Dut Informatique | Semestre 4

Document n°4/4 Manuel de maintenance

Crowdsensing - Groupe n°9

Tuteur: CHBEIR Richard
Commanditaire : ALLANI Sabri

FAGET Corentin TD1
MADEIRA Romann TD2
RAAPOTO Gabin TD2

Table des matières

Table des Figures

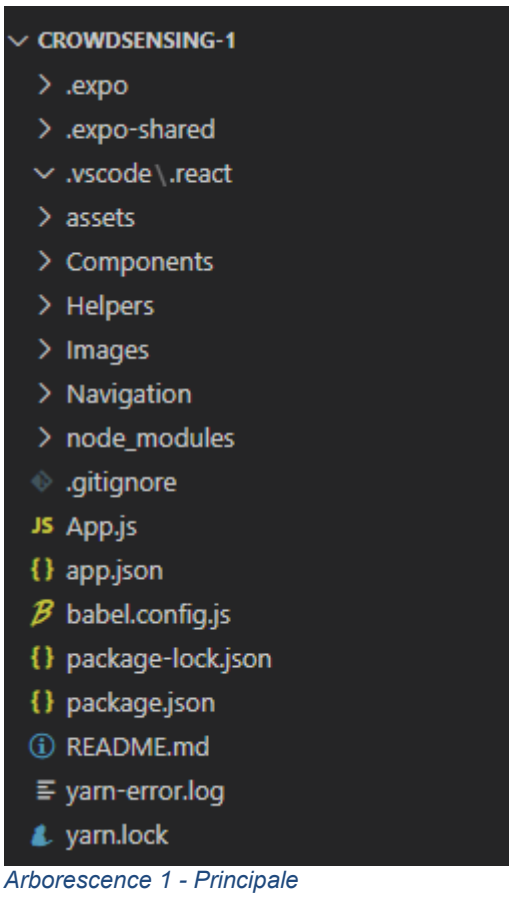
Préambule

Afin de pouvoir comprendre du mieux possible ce manuel de maintenance il est important d'avoir un minimum de connaissance concernant le Framework React Native et de son fonctionnement. Nous partons du postulat que les développeurs lisant ce manuel sont à l'aise avec ce Framework.

Toutefois si des points semblent flou a certain moment nous vous recommandons de consulter les cours d'openclassrooms qui seront susceptibles de vous aider lorsque vous serez en difficulté.

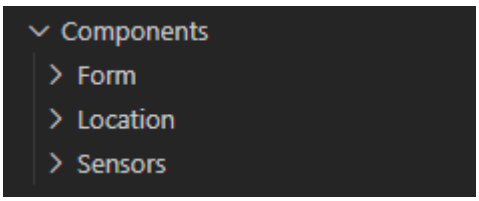
https://openclassrooms.com/fr/courses/4902061-developpez-une-application-mobile-react-native

Organisation du code



Arborescence 1 - Principale

Notre arborescence est composée de différents dossiers, nous allons voir plus en détail leurs rôles au fur et à mesure. Le fichier App.js représente le main de notre application et enfin le package.json détenant les modules utilisés ainsi que leurs versions.



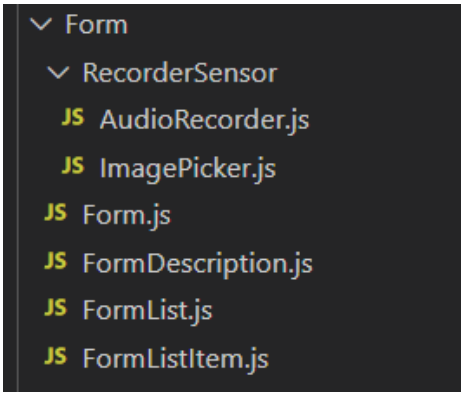
Arborescence 2 - Components

Components

Contenu

Le premier dossier représente les components de notre application, qui est lui-même composée de trois dossiers Forms, Location et Sensors. C'est trois dossiers sont les plus importants de l'application, nous allons donc les voirs en détails

Form



Arborescence 3 - Form

Contenu

Le fichier Form contient tous les fichiers relatifs aux formulaires et aux capteurs disponibles pour ces formulaires.

