nous avons 7 classes dans notre diagramme UML. Les classes Patient et Medecin héritent de la classe Utilisateur. Un patient possède une fréquence cardiaque et une valeur d'accélération sur X, Y et Z. Les données simulées dans simulFc() et simulX(), simulY() et simulZ() sont envoyées dans HistoriqueFc et HistoriqueAccelerometre qui les stockent dans des listes et qui s'afficheront sur un histogramme dans la page suivi du patient. Les classes Fc et Accelerometre sont des entités provisoires car elles sont là uniquement pour la simulation. En effet, si on poursuit notre projet l'année prochaine nous modifierons, sans doute, ce diagramme afin de récupérer les données du bracelet, car nous n'aurons, normalement, plus de méthodes de simulations aléatoires puisque les données arriveront en temps réel.

3. Structure de l’application

Nous avons pris comme modèle la correction du tp galerie. Grâce à cela nous avons pu ordonner plus facilement nos fichiers Java, Javascript, HTML, CSS… (structure de l’application).

Pour l’aspect client, nous avons utilisé CodeSandbox afin d’avoir un environnement partagé et de pouvoir travailler tous en même temps sur les fichiers CSS, HTML et JS. Cependant, après avoir terminé cette partie nous avons dû implémenter toutes nos pages, faites sur CodeSandbox, sur NetBeans afin de les ajouter au projet et de les mettre sur notre dépôt GitHub. Cela a été compliqué et nous nous sommes rendu compte que commencer à coder sur CodeSandbox alors qu’il fallait mettre tout le projet sur Github par la suite n’était pas une très bonne idée. En effet, ajouter nos pages HTML et les faire coïncider avec le modèle du tp Galerie a été plus compliqué que prévu.

Nous avons une vingtaine de pages HTML liées à une page JavaScript et à une page CSS. Ces pages correspondent aux différentes interfaces que nous avons sur notre site internet AutisMono. Parmi elles il y a donc la page d’accueil, la page d’explication de ce qu’est AutisMono, la page de connexion et la page d’inscription. Ensuite, pour le côté patient, il y a la page de suivi avec les diagrammes de suivi de l’évolution du rythme cardiaque, du nombre de crises et également des données de l’accéléromètre du patient. Il y a aussi, la page contact pour que le patient puisse contacter son médecin et enfin la page avec ses données personnelles qu’il peut modifier à tout moment. Côté professionnel de santé, il y a la page de ses données personnelles qu’il peut aussi modifier à tout moment comme le patient et il y a également la page qui lui permet d’avoir accès aux dossiers de ses patients. Pour voir toutes ces pages, il y aura à la fin de ce rapport dans l’annexe les liens vers notre CodeSandbox et notre dépôt Github.

Côté back-end, nous avons fait en sorte que les médecins et patients puissent s’inscrire, aient accès au suivi du patient, à leurs données personnelles… En clair, nous avons fait au mieux pour que toutes nos pages HTML soient fonctionnelles.

4. Explications du code

a. Les controllers

Les controllers ont tous la même utilité, ils contiennent les fonctions que nous utilisons dans notre application. Expliquer les fonctions d'un controller revient à expliquer les fonctions de tous les controllers. Nous allons donc expliquer le fonctionnement du controller PatientController.

```
@GetMapping(path = "show")
public String afficheToutesLesDonnées(Model model) {
    //model.addAttribute("patients", dao.findById(3));
    return "DonneesPatient";
}

@GetMapping(path = "showAll")
public String afficheTousLesPatients(@AuthenticationPrincipal Medecin medecin, Model model) {
    model.addAttribute("patients", dao.findByNommedecin(medecin.getNom()));
    return "listePatients";
}


@GetMapping(path = "add")
public String montreLeFormulairePourAjout(@ModelAttribute("patient") Patient patient) {
    return "formulairePatient";
}
```

