step2 フロントエンド

step1 お疲れ様でした。 これからstep2にのフロントエンドを書いていきたいと思います。

step2も同様に生のJavaScriptだけで書いていきます。 今回はリアルタイムのオープンチャットを書いていきます。

step1で作ったチャットbotとチャットできる機能もせっかくなので使えるようにしていきましょう。

仕様

- htmlファイルとjsファイルで構成されている。
- トップページからチャットbotとオープンチャットにアクセスできる
- チャットbotとUIは殆ど同一
- オープンチャットはニックネームをつけることができる。
- htmlファイルは`index.html,cahtroom.html,bot.html`で全て同一フォルダ上に存在し、それぞれトップページ、オープンチャット、チャットbotとのチャット用のhtmlファイルである。
- step1.jsにstep1のコードをコピペして用いて良い。index.jsは今回新たに書くファイルである。

リアルタイムチャットを複数人で行うためにWebSocketを用います。またstep1と同様に使用を元にチャレンジできる人はチャレンジしてみてください。質問タイムやテキストでもいいので自力でやってくれた人には添削等します。

ではコードを書いていきたいと思います。

ディレクトリ構成から

```
/root
|-backend
|-front
    |-index.html
    |-cahtroom.html
    |-bot.html
    |-index.js
    |-step1.js
```

ディレクトリ構成は上記のようになります。 今回からは明確にフロントエンドとバックエンドを分けるためにfrontディレクトリを用意します。

index.htmlから書いていきます。

```
<!DOCTYPE html>
<head>
    <title>step2</title>
    <link rel="stylesheet" href="./style.css">
</head>
<body>
    <h1>step2</h1>
    <a href="./chatroom.html">chatroom</a>
```

```
    <a href="./bot.html">botとお話し</a>
</body>
```

このように書いてください。 トップページは特に機能はなく、それぞれの機能のページまでの画面遷移の役割を担わせることにしました。 aタグを用いてリンクを設定してください。リンク先はファイルの中の相対パスを用いて指定しています。

次はchatroom.htmlです。

```
<!DOCTYPE html>
<head>
    <title>オープンチャットルーム</title>
</head>
<body>
    <h1>オープンチャットルーム</h1>
    <input type="text" id="message" placeholder="メッセージを入力" >
    <button id="send" onclick="" >送信</button>
    <div id="chatSpace"></div>
</body>
<script type="text/javascript" src="./index.js"></script>
```

この様に配置してください。step1と同じくheadタグ内には最低限の記述にしています。 scriptタグでJavaScriptを読み込んでいます。htmlファイル内にJSを直接書くことはほとんどありません。 原則scriptタグで読み込んでください(htmlファイル自体を書くことが時代的に少なくなりつつありますが)。

step1と共通する機能であるメッセージの送信とdomへの反映のためのタグは最初に記述してしまいましょう。 htmlに必要な残りの要素は仕様の中から**ニックネームの登録**であると読み取れるかと思います。 下記にこれからの方針を示します

> **方針**
>
> 1. ニックネームを登録するためのdomを配置する。
> 2. ニックネームはサーバーに登録するものではないのでフロントでその状態を管理する
> 3. chat botの時とは違うwebsocketなので接続を確立する
> 4. サーバからの配信に対応する処理を書く
> 5. domへの反映等はstep1と同じ考え方
> 6. サーバから送られてきたデータをフロントで扱いやすく整形する(step1にもありますが。)

方針の1番目のニックネームを登録するためのdomを配置していきましょう。

```
<body>
    <h1>オープンチャットルーム</h1>
    <h3 id="nameArea"></h3>
    <input type="text" id="name" placeholder="ニックネームを設定"/>
    <button id="setName" >設定</button>
    <input type="text" id="message" placeholder="メッセージを入力" disabled>
    <button id="send" disabled>送信</button>
```

```
    <div id="chatSpace"></div>
</body>
```

chatroom.htmlのbody内部をこんな感じに編集してください。 ニックネームを設定する部分と設定するためのボタンを設置していきましょう。またニックネームを編集するまでは送信ができない様にするためにdisabled属性を付与してください。 これで方針の①は達成です。

