DESARROLLO DE APLICACIONES WEB AVANZADO

LABORATORIO N° 08

Chat con Bootstrap, deploy Now.sh

Alumno(s):		***Nota***	
Grupo:		***Ciclo:V***	

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Realiza un deploy con Now.sh					
Utiliza Bootstrap para interfaz de chat					
Desarrolla aplicaciones web con socket					
Realiza con éxito lo propuesto en la tarea.					
Es puntual y redacta el informe adecuadamente					

Laboratorio 08:
Chat con Bootstrap, deploy Now.sh

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento de Socket.io
- Desarrollar aplicaciones web con el framework Bootstrap
- Implementación correcta de Now.sh

Seguridad:

- Ubicar maletines y/o mochilas en el gabinete del aula de Laboratorio.
- No ingresar con líquidos, ni comida al aula de Laboratorio.
- Al culminar la sesión de laboratorio apagar correctamente la computadora y la pantalla, y ordenar las sillas utilizadas.

Equipos y Materiales:

- Una computadora con:
  - Windows 7 o superior
  - VMware Workstation 10+ o VMware Player 7+
  - Conexión a la red del laboratorio

- Máquinas virtuales:
  - Windows 7 Pro 64bits Español - Plantilla

- Instalador de node.js

Procedimiento:

Lab Setup

1. **Configuración inicial del proyecto**
   1.1. Copie el contenido de la carpeta **lab07** (el laboratorio anterior) a excepción de la carpeta **node_modules** dentro de una carpeta llamada **lab08**.
   1.2. Durante este proyecto haremos uso de varias librerías, por lo que para facilitar las cosas, modificaremos nuestro **package.json** agregando las siguientes líneas:

```
{
  "name": "lab06",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@fortawesome/fontawesome-svg-core": "^1.2.18",
    "@fortawesome/free-brands-svg-icons": "^5.8.2",
    "@fortawesome/free-solid-svg-icons": "^5.8.2",
    "@fortawesome/react-fontawesome": "^0.1.4",
    "axios": "^0.18.0",
    "bootstrap": "^4.3.1",
    "i": "^0.3.6",
    "md5": "^2.2.1",
    "npm": "^6.9.0",
    "react": "^16.8.6",
    "react-bootstrap": "^1.0.0-beta.8",
    "react-dom": "^16.8.6",
    "react-helmet": "^5.2.1",
    "react-router": "^5.0.0",
    "react-router-dom": "^5.0.0",
    "react-scripts": "2.1.8",
    "socket.io-client": "^2.2.0"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject",
    "now-build": "react-scripts build"
  },
  "eslintConfig": {
    "extends": "react-app"
  },
```

Las librerías fortawesome son una serie de dependencias para utilizar los íconos de Font Awesome, cuyo listado puede encontrar en el siguiente enlace:

https://fontawesome.com/

Agregaremos Socket.Io para conectar nuestro chat y también md5 para cifrar nuestro correo. Haremos un uso de gravatar (no se preocupe, será explicado en el laboratorio) por lo que md5 caerá perfecto.

React-helmet es una librería que permite modificar las cabeceras de la ruta abierta. Esto es muy útil para el SEO (Search Engine Optimization) aunque en nuestro caso lo usaremos para modificar los estilos globales.

Así mismo, aprovecharemos para modificar la sección de scripts, donde agregaremos la función build con el apelativo now-build, esta parte es exclusiva del deploy del final de laboratoio.

1.3. Modificaremos la vista **welcome.js** para agregar un enlace al chat.

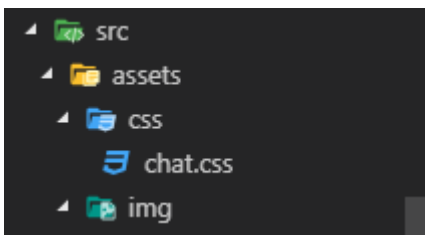
```
const Welcome = props => {
    const userName = localStorage.getItem('userName');
    return (
        <div name="container">
            <div className="jumbotron">
                <h1>Bienvenido {userName}!</h1>
                <button
                    className="btn btn-primary"
                    onClick={() => {
                        props.history.push('/chat');
                    }}
                >
                    Vamos al Chat!
                </button>
            </div>
        </div>
    );
};
```