Figure 4.a.1 : Fonctions simples

Voici les 3 premières fonctions présentes dans notre controller : ce sont des fonctions qui retournent une page HTML lorsque la requête est demandée. Ces fonctions sont présentes dans tous les controllers.
Les fonctions qui sont spécifiques sont dans LoginAndRegistrationController :

```
@GetMapping("/registrationpatient")
public String registrationpatient(Model model) {
    model.addAttribute("userForm", new Patient());

    return "registrationpatient";
}

@PostMapping("/registrationpatient")
public String registrationpatient(@Valid @ModelAttribute("userForm") Patient userForm, BindingResult bindingResult) {
    userValidator.validate(userForm, bindingResult);

    if (bindingResult.hasErrors()) {
        return "registrationpatient";
    }

    userService.savepatient(userForm);

    securityService.autoLogin(userForm.getUsername(), userForm.getPasswordConfirm());

    return "redirect:/welcome";
}
```

Figure 4.a.2 : Fonctions qui permettent la connexion d'un patient

La première fonction renvoie à la page d'inscription d'un patient, et la deuxième fonction vérifie si les données enregistrées dans le formulaire d'inscription sont valides puis renvoie sur la page d'accueil, dans le cas contraire, la fonction ne valide pas l'inscription et reste sur cette page d'inscription.

b. Les Repositories

```
package AutisMono.dao;

import AutisMono.entity.Utilisateur;
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<Utilisateur, Long> {
    Utilisateur findByUsername(String username);
}
```

Figure 4.b.1 : La classe UserRepository

Voici la classe UserRepository, elle utilise le JpaRepository qui contient beaucoup de fonctions possibles, comme celle que nous voyons ci-dessus qui permet de retrouver un utilisateur par son nom d'utilisateur. Tous nos repositories sont de cette forme.

c. Les Entités

```
@Entity
// Lombok
@Getter
@Setter
@NoArgsConstructor
@RequiredArgsConstructor
@ToString
public class Utilisateur implements UserDetails {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Setter(AccessLevel.NONE)
    private Integer id;

    @NonNull // Lombok
    // Contraintes de taille
    @Size(min = 6, max = 32)
    private String username;

    @NonNull // Lombok
    private String password;

        @NonNull
    private String nom;

    @NonNull
    private String prenom;

    @NonNull
    private String adresse;

     @NonNull
    private String ville;

    @NonNull // Lombok
    @Email // Doit avoir la forme d'une adresse email
    private String email;

    @NonNull
    private String numtel;

    @Transient // Non enregistré dans la BD
    private String passwordConfirm;


@ManyToMany(fetch = FetchType.EAGER)
@Setter(AccessLevel.NONE)
private List<Role> roles = new LinkedList<>();

@Override
public Collection<? extends GrantedAuthority> getAuthorities() {
    List<SimpleGrantedAuthority> authorities = new ArrayList<>();

    for (Role role : roles) {
        authorities.add(new SimpleGrantedAuthority(role.getName()));
    }
    return authorities;
}


@Override
public boolean isAccountNonExpired() {
    return true;
}

@Override
public boolean isAccountNonLocked() {
    return true;
}

@Override
public boolean isCredentialsNonExpired() {
    return true;
}

@Override
public boolean isEnabled() {
    return true;
}
```

Figure 4.c.1 : Classe Utilisateur

Les entités se ressemblent toutes, elles contiennent leurs getters, setters, constructeurs, etc… (tous les @ au début de la classe). Cette classe contient aussi les attributs qui composent l'entité mais aussi des fonctions booléennes qui vérifient si le compte est expiré ou non.

d. Les Services et le Validator

Les services et le validator sont utilisés pour le login, ce code nous a été envoyé par M. Bastide pour nous aider dans la réalisation de ce projet.
Nous avons seulement modifié quelques fonctions pour adapter ce code à notre application, et ainsi, enregistrer des patients et des médecins.

e. Le Front-end

Comme dit dans l'introduction, le front-end de notre application a eu plusieurs phases de développement. La première phase aura été une mise en place trop complexe et trop brouillon. Des pages HTML sans aucun sens, et plusieurs pages CSS qui n'étaient pas vraiment efficaces. Le Front-end à alors dû être "reboot" pour avoir un environnement plus simple et clair.

