

EMBOSSEUSE BRAILLE

DOSSIER TECHNIQUE

PARTIE INFORMATION ET COMMUNICATION



Dossier Technique de **MOSELLE Keziah**
Projet SIN - Lycée Blaise Pascal
Année scolaire 2017-2018



Disponible sur



SOMMAIRE

Partie A - Définition d'ensemble du projet (page 3)

1. Introduction au braille (page 3)
2. Qu'est-ce qu'une embosseuse ? (page 3)
3. Problématique (page 3)

Partie B - Cahier des charges (page 4)

Partie C - Choix de solutions (page 5-6)

1. Matériel + Serveur (page 5)
2. Application (page 6)

Partie D - Modélisation et simulation (page 7)

1. Maquette de l'application (page 7)
2. Simulation de l'application (page 7)

Partie E - Développement (page 8-10)

1. Serveur (page 8)
 - a. Protocole HTTP et WebSocket (page 8)
 - b. Notifications (page 8)
 - c. Encoder un caractère (page 8)
2. Client (Application/IHM) (page 9)
 - a. Introduction à React Native (page 9)
3. Conclusion (page 10)

Partie F - Annexes (page 11-33)

1. Le braille (page 11)
2. SysML (page 12-15)
3. Mesures HTTP/WebSocket (page 16-17)
4. Code source (page 18-33)

Partie A - Définition d'ensemble du projet

1. Introduction au braille

Le braille est une manière de transmettre du texte sous forme physique inventé par Louis Braille lorsqu'il a perdu la vue. Il a inventé un alphabet de 6 points et qui traduit tous les caractères de la langue Française (Louis Braille étant français).

Il y a au total 64 caractères possibles en braille, ce qui permet de traduire l'alphabet, des accents, les ponctuations et des symboles. Il existe des préfixes dans le but d'élargir cette possibilité.

2. Qu'est-ce qu'une embosseuse ?

Une embosseuse est une imprimante qui transcrit le texte d'un fichier informatique en caractères brailles sur papier et qui est destinée aux aveugles ou aux personnes malvoyantes.

3. Problématique

Suite à une visite à la fondation IDS Le Phare d'Illzach (Institut s'occupant des malvoyants) nous avons pu voir leur manière de vivre et d'apprendre, nous nous sommes rendu compte que leur embosseuse braille était très prisé par les professeurs de ce fait ils ne pouvaient pas l'utiliser quand ils le souhaitent, nous avons eu donc l'idée de créer une embosseuse braille portable à un coût fortement réduit.

Thème sociétal : La santé

Problématique : Comment améliorer l'apprentissage des personnes à déficiences visuelles ?

Partie B - Cahier des charges

(Sous forme de tableau)

Fonction	Critères d'appréciations	Niveaux d'appréciations	Limite d'acceptation
FP1 : Embosser	Déformer la feuille Respecter les normes du braille	Lisible par une personne à déficiences visuelles	$\pm 15\%$ mm
FP2 : Autonomie	Doit pouvoir être autonome durant 2h (en fonctionnement)	Assure une utilisation durant une journée de cours	$2h \pm 1h$
FP3 : Poids	Ne doit pas être lourd à porter	Assure la portabilité	Poids = $2kg \pm 1kg$
FP4 : Encoder	Encoder tout l'alphabet français	Alphabet, accents et caractères numériques	N/A
FP5 : Communiquer	Wi-Fi NFC	Rapidité et fonctionnement sans-fil	Distance = 35m 10cm max
FC1 : Reconnaissance vocale	Transcrire de la voix en texte	Simplifier l'utilisation de l'application	$T < 5s \pm 1s$
FC2 : Notifications	Pouvoir recevoir des notifications en temps réel	Retour visuel, sonore et sensoriel	N/A




GRAS : Mes fonctions à réaliser




FP : Fonction Principale

FC : Fonction complémentaire

Partie C - Choix de solutions

1. Serveur

Critères / Solutions			
Coût (Coef 2)	4/5	3/5	5/5
Consommation d'énergie	4/5	3/5	5/5
Rapidité de traitement	5/5	5/5	3/5
Communication	5/5	4/5	2/5
Total	22	18	20

Critères / Solutions			
Requêtes/s	5/5	3/5	5/5
Bas niveau (Coef 2)	5/5	0/5	5/5
Orienté événement	5/5	2/5	4/5
Communication	5/5	3/5	5/5
Total	25	8	24

NodeJS a été choisis grâce à :

Rapidité,

Bas niveau (Accès à la partie matériel),

Écosystème riche (Grâce à son gestionnaire de paquets NPM *Node Package Manager*),




Orienté événement

Asynchrone



Node.JS est une plateforme logicielle conçue pour exécuter du JavaScript côté serveur, il a la particularité d'être orienté événement, entrée/sortie non-bloquant (Asynchrone) ce qui permet à Node de gagner en vitesse d'exécution. C'est une technologie qui a vite été utilisée par les grandes entreprises telles que Netflix, PayPal etc.... qui ont réussi à diminuer de moitié leurs latences. Comme vous pouvez le voir Node est évolutif, il peut être utilisé pour toute tailles de projet, il sera donc utilisé en tant que serveur dans le but de traiter les requêtes d'impression via l'application ou le site. Le serveur desservira également le site et nous permettra enfin de communiquer avec l'Arduino.

2. Application

Critères / Solutions	 React	 Ionic	 Java
Fluidité (Coef 2)	5/5	3/5	5/5
Ergonomie	5/5	5/5	2/5
Multi-plateforme (Coef 2)	5/5	5/5	0/5
Coût	5/5	4/5	4/5
Popularité	5/5	3/5	5/5
Total	50	39	14

React Native a été choisis grâce à :

Natif,

Multi-plateforme (iOS & Android),

Écosystème riche (Communauté active),

Permet d'intégrer des composants écrits en Java, Swift ou Objective-C,

Accès à l'API du téléphone (Caméra, microphone etc...),

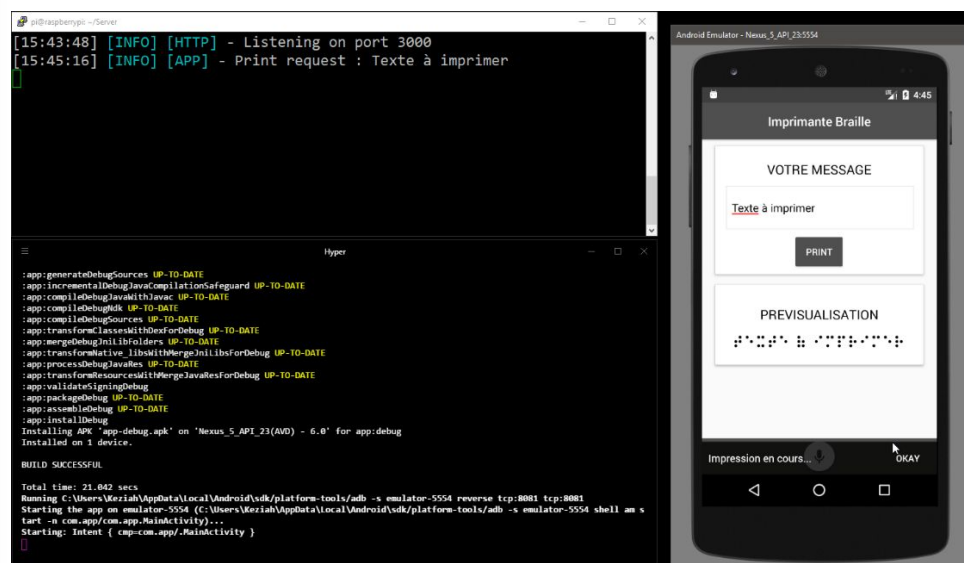
Développé par l'équipe de Facebook (Utilisé par Facebook, Instagram, AirBnb, Skype...) ce qui garantit le développement de cette technologie



React Native est un Framework JavaScript qui va nous permettre de développer des applications mobiles natives sous Android ET iOS en utilisant la librairie première du nom ReactJS. Cette combinaison va nous permettre de créer des composants (ReactJS) et dans ces composants nous allons pouvoir écrire leur logique (Des fonctions propres au composant) et faire interagir les composants entre eux d'après une architecture hiérarchique. React Native va s'occuper de transpiler notre code en Java, Swift ou Objective-C selon la plateforme souhaitée et ainsi obtenir une vraie application mobile native.

Partie D - Modélisation et simulation

1. Maquette de l'application
2. Simulation de l'application (AVD)



Partie E - Développement

1. Serveur

a. Protocole HTTP et WebSocket (Mesures)

Le rôle primaire du serveur est de faire passerelle entre l'application et la partie opérative, pour se faire nous avons deux protocoles : HTTP et WebSocket. J'ai donc décidé d'effectuer des mesures pour différencier lequel sera le plus efficace en terme de poids de la trame et de requêtes/s.