1.4. Así mismo, nos corresponde agregar Chat.js a la invocación de las rutas en **App.js**

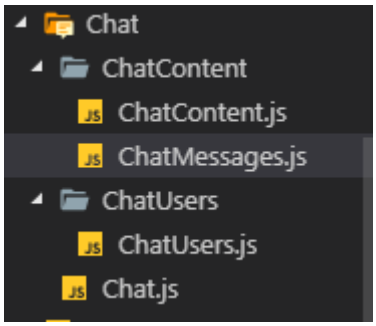
```
import Layout from './hoc/Layout/Layout';
import Welcome from './views/Welcome';
import Chat from './views/Chat/Chat';
import NotFound from './views/NotFound';

class App extends Component {
    render() {
        return (
            <BrowserRouter>
                <Layout>
                    <Switch>
                        <Route path="/" exact component={Home} />
                        <Route path="/details" component={Details} />
                        <Route path="/login" component={Login} />
                        <Route path="/welcome" component={Welcome} />
                        <Route path="/chat" component={Chat} />
                        <Route
                            path="/logout"
```

2. Creación de vista Chat

2.1. Para facilitar su labor, descargue el archivo **chat.css** (adjunto con este laboratorio) y péguelo dentro de la carpeta **css** que será creada dentro de **assets**.



2.2. A continuación, crearemos la carpeta **Chat** dentro de **views**, que a su vez contendrá la siguiente estructura de archivos dentro de sí



2.3. Empezaremos llenando el contenido de **Chat.js**

```
import React, { Component, Fragment } from 'react';
import { Helmet } from 'react-helmet';

import ChatContent from './ChatContent/ChatContent';
import ChatUsers from './ChatUsers/ChatUsers';

import '../../assets/css/chat.css';

class Chat extends Component {
    render() {
        return (
            <Fragment>
                <Helmet>
                    <title>Chat Tecsup</title>
                    <style type="text/css">{`
                        body,
                        html {
                            height: 100%;
                            margin: 0;
                            background: #7f7fd5;
                            background: -webkit-linear-gradient(to right, #91eae4, #86a8e7, #7f7fd5);
                            background: linear-gradient(to right, #91eae4, #86a8e7, #7f7fd5);
                        }
                    `}</style>
                </Helmet>
```

Esta es la primera invocación que hacemos de **Helmet**. Puede apreciar que podremos cambiar el título de la página web y también estilos globales gracias a este componente.

```
                <div
                    className="container-fluid h-100"
                    style={{
                        margin: '10px 0'
                    }}
                >
                    <div className="row justify-content-center h-100">
                        <div class="col-md-4 col-xl-3 chat">
                            <ChatUsers />
                        </div>
                        <div class="col-md-8 col-xl-6 chat">
                            <ChatContent />
                        </div>
                    </div>
                </div>
            </Fragment>
        );
    }
}

export default Chat;
```

2.4. Antes de continuar con la creación del componente ChatUsers (que mostrará el listado de usuarios conectados) crearemos el archivo **avatar.js** dentro de la carpeta **utils**.

```
import md5 from 'md5';

const getAvatar = email => {
    return `https://www.gravatar.com/avatar/${md5(email)}?d=identicon`;
};

export default getAvatar;
```

Gravatar es un servicio de avatares en línea, es decir, imágenes de perfil. Cuando le pasamos como argumento nuestro correo cifrado en md5, nos devolverá un avatar de la cuenta de correo que hayamos puesto pública, facilitando la obtención de una imagen para nuestros nuevos usuarios.

2.5. Ahora sí, llenaremos el contenido de **ChatUsers.js**

```
import React, { Component } from 'react';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import { faSearch } from '@fortawesome/free-solid-svg-icons';

import getAvatar from '../../../utils/avatar';

class ChatUsers extends Component {
    render() {
        return (
            <div className="card mb-sm-3 mb-md-0 contacts_card">
                <div className="card-header">
                    <div className="input-group">
                        <input
                            type="text"
                            placeholder="Buscar..."
                            name=""
                            className="form-control search"
                        />
                        <div className="input-group-prepend">
                            <span className="input-group-text search_btn">
                                <FontAwesomeIcon icon={faSearch} />
                            </span>
                        </div>
                    </div>
                </div>
```

