Internship Paper Report

Given tasks:

- WebChat module uses botframework-webchat and bundles with Rollup.JS
- Write a paper report and submit to Github.

---

botframework-webchat

https://github.com/microsoft/BotFramework-WebChat

Description:

Botframework-webchat component is a highly-customizable web-based client for the Bot Framework SDK. This framework provides a UI, which can be used to connect with The Bot. A user can have multiple Bot with a specific functionality and mission (chat, support, ...).

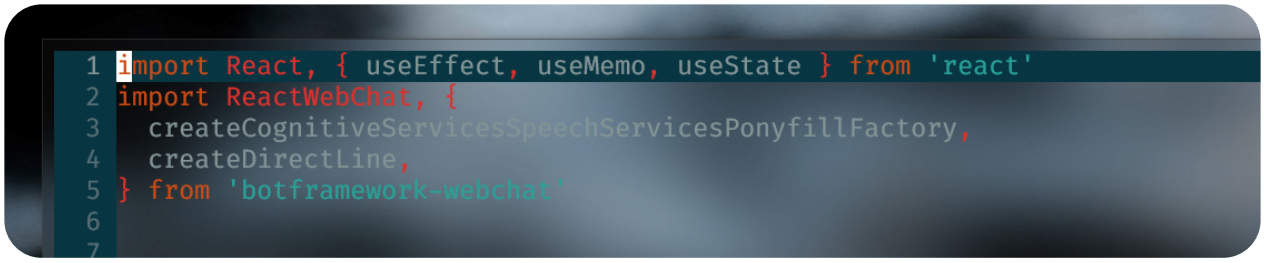
Install:

npm i botframework-webchat

Usage:

We're gonna use React Hook to implement with botframework-webchat.

First of all, import these modules.



```
import React, { useEffect, useMemo, useState } from 'react'
import ReactWebChat, {
  createCognitiveServicesSpeechServicesPonyfillFactory,
  createDirectLine,
} from 'botframework-webchat'
```

- *useEffect* will be called one time only after all the states has been init.
- *useMemo* will check whether the states has been update values. If it was, it would trigger the inside function (useMemo is useful, because we don't want React to render too many times.
- *ReactWebChat* is the main component to call the Web Chat UI.
- *createDirectLine* is the main function to connect with Bot.
- createCognitiveServicesSpeechServicesPonyFillFactory allows user to use voice chat.

```
// states field.
const [tokens, setTokens] = useState<ITokens>({
  directLine: { token: '' },
  speech: { authorizationToken: '', region: '' },
})
```

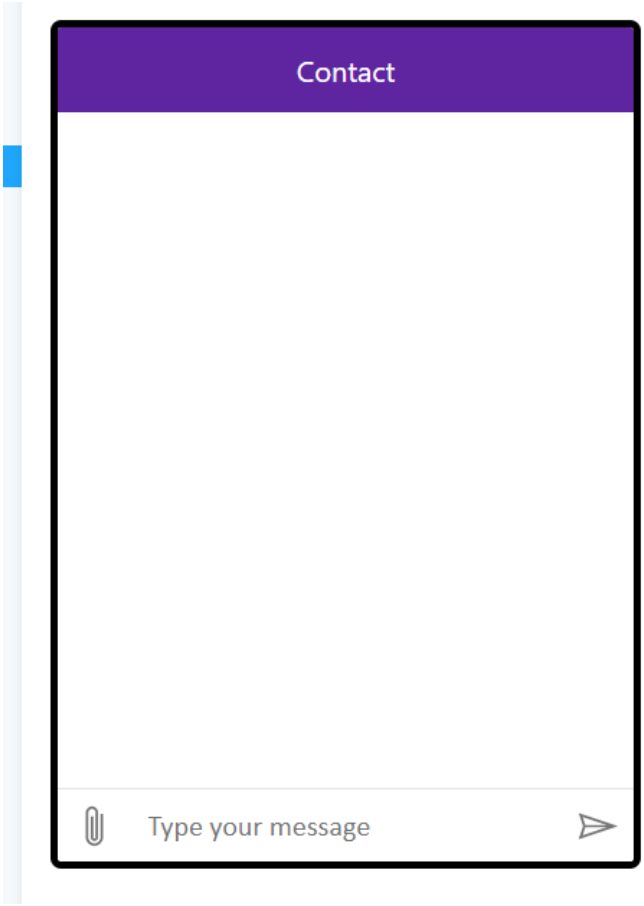
- *tokens* will holds all the tokens from speech and direct line, instead of using 2 states to hold the tokens, we only need to use one so that after state has been updated, React will render only 2 times (The philosophy about this is to optimize as much as you can).
- *setTokens will update the values.*

- *onFetch* is the main function to be invoked after useEffect has been inititialize, it will also trigger these 2 inside functions for fetching direct-line and speech tokens.

- After the tokens has been updated, we'll use the useMemo hooks to trigger the function inside, directLine instance holds the function createDirectLine(), webSpeechPonyfillFactory holds the create...Factory() function.