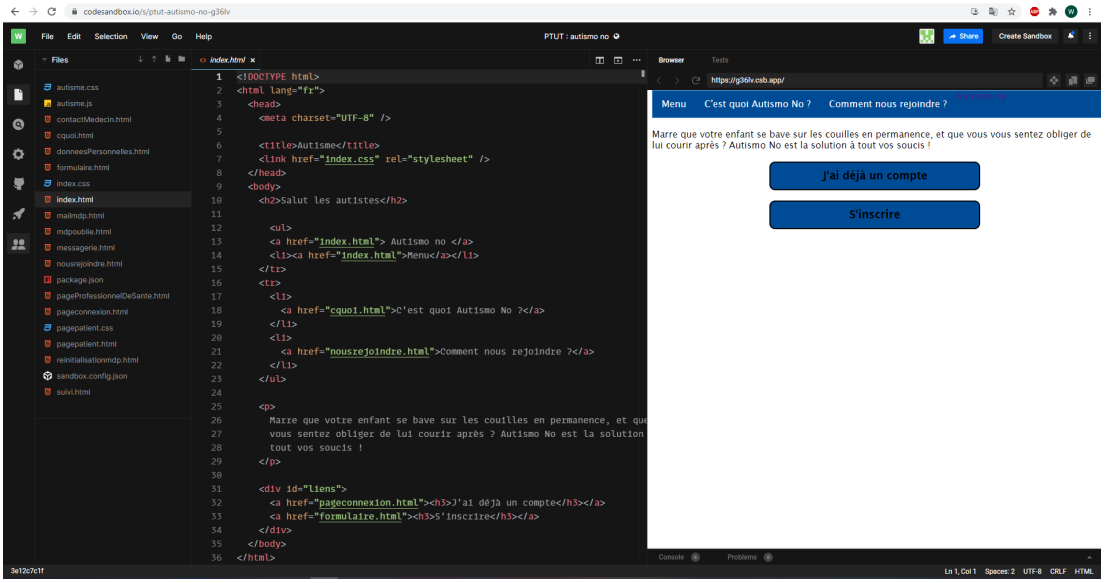


Figure 4.e.1 : Première version du code, totalement obsolète.

C’est pour cela que l’un d’entre nous a pris la décision de repartir du début, créant alors une nouvelle base plus saine sur laquelle travailler.