(Voir Annexes - figure n°9,10,11 pour voir les résultats des mesures)

J'ai donc décidé d'utiliser le protocole WebSocket car en plus de ses performances bien supérieur au protocole HTTP, il dispose d'une connexion bidirectionnel ce qui signifie que nous pouvons envoyer des trames sans que le client doit envoyer une requête au préalable. Le seul défaut du protocole WebSocket est le délai de connexion qui prend aux alentours de 5 secondes en moyenne, mais une fois connecté ce n'est plus un problème.

b. Notifications

Grâce au protocole WebSocket nous allons pouvoir intégrer des notifications en temps réel. Un algorithme représentant le fonctionnement serait:

SI événement_de_notifications
AFFICHER notification

c. Encoder un caractère

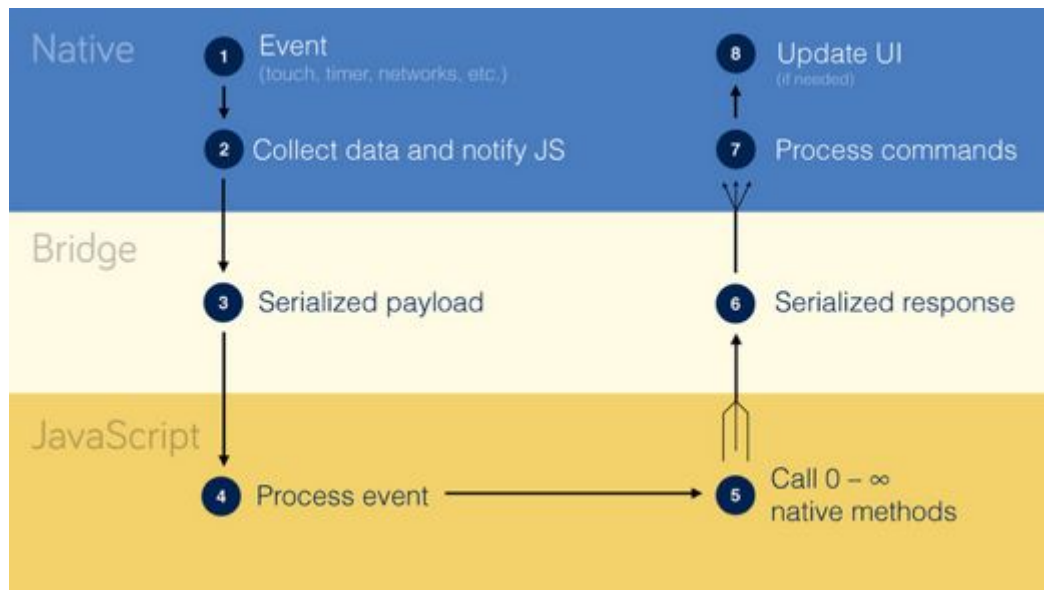
Pour encoder un caractère nous avons décidé de découper un caractère en deux parties :

- La première partie détermine si le caractère est en minuscule, majuscule ou alors si c'est un chiffre, pour cela nous avons décidé d'utiliser trois symboles : Si c'est une majuscule : !, un chiffre : * et : pour tous les autres types dont les minuscules.
- La deuxième partie déterminera de quelle lettre il s'agira, pour cela nous codons le caractère sur 6 bits selon l'ordre des cellules brailles comme suit :

D'après cet ordre, si une cellule est remplie nous mettrons 1 et si elle est vide 0. Exemple pour la lettre 'a' cela donnerait :100000.

2. Client (Application/IHM)

a. Introduction à React Native



Nous avons 3 threads (permet d'exécuter certaines tâches en étant isolé) dans une application React Native, le Main ou Native, le Bridge et le JavaScript.

Le thread Native écoute tous les événements en relation avec le téléphone (écran tactile, requêtes réseaux etc...) ensuite ce thread va essayer de communiquer avec le JavaScript thread, mais il ne peut pas communiquer avec le même type de données.

C'est là que le Bridge thread apparaît : Il permet de sérialiser (il va mettre les données sous forme binaire) ces données et ainsi pouvoir être récupéré par le JavaScript thread. Le JavaScript thread.

Et enfin le JavaScript thread va exécuter la logique du composant (des fonctions) pour retourner le résultat au Native thread et ce thread va actualiser l'application en modifiant son apparence par exemple.

Ces threads ont plusieurs caractéristiques :

- Asynchrone
 - Permet une communication asynchrone entre les threads. Ce qui signifie qu'ils ne vont jamais se bloquer.
- Batched
 - Les transferts des messages se font de manière optimisée en les envoyant par paquet.
- Sérialisé
 - Les deux threads ne s'échangent jamais les mêmes données car ils ne travaillent pas avec le même type de données. Ils s'échangent donc des messages sérialisés.

3. Conclusion

Durant la réalisation de ce projet j'ai rencontré un problèmes majeur : l'implémentation de la reconnaissance vocale via l'API de Google Speech car l'accès à leur service (API) est payant et complexe à implémenter au sein de React Native, ce qui m'a empêché de réaliser cette tâche. J'ai également été confronté à des retards sur l'avancement du projet définis par un diagramme de Gantt à cause du temps d'apprentissage des technologies notamment l'environnement de React Native qui est simple à écrire mais difficile à simuler (problèmes de compilation dû à des fichiers de configurations natifs), choisir la bonne structure et les dépendances, ce qui m'a empêché de rajouter le NFC à cause du manque de temps.

Les tâches qui m'ont été assignées au sein du cahier des charges sont : FP4 (Encoder), FP5 (Communiquer), FC1 (Reconnaissance Vocale) et FC2 (Notifications). J'ai pu réaliser la FP2 car tout l'alphabet français a bien été encodé. La FP3 n'a pas été complété car il manque la communication via NFC, mais pour le Wi-Fi je l'ai bien réalisé avec une distance maximum de 90 mètres ! Supérieur aux cahiers des charges de 55 mètres, la FC1 n'a pas pu être réalisée dû aux raisons évoquée ci-dessus. Mais j'ai pu réaliser la FC2 (Notifications).

La façon dont a été conçu notre embosseuse nous permettra d'ajouter des fonctionnalités très simplement, notamment grâce au modèle du Raspberry Pi, qui nous permet d'ajouter de multiples moyens de communications dans le but d'en couvrir au maximum (Bluetooth Low Energy, Near Field Communication, Ethernet), il était également prévu de rajouter un mode conversation sur l'application qui permettra aux aveugles de communiquer avec des personnes à distance librement grâce à la reconnaissance vocale (Speech To Text) et lorsqu'un message est reçu il lui sera retransmis par synthèse vocale. (Text To Speech)

Il manque actuellement l'implémentation des préfixes dans la prévisualisation (pour les lettres majuscules, chiffres...) qui n'est pas présente à cause de la police d'écriture utilisée.

L'organisation du projet s'est très bien passée, notre groupe est cohérent car nous nous complétons grâce à nos différentes compétences. Nous avons utilisé un diagramme de Gantt pour gérer notre temps et j'ai utilisé la méthode Kanban durant le développement relié à notre repository Github qui nous permettait de relier notre tableau à nos "issues" (bogues ou fonctionnalités) et ainsi de les corriger ou les ajouter.

Nous sommes motivés à développer ce projet dans le futur car il possède de multiples utilités, c'est pour cela que nous avons déposé une enveloppe soleau.

Partie F - Annexes

1. Le braille

a	b	c	d	e	f	g	h	i	j
k	l	m	n	o	p	q	r	s	t
u	v	w	x	y	z				

à	â	ç	è	é	ê	ë
î	ï	ô	œ	ù	û	ü

Figure n°1: L'alphabet braille (Non complet)

2. SysML

JAUNE : APPLICATION / SITE

ORANGE : SERVEUR

ROUGE : PARTIE OPÉRATIVE

VIOLET : API (Application
Programming Interface)

