

การสร้างหน้า Login ด้าน Client

สร้างไคลเอนต์สำหรับโปรเจกใหม่

- สร้างไคลเอนต์ loginExample
- รัน npm init เพื่อสร้าง package.json
- ย้ายไปยังไคลเอนต์ loginExample
- สร้างไคลเอนต์ client และสร้าง react app โดยใช้คำสั่ง create-react-app client

หน้า Homepage และ หน้า login ในส่วนของ client

การใช้ react-router เพื่อสร้าง routing path

React Router แบ่งออกเป็น 3 packages คือ react-router, react-router-native และ react-router-dom โดย react-router เป็นส่วน core-routing components และฟังก์ชัน และอีก 2 packages สนับสนุนการทำงานภายใต้สิ่งแวดล้อมของ browser และ mobile กรณีของเว็บ เราใช้ react-router-dom

- ติดตั้ง react-router-dom: npm install --save react-router-dom

สำหรับ Browser ใช้ BrowserRouter เพื่อสนับสนุนการร้องขอหน้าเพจที่เป็น dynamic (Dynamic Website) และใช้ <HashRouter> สำหรับหน้าเว็บสถิตย (Static Website)

การใช้ Routing:

Router component ภายในมีเพียง 1 child element เราจึงใช้ App เป็น child สำหรับหน้าหลักของ react (index.js)

```
import React from 'react';
import ReactDOM from 'react-dom';
import { BrowserRouter } from 'react-router-dom';
import App from './App';
import registerServiceWorker from './registerServiceWorker';

ReactDOM.render(
  <BrowserRouter><App /></BrowserRouter>,
  document.getElementById('root')
);
registerServiceWorker();
```

- กำหนด routing สำหรับไปยังหน้าเพจต่าง ๆ โดยใช้ Route ของ react-router-dom ซึ่งสามารถกำหนดเส้นทางและ component ที่จะอ้างถึง ด้วย props: path และ component ตามลำดับ เช่น ถ้าต้องการกำหนดเส้นทาง /login ให้อ้างไปยัง component LoginPage ทำได้โดย
<Route path="/login" component={LoginPage} />
ซึ่งจะ match ทั้ง /login หรือ /login/user ฯลฯ แต่จะไม่ match กับ / แต่ถ้าต้องการให้ match แบบตรงตัว

เท่านั้นให้ระบุ exact เพิ่มไว้ด้วย

กรณีตัวอย่างกำหนดให้ App.js มี routing 2 เส้นทางคือ root (/) ไปยัง HomePage และ /login ไปยัง LoginPage จะได้โค้ดเป็น

```
import React from 'react';
```

```
import { Route } from 'react-router-dom';
import HomePage from './components/pages/HomePage';
import LoginPage from './components/pages/LoginPage';

const App = () => (
  <div className="ui container" >
    <Route exact path="/" component={HomePage} />
    <Route exact path="/login" component={LoginPage} />
  </div>
);
export default App;
```

รายละเอียดเพิ่มเติมอ่านได้ที่ <https://reacttraining.com/react-router/web/guides/philosophy>

- การกำหนดการเชื่อม (link) สำหรับเป็น anchor element เพื่อให้สามารถคลิกแล้วย้ายไปยัง component เหล่านั้นได้ กำหนดได้โดยใช้ component: Link ของ react-router-dom ซึ่งจะ load เพียงส่วนของ component นั้น (แตกต่างจาก <a> ซึ่งจะ load หรือ reload ทั้งหน้าเพจใหม่)

```
import React from 'react';
import { Link } from 'react-router-dom';

const HomePage = () => (
  <div>
    <h1>Home</h1>
    <Link to="/login">Login</Link>
  </div>
);
export default HomePage;
```

Home

Login

สร้างหน้า HomePage อย่างง่าย

```
import React from 'react';
import { Link } from 'react-router-dom';

const HomePage = () => (
  <div>
    <h1>Home</h1>
    <Link to="/login">Login</Link>
  </div>
);
export default HomePage;
```

เพิ่มการตกแต่งโดย semantic-ui-react

ดูเพิ่มเติมที่ (<https://semantic-ui.com/>, <https://react.semantic-ui.com/introduction>)

- ติดตั้ง semantic-ui-react: npm install --save semantic-ui-react

semantic-ui-react เป็น UI framework สำหรับการทำงานกับ react มีองค์ประกอบ UI ต่าง ๆ เช่น Menu, Form ฯลฯ เพื่อการใช้งาน

การตกแต่ง container:

- เพิ่มการใช้ stylesheet ใน index.js โดย import 'semantic-ui-css/semantic.min.css';

Home

Login

- ปรับแต่ง container ของ App โดยกำหนด class ให้ div เป็น `<div className="ui container">`

เพิ่มเมนู

- สร้างเมนู NavMenu (NavMenu.js) ดูการใช้เมนูที่ <https://react.semantic-ui.com/collections/menu>
Menu.Item มี prop as ใช้เพื่อกำหนดชื่อฟังก์ชันหรือ element ที่จะให้ทำงาน เราจึงสามารถใช้ Link และกำหนด prop อื่น เช่น to exact ให้กับมันได้

```
import React from 'react';
import { Menu } from 'semantic-ui-react';
import { Link } from 'react-router-dom';

const NavMenu = () => (
  <Menu pointing secondary>
    <Menu.Item as={Link} exact to="/">Home</Menu.Item>
    <Menu.Menu position='right'>
      <Menu.Item as={Link} exact to="/login">Login</Menu.Item>
    </Menu.Menu>
  </Menu>
);

export default NavMenu;
```