方針の②から⑥を達成するために、javascriptを書いていきましょう。

最初に参照する参照するDOMを定義していきます。 ③に関する変数も定義したいと思います。

```
//関数内からwsを参照出来る様にするために一旦nullで定義(変数のスコープについて調べてみても良いかも)
let ws = null;
//参照するdomを定数に格納
const inputNickname = document.getElementById("name")
const nameSettingButton = document.getElementById("setName")
const sendButton = document.getElementById("send")
const msgText = document.getElementById("message")
const nameArea = document.getElementById("nameArea")
```

ニックネームの登録とメッセージの送受信に関するdomをここで全体に向けて定義していきます。 websocketを使えるためにwebsocketの為の空の変数を定義していきます。

最初に名前を設定するための関数を定義していきます

```
const nameSetting = () => {
    const nickname = inputNickname.value
    //名前は一度しか使用しないのでdomを削除するためのリストを作成
    const removableDoms = [inputNickname,nameSettingButton]
    //disabledを削除したいdomのリストを作成
    const abeledDoms = [sendButton,msgText]
    nameArea.innerText = nickname
    removeDom(removableDoms)
    abledChatSpace(abeledDoms)
}
```

次に使用後のdomを削除してニックネームの変更をできない様にしていきましょう。 domの削除はremove()で行うことができます。 複数のdomを削除する必要があるので関数化してしまいましょう。

```
//使わないdomを削除するための関数
//doms=array
const removeDom = (doms) => {
    doms.forEach((dom) => dom.remove())
}
//domのdisable属性を削除するための関数
const abledChatSpace = (doms) => {
```

```
    doms.forEach((dom) => dom.disabled = false)
}
```

この様になります。domを削除したら(ニックネームを設定したら)、メッセージの送信をできる様にするためにdisabled属性を消す関数も同時に定義しいました。

> forEachはjavascriptにおける配列操作のメソッドです。これは配列の各要素に処理を加えることができます。**何も返しません。** mapやfilter関数などforEachと同じように配列操作のメソッドは多く存在するので積極的に使いましょう。forやwhileで処理するのはベターではありません。ダサいです。 参考:https://qiita.com/itagakishintaro/items/29e301f3125760b81302

次はチャットのログをdomに反映させる関数を作っていきましょう。

```
//data = string
const updateLog = ( data ) => {
    //stringで帰ってくるのでjson形式にする。
    data = JSON.parse(data)
    const p = document.createElement('p');
    p.innerText = `${data.nickname} : ${data.message}`
    chatSpace.appendChild(p)
}
```

上記の様になります。文字列としてデータがサーバより取得できるので、 JSON形式にして扱いやすくします。その後はstep1と同じ形式でdomに反映させていきます。

注:今回はWebSocketを使っているためinput要素のvalueを直接domにレンダリングする関数は作成しません。

次に今回の肝であるサーバとwebsocket接続を確立する関数を作成していきます。 またこの関数の中でサーバからのメッセージを監視します

```
const connectWsServer = () => {
    //webSocketのクラスをインスタンス化する。(最初に作っておいた変数wsに代入する)
    ws = new WebSocket(`ws://localhost:8080/chat?
nickname=${nameArea.innerText}`)
    //wsの接続が確立された時の処理
    ws.onopen = function() {
        //wsからmessageが来た時の処理。
        ws.onmessage = function( msg ) {
            updataLog( msg.data ) ;
        }
    }
}
```

このような関数になります。処理としては少し気持ち悪い気もしますが、webSocketはサーバからのmessageをlistenすることや接続状態に応じて処理をすることになります。そのためイベント駆動的な関数の書き方をすることになります。

最後にwebsocketのエンドポイントに対してメッセージを送信するための処理を書いていきます。

```
//入力されたテキストをjson形式の文字列に変換
const getMessage = () => {
    const msg = {
        message: msgText.value
    }
    return JSON.stringify(msg)
}
//wsのエンドポイントに上の関数をの戻り値を送信する
const sendMessage = () => {
    //メッセージが空の場合は送信できない様にする。
     if (!msgText.value) {
        alert("文字を入力してから送信してください")
        return
    }
    const msg = getMessage()
    ws.send(msg)
}
```

以上がwebsocketを用いた送信の為の処理です。 最後にEventListnerで各ボタンのイベントを監視し、イベントが起きたタイミングで関数が発火する様にしていきましょう。

```
sendButton.addEventListener( 'click', sendMessage)
nameSettingButton.addEventListener("click", nameSetting)
nameSettingButton.addEventListener("click",connectWsSer ver)
```

これで完成になります。onClick属性を指定するやり方とは別にこの設定方法もあるので覚えておくといいでしょう。