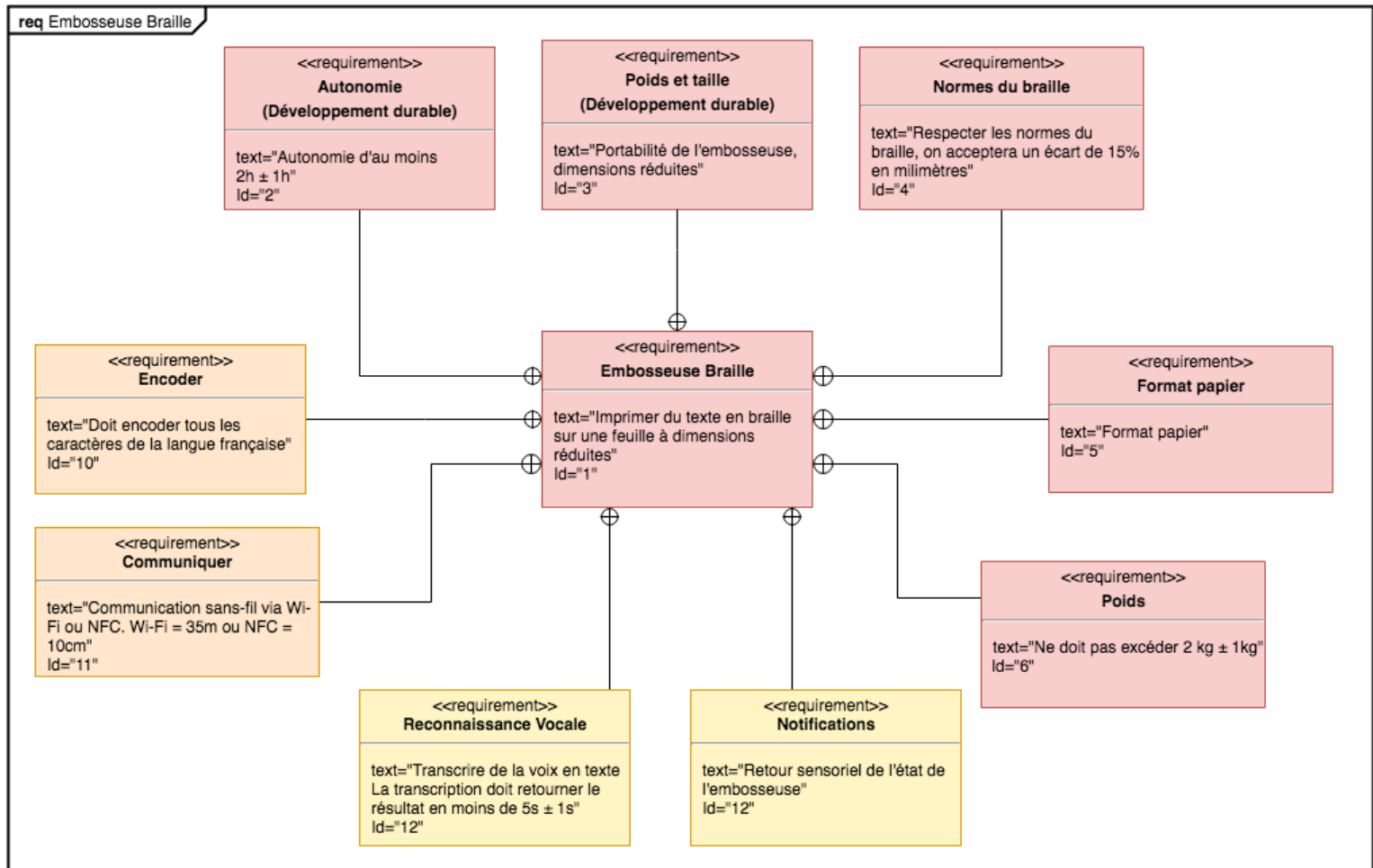


Figure n°2 : Diagramme d'exigences de l'embosseuse

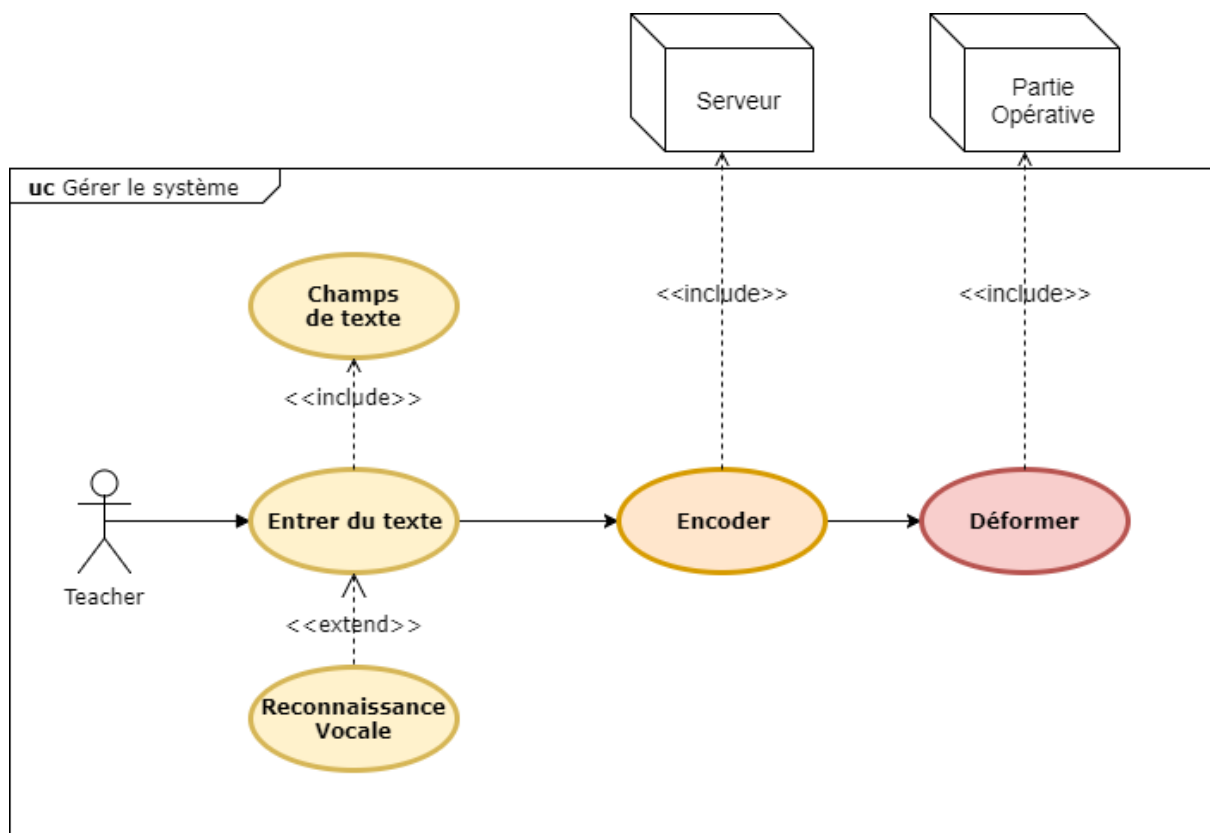


Figure n°3: Cas d'utilisation général

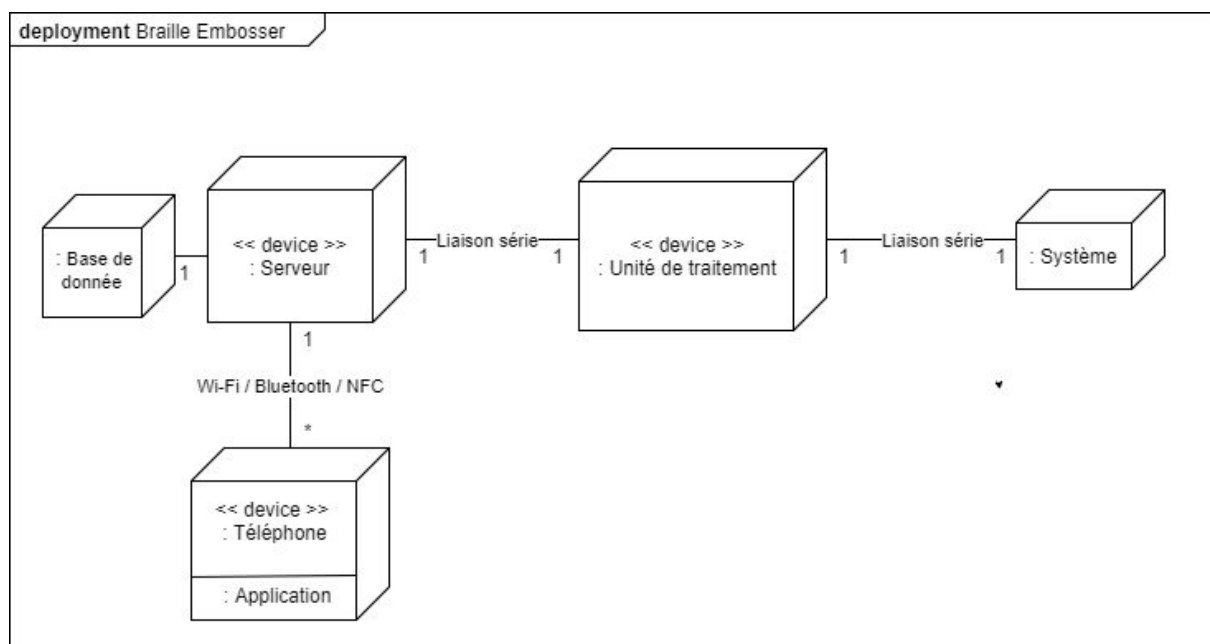


Figure n°4: Diagramme de déploiement général
(Hors programme, mais représente bien le système général)