เพิ่มโค้ดใน App ให้ใช้ Menu

```
import React from 'react';
import { Route } from 'react-router-dom';
import HomePage from './components/pages/HomePage';
import LoginPage from './components/pages/LoginPage';
import NavMenu from './components/pages/NavMenu';

const App = () => (
  <div className="ui container">
    <header>
      <NavMenu />
    </header>
    <section>
      <Route path="/" exact component={HomePage} />
      <Route path="/login" exact component={LoginPage} />
    </section>
  </div>
);

export default App;
```

แก้ไขให้เมนูแสดง active link

และเพื่อกำหนด link ที่ active สามารถใช้ NavLink ของ router แทน Link โดยกำหนด prop `activeClassName='active'` เพื่อให้เมื่อเปลี่ยนไปยัง link ที่หน้าตรงกันนั้นจะแสดงให้เห็นว่า active ด้วย

```
import React from 'react';
import { Menu } from 'semantic-ui-react';
import { NavLink } from 'react-router-dom';

const NavMenu = () => (
```

```

    <Menu pointing secondary>
      <Menu.Item as={NavLink} exact to="/" activeClassName='active'>Home</Menu.Item>
      <Menu.Menu position='right'>
        <Menu.Item as={NavLink} exact to="/login"
activeClassName='active'>Login</Menu.Item>
      </Menu.Menu>
    </Menu>
  );
}

export default NavMenu;

```

สร้าง react: Login Form โดยใช้ semantic-ui-react ช่วย

ดูเพิ่มเติมที่ <https://react.semantic-ui.com/collections/form>

- สร้างฟอร์มสำหรับการ login โดยมีข้อมูล state คือ email และ password

```

import React from 'react';
import { Form, Button } from 'semantic-ui-react';

class LoginForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      data: { email: '', password: '' },
    };
  }

  onChange = (e) => {
    this.setState({ data: { ...this.state.data, [e.target.name]: e.target.value } });
  }

  render() {
    const { data } = this.state;
    return (
      <Form>
        <Form.Field>
          <label htmlFor="email">Email
            <input
              type="email"
              id="email"
              name="email"
              placeholder="your@email.com"
              value={data.email}
              onChange={this.onChange}
            />
          </label>
        </Form.Field>
        <Form.Field>
          <label htmlFor="password">Password
            <input
              type="password"
              id="password"
              name="password"
              placeholder="password"
              value={data.password}
              onChange={this.onChange}
            />
          </label>
        </Form.Field>
      </Form>
    );
  }
}

```

```

        />
      </label>
    </Form.Field>
    <Button>Submit</Button>
  </Form>
);
}
}
export default LoginForm;

```

เพิ่มเติมให้ตรวจสอบความถูกต้องของข้อมูลเมื่อ submit ฟอรั่ม โดยใช้ validator

- ติดตั้ง validator และ prop-types: `npm install --save validator prop-types`
- สร้าง component `InlineError` เพื่อแสดงข้อความ error โดย `InlineError` ใช้ `PropTypes` ตรวจสอบว่าได้ส่ง prop ที่เหมาะสม ในที่นี้คือ `text` ที่มีชนิดเป็น `string` มาให้ component นี้

```

import React from 'react';
import PropTypes from 'prop-types';

const InlineError = ({ text }) => <span style={{ color: '#9f3a38' }}>{ text }</span>;

InlineError.propTypes = {
  text: PropTypes.string.isRequired,
};

export default InlineError;

```

- ปรับปรุงการทำ submit ฟอรั่มให้ตรวจสอบข้อมูล โดยเพิ่มฟังก์ชัน `onSubmit` ตรวจสอบความถูกต้องโดยเรียกใช้ฟังก์ชัน `validate`

```

onSubmit = () => {
  const errors = this.validate(this.state.data);
  this.setState({ errors });
}

```

- ฟังก์ชัน `validate` กรณีพบความผิดพลาด เพิ่ม `errors` ที่ผิดพลาดนั้นเข้ากับ `state` ของ component

```

validate = (data) => {
  const errors = {};
  if (!Validator.isEmail(data.email)) errors.email = 'Invalid email';
  if (!data.password) errors.password = 'Cannot be blanked';
  return errors;
}

```

- เพิ่มข้อความกรณีที่มีความผิดพลาดเข้ากับแต่ละ element ของฟอรั่ม เช่นกรณี email ผิดพลาด

```
{errors.email && <InlineError text={errors.email} />}
```

โค้ดของ LoginForm เมื่อปรับปรุงแล้วจะได้ดังนี้

```

import React from 'react';
import { Form, Button } from 'semantic-ui-react';
import Validator from 'validator';
import InlineError from './InlineError';

```

```

class LoginForm extends React.Component {
  constructor(props) {
    super(props);
    this.state = {
      data: { email: '', password: '' },
      loading: false,
      errors: {},
    };
  }

  onChange = (e) => {
    this.setState({ data: { ...this.state.data, [e.target.name]: e.target.value } });
  }
  onSubmit = () => {
    const errors = this.validate(this.state.data);
    this.setState({ errors });
  }

  validate = (data) => {
    const errors = {};
    if (!Validator.isEmail(data.email)) errors.email = 'Invalid email';
    if (!data.password) errors.password = 'Cannot be blanked';
    return errors;
  }

  render() {
    const { data, errors } = this.state;
    return (
      <div>
        <h1>Login Page</h1>
        <div>
          <h1>Login Page</h1>
          <Form onSubmit={this.onSubmit}>
            <Form.Field error={!!errors.email}>
              <label htmlFor="email">Email
                <input
                  type="email"
                  id="email"
                  name="email"
                  placeholder="your@email.com"
                  value={data.email}
                  onChange={this.onChange}
                />
                {errors.email && <InlineError text={errors.email} />}
              </label>
            </Form.Field>
            <Form.Field error={!!errors.password}>
              <label htmlFor="password">Password
                <input
                  type="password"
                  id="password"
                  name="password"
                  placeholder="password"
                  value={data.password}
                  onChange={this.onChange}
                />
                {errors.password && <InlineError text={errors.password} />}
              </label>
            </Form.Field>
          </Form>
        </div>
      </div>
    );
  }
}