Form.js

Contient l'affichage du formulaire.

FormDescription.js

Contient l'affichage de la description du formulaire sélectionné.

FormList.js

Contient l'affichage de la liste des formulaires.

FormListItem.js

Contient le template des items de la liste affichée dans FormList.js.

RecorderSensor/AudioRecorder.js

Contient l'affichage de l'enregistreur audio.

RecorderSensor/ImagePicker.js

Contient l'affichage relatif à la récupération d'image (via la galerie ou la caméra).

Classes

Form.js

_displayFormList

```
_displayFormList = () => {
    //Return to FormList sreen
    this.props.navigation.navigate("Forms");
}
```

Retourne à l'écran de la liste des formulaires.

getAnswer

```
getAnswer = (answer, question) => {
    //Add the aswer param to the answers
    const answers = this.state.answers;
    answers[question.name] = answer;
    this.setState({ answers: answers });
}
```

Fonction 1 - getAnswer

Ajoute la réponse (answer) de l'utilisateur passé en paramètre dans le state 'answers' avec comme clé le nom de la question.

_submitForm

```
_submitForm(){
    //Currently this just displays form answer to the console in JSON format
    //But this will send it to a database in a future version
    console.log(JSON.stringify(this.state.answers));
    //Show a toast to inform the user that the form are submit
    this.toast.show('Form submit !');
    //Wait 500ms that the toast has been shown
    setTimeout(() => {  this._displayFormList(); }, 500);

}
```

Fonction 2 - _submitForm

Pour l'instant cette fonction affiche juste les réponses de l'utilisateur au format JSON mais dans la future version, il l'enverra vers une base de données. La fonction affiche aussi un message pour avertir l'utilisateur que le formulaire a été envoyé et redirige l'utilisateur vers la liste des formulaires

_displayQuestionsForm

```
_displayQuestionsForm(form){
    var input; //Input use to get data from the user
    return(
        //Get all questions from the form
        form.questions.map((question, index) => {
            //Get input type
            switch (question.type) {
                case "text":
                    input = <TextInput
                        style={styles.text_input}
                        returnKeyType="done"
                        multiline={true}
                        blurOnSubmit={true}
                        onSubmitEditing={()=>{Keyboard.dismiss()}}
                        onChangeText={(answer) => this.getAnswer(answer, question)}
                    />
                    break;

                case "audioRecord":
                    input = <AudioRecorder question={question} getAnswer={this.getAnswer}/>
                   break;

                case "image":
                    input = <ImagePicker question={question} getAnswer={this.getAnswer}/>
                    break;

                default:
                    break;
            }
            //Display data title and his input
            return(
                <View style={styles.container} key={index}>
                    <Text style={styles.title_question}> {question.title} </Text>
                    {input}
                </View>
            );
        })
    );
}
```

Affiche toutes les questions contenues dans le formulaire passé en paramètre et affiche un input en fonction du type de donnée de la question.

FormDescription.js

_displayForm

```
displayForm = (form) => {
    //Open a new screen with the form selected
    this.props.navigation.navigate("Form", { form: form });
}
```

Fonction 4 - _displayForm

Permet d'afficher l'écran 'Form' qui affiche le formulaire passé en paramètre.