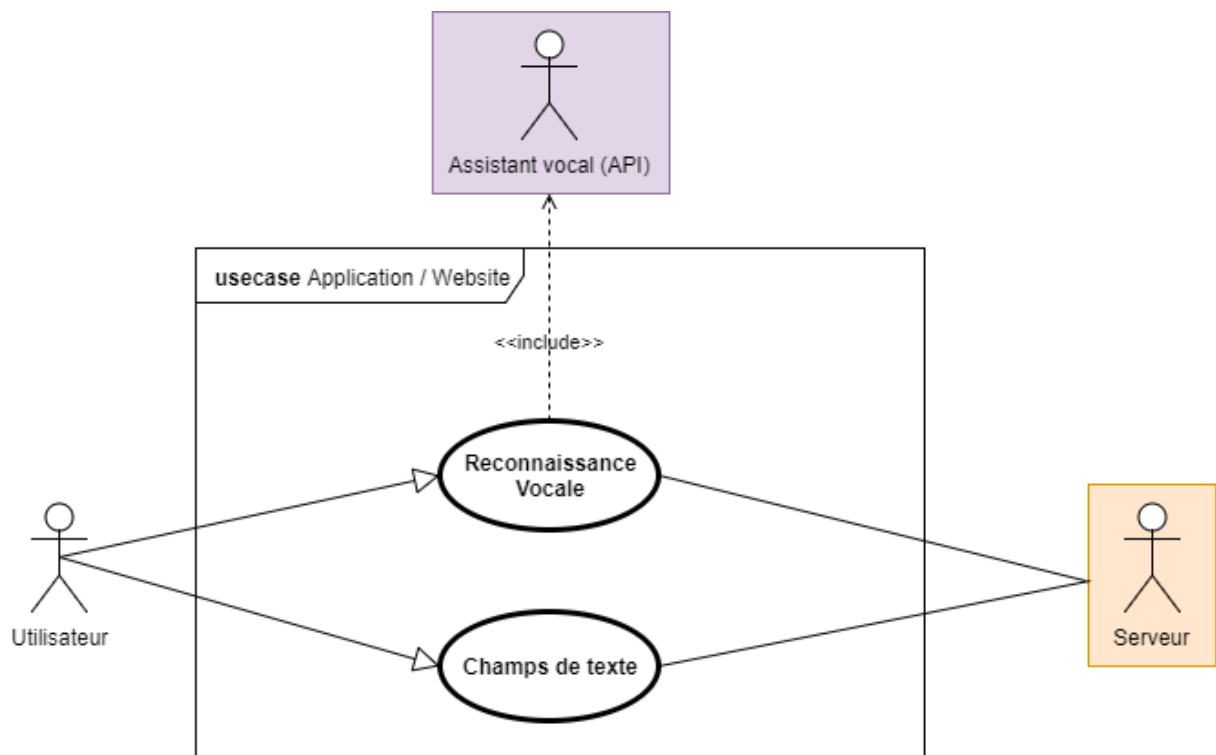


Figure n°5: Cas où l'utilisateur souhaite embosser grâce à l'application ou le site