```

```

        <Button primary>Submit</Button>
      </Form>
    </div>
  );
}
}

```

```
export default LoginForm;
```

ทดสอบการทำงาน กรณีไม่กรอกข้อมูล กรอก email ไม่ถูกต้อง กรณีกรอกข้อมูลถูกต้อง

- ปรับปรุงโค้ด onSubmit กรณีไม่มีข้อมูลผิดพลาดให้ส่งข้อมูลกลับไปยังหน้าเรียกใช้ component นี้ผ่านฟังก์ชัน this.props.submit

```

onSubmit = () => {
  const errors = this.validate(this.state.data);
  this.setState({ errors });
  if (Object.keys(errors).length === 0) {
    this.props.submit(this.state.data);
  }
}

```

- เพิ่มโค้ดการตรวจสอบการส่ง props submit ที่มีชนิดเป็นฟังก์ชัน มาให้ โดย
- ```
import PropTypes from 'prop-types';
```
- และเพิ่มโค้ดการตรวจสอบใน LoginForm.js ด้วย

```

LoginForm.propTypes = {
 submit: PropTypes.func.isRequired,
};

```

## สร้าง react: LoginPage ที่รับข้อมูลจาก LoginForm

```

import React from 'react';
import LoginForm from '../forms/LoginForm';

class LoginPage extends React.Component {
 submit = (data) => {
 console.log(data);
 }
 render() {
 return (
 <LoginForm submit={this.submit} />
);
 }
}

export default LoginPage;

```

## การสร้าง api ด้าน Server

### เปิด console สำหรับทำงานกับ Server

- ย้ายไปยังไดเรกทอรี loginExample
- ติดตั้ง express, path, body-parser
- สร้างไดเรกทอรี server

### สร้าง Server เพื่อให้ทำงานกับ react client

- สร้างไฟล์ index.js ภายใต้อไดเรกทอรี server

```
var express = require('express');
var path = require('path');
var bodyParser = require('body-parser');

var app = express();
app.use(bodyParser.json()); // for parsing application/json
app.use(bodyParser.urlencoded({ extended: true }));
// for parsing application/x-www-form-urlencoded

var port = 4000;

app.post('/api/signin', (req, res) => {
 res.status(404).json({errors: {global: "Invalid credential"}});
});

app.get("/*", (req, res) => {
 res.sendFile(path.join(__dirname, "index.html"));
});

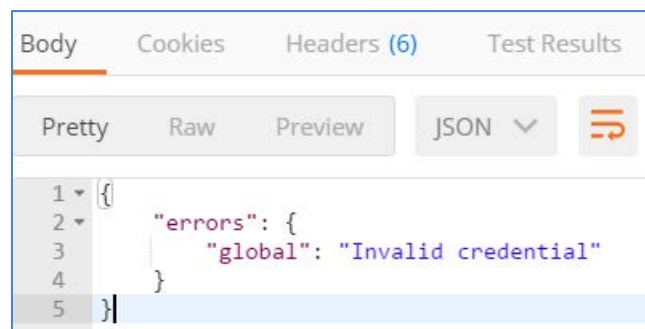
app.listen(port, function() { console.log(`start http server on ${port}`); });
```

- แก้ไข package.json ของ server ในส่วน scripts ให้เป็น

```
"scripts": {
 "start": "node server/index.js",
 "server": "nodemon server/index.js"
},
```

ทดสอบการทำงานกับ postman โดยเรียก post method บน host- <http://localhost:4000/api/signin>

ผลลัพธ์ที่ได้ควรเป็น



ย้อนกลับไปแก้ไขโค้ดของ react client



โดยให้ฟังก์ชัน submit ของ LoginPage.js ร้องขอไปยัง express server

```
submit = (data) => {
 // verify user with the server
 return fetch('/api/signin', {
 method: 'POST',
 body: JSON.stringify(data),
 headers: new Headers({ 'Content-Type': 'application/json' })),
 })
 .then((resp) => {
 const json = resp.json();
 if (resp.status >= 200 && resp.status < 300) {
 return json;
 }
 return json.then((err) => { throw err; });
 });
};
```

ซึ่งเมื่อเรา return fetch ที่ได้ซึ่งอยู่ในรูปของ Promise ทำให้ LoginForm ซึ่งเรียกใช้ฟังก์ชัน submit สามารถประมวลผลในกรณีผิดพลาดต่อไปได้ ทำให้แก้ไขโค้ดในส่วนฟังก์ชัน onSubmit ใน LoginForm.js ได้เป็น

|     |                                                |
|-----|------------------------------------------------|
| 1.  | onSubmit = () => {                             |
| 2.  | const errors = this.validate(this.state.data); |
| 3.  | this.setState({ errors });                     |
| 4.  | if (Object.keys(errors).length === 0) {        |
| 5.  |                                                |
| 6.  | this.props.submit(this.state.data)             |
| 7.  | .catch((err) => {                              |
| 8.  | this.setState({ errors: err.errors });         |
| 9.  | });                                            |
| 10. | }                                              |
| 11. | }                                              |