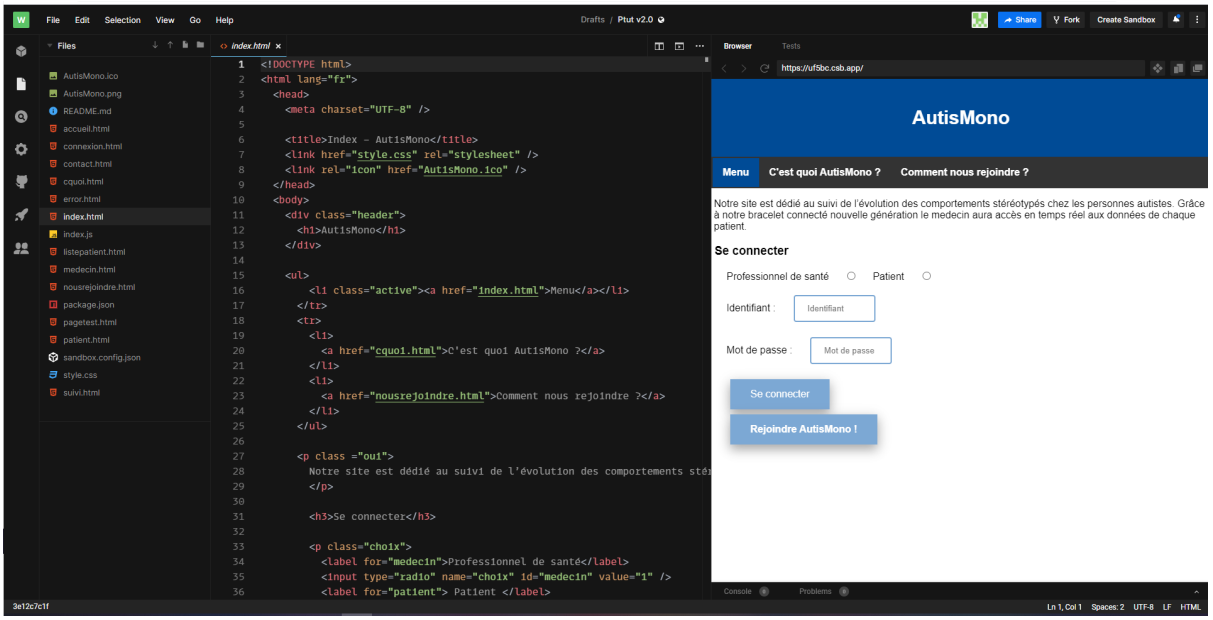


Figure 4.e.2 : Nouvelle version du code, plus claire.

C’est alors avec cette nouvelle base que nous sommes arrivés au résultat final.
Le CSS se dévoile sous la forme d’un seul fichier. Dans ce fichier, il a la simple mise en forme du site ainsi que l’animation du Graph de BPM. Ce dernier sera dans le futur à décliner dans le fichier Javascript afin de pouvoir le mettre à jour en temps réel.

```css
.circle-wrap .circle .mask,
.circle-wrap .circle .fill {
  width: 150px;
  height: 150px;
  position: absolute;
  border-radius: 50%;
}

.circle-wrap .circle .mask {
  clip: rect(0px, 150px, 150px, 75px)
}

.circle-wrap .circle .mask .fill {
  clip: rect(0px, 75px, 150px, 0px);
  background-color: red;
}
.circle-wrap .circle .mask.full,
.circle-wrap .circle .fill {
  animation: fill ease-in-out 3s;
  transform: rotate(120deg);
}

@keyframes fill {
  0% {
    transform: rotate(0deg);
  }
  100% {
    transform: rotate(120deg);
  }
}

.circle-wrap .inside-circle {
  width: 130px;
  height: 130px;
  border-radius: 50%;
  background: #fff;
  line-height: 130px;
  text-align: center;
  margin-top: 10px;
  margin-left: 10px;
  position: absolute;
  z-index: 100;
  font-weight: 700;
  font-size: 2em;
```

Figure 4.e.3 : Code de l'animation du cercle autour du BPM.

Pour ce qui est du javascript, ce dernier nous sert à afficher les graphiques, ainsi qu'à générer un rythme cardiaque aléatoire (en effet, sur le suivi, les valeurs varient fortement, car elles sont aléatoires, donc le graphique s'excite un peu).

```
function makeGraph(container, labels) {
  container = document.getElementById(container);
  labels = document.getElementById(labels);
  var dnl = container.getElementsByTagName("li");
  for (var i = 0; i < dnl.length; i++) {
    var item = dnl.item(i);
    var value = item.innerHTML;
    var color = (item.style.background = color);
    var content = value.split(":");
    value = content[0];
    item.style.height = value + "px";
    item.style.top = 199 - value * 10 + "px";
    item.style.left = i * 50 + 20 + "px";
    item.style.height = value * 10 + "px";
    item.innerHTML = value;
    item.style.visibility = "visible";
    color = content[2];
    if (color !== false) item.style.background = color;
    labels.innerHTML =
      labels.innerHTML +
      "<span style='margin:8px;background:" +
      color +
      "'>" +
      content[1] +
      "</span>";
  }
}

window.onload = function () {
  makeGraph("graph", "labels");
};
```

Figure 4.e.4 : Code de la fonction JS qui affiche le graphe du nombre de crises stéréotypées en fonction des jours.