_displaySensorsForm

```
displaySensorsForm(form){
    return(
        //Display all sensors use in the form
        form.sensors.map((sensor) => {
            return(<Text style={styles.text} key={sensor.toString()}> -{sensor}</Text>);
        })
    );
}
```

Fonction 5 - _displaySensorsForm

Affiche les capteurs utilisé dans le formulaire

FormList.js

_setInfoPopUpVisibility

```
setInfoPopUpVisibility(){
    //Change the pop-up visibility
    this.setState({ modalVisible: !this.state.modalVisible });
}
```

Fonction 6 - _setInfoPopUpVisibility

Modifie- e state ‘modalVisible’ pour afficher ou cacher le pop-up d’information. Si ‘modalVisible’ égale ‘true’ le pop-up est visible et si ‘modalVisible’ égale ‘false’ le pop-up est cachée.

displayFormDescription

```
displayFormDescription = (form) => {
    //Open a new screen with the selected form
    this.props.navigation.navigate("Description", { form: form });
}
```

Fonction 7 - displayFormDescription

Permet d’afficher l’écran ‘Description’ qui affiche le formulaire passé en paramètre. Ce formulaire est le formulaire de la liste sélectionnée par l’utilisateur.

getColors

```
getColors(){

    let colors = []; //colors array
    let i = 0;
    //Get colors from all forms. It's use in the FormListItem for the design
    forms.forEach(form => {
        colors[i] = form.color;
        i++;
    });
    return colors;
}
```

Récupère la couleur de chaque formulaire.

Fonction 8 - getColors

FormListItem.js

_getColorNext

```
_getColorNext(){
    const colors = this.props.getColors(); //Array with all forms color
    //Get next form color
    let index = colors.indexOf(this.props.form.color);
    if(index == colors.length-1){
        return '#441d59';
    }
    return colors[index+1];
}
```

Fonction 9 - _getColorNext

Récupère la couleur du formulaire suivant pour pouvoir l'afficher dans le coin du formulaire courant.

RecorderSensor/AudioRecorder.js

toggleStopwatch

```
toggleStopwatch() {
    //Stop the reset and start the stop watch
    this.setState({stopwatchStart: !this.state.stopwatchStart, stopwatchReset: false});
}
```

Fonction 10 - toggleStopwatch

Stop la remise à zéro du chronomètre et change son état (en marche ou arrêté). Si stopwatchStart égale true le chronomètre est en marche et si stopwatchStart égale false le chronomètre s'arrête.

resetStopwatch

```
resetStopwatch() {
    //Stop the stopwatch and start the reset
    this.setState({stopwatchStart: false, stopwatchReset: true});
}
```

Stop le chronomètre et le remet à zéro.

Fonction 11 - resetStopwatch

_startRecording

```
startRecording = async () =>  {
    try {
        //Ask permission to use Audio sensor
        console.log('Requesting permissions..');
        await Audio.requestPermissionsAsync();
        await Audio.getPermissionsAsync();
        await Audio.setAudioModeAsync({
            allowsRecordingIOS: true,
            playsInSilentModeIOS: true,
        });
        //Start audio recording
        const recording = new Audio.Recording();
        await recording.prepareToRecordAsync(Audio.RECORDING_OPTIONS_PRESET_HIGH_QUALITY);
        await recording.startAsync();
        this.setState({ recording: recording, isRecording:true });

        //Start and reset timer
        this.resetStopwatch()
        this.toggleStopwatch();

    } catch (err) {
        //Send the error to the log
        console.error('Failed to start recording', err);
    }
}
```

Fonction 12 - _startRecording

Demande la permission d'accès au microphone à l'utilisateur. Commence l'enregistrement audio et démarre le chronomètre.

_stopRecording

```
stopRecording = async () => {
    const getAnswer = this.props.getAnswer; //Function to send record to the form
    const recording = this.state.recording; //Current recording

    //Stop audio record
    this.setState({ recording: undefined, isRecording:false });
    await recording.stopAndUnloadAsync();
    //Get uri record
    const uri = recording.getURI();
    this.setState({ uri: uri });
    //Send record to the form
    getAnswer(uri, this.props.question);
    //Stop stopwatch
    this.toggleStopwatch();
}
```

Fonction 13 - _stopRecording

Arrête l'enregistrement audio et le chronomètre. Envoie l'uri de l'enregistrement dans les réponses du formulaire.