ปรับปรุงโค้ดของ LoginForm ในส่วนของ Form แสดงการ Loading และแสดง Error หากได้รับความผิดพลาดตอบกลับมาจากเซิร์ฟเวอร์

```
<Form onSubmit={this.onSubmit} loading={loading}>
 { errors.global && (
 <Message negative>
 <Message.Header>Cannot verify your login</Message.Header>
 <p>{errors.global}</p>
 </Message>) }
 <Form.Field error={!errors.email}>
// ละส่วนที่เหลือ
```

ในตอนนี้ หากทดลองให้ทำงาน เมื่อผู้ใช้กรอกข้อมูลผ่านการตรวจสอบจาก validate ได้ (ผ่านเงื่อนไขในบรรทัดที่ 4) จะเรียกฟังก์ชัน submit ที่ได้จาก props ทำให้ไปเรียกการทำงานของฟังก์ชัน submit ใน LoginPage และได้รับผลตอบรับเป็น 404 error (ตามการทำงานของส่วน /api/signin ของเซิร์ฟเวอร์)

Home

Login Page

Email

wrong@email.com

Password

••••

Submit

ชื่อผู้ใช้ไม่ถูกต้อง แสดงความผิดพลาด

Home

Login Page

Cannot verify your login  
Invalid credential

Email

wrong@email.com

Password

••••

## แก้ไข Server เพื่อให้ตรวจสอบ email และ password

แก้ไขการทำงานของเซิร์ฟเวอร์ให้ตรวจสอบการยืนยันตน หากผิดพลาดส่ง HTTP Response status 404 ไม่เช่นนั้นคืน status 200 พร้อมข้อมูล email ของ user (ซึ่งถ้าต้องการ จะสามารถเพิ่มข้อมูลอื่นๆ ได้ในอนาคต)

โดยจะสมมติผู้ใช้ที่มี email เป็น [yao@cs.com](#) และมี password เป็น mySecret และเมื่อเข้ารหัสโดยใช้ bcrypt จะมีรหัสที่ใช้เก็บใน password ดังโค้ดในบรรทัดที่ 4

1.	<code>var encrypt = require('./encryptModule');</code>
2.	<code>var database = [{</code>
3.	<code>  email: 'yao@cs.com',</code>
4.	<code>  password: '\$2a\$10\$LiliyJwXf93YanqqKpHZQ.jkjdPwBsvlsIUhsa0bcizT5ImwDex/e',</code>
5.	<code>  plaintext: 'mySecret',</code>
6.	<code>  }];</code>
7.	
8.	<code>var express = require('express');</code>
9.	<code>var path = require('path');</code>
10.	<code>var bodyParser = require('body-parser');</code>
11.	
12.	<code>var app = express();</code>
13.	<code>app.use(bodyParser.json()); // for parsing application/json</code>
14.	<code>app.use(bodyParser.urlencoded({ extended: true })); // for parsing application/x-www-form-urlencoded</code>
15.	
16.	<code>var port = 4000;</code>
17.	
18.	<code>var checkLogin = async function(user, pwd) {</code>
19.	<code>  var dbUser = database.find(({email}) =&gt; email === user);</code>
20.	<code>  if (!dbUser) return false;</code>
21.	<code>  // return encrypt.compareSyncPassword(pwd, dbUser.password);</code>
22.	<code>  return encrypt.comparePassword(pwd, dbUser.password);</code>
23.	<code>}</code>
24.	
25.	<code>app.post('/api/signin', (req, res) =&gt; {</code>
26.	<code>  const {email, password} = req.body;</code>
27.	<code>  if (!email    !password) {</code>
28.	<code>    res.status(404).json({errors: {global: "Invalid credential"}});</code>
29.	<code>  } else {</code>
30.	<code>    checkLogin(email, password)</code>

31.	.then(checked => {
32.	if (checked) {
33.	console.log(email + " logged in");
34.	res.status(200).json({user: {email: email}});
35.	}
36.	else {
37.	res.status(404).json({errors: {global: "Invalid credential"}});
38.	}
39.	});
40.	}});
41.	
42.	app.get("/*", (req, res) => {
43.	res.sendFile(path.join(__dirname, "index.html"));
44.	});
45.	
46.	app.listen(port, function() { console.log(`start http server on \${port}`); });

ส่วนโค้ดการทำ encrypt และ compare รหัส password (encryptModule.js)

1.	var bcrypt = require('bcrypt');
2.	
3.	var encrypt = {
4.	encryptPwd: function(password, callback) {
5.	bcrypt.genSalt(10, function(err, salt) {
6.	if (err) return callback(err);
7.	bcrypt.hash(password, salt, function(err, hash) {
8.	return callback(err, hash);
9.	});
10.	}},
11.	
12.	comparePassword: async function(password, encPwd) {
13.	return await bcrypt.compare(password, encPwd);
14.	},
15.	compareSyncPassword: function(password, encPwd) {
16.	return bcrypt.compareSync(password, encPwd);
17.	},
18.	}
19.	
20.	module.exports = encrypt;

## แก้ไข Client ให้รองรับการยืนยันตนกรณี account ถูกต้อง

เพื่อเลือกแสดงกรณีที่ login ได้ถูกต้องให้ย้ายการทำงานไปยังหน้า SecretPage เราสามารถใช้เงื่อนไขสำหรับการแสดงผล (Conditional Rendering) (ดูเพิ่มเติมที่ <https://reactjs.org/docs/conditional-rendering.html>) เพื่อเลือกแสดงผลตามเงื่อนไข