Este componente (aún falta completar la última parte, pero aprovecho para explicar) FontAwesomeIcon hace uso de la librería ya antes mencionada. Es muy útil ya que tiene un catálogo enorme de íconos gratuitos y de fácil uso como se puede apreciar.

```
                <div className="card-body contacts_body">
                    <ui className="contacts">
                        <li>
                            <div className="d-flex bd-highlight">
                                <div className="img_cont">
                                    <img
                                        src={getAvatar('fnaquiravargas@gmail.com')}
                                        alt="user"
                                        className="rounded-circle user_img"
                                    />
                                    <span className="online_icon offline" />
                                </div>
                                <div className="user_info">
                                    <span>Juan Perez</span>
                                    <p>Juan se conectó 7 mins atrás</p>
                                </div>
                            </div>
                        </li>
                    </ui>
                </div>
                <div className="card-footer" />
            </div>
        );
    }
}

export default ChatUsers;
```

2.6. Llenaremos el contenido de **ChatMessages.js.** Por el momento, este componente nos mostrará solamente dos imágenes, puede cambiar los correos puestos por algún correo de su preferencia.

```
import React from 'react';

import getAvatar from '../../../utils/avatar';

const ChatMessages = props => {
    let email = 'fnaquiravargas@gmail.com';
    let classMessage = 'd-flex justify-content-end mb-4';
    if (props.self === 1) {
        email = 'fnaquira@tecsup.edu.pe';
        classMessage = 'd-flex justify-content-start mb-4';
    }
    return (
        <div className={classMessage}>
            <div className="img_cont_msg">
                <img
                    src={getAvatar(email)}
                    alt={email}
                    className="rounded-circle user_img_msg"
                />
            </div>
            <div className="msg_cotainer">
                {props.line}
                <span className="msg_time">8:40 AM, Today</span>
            </div>
        </div>
    );
};

export default ChatMessages;
```

3. Conexión de Socket.IO

3.1. Vamos a crear un archivo auxiliar para el manejo de sockets. Cree el archivo **socket.js** dentro de **utils** con el siguiente contenido.

```
import openSocket from 'socket.io-client';
const socket = openSocket('http://localhost:5000');

const receiveMessage = cb => {
    socket.on('message', line => cb(line));
};

const emmitMessage = line => {
    socket.emit('message', line);
};

export { receiveMessage, emmitMessage };
```

Note que estamos utilizando por el momento las funciones ya implementadas de socket.io en el curso de **Aplicaciones Móviles Multiplataforma**, lo que indica que el chat de ahora será compatible con el de su aplicación en React Native.

3.2. Empezamos a llenar el contenido de **ChatContent.js**, que es quien contendrá la funcionalidad de crear y recibir mensajes. De hecho, la primera parte es el manejo del estado y como agregar o recibir nuevos mensajes.

```
import React, { Component } from 'react';
import { FontAwesomeIcon } from '@fortawesome/react-fontawesome';
import {
    faLocationArrow,
    faPaperclip
} from '@fortawesome/free-solid-svg-icons';

import getAvatar from '../../../utils/avatar';
import { receiveMessage, emmitMessage } from '../../../utils/socket';

import ChatMessages from './ChatMessages';

class ChatContent extends Component {
    state = {
        text: '',
        lines: []
    };
    componentDidMount() {
        receiveMessage(this.getmessage);
    }
    componentDidUpdate() {
        const objDiv = document.getElementById('msg_card_body');
        objDiv.scrollTop = objDiv.scrollHeight;
    }
    inputHandler = e => {
        this.setState({ text: e.target.value });
    };
    submitHandler = e => {
        e.preventDefault();
        emmitMessage(this.state.text);
        const newLines = [...this.state.lines];
        newLines.push({ line: this.state.text, self: 0 });
        this.setState({
            text: '',
            lines: newLines
        });
    };
```

```
    getmessage = line => {
        const newLines = [...this.state.lines];
        newLines.push({ line: line, self: 1 });
        this.setState({ lines: newLines });
    };
    render() {
        return (
            <div className="card">
                <div className="card-header msg_head">
                    <div className="d-flex bd-highlight">
                        <div className="img_cont">
                            <img
                                src={getAvatar('fnaquiravargas@gmail.com')}
                                alt="user"
                                className="rounded-circle user_img"
                            />
                            <span className="online_icon" />
                        </div>
                        <div className="user_info">
                            <span>Chat con Juan Perez</span>
                            <p>177 Mensajes</p>
                        </div>
                    </div>
                </div>
                <div className="card-body msg_card_body" id="msg_card_body">
                    {this.state.lines.map(item => {
                        return <ChatMessages {...item} />;
                    })}
                </div>
                <div className="card-footer">
                    <form onSubmit={this.submitHandler}>
                        <div className="input-group">
                            <div className="input-group-append">
                                <span className="input-group-text attach_btn">
                                    <FontAwesomeIcon icon={faPaperclip} />
                                </span>
                            </div>
```

```
                            <input
                                value={this.state.text}
                                onChange={this.inputHandler}
                                className="form-control type_msg"
                                placeholder="Escribe algo..."
                            />
                            <div className="input-group-append">
                                <span
                                    className="input-group-text send_btn"
                                    onClick={this.submitHandler}
                                >
                                    <FontAwesomeIcon icon={faLocationArrow} />
                                </span>
                            </div>
                        </div>
                        <button type="submit" style={{ display: 'none' }} />
                    </form>
                </div>
            </div>
        );
    }
}

export default ChatContent;
```