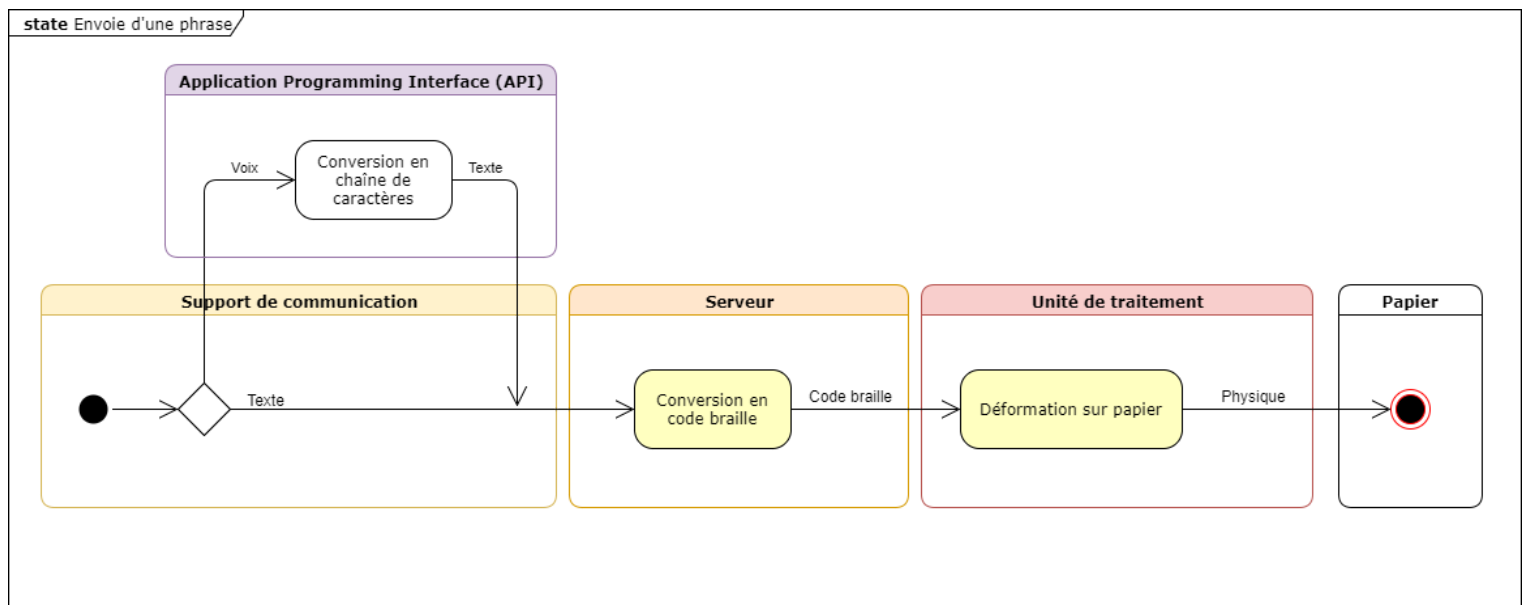


Figure n°6: Les différents états de l'embosseuse

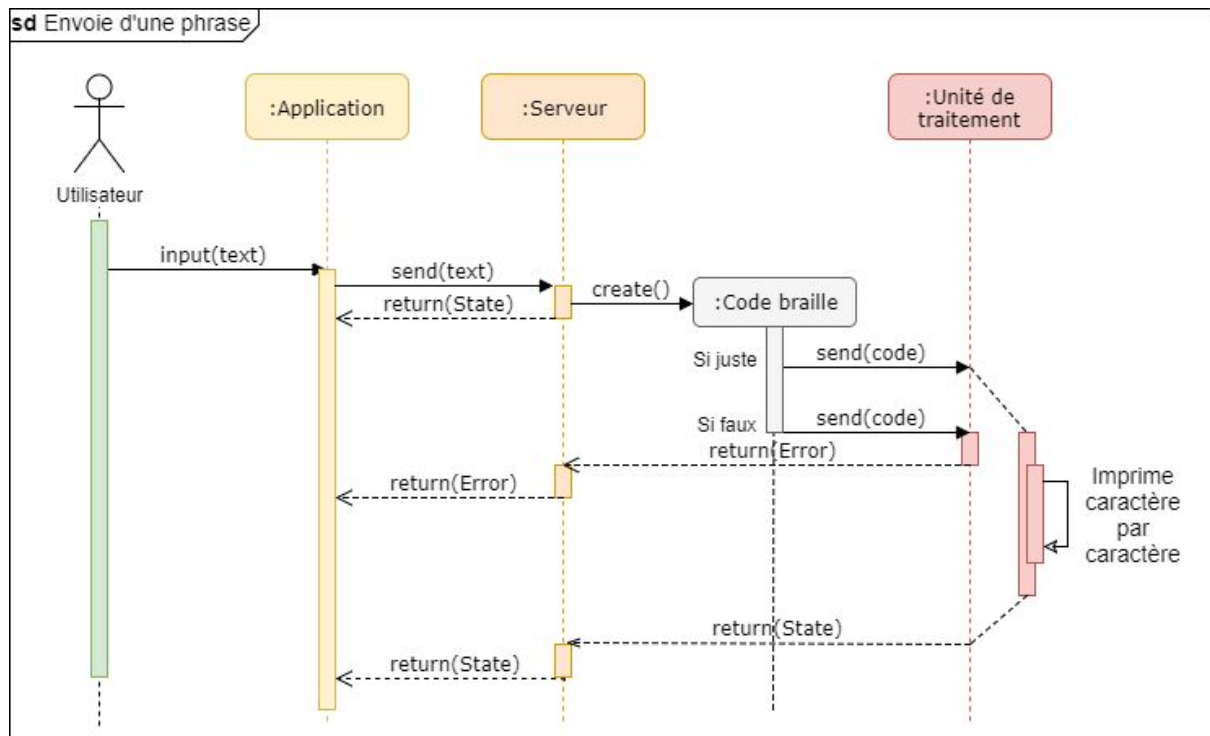


Figure n°7: Diagramme de séquence dans le cas où l'on utilise le champs de texte

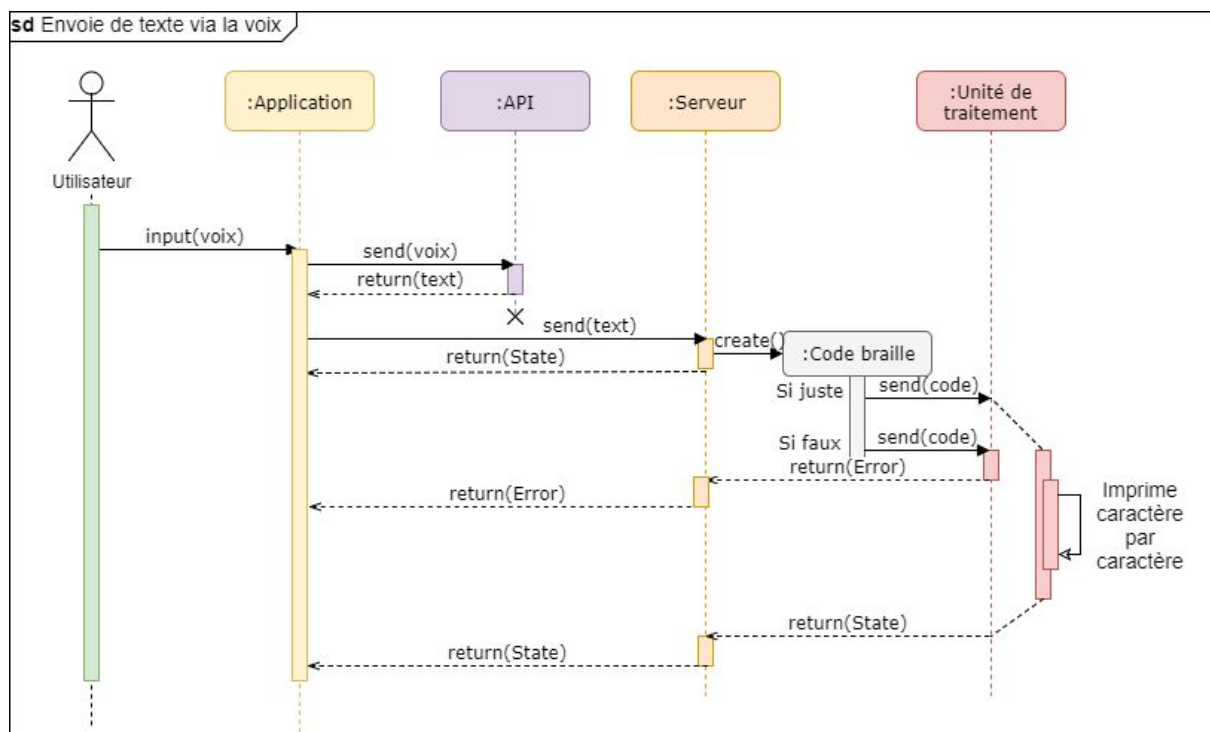


Figure n°8: Diagramme de séquence dans le cas où l'on utilise la voix

3. Mesures HTTP/WebSocket

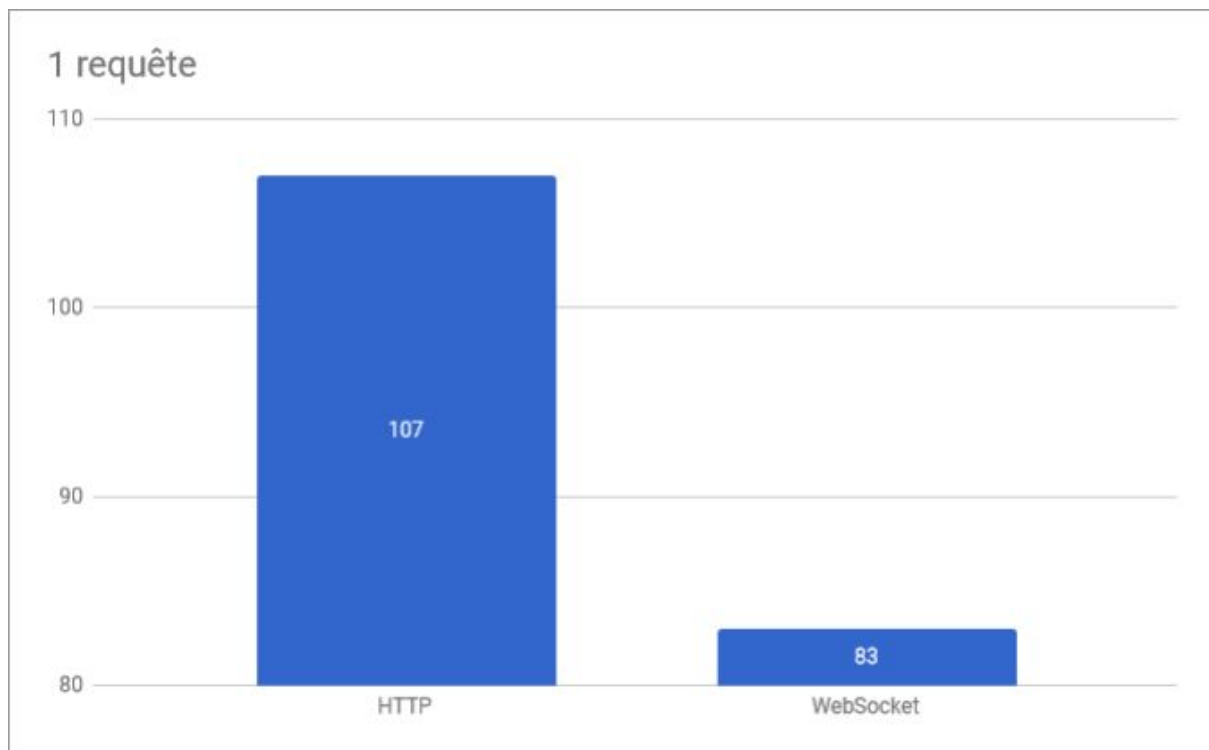


Figure n°9: Comparaison de la vitesse pour 1 trame selon le protocole HTTP et WebSocket (en ms)

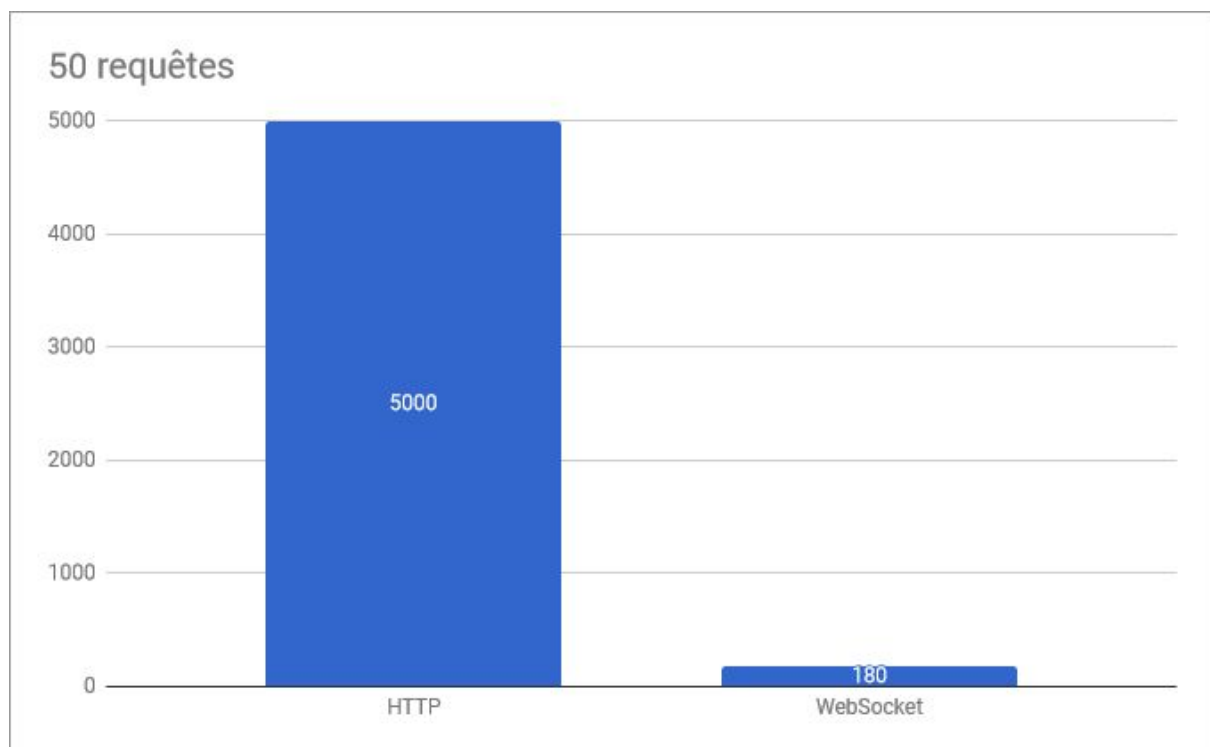


Figure n°10: Comparaison de la vitesse pour 50 trames (en ms)