- We call the ReactWebChat component and pass all those props in.

Result:



Storybook

https://github.com/storybookjs/storybook

Description:

Storybook is like a tool, it let the users to see a specific component in a larger scale with the help of UI display. It make the design development between team members become more easy.

Install:

Due to my project has to be bundled with Rollup, I'm gonna install Storybook from scratch. Because Storybook currently is using with Webpack5 so we need to install:

```
npm i -D @babel/core babel-loader @storybook/react @storybook/addon-actions \
@storybook/addon-links @storybook/addon-essentials @storybook/builder-webpack5 \
@storybook/manager-webpack5
```

Usage:

After we installed those modules, open your IDE and add these code to this file.

.storybook/main.js

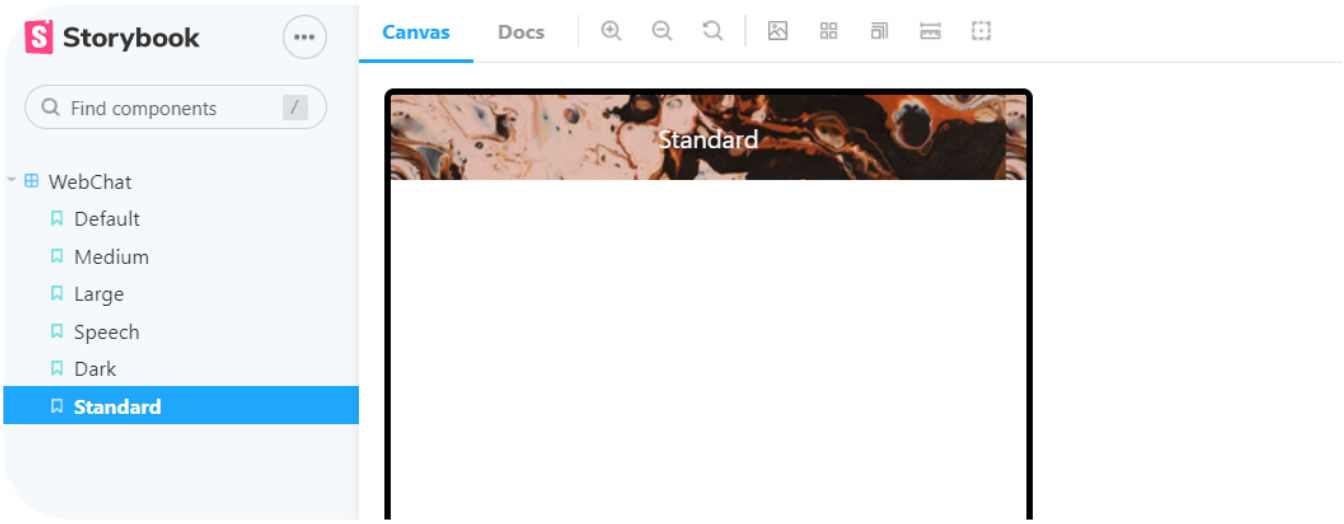
```
module.exports = {
  stories: ['../src/**/*.stories.mdx', '../src/**/*.stories.@(js|jsx|ts|tsx)'],
  addons: ['@storybook/addon-links', '@storybook/addon-essentials'],
  framework: '@storybook/react',
  core: {
    builder: 'webpack5',
  },
}
```

- *main.js* is the configuration file of Storybook, the first line tells storybook where it's gonna load story files.
- *addons* add some plugins to storybook UI.
- *framework* indicates which framework storybook gonna use.
- *core* of couse, webpack5.

```
"start:storybook": "start-storybook -p 6006",
```

- Add this line to package.json and run this, a UI Storybook will pop up in the browser.

Result:



Rollup.JS

https://www.npmjs.com/package/rollup

Description:

- Rollup is one of the most popular bunlders, which is being used to wrap all the source codes up. Rollup is mainly used to bundle the souce code into module, due to its funcionality to split into esm, cjs or umd type.

- In this example, I'm gonna use Rollup with Typescript (because I always want to learn Typescript and I think it's also a good practice to use it for the rest of my life).

Tutorial:

- First of all, install these modules so that our project can use with TypeScript.

npm i -D typescipt @types/react

- After installing, create *tsonconfig.json* in your *root* project and add these codes.

```
{
  "compilerOptions": {
    "declaration": true,
    "declarationDir": "build",
    "module": "esnext",
    "target": "es5",
    "lib": ["es6", "dom", "es2016", "es2017"],
    "sourceMap": true,
    "jsx": "react",
    "moduleResolution": "node",
    "allowSyntheticDefaultImports": true,
    "esModuleInterop": true
  },
  "include": ["src/**/*"],
  "exclude": [
    "node_modules",
    "build",
    "src/**/*.stories.tsx",
    "src/**/*.test.tsx"
  ]
}
```

- *declaration:* tells the TypeScript Compiler to create the *.d.ts for each JavaScript file.
- *declarationDir:* tells the TypeScript Compiler to put all those *.d.ts from *declaration* into that folder.
- *module:* specifies which project Typescript is facing on.
- *target:* if the current project is React, then it's definitely ES5.
- sourceMap: tells the Typescript Compiler create the *.js.map file for Snippet only.
- *moduleResolution:* tells the TypeScript Compiler to follow Node rules.
- *allowSyntheticDefaultImports*: tells the TypeScript that the ES5 and CommonJS import and export are different but it still allow to do that.