3.3. Hasta este momento, su chat ya estará listo para ser probado. Pruebe duplicando pestañas o incluso conectando la aplicación de su celular y probando la conversación entre las interfaces.

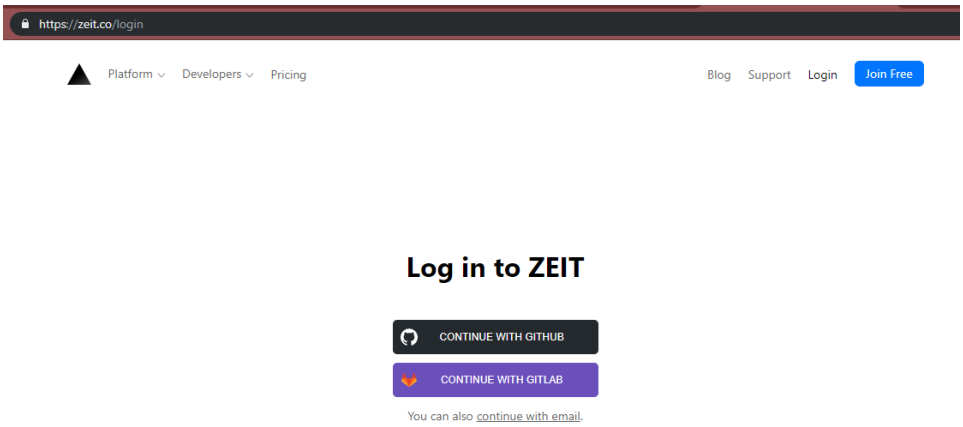
4. Deploy con now.sh

4.1. Now.sh es un producto de Zeit y nos ayuda a levantar micro servicios en la nube sin tener que sufrir con configuraciones tortuosas. Para utilizarlo, vamos a instalarlo globalmente.

```
$ npm install -g now
```

4.2. Crearemos una cuenta en el sitio oficial del servicio.

https://zeit.co/login



4.3. Ahora copiaremos el archivo **now.json** (adjunto con este laboratorio) en la raíz de nuestro proyecto y ejecutaremos el comando **now**. Esto será suficiente para hacer el deploy de la aplicación de React y tener un sitio web corriendo, inclusive con certificado digital incluido.

```
favic@INNISTRAD:/mnt/c/Users/favio/code/tecsup/dawa/lab08$ now
> UPDATE AVAILABLE The latest version of Now CLI is 15.2.0
> Read more about how to update here: https://zeit.co/update-cli
> Changelog: https://github.com/zeit/now-cli/releases/tag/15.2.0
> Deploying /mnt/c/Users/favio/code/tecsup/dawa/lab08 under fnaquiravargas
> Using project dawa-react
> Synced 1 file (936B) [1s]
> https://dawa-react-2fq0whuh8.now.sh [v2] [2s]
┌ package.json        Ready               [1m]
├── asset-manifest.json (1.18KB)
├── favicon.ico (3.78KB)
├── index.html (2.08KB)
├── manifest.json (306B)
├── precache-manifest.38cfff0c4b16d4e43257cd7f63368370.js (1.11KB)
└── 16 output items hidden
> Ready! Aliased to https://dawa-react.fnaquiravargas.now.sh [1m]
```

4.4. Now.sh nos permite hacer deploy no solamente de sitios en React, sino de sitios estáticos, soluciones en node, php, go, gatsby, y muchas más que pueden ser encontradas en su documentación oficial.

https://zeit.co/examples/

6. **Finalizar la sesión**
   6.1. Apagar el equipo virtual
   6.2. Apagar el equipo

Conclusiones:

Indicar las conclusiones que llegó después de los temas tratados de manera práctica en este laboratorio.