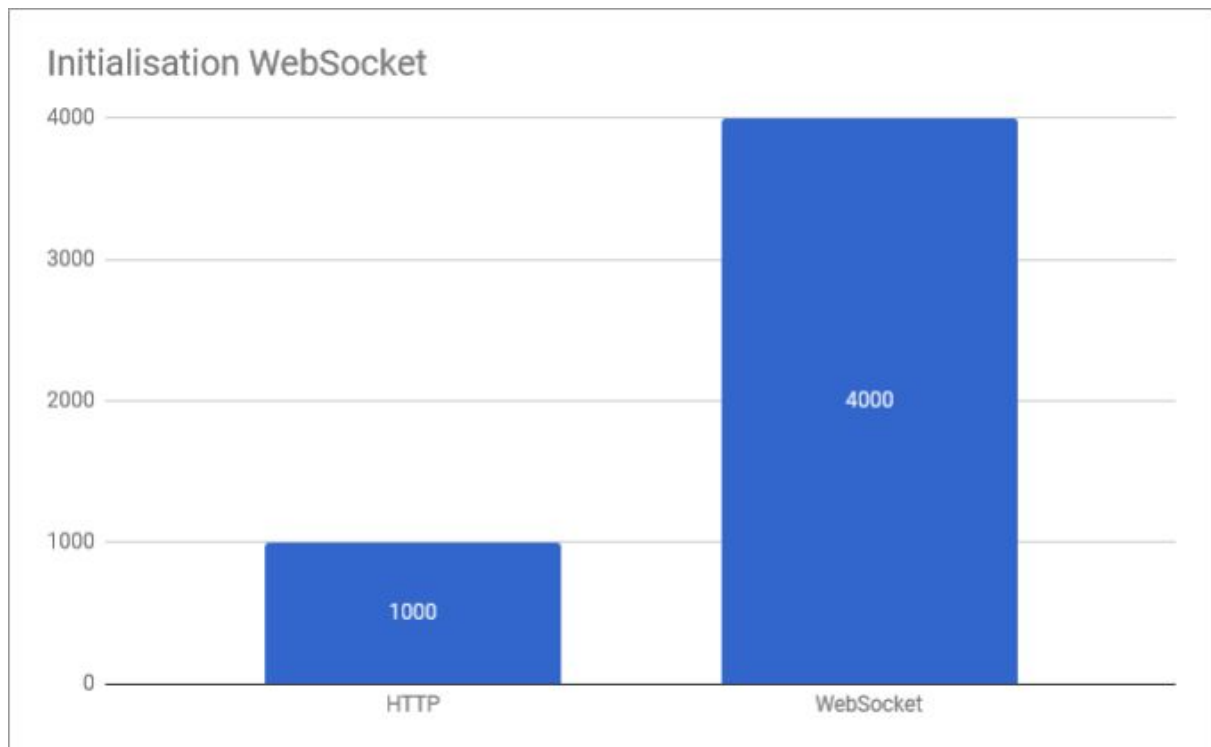


Figure n°11: Comparaison en prenant en compte l'initialisation de la connexion bidirectionnel du protocole WebSocket sur 10 requêtes (en ms)

4. Code source (également disponible sur <https://github.com/dream-io/braille-embosser>)

a. Application

```
1 {
2   "name": "braille-embosser",
3   "version": "1.0.1",
4   "private": true,
5   "scripts": {
6     "start": "node node_modules/react-native/local-cli/cli.js start",
7     "build:android": "cd android && gradlew assembleRelease"
8   },
9   "rnpm": {
10    "assets": [
11      "./assets/fonts/"
12    ]
13  },
14  "dependencies": {
15    "native-base": "2.4.4",
16    "react": "16.3.1",
17    "react-native": "0.55.4",
18    "react-native-push-notification": "3.0.2",
19    "socket.io-client": "2.1.0"
20  },
21  "devDependencies": {
22    "babel-jest": "22.4.3",
23    "babel-preset-react-native": "4.0.0",
24    "jest": "22.4.3",
25    "react-test-renderer": "16.3.1"
26  },
27  "jest": {
28    "preset": "react-native"
29  }
30 }
```

Figure n° 12: Fichier de configuration de l'application

(Le nom, la version, les scripts qui permettent de lancer facilement une commande, rnpm qui liera le dossier "fonts" à notre projet, les dépendances du projet, les dépendances durant le développement...)

```

1 /**
2  * Braille Embosser
3  *
4  * @category Education
5  * @author Keziah MOSELLE <keziahmoselle@protonmail.com>
6  * @link https://github.com/dream-io/braille-embosser
7  */
8
9 import React from 'react';
10 import {View,StyleSheet} from 'react-native';
11 import {
12   StyleProvider,
13   Root,
14   Container,
15   Header,
16   Title,
17   Left,
18   Button,
19   Icon,
20   Right,
21   Body,
22   Content,
23   Text,
24   Card,
25   CardItem,
26   Form,
27   Item,
28   Label,
29   Input,
30   Footer,
31   FooterTab,
32   Toast,
33   Fab,
34   Spinner,
35   Tab,
36   Tabs
37 } from 'native-base';
38 // Récupère le thème de couleur de Native Base
39 import getTheme from './native-base-theme/components';
40 import commonColor from './native-base-theme/variables/commonColor';
41 import SocketIOClient from 'socket.io-client';
42 import PushNotification from 'react-native-push-notification';
43
44 export default class App extends React.Component {
45
46   constructor(props)
47   {
48     super(props);
49
50     // State
51     this.state = {
52       text: undefined,
53       status: '',
54       showToast: false,
55       connected: false,
56       ipAddress: 'http://192.168.43.73:8080',
57       printing: false,
58       count: 0,
59       mode: 'papier'
60     };
61
62     // Connexion WebSocket
63     this.socket = SocketIOClient(this.state.ipAddress);
64
65     // Lorsqu'on se connecte, assigne connected à true et affiche un toast
66     this.socket.on('connect', () =>
67     {
68       this.setState({ connected: true });
69     });
70

```

Figure n°13 : Fichier principal de l'application [1/7]

```

71 // Lorsque l'événement 'S0' se déclenche : affiche une notification
72 this.socket.on('S0', (data) =>
73 {
74     // Increment count var
75     this.setState(previousState => {
76         return {count: previousState.count + 1};
77     });
78     Toast.show({
79         text: 'Caractère embossé.',
80         position: 'top',
81         buttonText: 'Ok'
82     });
83 });
84
85 // Lorsque l'événement 'S1' se déclenche : affiche une notification
86 this.socket.on('S1', (data) =>
87 {
88     this.setState({printing: false});
89     // Affiche une notification push, même quand on quitte l'application
90     PushNotification.localNotification({
91         title: "Embosseuse Braille",
92         message: "Votre phrase a été embossé.",
93         bigText: ` "${this.state.text}" a été embossé.`,
94         largeIcon: "done",
95         smallIcon: "ic_launcher",
96         color: 'black',
97         vibrate: true,
98         vibration: 300,
99     });
100 });
101
102 // Lorsque la connexion se ferme, on force la reconnexion
103 this.socket.on('disconnect', () => {
104     this.setState({connected: false});
105     this.socket.open();
106     Toast.show({
107         text: 'Reconnexion...',
108         position: 'top',
109         buttonText: 'Ok'
110     });
111 });
112
113 }
114
115 // Fonction qui enverra le texte à imprimer au serveur
116 print = () =>
117 {
118     // On vérifie la connexion
119     if (this.state.connected)
120     {
121         // Si le champs est vide
122         if (this.state.text === undefined || this.state.text === '')
123         {
124             // Informe l'utilisateur de remplir le champs
125             Toast.show({
126                 text: 'Le champs est vide !',
127                 position: 'top',
128                 buttonText: 'Ok'
129             });
130         }
131         // Si le champs n'est pas vide
132         else
133         {
134             // Envoie l'événement 'print' et envoie les données
135             this.socket.emit('print', this.state.text);
136             this.setState({printing: true});
137             // Informe l'utilisateur que l'embosseuse va commencer
138             Toast.show({
139                 text: 'Embossage en cours...',
140                 position: 'top',
141                 buttonText: 'Ok'
142             });
143         }
144     }
145     else