เพิ่มเติม Route เพื่อให้กรองสิทธิ์ก่อนเข้าใช้ และสร้างหน้าเพจตัวอย่างที่จะใช้ได้เมื่อมีสิทธิ์เท่านั้น โดย

- สร้าง PrivateRoute เพื่อใช้เป็นตัวครอบ (wrap) หน้าเพจที่ต้องผ่านการตรวจสอบสิทธิ์จึงจะมีสิทธิ์เข้าใช้ได้ กรณีที่ยังไม่มีสิทธิ์ให้ทำ Redirect ไปยัง path ที่ต้องการ (เช่น LoginPage)

ฟังก์ชัน PrivateRoute.js

```
import React from 'react';
import { Route, Redirect } from 'react-router-dom';
```

```
import PropTypes from 'prop-types';

const PrivateRoute = ({ component: Component, redirectTo, ...rest }) => {
 const { isAuthenticated } = rest;
 return (
 <Route
 {...rest}
 render={ (routeProps) => { return isAuthenticated ?
 (<Component {...routeProps} />) :
 (<Redirect to={{ pathname: redirectTo, state: { from: routeProps.location } }}
 />);
 }
 />
);
};

PrivateRoute.propTypes = {
 isAuthenticated: PropTypes.bool.isRequired,
};

export default PrivateRoute;
```

- สร้างหน้าเพจ SecretPage เพื่อใช้เป็นตัวอย่างหน้าที่ต้องมีสิทธิ์ก่อนจึงเข้าใช้งานได้

หน้าเพจ SecretPage.js

```
import React from 'react';

const SecretPage = () => (
 <div>
 <h1>SecretPage</h1>
 This is a special page; can be used after login.
 </div>
);

export default SecretPage;
```

### แก้ไขโค้ดของ App.js

- เปลี่ยนจากฟังก์ชันคอมโพเนนต์เป็นคลาส เพื่อรองรับการทำงานกับ state
- กำหนด state สำหรับ user และการมีสิทธิ์ (isAuthorized) พร้อมกำหนดฟังก์ชันสำหรับเปลี่ยนค่า setAuth ใน App ซึ่งเป็น parent component
- เปลี่ยน Route ของ LoginPage ให้ส่ง props ฟังก์ชันและค่าการมีสิทธิ์จาก parent ไปได้
- กำหนด PrivateRoute สำหรับหน้า SecretPage เพื่อเป็นตัวอย่างการทำงาน

โค้ดใหม่ของ App.js ที่ได้เป็นดังนี้

```
import React from 'react';
import { Route } from 'react-router-dom';

import HomePage from './components/pages/HomePage';
import LoginPage from './components/pages/LoginPage';
import SecretPage from './components/pages/SecretPage';
import NavMenu from './components/pages/NavMenu';
```

```

import PrivateRoute from './PrivateRoute';

class App extends React.Component {
 state = { isAuthenticated: false, user: {} };
 setAuth = (data) => {
 this.setState({ isAuthenticated: true, user: data.user });
 };

 render() {
 const state = this.state;
 return (
 <div className="ui container">
 <header>
 <NavMenu />
 </header>
 <section>
 <Route path="/" exact component={HomePage} />
 <Route path="/login" exact render={ () =>
 <LoginPage setAuth={this.setAuth} isAuthenticated={state.isAuthenticated}
 />
 <PrivateRoute path="/secretPage" component={SecretPage} user={state.user}
 redirectTo="/login" isAuthenticated={state.isAuthenticated}
 />
 </section>
 </div>
);
 }
}

export default App;

```

### แก้ไขโค้ดของ LoginPage.js

เพื่อให้รองรับเงื่อนไขการเลือกแสดง Component เราจะปรับปรุงโค้ดของ LoginPage ใหม่ โดย

- สร้างตัวแปร สำหรับกำหนดเงื่อนไข authResponse โดยให้มีค่าเริ่มต้นเป็นค่าที่ได้รับจาก props.isAuthenticated
- เพิ่มโค้ดส่วนตรวจสอบกรณีรหัสผู้ใช้ถูกต้อง
- สร้างเงื่อนไขการแสดงผล ถ้าผ่านตรวจสอบสิทธิ์ย้ายไปหน้า SecretPage ถ้าไม่เช่นนั้นไปยัง LoginForm

แก้ไขหน้า LoginPage.js ให้รองรับเงื่อนไข

```

import React from 'react';
import PropTypes from 'prop-types';
import { Redirect } from 'react-router-dom';

import LoginForm from '../forms/LoginForm';

class LoginPage extends React.Component {
 constructor(props) {
 super(props);
 this.state = { authResponse: props.isAuthenticated };
 }

```

```

// verify user with the server
submit = (data) => {
 return fetch('/api/signin', {
 method: 'POST',
 body: JSON.stringify(data), // can use either `string` or {object}!
 headers: new Headers({ 'Content-Type': 'application/json' }),
 })
 .then((resp) => {
 const json = resp.json();
 if (resp.status >= 200 && resp.status < 300) {
 json.then((user) => {
 this.props.setAuth(user);
 this.setState({ authResponse: true });
 });
 } else {
 return json.then((err) => { throw err; });
 }
 });
}

render() {
 const { authResponse } = this.state;
 if (!authResponse) {
 return (<LoginForm submit={this.submit} />);
 }
 // else
 return (<Redirect to={{ pathname: '/secretPage', }} />);
}
}

LoginPage.propTypes = {
 setAuth: PropTypes.func.isRequired,
 isAuthenticated: PropTypes.bool.isRequired,
};

export default LoginPage;