RecorderSensor/ImagePicker.js

_pickImage

```
_pickImage = async () => {
    //Ask storage permission to user
    const { status } = await Picker.requestMediaLibraryPermissionsAsync();
    if (status !== 'granted') {
      alert('Sorry, we need camera roll permissions to make this work!');
    } else {
        //If permission allowed -> pick picture from storage
        let result = await Picker.launchImageLibraryAsync({
            mediaTypes: Picker.MediaTypeOptions.All,
            allowsEditing: true,
            aspect: [4, 3],
            quality: 1,
          });

          if (!result.cancelled) {
              //If picture has been picken -> send to the form
            this._setImage(result.uri);
            this.props.getAnswer(result.uri, this.props.question);
          }
    }
};
```

Fonction 14 - _pickImage

Récupère la permission de l'utilisateur pour utiliser la galerie photo du téléphone. Cette fonction récupère aussi l'image sélectionnée par l'utilisateur et l'envoie dans le formulaire via la fonction getAnswer.

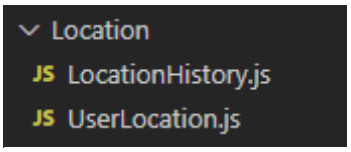
_takePicture

```
_takePicture = async () => {
    //Ask camera permission to user
    const { status } = await Picker.requestCameraPermissionsAsync();
    if (status !== 'granted') {
      alert('Sorry, we need camera roll permissions to make this work!');
    } else {
        //If permission allowed -> Take picture
        let result = await Picker.launchCameraAsync({
            mediaTypes: Picker.MediaTypeOptions.All,
            allowsEditing: true,
            aspect: [4, 3],
            quality: 1,
          });

          if (!result.cancelled) {
            //If picture has been taken -> send to the form
            this._setImage(result.uri);
            this.props.getAnswer(result.uri, this.props.question);
          }
    }
}
```

Fonction 15 - _takePicture

Récupère la permission de l'utilisateur pour utiliser la caméra du téléphone. Cette fonction récupère aussi la photo prise par l'utilisateur et l'envoie dans le formulaire via la fonction getAnswer.

Location



Arborescence 4 - Location

Contenu

Ce dossier contient les fichiers nécessaires à la gestion de la localisation dans le deuxième onglet de l'application.

LocationHistory.js

LocationHistory.js contient l'affichage de l'historique.

UserLocation.js

UserLocation.js contient l'affichage de la map.

Classes

UserLocation.js

constructor(props)

Constructeur de la classe, initialise les states de la classe (comme des variables propres à la classe)

_getLocation()

Permet de récupérer la localisation actuelle de l'utilisateur, appelée lors d'un appuie sur le bouton "Save position".

```
_getLocation = async () => {
    const {status} = await Permissions.askAsync(Permissions.LOCATION);
    if(status != 'granted'){
        console.log('PERMISSION NOT GRANTED');
        this.setState({
            errorMessage: 'PERMISSION NOT GRANTED'
        })
    }
    const userLocation = await Location.getCurrentPositionAsync();

    this.setState({
        currentLocation: userLocation,
        region: {
            latitude: userLocation.coords.latitude,
            longitude: userLocation.coords.longitude,
            latitudeDelta: 0.05,
            longitudeDelta: 0.05,
        }
    })
}
```

Fonction 16 - _getLocation

_recordLocation()

Gère l'appel des autres fonctions permettant de faire un enregistrement de localisation. Est appelé lors de l'appuie du bouton record.

```
async _recordLocation() {
    if(this.state.isRecording){
        //Stop recording location
        clearInterval(this.state.interval);

        if(this.state.recordedLocations.length > 0){
            await this._storeRecord();
        }
        this.setState({
            recordedLocationsTimestamp: 0,
            isRecording: false,
            recordButtonImage: require('../../Images/record-button-white.png'),
        })
    } else {
        this.setState({
            isRecording: true,
            recordedLocations: [],
        })

        await this._recordLocationInterval();

        this.setState({
            recordButtonImage: require('../../Images/stop-button-white.png'),
        })

        this.state.interval = setInterval(async () => {
            await this._recordLocationInterval();
        }, 5000);
    }
}
```