```

```

146   {
147     // Si on a remplis le champs mais que nous ne sommes pas connecté, on informe l'utilisateur
148     Toast.show({
149       text: 'Vous n\'êtes pas connecté au RPI.',
150       position: 'top',
151       buttonText: 'Ok'
152     });
153   }
154 }
155
156 handleChangeTab = () =>
157 {
158   this.setState({text: undefined});
159   if (this.state.mode === 'papier')
160   {
161     this.setState({mode: 'adhésif'});
162     this.socket.emit('changePaper', {type: 'Adhésif'});
163   }
164   else
165   {
166     this.setState({mode: 'papier'});
167     this.socket.emit('changePaper', {type: 'Normal'});
168   }
169 }
170
171 // Fonction qui retournera du texte en fonction des ondes sonores
172 voiceRecognition = () =>
173 {
174   Toast.show({
175     text: 'En développement...',
176     position: 'top',
177     buttonText: 'Ok'
178   });
179 }
180
181 // Fais avancer le papier manuellement
182 paperForward = () =>
183 {
184   // Vérification de la connexion
185   if (this.state.connected)
186   {
187     // Envoie d'un événement 'paperForward'
188     this.socket.emit('paperForward');
189     // Informe l'utilisateur
190     Toast.show({
191       text: 'Le papier avance...',
192       position: 'top',
193       buttonText: 'Ok'
194     });
195   }
196   else
197   {
198     // Si on n'est pas connecté on informe l'utilisateur
199     Toast.show({
200       text: 'Vous n\'êtes pas connecté au RPI',
201       position: 'top',
202       buttonText: 'Ok'
203     });
204   }
205 }

```

[3/7]

```

206
207 // Demande à l'Arduino de redémarrer
208 restart = () =>
209 {
210     // On vérifie la connexion
211     if (this.state.connected)
212     {
213         // Envoie l'événement 'RESTART'
214         this.socket.emit('restart');
215         // Informe l'utilisateur
216         Toast.show({
217             text: 'Redémarrage...',
218             position: 'top',
219             buttonText: 'Ok'
220         });
221     }
222     else
223     {
224         // Si on n'est pas connecté on informe l'utilisateur
225         Toast.show({
226             text: 'Vous n\'êtes pas connecté au RPI',
227             position: 'top',
228             buttonText: 'Ok'
229         });
230     }
231 }
232
233 // Demande à l'Arduino de RESET
234 reset = () =>
235 {
236     // On vérifie la connexion
237     if (this.state.connected)
238     {
239         // Envoie l'événement 'RESTART'
240         this.socket.emit('reset');
241         // Informe l'utilisateur
242         Toast.show({
243             text: 'Réinitialisation...',
244             position: 'top',
245             buttonText: 'Ok'
246         });
247     }
248     else
249     {
250         // Si on n'est pas connecté on informe l'utilisateur
251         Toast.show({
252             text: 'Vous n\'êtes pas connecté au RPI',
253             position: 'top',
254             buttonText: 'Ok'
255         });
256     }
257 }
258

```

[4/7]


```

259 render() {
260   return (
261
262     <StyleProvider style={getTheme(commonColor)}>
263       <Root>
264         <Container>
265
266           <Header>
267             <Left style={styles.padding}>
268               <Icon style={styles.icon} name='print'></Icon>
269             </Left>
270
271             <Right>
272               <Icon style={styles.icon} name={this.state.connected ? 'ios-checkmark-circle' : 'ios-close-circle'}></Icon>
273             </Right>
274           </Header>
275
276           <Tabs initialPage={0} onChangeTab={this.handleChangeTab}>
277             <Tab heading="PAPIER">
278               <Content padder>
279                 <Card>
280                   <CardItem cardBody>
281                     <Item>
282                       <Input
283                         multiline={true}
284                         numberOfLines={2}
285                         placeholder="Entrer le texte à embosser..."
286                         onChangeText={(text) => this.setState({text})}
287                       />
288                     </Item>
289                   </CardItem>
290                 </Card>
291                 {this.state.text ? <Card><CardItem cardBody><Text style={styles.braille}>
292                   {this.state.text}
293                 </Text></CardItem></Card>
294                   : null}
295               </Content>
296             </Tab>
297             <Tab heading="ADHESIF">
298               <Content padder>
299                 <Card>
300                   <CardItem cardBody>
301                     <Item>
302                       <Input
303                         multiline={true}
304                         numberOfLines={2}
305                         placeholder="Entrer le texte à embosser..."
306                         onChangeText={(text) => this.setState({text})}
307                       />
308                     </Item>
309                   </CardItem>
310                 </Card>
311                 {this.state.text ? <Card><CardItem cardBody><Text style={styles.braille}>
312                   {this.state.text}
313                 </Text></CardItem></Card>
314                   : null}
315               </Content>
316             </Tab>
317           </Tabs>
318

```

[5/7]

```

319     <Footer>
320       <FooterTab>
321         <Button vertical active onPress={this.restart}>
322           <Icon name="md-power" />
323           <Text>Reboot</Text>
324         </Button>
325         <Button vertical active onPress={this.reset}>
326           <Icon name="md-refresh" />
327           <Text>Réinit</Text>
328         </Button>
329         <Button vertical active onPress={this.paperForward}>
330           <Icon active name="md-redo" />
331           <Text>Avancer</Text>
332         </Button>
333         <Button vertical active onPress={this.voiceRecognition}>
334           <Icon name="mic" />
335           <Text>Voix</Text>
336         </Button>
337       </FooterTab>
338     </Footer>
339
340     {this.state.printing ?
341       <Fab
342         style={styles.fab}
343         position="bottomRight">
344           <Spinner color="white"/>
345         </Fab> :
346       <Fab
347         style={styles.fab}
348         position="bottomRight"
349         onPress={this.print}>
350           <Icon name="print" />
351         </Fab>}
352
353   </Container>
354 </Root>
355 </StyleProvider>
356 );
357 }
358 }
359
360 const styles = StyleSheet.create({
361   header: {
362     flexDirection: "row",
363     justifyContent: "center",
364     alignItems: "center",
365   },
366   headerPanel: {
367     height: 50,
368     backgroundColor: "#4F4F4F",
369     flexDirection: "row",
370     justifyContent: "center",
371     alignItems: "center",
372   },
373   headerTitle: {
374     color: "FFF",
375   },
376   center: {
377     flexDirection: "column",
378     justifyContent: "center",
379     alignItems: "center",
380   },
381   footer: {
382     height: 45,
383     backgroundColor: "#4F4F4F",
384   },
385   btnContainer: {
386     flexDirection: "row",
387     justifyContent: "center",
388     marginTop: -60,
389   },

```



```
390  braille: {
391    fontFamily: "BrailleNormal",
392    fontSize: 22,
393    color: '#333',
394    padding: 10,
395    lineHeight: 40,
396    letterSpacing: 5,
397  },
398  h2: {
399    fontFamily: "RobotoMono",
400  },
401  headerH2: {
402    color: "#FFF",
403    fontFamily: "RobotoMono",
404  },
405  fab: {
406    backgroundColor: '#5067FF',
407    marginBottom: 50,
408  },
409  padding: {
410    padding: 10,
411  },
412  icon: {
413    color: '#FFF',
414    fontSize: 26,
415    marginLeft: 4,
416    marginRight: 4,
417  }
418 }));
```

[7/7]

b. Serveur

```
1 {
2   "name": "server",
3   "version": "1.0.0",
4   "main": "App.js",
5   "scripts": {
6     "start": "nodemon server"
7   },
8   "author": "KeziahMoselle",
9   "dependencies": {
10    "express": "^4.16.2",
11    "leaf-logger": "^3.2.1",
12    "serialport": "^6.0.4",
13    "socket.io": "^2.0.4"
14  },
15  "devDependencies": {
16    "nodemon": "^1.17.5"
17  }
18 }
```

Figure n°14: Fichier de configuration du serveur

(Le nom, la version, le fichier principal à lancer, les scripts, l'auteur, les dépendances, les dépendances durant le développement)