```

เมื่อทดสอบ โดยการ login ให้ถูกต้องจะย้ายการทำงานไปยังหน้า SecretPage และหากกดเมนู Home และกด login ในหน้า Home หรือกด Login ที่เมนูจะแสดงหน้า SecretPage อย่างไรก็ตามหากทำ refresh หน้าจะพบว่า state ของการมีสิทธิ์นั้นจะถูก reset ต้อง login ใหม่

<https://css-tricks.com/react-router-4/>

## การทำ Logout และ Register ผู้ใช้ใหม่

ปรับปรุง NavMenu ให้เปลี่ยนจาก Login เป็น Dropdown เมื่อผ่านการยืนยันตัวตนแล้ว

```
import React from 'react';
import { Menu, Dropdown, Image, Icon } from 'semantic-ui-react';
import { NavLink, Link } from 'react-router-dom';
import PropTypes from 'prop-types';
import avatar from '../images/avatar.jpg';

const NavMenu = (props) => {
 const trigger = (

 {props.user && <Image avatar src={avatar} /> Hello,
{props.user.name}}
 {!props.user && <Icon name="user" /> Hello, User}

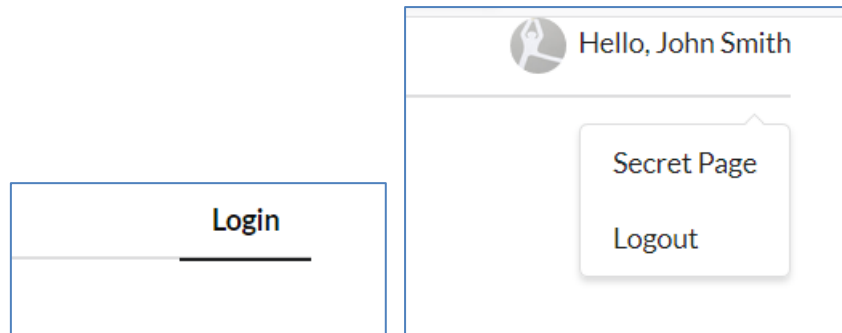
);

 return (
 <Menu pointing secondary>
 <Menu.Item as={NavLink} to="/" activeClassName="active">Home</Menu.Item>
 <Menu.Menu position="right">
 {!props.isAuthenticated &&
 <Menu.Item as={NavLink} to="/login"
activeClassName="active">Login</Menu.Item>}
 {props.isAuthenticated &&
 <Dropdown trigger={trigger} pointing="top right" icon={null}>
 <Dropdown.Menu>
 <Dropdown.Item as={Link} to="/secretPage">Secret Page</Dropdown.Item>
 <Dropdown.Item onClick={() =>
props.handler.resetToken()}>Logout</Dropdown.Item>
 </Dropdown.Menu>
 </Dropdown>
 }
 </Menu.Menu>
 </Menu>
);
};

NavMenu.propTypes = {
 handler: PropTypes.shape({
 resetToken: PropTypes.func,
 }).isRequired,
 isAuthenticated: PropTypes.bool.isRequired,
 user: PropTypes.shape({
 name: PropTypes.string,
 }),
};

NavMenu.defaultProps = {
 user: {},
};
export default NavMenu;
```

ตัวอย่างภาพแสดงให้ Login และแสดงผู้ใช้เมื่อผ่านการยืนยันตัวตนแล้ว



ปรับหน้าเพจ App.js ให้มีฟังก์ชัน resetToken และส่ง props นี้มาให้ NavMenu พร้อมนำ setAuth เข้ามารวมไว้ใน parentHandler แทน นอกจากนี้เพิ่ม checkToken เพื่อให้ตรวจสอบว่ามี token อยู่หรือไม่ ถ้ามีให้กำหนดให้ผ่านการยืนยันตัวตนด้วย

```
import React from 'react';
import { Route } from 'react-router-dom';
import HomePage from './components/pages/HomePage';
import LoginPage from './components/pages/LoginPage';
import SecretPage from './components/pages/SecretPage';
import NavMenu from './components/pages/NavMenu';
import { PrivateRoute } from './PrivateRoute';

class App extends React.Component {
 state = { isAuthenticated: false, user: {} };
 componentDidMount() {
 this.parentHandler.checkToken();
 }

 parentHandler = {
 setAuth: (data) => {
 const { isAuthenticated } = this.state;
 if (!isAuthenticated) {
 this.setState({ isAuthenticated: true, user: data.user });
 sessionStorage.setItem('token', data.token);
 }
 },
 resetToken: () => {
 const { isAuthenticated } = this.state;
 if (isAuthenticated) {
 this.setState({ isAuthenticated: false, user: {} });
 sessionStorage.removeItem('token');
 }
 },
 checkToken: () => {
 const token = sessionStorage.getItem('token');
 if (token !== null) {
 this.setState({ isAuthenticated: true, user: {} });
 }
 },
 },
}

render() {
 const { state } = this;
 return (
 <div className="ui container">
 <header>
 <NavMenu
```



```

 handler={this.parentHandler}
 isAuthenticated={state.isAuthenticated}
 user={state.user.name}
 />
 </header>
 <section>
 <Route path="/" exact component={HomePage} />
 <Route
 path="/login"
 exact
 render={() =>
 (<LoginPage
 handler={this.parentHandler}
 isAuthenticated={state.isAuthenticated}
 />)
 }
 />
 <PrivateRoute
 path="/secretPage"
 component={SecretPage}
 user={state.user}
 redirectTo="/login"
 isAuthenticated={state.isAuthenticated}
 />
 </section>
 </div>
);
}
}
export default App;

```

เมื่อทดสอบจะพบว่า หลังจากผู้ใช้ login ผ่านแล้ว ถ้ากด refresh หน้าเพจจะยังคงการ login ของผู้ใช้ได้

