l'école d'ingénierie informatique

EPSI BORDEAUX - I1 EISI

Module Atelier - Architecture applicative (2/9)

Code Module	Durée	Titre Diplôme	Bloc de Compétences	Promotion	Auteur
ARCE842 - DEVE702	20h	EISI / RNCP 35584	Concevoir & Développer des solutions applicatives métiers	2023/2024	Julien COURAUD

3. Setup du projet et de l'environnement de développement.

3.1 Créer un projet Expo / React Native avec un starter-kit

Assurez-vous d'avoir Node.js installé sur votre système. Vous pouvez télécharger la dernière version depuis le site officiel.

Nous utiliserons le template 'with-socket-io' proposé par Expo pour directement travailler sur un socle applicatif ayant des exemples de connexion entre des clients et un serveur WebSocket.

Starter Kit: https://github.com/expo/examples/tree/master/with-socket-io

Il s'agit d'un exemple simple où le serveur renvoie simplement l'heure aux clients en écoute du socket `time-msg`.

Vous pouvez utiliser ce site pour télécharger directement le sous dossier 'with-socket-io': https://download-directory.github.io/

3.2 Lancer l'application en mode développement

Le starter-kit 'with-socket-io' de Expo contient la partie cliente et la partie serveur au sein du même répertoire.

Nous allons donc devoir lancer les deux en même temps pour que nos entités fonctionnent entres elles.

A la racine du projet, nous lançons le client mobile Expo après avoir installé ses dépendances

```
npm install
npx expo install @expo/metro-runtime
npx expo start
```

Une fois le client lancé, vous aurez plusieurs possibilité pour émuler votre application (browser localhost, Android, iOS, etc..) ou accès à plusieurs fonctionnalités utiles (debugger, reload app, etc..).

Vous aurez également accès à un QRCode qui une fois scanné avec votre smartphone vous permettra d'assister en direct au rendu de votre application (https://expo.dev/client)

Puis nous lançons le serveur à la racine du dossier 'backend' après avoir installé ses dépendances:

```
npm install
npm run start
```

Une fois vos entités lancés, vous devriez voir apparaître à l'écran l'heure du serveur, et dans la console Serveur des informations sur les clients connectés.

La connexion 'localhost' peut poser un soucis sur le réseau local et les entités peuvent avoir du mal à se trouver. Vous pouvez utiliser ce code pour gérer les différentes conditions là où sont instanciés les sockets:

```
console.log('Emulation OS Platform: ', Platform.OS);
// Also usable : "http://10.0.2.2:3000"
export const socketEndpoint = Platform.OS === 'web' ? "http://localhost:3000" : "http://172.20.10.3:3000";
// '172.20.10.3' must be replaced by your intern IP adress
```

3.3 Configurer Android Studio pour l'émulation de l'application React Native.

Pour émuler notre application également sur Android Studio, nous devrons suivre cette documentation:
https://docs.expo.dev/workflow/android-studio-emulator/

4. WebSockets et communication Client / Server

4.1 Restructuration de l'arborescence des fichiers

Ce starter-kit Expo 'with-socket-io' possède un dossier pour le serveur et toute sa logique Client est située dans le 'App.js' (gestion des sockets, etc…).

Afin de batir une architecture complexe, nous devons restructurer notre arborescence de sorte à prévoir l'embarquement de fonctionnalités sur notre plateforme.

Nous avons besoin d'un dossier 'app' (appelez-le comme bon vous semble) qui contiendra la logique côté Client.

- Installation des dépendances de routing côté Client:

```
npm install --save @react-navigation/native
npm install --save @react-navigation/stack
```

- Déplacez le contenu de 'App.js' à l'interieur d'un dossier fichier './app/screens/home.screen.js' et appelez votre composant `HomeScreen`.

- Utilisez `NavigationContainer` et les fonctionnalités de `Stack` de `@react-navigation` pour définir votre premier niveau d'appels à vous routes.

```
// ./App.js
```

```
import React from 'react';
import { LogBox } from 'react-native';
import { NavigationContainer } from '@react-navigation/native';
import { createStackNavigator } from '@react-navigation/stack';
import HomeScreen from './app/screens/home.screen';

const Stack = createStackNavigator();
LogBox.ignoreAllLogs(true);

function App() {
  return (
      <NavigationContainer>
        <Stack.Navigator initialRouteName="HomeScreen">
          <Stack.Screen name="HomeScreen" component={HomeScreen} />
        </Stack.Navigator>
      </NavigationContainer>
  );
}

export default App;
```