Example:

```
const axios = require('axios')        #        import axios from 'axios'
```

- *esModuleInterop:* tells the TypeScript Compiler that it allow to use *asterisk (*)* to import modules.

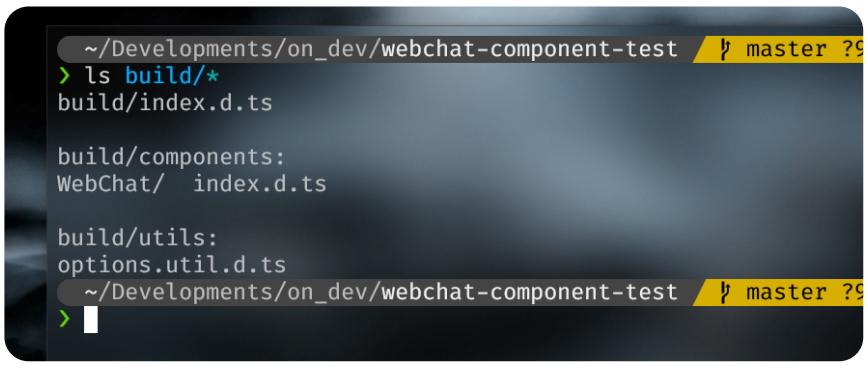
Example:

```
import * as React from 'React';
```

- After creating the *tsconfig.json.* In your terminal enter this command.

```
$    tsc
```

- You will notice that there's a new folder called *build,* which is the location folder we specified in the *tsconfig.json.*



- Greate, now the *build* folder has been created, we will bundle the *WebChat* component (source code) with *Rollup.JS*, go install these dependencies.

```
npm i -D rollup @rollup/plugin-node-resolve @rollup/plugin-common-js rollup-plugin-typescript2 \
rollup-plugin-postcss rollup-plugin-peer-deps-external
```

- *rollup:* main module of Rollup.JS.
- *@rollup/plugin-node-resolve:* the rollup plugin will locate the third party modules inside your *node_modules* folder.
- *@rollup/plugin-commonjs:* this plugin will behave like *Babel* except it'll parse the the old JavaScript into the new one, which is ECMAScript5.
- *rollup-plugin-typescript2:* this plugin allow the Rollup to have the abilities to bundle with TypeScript files.
- *rollup-plugin-peer-deps-external:* this plugin will tell the after-bundling Rollup to tell the user if their projects have all the necessary dependencies.

Example:

```
  },
  "peerDependencies": {
    "botframework-webchat": "≥4.14.0",
    "prop-types": "≥15.0.0",
    "react": "≥17.0.0"
  }
}
```

- *rollup-plugin-postcss:* this plugin allow the Rollup bundle to have the abilities to bundle with CSS files.

- OK! After installing all those things, let's create *rollup.config.js* file and adds these codes.

```
import peerDepsExternal from 'rollup-plugin-peer-deps-external'
import resolve from '@rollup/plugin-node-resolve'
import commonjs from '@rollup/plugin-commonjs'
import typescript from 'rollup-plugin-typescript2'
import { terser } from 'rollup-plugin-terser'
import postcss from 'rollup-plugin-postcss'

const packageJson = require('./package.json')
```

```
export default {
  input: 'src/index.ts',
  output: [
    {
      file: packageJson.main,
      format: 'cjs',
      sourcemap: true,
    },
    {
      file: packageJson.module,
      format: 'esm',
      sourcemap: true,
    },
  ],
  plugins: [
    peerDepsExternal(),
    resolve(),
    commonjs(),
    typescript({ useTsconfigDeclarationDir: true }),
    postcss(),
    terser(),
  ],
}
NORMAL  master  rollup.config.js
```

- *input:* the entries file so Rollup can bundle.
- *output:* because we want our module can use both in non-ecma5 and ecma5, we w-want to split the outputs and have different formats.
- *plugins:* all the plugins you've installed put it all in there.
- The final step, in *package.json* file add these lines.

```
  "main": "build/index.js",
  "module": "build/index.es.js",
  "files": [
    "build"
  ],
```

```
  "build": "rollup -c",
```

- *main* is for CommonJS and *module* is for ECMAScript5.
- After all of that, hit *npm run build* and publish it to NPM and then it's good to go.
- After publishing it to NPM, from your *test-react-project* hit *npm i webchat-component-test,* inside your *App.js.*