ปรับปรุงการทำงานในเรื่องการทำ Fetch (การร้องขอไปยัง server)

สร้างไฟล์ api.js (ในที่นี้ไว้โฟลเดอร์เดียวกับ App.js) ด้าน react เพื่อช่วยในการร้องขอไปยัง Node Server

```
import _fetch from './utils/timeoutFetch';

export default {
 user: {
 signin: credentials =>
 _fetch('/api/signin', {
 method: 'POST',
 body: JSON.stringify(credentials),
 headers: new Headers({ 'Content-Type': 'application/json' }),
 }),
 register: user =>
 _fetch('/api/user', {
 method: 'POST',
 body: JSON.stringify(user),
 headers: new Headers({ 'Content-Type': 'application/json' }),
 }),
 validateToken: token =>
 _fetch('/api/validateToken', {
 method: 'POST',
 headers: new Headers({
 'Authorization': token,
 'Content-Type': 'application/x-www-form-urlencoded',
 }),
 body: token,
 }),
 },
 products: {
 fetchAll: () => _fetch("/api/products").then(res => res.json()),
 },
};
```

โดยกำหนดฟังก์ชัน timeoutFetch (โค้ดใน api.js อ้างถึงด้วยชื่อ \_fetch) ให้รองรับกับ timeout กรณีที่การร้องขอใช้เวลานานเกินไป ฟังก์ชันนี้ใช้การสร้าง Promise ซึ่งเป็น async โดยมีพารามิเตอร์เป็นฟังก์ชัน callback ซึ่งส่งพารามิเตอร์ให้ 2 ตัวคือ resolve และ reject โดยวัตถุ resolve ใช้เพื่อการทำงานกรณีสำเร็จ และ reject ใช้ในกรณีไม่สำเร็จ

นอกจากนี้ยังใช้ timeout เพื่อให้ยกเลิกการทำงานหากรอนานเกินไป

```
// credit code modified from: https://davidwalsh.name/fetch-timeout

export default function timeoutFetch(input, init = {}, fetch_timeout = 8000) {
 let didTimeout = false;

 return new Promise((resolve, reject) => {
 const timeout = setTimeout(() => {
 didTimeout = true;
 const err = { errors: { global: 'Request timed out' } };
 reject(err);
 }, fetch_timeout);

 fetch(input, init)
 .then((response) => {
 // Clear the timeout as cleanup
 clearTimeout(timeout);
 });
 });
}
```

```

 if (!didTimeout) {
 console.log('fetch good! ', response);
 if (response.status >= 500) {
 const err = { errors: { global: 'Comeback later' } };
 reject(err);
 } else resolve(response);
 }
 })
 .catch((err) => {
 console.log('fetch failed! ', err);

 // Rejection already happened with setTimeout
 if (didTimeout) return;
 // Reject with error
 reject(err);
 });
});
}

```

แก้ไขให้ LoginPage.js ใช้งาน api แทน

```

import React from 'react';
import PropTypes from 'prop-types';
import { Redirect } from 'react-router-dom';

import LoginForm from '../forms/LoginForm';
import api from '../api';

class LoginPage extends React.Component {
 constructor(props) {
 super(props);
 this.state = { authResponse: props.isAuthenticated };
 }
 // verify user with the server
 submit = (data) => {
 return api.user.signin(data)
 .then((resp) => {
 const json = resp.json();
 if (resp.status >= 200 && resp.status < 300) {
 json.then((user) => {
 this.props.handler.setAuth(user);
 this.setState({ authResponse: true });
 });
 } else {
 return json.then((err) => { throw err; });
 }
 });
 }

 render() {
 const { authResponse } = this.state;
 if (!authResponse) {
 return (<LoginForm submit={this.submit} />);
 }
 return (<Redirect to={{ pathname: '/secretPage' }} />);
 }
}

LoginPage.propTypes = {
 handler: PropTypes.shape({

```

```

 setAuth: PropTypes.func.isRequired,
 resetToken: PropTypes.func,
 checkToken: PropTypes.func.isRequired,
 }).isRequired,
 isAuthenticated: PropTypes.bool.isRequired,
};

```

```
export default LoginPage;
```

เมื่อทดสอบจะพบว่าสามารถ login และเมื่อผ่านการยืนยันตัวตนแล้ว ทำ refresh หน้าจอก็ยังมีสถานะผ่านการยืนยันตัวตนได้อยู่

สร้างหน้าลงทะเบียน Register

ไฟล์ RegisterPage.js

```

import React from 'react';
import PropTypes from 'prop-types';
import { Redirect } from 'react-router-dom';
import RegisterForm from '../forms/RegisterForm';
import ConfirmedLogoutModal from '../forms/ConfirmedLogoutModal';
import api from '../api';

class RegisterPage extends React.Component {
 state = {
 redirectToReferrer: false,
 isAuthenticated: this.props.isAuthenticated,
 user: this.props.user,
 };
 setRedirect = () => {
 this.setState({ redirectToReferrer: true });
 }
 setLogout = () => {
 this.setState({ isAuthenticated: false });
 this.props.handler.resetToken();
 }
 submit = (data) => {
 return api.user.register(data)
 .then((resp) => {
 const json = resp.json();
 if (resp.status >= 200 && resp.status < 300) {
 json.then((user) => {
 this.setState({ redirectToReferrer: true });
 });
 } else {
 return json.then((err) => { throw err; });
 }
 });
 }

 render() {
 const { user, redirectToReferrer, isAuthenticated } = this.state;
 if (isAuthenticated && !redirectToReferrer) {
 return (<ConfirmedLogoutModal
 confirm={this.setLogout}
 cancel={this.setRedirect}
 user={user}