5. Etapes du projet

a. Problèmes rencontrés

Pendant le déroulement de ce projet, au second semestre, nous avons malheureusement rencontré de nombreux problèmes. Nous avons réussi à en régler quelques-uns mais certains n'ont pas été résolus.
Certains problèmes ont été fréquents, par exemple, lorsque quelqu'un "push" et qu'on veut "pull" ensuite ça fait (parfois) une erreur "head is detached". Nous avons également eu beaucoup de mal à déployer notre application sur Heroku.
Tout d'abord, concernant la simulation des données du bracelet connecté. Nous ne savons pas comment simuler les données d'un accéléromètre. En effet, nous n'avons pas de bases qui nous permettrait de savoir quelles valeurs sont "normales" et lesquelles peuvent être "anormales". Pour ce qui est de la simulation du rythme cardiaque, nous avons créé une fonction qui permet d'avoir des valeurs aléatoires entre 50 et 220 bpm, cette fonction n'est pas reliée à la base de données car elle fonctionne seulement sur la page "suivi" donc elle n'est pas liée au patient. De plus, ces données ne sont pas "réalistes" car elles peuvent passer de 50 à 200 bpm d'un coup ce qui n'arrive jamais dans la vraie vie.
Ensuite, la valeur de la fréquence cardiaque n'est pas reliée au recensement du nombre de stéréotypies. En effet, comme nous n'avons pas de données sur lesquelles se baser nous ne pouvons pas faire correspondre la fréquence cardiaque et une crise.

b. Prolongements possibles

Dans les prolongements possibles qui pourraient être réalisés l'année prochaine, nous avons pour commencer régler les problèmes auxquels nous avons fait face. Mais également, nous aimerions ajouter un bracelet connecté au projet qui permettrait de supprimer les valeurs aléatoires et de récupérer les données de l'accéléromètre et du cardiofréquencemètre en temps réel.

Nous souhaiterions aussi que lorsqu'un nouveau patient ou un nouveau médecin s'inscrit, ses données s'enregistrent dans la base de données et ne disparaissent pas dès que l'on ferme notre site.

Ensuite, nous aimerions ajouter l'envoi d'un mail automatique lorsqu'on remplit un formulaire, ce dernier est pour la page "mot de passe oublié", avoir un mail automatique qui s'envoit pour récupérer et changer son mot de passe.

Pour finir, nous aimerions ajouter une messagerie sécurisée et interne à notre application pour que les patients puissent communiquer directement avec leur médecin et inversement.

Tables des figures

Annexes

lien de notre codeSandbox : https://codesandbox.io/s/ptut-v20-uf5bc?file=/index.html
lien vers notre dépôt Github : https://github.com/bendcst/PTUTAutisme
lien vers Heroku : https://autismono.herokuapp.com/

Pour les trois choses que nous aimerions faire si un prolongement du projet se fait, nous avons trouvé des documentations afin de nous aider pour la suite.
Voici les liens des sites que nous avons trouvé :

- Rapport d'un projet sur des capteurs qui mesure la fc : https://activcollector.clermont.inra.fr/Finder2E/Presentation/inra/RapportINRA_CP&AW.pdf

- Récupérer les données d'un formulaire pour les mettre dans la base de données en PHP : https://www.pierre-giraud.com/php-mysql-apprendre-coder-cours/recuperer-manipuler-donnee-formulaire/

- Envoyer un mail automatique lorsqu'on rempli un formulaire (mot de passe oublié) : http://www.fobec.com/tuto/948/envoyer-email-partir-formulaire.html