1	{	
2	" "	"000000",
3	"a"	"100000",
4	"b"	"110000",
5	"c"	"100100",
6	"d"	"100110",
7	"e"	"100010",
8	"f"	"110100",
9	"g"	"110110",
10	"h"	"110010",
11	"i"	"010100",
12	"j"	"010110",
13	"k"	"101000",
14	"l"	"111000",
15	"m"	"101100",
16	"n"	"101110",
17	"o"	"101010",
18	"p"	"111100",
19	"q"	"111110",
20	"r"	"111010",
21	"s"	"011100",
22	"t"	"011110",
23	"u"	"101001",
24	"v"	"111001",
25	"w"	"010111",
26	"x"	"101101",
27	"y"	"101111",
28	"z"	"101011",
29	"0"	"001111",
30	"à"	"111011",
31	"â"	"100001",
32	"ç"	"111101",
33	"è"	"011101",
34	"é"	"111111",
35	"ê"	"110001",
36	"ë"	"110101",
37	"î"	"100101",
38	"ï"	"110111",
39	"ô"	"100111",
40	"œ"	"010101",
41	"ù"	"011111",
42	"û"	"100011",
43	"ü"	"110011",
44	"1"	"100001",
45	"2"	"110001",
46	"3"	"100101",
47	"4"	"100111",
48	"5"	"100011",
49	"6"	"110101",
50	"7"	"110111",
51	"8"	"110011",
52	"9"	"010101",
53	". "	"010011",
54	", "	"010000",
55	"?"	"010001",
56	"; "	"011000",
57	":"	"010010",
58	!"	"011010",
59	(" "	"011001",
60) "	"001011",
61	"\"	"011011",
62	"_ "	"001001",
63	"' "	"001000",
64	}	

Figure n°15: L'alphabet braille encodé

```
1 {  
2   "success":  
3   {  
4     "S0": "Caractère embossé.",  
5     "S1": "Phrase embossée.",  
6     "PAPERF": "Demande pour avancer le papier."  
7   },  
8   "errors":  
9   {  
10    "E0": "Format de données incorrect.",  
11    "E1": "Préfixe incorrect (:!* uniquement).",  
12    "E2": "Vérification du caractère (0/1 uniquement).",  
13    "E3": "Recharger l'emboseuse en papier."  
14  }  
15 }
```

Figure n°16: Les différents messages

```

1 /**
2  * Braille Embosser
3  *
4  * @category Education
5  * @author Keziah MOSELLE <keziahmoselle@protonmail.com>
6  * @link https://github.com/dream-io/braille-embosser
7  */
8
9 /**
10 * Dependencies
11 */
12 // Logger
13 const Logger = require('leaf-logger');
14 // ExpressJS
15 const express = require('express');
16 const app = express();
17 // HTTP
18 const server = require('http').Server(app);
19 // Body Parser
20 const bodyParser = require('body-parser');
21 // Socket.io
22 const io = require('socket.io')(server);
23 // SerialPort
24 const SerialPort = require('serialport');
25
26 // JSON Files
27 const braille = require('./data/braille.json');
28 const status = require('./data/status.json');
29
30 /**
31 * Constants / Variables
32 */
33 // HTTP
34 const webPort = 8080;
35 // SerialPort
36 const portPath = '/dev/ttyS0',
37       baudRate = 9600,
38       encodeType = 'utf8';
39 // Application
40 var charIndex = 0,
41     encoded = "",
42     encodedLength = 0,
43     isPrinting = false,
44     reOpen,
45     queue = [];
46
47 /**
48 * Code
49 */
50 // SerialPort
51 const Serial = new SerialPort(portPath, {
52   baudRate: baudRate,
53   dataBits: 8,
54   parity: 'none',
55   stopBits: 1,
56   flowControl: false
57 }).setEncoding(encodeType);
58
59 const Parser = Serial.pipe(new SerialPort.parsers.Readline());
60
61 // HTTP
62 server.listen(webPort, () => {
63   Logger.info(`Server listening on ${webPort}`, 'HTTP')
64 });
65

```

Figure n°17: Fichier principal du serveur [1/5]

```

66 // ExpressJS
67 app.use(bodyParser.json());
68
69 // SerialPort
70 Serial.on('open', () =>
71 {
72     setTimeout(() =>
73     {
74         Logger.success(`Port ${portPath} ouvert`, 'SerialPort');
75         Logger.warn(`Reset Arduino`, 'Application');
76         send('RESET');
77     }, 2500)
78 });
79
80 Serial.on('close', () =>
81 {
82     Logger.error(`Port ${portPath} fermé`, 'SerialPort');
83
84     reOpen = setInterval(() =>
85     {
86         Serial.open((err) =>
87         {
88             if (err) {
89                 Logger.error('Impossible d\'ouvrir', 'SerialPort');
90                 return;
91             }
92             clearInterval(reOpen);
93         })
94     }, 5000)
95 });
96
97 Parser.on('data', (data) =>
98 {
99     data = Array.from(data);
100     data.pop();
101     data = data.join('');
102
103     Logger.info(`Données reçus: ${data}`, 'SerialPort');
104
105     if (data == 'S0') {
106         if (charIndex != encodedLength) {
107             io.sockets.emit('S0', 'Caractère embossé.');
```

[2/5]

```

125
126 // ExpressJS
127 app.post('/print', (req, res) =>
128 {
129     let message = req.body.text;
130
131     Logger.info(`Print requested -> ${message}`, 'ExpressJS (HTTP)');
132
133     encoded = encode(message);
134     print();
135
136     req.send({
137         'status': 0
138     });
139 })
140
141 // Socket.io
142 io.on('connection', (socket) =>
143 {
144     Logger.info(`Utilisateur #${socket.id} s'est connecté`);
145
146     socket.on('print', (message) =>
147     {
148         Logger.info(`Nouvelle requête: ${message}, ajoutée à la queue`, 'WebSocket');
149         queue.push(message);
150         console.log(queue);
151     })
152
153     socket.on('changePaper', (data) => {
154         let type = data.type;
155         Logger.info(`Utilisation de papier ${type}`, 'WebSocket');
156         switch (type) {
157             case "Adhésif":
158                 send('PADH');
159                 break
160             case "Normal":
161                 send('PNOR');
162                 break
163             default:
164                 Logger.error('Mauvais type de papier', 'WebSocket');
165                 break
166         }
167     });
168
169     socket.on('paperForward', () => {
170         Logger.info(`Paper forward requested`, 'WebSocket');
171         send('PAPERF');
172     })
173
174     socket.on('restart', () => {
175         Logger.info(`Arduino restart requested`, 'WebSocket');
176         send('RESTART');
177     })
178
179     socket.on('reset', () => {
180         Logger.info(`Arduino reset requested`, 'WebSocket');
181         send('RESET');
182     })
183
184 })

```

[3/5]


```

185
186 /**
187  * Envoie un caractère à l'Arduino
188  * @param {string} message
189  * @param {function} callback
190  */
191 function send(message, callback = null)
192 {
193     Serial.write(message, () =>
194     {
195         if (callback === null)
196         {
197             Logger.success(`Envoie: ${message} à ${portPath}`, 'Application');
198         } else {
199             callback();
200         }
201     });
202 }
203
204 /**
205  * Prints the next character from encoded array
206  */
207 function print()
208 {
209     send(encoded[charIndex], () =>
210     {
211         Logger.success(`Requête: ${encoded[charIndex]}. En attente d'une réponse`, 'Application');
212     })
213 }
214
215 /**
216  * Encode string to braille characters
217  * @param {string} text
218  */
219 function encode(text)
220 {
221     isPrinting = true;
222     charIndex = 0;
223     encodedLength = -1;
224     var stringArray = [];
225     text.split('').forEach((char) =>
226     {
227         encodedLength++;
228         if (isWhiteSpace(char))
229         {
230             stringArray.push(":000000");
231         } // ESPACE
232         else if (Number.isInteger(parseInt(char)))
233         {
234             stringArray.push(`*${braille[char]}`);
235         } // NUMERIQUE
236         else if (isUpperCase(char))
237         {
238             stringArray.push(`!${braille[char.toLowerCase()]}`);
239         } // MAJUSCULE
240         else
241         {
242             stringArray.push(`:${braille[char]}`);
243         } // MINUSCULE
244     })
245     return stringArray;
246 }
247
248 /**
249  * Returns `true` if argument is a capital letter
250  * @param {string} text
251  * @returns {boolean}
252  */
253 function isUpperCase(text)
254 {
255     return (text == text.toUpperCase() && !/[~`!#$%^&*+=\~\[\]\'\',/{}|\\"<=>\?./g.test(text));
256 }

```



```
257
258 /**
259  * Returns `true` if argument is a space
260  * @param {string} text
261  * @returns {boolean}
262  */
263 function isWhiteSpace(text)
264 {
265     return text.indexOf(' ') >= 0;
266 }
267
268 // Launch printing queue
269 setInterval(() => {
270     if (!isPrinting) {
271         if (queue.length > 0) {
272             encoded = encode(queue.pop());
273             print();
274         }
275     }
276 }, 1000);
```

[5/5]