Fonction 17 - _recordLocation

_recordLocationInterval()

Récupère la localisation courante de l'utilisateur, la formate en JSON et l'ajoute dans un tableau. Cette fonction est appelée par la fonction _recordLocation() toutes les 5 secondes.

```
_recordLocationInterval = async () =>{
    var currentLocation = await Location.getCurrentPositionAsync();

    if(this.state.recordedLocationsTimestamp == 0){
        this.state.recordedLocationsTimestamp = currentLocation.timestamp
    }

    var locationJSON = ["location_" + JSON.stringify(currentLocation.timestamp),currentLocation];

    this.state.recordedLocations.push(locationJSON);
    this.setState({
        recordedLocations: this.state.recordedLocations,
    });
}
```

Fonction 18 - recordLocationInterval

_storeRecord() & _storeData()

Ces fonctions permettent de stocker les données d'une localisation ou d'un enregistrement de localisation.

```
async _storeRecord(){
    try {
        let key = "locationRecord_" + JSON.stringify(this.state.recordedLocationsTimestamp);
        let value = JSON.stringify(this.state.recordedLocations);
        await AsyncStorage.setItem(key, value);
        this.toast.show('Record saved !');
    } catch (e) {
      alert(e)
    }
}
```

Fonction 19 - _storeRecord

_getDate(timestamp)

Retourne la date correspondante au timestamp donné. format : Mois Jour Année, hh:mm:ss

_resetMarkers()

Efface tous les marqueurs affichés sur la map. Est appelée par le bouton "erase".

_displayHistory()

Renvoi vers la page d'historique de localisation dans le fichier "LocationHistory.js"

returnData(latitude, longitude, date)

Fonction publique, permettant de faire passer des variables entre la vue de la map et celle de l'historique.

```
returnData(latitude, longitude, date) {
    var coordinate = {
        latitude: parseFloat(latitude),
        longitude: parseFloat(longitude),
        date: date,
    }
    this.state.mapMarkers.push(coordinate);
    this.setState({
        mapMarkers: this.state.mapMarkers
    });
}
```

Fonction 20 - returnData

LocationHistory.js

_onShare(latitude)

Affiche les options de partage lorsque l'on souhaite partager une localisation de l'historique.

```
_onShare = async (latitude) => {
    try {
      const result = await Share.share({
        message:
            JSON.stringify(latitude),
      });
      if (result.action === Share.sharedAction) {
        if (result.activityType) {
          // shared with activity type of result.activityType
        } else {
          // shared
        }
      } else if (result.action === Share.dismissedAction) {
        // dismissed
      }
    } catch (error) {
      alert(error.message);
    }
}
```

Fonction 21 - _onShare

_getPosHistory()

Récupère toutes les positions et enregistrements de position contenues dans les données de l'application. Elles sont ensuite stockées dans des variables (state) permettant ainsi leur traitement et affichage dans la page de l'historique.

_getDate(timestamp)

Retourne la date correspondante au timestamp donné. Format : Mois Jour Année, hh:mm:ss

_askToDelete(item,isARecord)

Envoi une demande de confirmation si l'utilisateur souhaite supprimer une localisation ou enregistrement de localisation.

```
_askToDelete(item,isARecord){
    Alert.alert(
        "Warning!",
        "Are you sure you want to delete the location?",
        [
            {
                text: "Cancel",
                onPress: () => {
                    return false;
                },
            },
            {
                text: "OK",
                onPress: () => this._deleteLocation(item,isARecord)
            }
        ]
    );
}
```

Fonction 22 - _askToDelete

_deleteLocation(item, isARecord)

Est appelé par "_askToDelete(item,isARecord)" dans le cas ou l'utilisateur confirme son souhait de suppression.

_listEmptyComponent()

Dans le cas où l'utilisateur n'a aucune donnée de localisation enregistrée dans le téléphone, cette fonction renverra du texte l'indiquant.