4.2 Préparation des fichiers nécéssaires au démarrage du projet

Frontend

Selon l'architecture vue en cours, vous pouvez déjà au sein du dossier 'app' préparer vos sous-dossiers qui recevront les futurs fonctionnalités à implémenter:

```
- websocket-server-folder /
- ...
- node_modules /
- package.json
- App.js
- app /
    - screens
        - home.screen.js
        - online-game.screen.js
        - vs-bot-game.screen.js
    - controllers
    - contexts
    - components
    - services
```

Backend

Côté serveur, nous garderons dans un premier temps le fichier 'index.js', qui sera notre porte d'entrée sur notre serveur et sa logique de jeu. Nous pouvons d'ore et déjà divisé notre fichier en 4 parties:

```
const app = require('express')();
const http = require('http').Server(app);
const io = require('socket.io')(http);

// ------------------------------------------------
// -------- CONSTANTS AND GLOBAL VARIABLES ----------
// ------------------------------------------------

// ---------------------------------
// -------- GAME METHODS ----------
// ---------------------------------

// ---------------------------------------
// -------- SOCKETS MANAGEMENT ----------
// ---------------------------------------
```

```
io.on('connection', socket => {
  console.log(`[${socket.id}] socket connected`);
  socket.on('disconnect', reason => {
    console.log(`[${socket.id}] socket disconnected - ${reason}`);
  });
});

setInterval(() => {
  io.sockets.emit('time-msg', { time: new Date().toISOString() });
}, 1000);

// ------------------------------------
// -------- SERVER METHODS ------------
// ------------------------------------

app.get('/', (req, res) => res.sendFile('index.html'));

http.listen(3000, function(){
  console.log('listening on *:3000');
});
```

4.3 Contexte React et connexions aux Sockets optimisées

Comme vu en cours, chaque entité et composant au sein du Frontend aura sa propre logique embarquée et ses propres responsabilités. Ils devront communiquer fréquemment avec le serveur WebSocket.

Nous allons utiliser les contextes React pour instancier une seule fois à l'arrivée sur l'application (au sein du `App.js`) la connexion socket Client/Server puis la rendre disponible à travers sorte de variable globale `SocketContext` via un Provider React.

Celle-ci sera disponible dans toute l'application et à n'importe quel moment via la méthode React `useContext()`

- Créer un fichier 'app/contexts/socket.context.js'

```
// app/contexts/socket.context.js

import React from "react";
import { Platform } from 'react-native';
import io from "socket.io-client";

console.log('Emulation OS Platform: ', Platform.OS);
// Also usable : "http://10.0.2.2:3000"
export const socketEndpoint = Platform.OS === 'web' ? "http://localhost:3000" : "http://172.20.10.3:3000";

export const socket = io(socketEndpoint, {
  transports: ["websocket"],
});;

export let hasConnection = false;

socket.on("connect", () => {
  console.log("connect: ", socket.id);
  hasConnection = true;
});

socket.on("disconnect", () => {
  hasConnection = false;
  console.log("disconnected from server"); // undefined
  socket.removeAllListeners();
});

export const SocketContext = React.createContext();
```

- Enrichissement du 'App.js'

```
// App.js

// ...
import { SocketContext, socket } from './app/contexts/socket.context';
// ...

function App() {
  return (
    <SocketContext.Provider value={socket}>
      <NavigationContainer>
      // ....
      </NavigationContainer>
    </SocketContext.Provider>
  );
}
```

- Exemple d'utilisation

```
import { useEffect, useState, useContext } from "react";
import { StyleSheet, Text, View } from "react-native";
import { SocketContext } from '../contexts/socket.context';

export default function HomeScreen() {

  const socket = useContext(SocketContext);

  const [time, setTime] = useState(null);

  useEffect(() => {

    socket.on("time-msg", (data) => {
      setTime(new Date(data.time).toString());
    });

    return () => {
      socket.disconnect();
      socket.removeAllListeners();
    };
  }, []);

  return (
    <View style={styles.container}>
      {!socket && (
        <>
          <Text style={styles.paragraph}>
            Connecting to websocket server...
          </Text>
          <Text style={styles.footnote}>
            Make sure the backend is started and reachable
          </Text>
        </>
      )}

      {socket && (
        <>
          <Text style={[styles.paragraph, { fontWeight: "bold" }]}>
            Server time
          </Text>
          <Text style={styles.paragraph}>{time}</Text>
        </>
      )}
    </View>
  );
}

const styles = StyleSheet.create({
  // ...
});
```