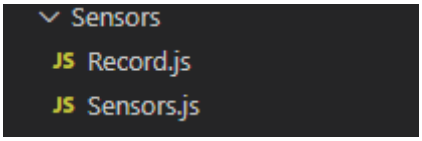
_returnToMapWithLocation(latitude, longitude, timestamp)

Quand un utilisateur clique sur une localisation, cette fonction sera appelée et retournera les données concernées à la map. Appelle la fonction returnData(latitude, longitude, date) de "UserLocation.js"

```
_returnToMapWithLocation = (latitude, longitude, timestamp) => {
    this.props.navigation.state.params.returnData(latitude,
                                                  longitude,
                                                  timestamp);
    this.props.navigation.goBack(null);
}
```

Fonction 23 - _returnToMapWithLocation

Sensors



Arborescence 5 - Sensors

Contenu

Les deux fichiers présents dans le component Sensors représentent chacun 1 page.

Record.js

Sensors.js représente la page ou l'on va pouvoir sélectionner les capteurs que l'on souhaite enregistrer

Sensors.js

Record.js quant à lui est la page qui va afficher les données récoltées des capteurs en temps réels

Classes

Record.js

L'une des fonctions les plus importantes dans cette classe est "onCheckedSmartphone(id)". Cette fonction permet de mettre à jour la liste des capteurs qui sont sélectionnés (avec le l'identifiant de la case à cocher passer en paramètre)

```
onCheckedSmartphone(id){
    const data=this.state.smartphoneData
    const index=data.findIndex(x => x.id === id)
    data[index].checked = !data[index].checked
    if (data[index].checked==true){
        this.state.selectedSensors.push(data[index].name)
    }
    else{
        this.state.selectedSensors.splice(this.state.selectedSensors.indexOf(data[index].name), 1);
    }
    this.setState(data)
}
```

Fonction 24 - onCheckedSmartphone

Cette méthode sera déclenchée lorsque que l'utilisateur cliquera sur une case à cocher

```
onPress={()=>this.onCheckedSmartphone(item.id)}
```

Ici le paramètre "item.id" représente l'identifiant de la case à cocher qui est appuyée.

Une fois que l'utilisateur souhaite accéder à la page d'enregistrement des capteurs cette fonction se lance '_displayRecord()'

```
_displayRecord = () => {
    if(this.state.selectedSensors.length > 0){
        this.setModalVisible();
    }
    else{
        alert("None sensors selected!")
    }
}
```

Fonction 25 - _displayRecord

Elle va vérifier s'il y a des capteurs qui sont cochés, si oui alors elle appelle une nouvelle fonction permettant d'afficher un popup qui va demander à l'utilisateur la permission d'utiliser les capteurs. Sinon il renvoi qu'un capteur n'a été coché.

La fonction qui affiche le pop-up est la suivante :

```
renderPopUp() {
    const { modalVisible } = this.state;
    return (
      <View style={styles.centeredView}>
        <Modal
          animationType="fade"
          transparent={true}
          visible={modalVisible}
        >
          <View style={styles.centeredView}>
            <View style={styles.modalView}>
              <Text style={styles.modalTitle}>Sensors permissions</Text>
              <Text style={styles.modalText}>Allow this app to access motion sensors? </Text>
              <Text style={styles.modalText}> Concerned : {this.state.selectedSensors.length} </Text>

              <TouchableOpacity style={styles.buttonAllowed} onPress={() => {this.setModalVisible(false); this._NextPage()}}>
                <Text style={styles.buttonText}>            Allow            </Text>
              </TouchableOpacity>

              <TouchableOpacity style={styles.buttonDeny} onPress={() => this.setModalVisible(false)}>
                <Text style={styles.buttonText}>            Deny            </Text>
              </TouchableOpacity>
            </View>
          </View>
        </Modal>
      </View>
    );
  }
```

Fonction 26 - RenderPopUp

On remarque qu'elle est composée de deux TouchableOpacity qui représente des boutons Autoriser et Refuser.

Si l'utilisateur autorise l'accès alors la fonction _NextPage() est appelée est permet d'afficher la page d'enregistrement des données des capteurs. Ainsi que la fonction setModalVisible avec en paramètre false qui va donc faire disparaître le pop-up.
Mais s'il refuse la fonction permettant de faire disparaître le pop-up est la seul appelé.

Sensors.js

Cette classe est un peu longue est complexe et est donc composé de la façon suivante, les fonctions permettant de gérer les capteurs sont au début du code avec la fonction

“subscribe()” qui va initialiser/montés les données des capteurs et les mettres à jour

“unsubscribe()” qui va les supprimer/désinstaller les données des capteurs

“componentDidMount()” qui est une fonction de base de React native, lorsque que l’utilisateur arrive ou lance la page cette fonction s'exécute automatiquement. Elle contient donc la fonction “subscribe()”

“componentWillUnmount()” qui est une fonction de base de React native, lorsque que l’utilisateur quitte la page cette fonction s'exécute automatiquement. Elle contient donc la fonction “unsubscribe()”

Nous avons par la suite les fonctions permettant de savoir quels capteurs nous devons affichés à l’utilisateur (par rapport aux capteurs coché à la page précédente)

```
checkSwitch=(param)=>{
    switch(param) {
    case 'Accelerometer':
        return ( this._renderAccelerometer() )

    case 'Barometer':
        return ( this.renderBarometer() )

    case 'Gyroscope':
        return ( this.renderGyroscope() )

    case 'Magnetometer':
        return ( this.renderMagnetometer() )

    case 'Pedometer':
        return ( this.renderPedometer() )
    }
}
```

Fonction 27 - checkSwitch

Cette fonction va vérifier si le paramètre passer correspond à un capteur, si oui elle l’affiche. Nous mettons dans le paramètre de cette fonction chaque élément de la liste contenant les capteurs cochés comme ceci :

```
renderSmartphoneSensorList(){
    return this.state.selected.map((item,key) => {
        return (
            <View key={key}>
                <Text  style={styles.text}>{"\n"}
                                           {item.key}
                                           {"\n"}
                                           {this.checkSwitch(item.key)}
                </Text>
            </View>
        )
      })
}
```

Fonction 28 - renderSmartphoneSensorList

Ici “this.state.selected” représente la liste des capteurs qui sont cochés et pour chaque élément présent dans la liste on affiche cette item.

Nous avons ensuite des fonctions relatives aux capteurs permettant de choisir leurs taux de mise à jour (rapide, moyen, lent)

Fonction 29 - taux de mise à jour Accéléromètre

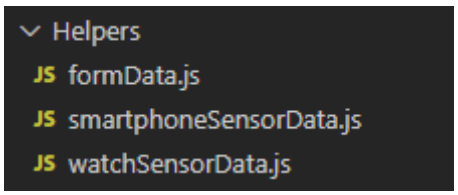
```
//Accelerometer
_slowAccelerometer(){
    Accelerometer.setUpdateInterval(350);
};

_mediumAccelerometer(){
    Accelerometer.setUpdateInterval(150);
};

_fastAccelerometer(){
    Accelerometer.setUpdateInterval(50);
};
```

Enfin, nous avons une multitude de fonctions permettant d'afficher des vues différentes (commençant par "render…") telles que la liste des capteurs, la liste des popUps, chronomètres.

```
render(){
    return(

        <View style={styles.main_container}>

            <ScrollView >
                <View style={styles.description_container}>
                    <Text style={styles.subhead}>Sensors selected :</Text>
                    {this.renderSmartphoneSensorList()}
                </View>
            </ScrollView>

            {this.renderStopWatch()}
            <TouchableOpacity style={styles.button} onPress={ () => { this.toggleStopwatch(); this.handleTimerComplete();} }>
                    <Text style={styles.text_button}>{!this.state.stopwatchStart ? "Start" : "Stop and save"}</Text>
            </TouchableOpacity>

            <View>
                {this.renderPopUp()}
            </View>
        </View>
    );
};
```

Et toute ces vues seront affichées dans la vue général de la classe

Helpers



Arborescence 6 - Helpers

Contenu

Le deuxième fichier de l'arborescence représente une bibliothèque de données. Pour mieux illustrer :

formData.js

Contient toutes les données des formulaires présents dans notre application (date de publication, titre, description…)

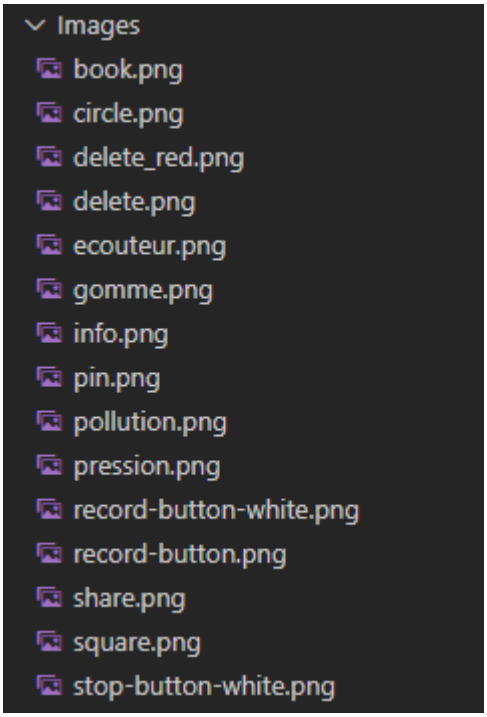
smartphoneSensorData.js

Contient les données des capteurs de téléphones (nom du capteur, permissions requises etc…)

Sensors.js

Tout comme le précédent, il contient les données des capteurs de montre connectés (nom du capteurs, permissions requises etc…)

Images

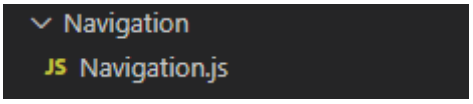


Arborescence 7 - Images

Contenu

Il n'y a pas grand-chose à dire ici, ce dossier contient toutes les images utilisées dans l'application

Navigation



Arborescence 8 - Navigation

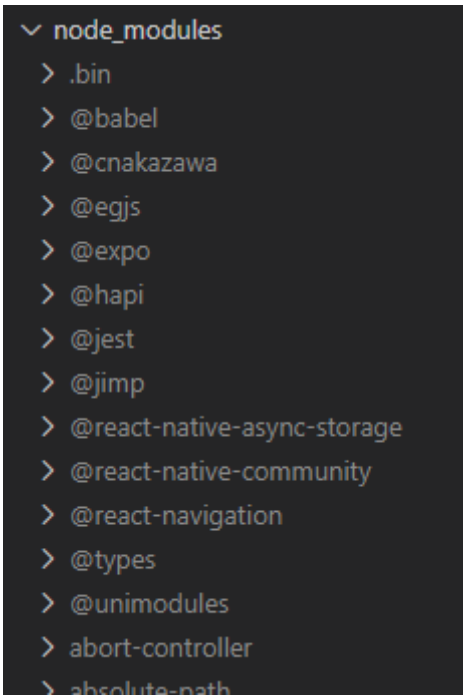
Contenu

Ce dossier contient un fichier permettant de naviguer entre les trois components (Form, Location, Sensors)

Navigation.js

Gère toutes les navigations entre nos pages (passer de la page des Formulaires à la page Localisation ou passer d'un historique des positions à la page de Localisation…)

node_modules



Arborescence 9 - node_modules

Contenu

Ce dernier dossier est le plus volumineux de l'application car il contient tous les modules que nous utilisons pour développer notre application.

Google Maps API

Lors du build de l'application il est nécessaire de définir la clé API.
Nous vous redirigeons vers le tutoriel suivant qui définit exactement les étapes à faire:

https://docs.expo.io/versions/latest/sdk/map-view/#deploying-to-a-standalone-app-on-android