```

```

 />);
 }
 if (redirectToReferrer) {
 return <Redirect to="/" />;
 }
 return (
 <div>
 <RegisterForm submit={this.submit} />
 </div>
);
}
}

RegisterPage.propTypes = {
 handler: PropTypes.shape({
 resetToken: PropTypes.func.isRequired,
 }).isRequired,
 isAuthenticated: PropTypes.bool.isRequired,
 user: PropTypes.shape({
 name: PropTypes.string,
 }),
};

RegisterPage.defaultProps = {
 user: {},
};

export default RegisterPage;

```

❗ไฟล์ RegisterForms.js

```

import React from 'react';
import PropTypes from 'prop-types';
import { Form, Button, Message } from 'semantic-ui-react';
import Validator from 'validator';
import InlineError from './InlineError';

class RegisterForm extends React.Component {
 state = {
 data: {
 email: '',
 password: '',
 passwordConfirmed: '',
 name: '',
 color: '',
 },
 loading: false,
 errors: {},
 };

 onChange = (e) => {
 this.setState({ data: { ...this.state.data, [e.target.name]: e.target.value } });
 }

 onSubmit = (e) => {
 e.preventDefault();
 const err = this.validate(this.state.data);
 this.setState({ errors: err });
 if (Object.keys(err).length === 0) {
 this.setState({ loading: true });
 }
 }

```

```

 this.props.submit(this.state.data)
 .then(null, (respErr) => {
 if (respErr.errors) {
 this.setState({ errors: respErr.errors, loading: false });
 }
 });
 }
}

validate = (data) => {
 const errors = {};

 if (!Validator.isEmail(data.email)) errors.email = 'Invalid email';
 if (!data.password) errors.password = 'Cannot be blank';
 if (data.password !== data.passwordConfirmed) errors.passwordConfirmed = 'password
must be confirmed';
 if (!data.name) errors.name = 'Cannot be blank';

 return errors;
};

render() {
 const { data, errors, loading } = this.state;
 return (
 <div>
 <h1>Register New Member</h1>
 <Form onSubmit={this.onSubmit} loading={loading}>
 { errors.global && (
 <Message negative>
 <Message.Header>Cannot verify your
login</Message.Header><p>{errors.global}</p>
 </Message>) }
 <Form.Field error={!!errors.email}>
 <label htmlFor="email">Email
 <input
 type="email"
 id="email"
 name="email"
 placeholder="your@email.com"
 value={data.email}
 onChange={this.onChange}
 />
 {errors.email && <InlineError text={errors.email} />}
 </label>
 </Form.Field>
 <Form.Field error={!!errors.name}>
 <label htmlFor="name">Name
 <input
 type="text"
 id="name"
 name="name"
 placeholder="Your name"
 value={data.name}
 onChange={this.onChange}
 />
 {errors.name && <InlineError text={errors.name} />}
 </label>
 </Form.Field>
 <Form.Field error={!!errors.password}>

```

```

 <label htmlFor="password">Password
 <input
 type="password"
 id="password"
 name="password"
 placeholder="password"
 value={data.password}
 onChange={this.onChange}
 />
 {errors.password && <InlineError text={errors.password} />}
 </label>
 </Form.Field>
 <Form.Field error={!errors.passwordConfirmed}>
 <label htmlFor="passwordConfirmed">Confirmed Password
 <input
 type="password"
 id="passwordConfirmed"
 name="passwordConfirmed"
 placeholder="passwordConfirmed"
 value={data.passwordConfirmed}
 onChange={this.onChange}
 />
 {errors.passwordConfirmed && <InlineError text={errors.passwordConfirmed}
 />}
 </label>
 </Form.Field>
 <Form.Field>
 <label htmlFor="color">Color
 <input
 type="text"
 id="color"
 name="color"
 placeholder="Your favorite color"
 value={data.color}
 onChange={this.onChange}
 />
 {errors.color && <InlineError text={errors.color} />}
 </label>
 </Form.Field>
 <Button primary>Submit</Button>
</Form>
</div>
);
}
}

RegisterForm.propTypes = {
 submit: PropTypes.func.isRequired,
};

export default RegisterForm;

```

แก้ไข App.js ให้รองรับกับการทำงานหน้า component ต่างๆ

```

import React from 'react';
import { Route } from 'react-router-dom';
import HomePage from './components/pages/HomePage';
import LoginPage from './components/pages/LoginPage';

```

```

import RegisterPage from './components/pages/RegisterPage';
import SecretPage from './components/pages/SecretPage';
import GuestPage from './components/pages/GuestPage';
import NavMenu from './components/pages/NavMenu';
import { PrivateRoute, GuestRoute } from './PrivateRoute';
import api from './api';

class App extends React.Component {
 state = { isAuthenticated: false, user: {} };

 componentDidMount() {
 const token = sessionStorage.getItem('token');
 if (token !== null && !this.state.isAuthenticated) {
 this.fetchToken = this.parentHandler.checkToken(token);
 }
 }

 parentHandler = {
 setAuth: (data) => {
 const { isAuthenticated } = this.state;
 if (!isAuthenticated) {
 this.setState({ isAuthenticated: true, user: data.user });
 sessionStorage.setItem('token', data.token);
 }
 },
 resetToken: () => {
 const { isAuthenticated } = this.state;
 if (isAuthenticated) {
 this.setState({ isAuthenticated: false, user: {} });
 sessionStorage.removeItem('token');
 }
 },
 register: () => {
 const { isAuthenticated } = this.state;
 if (isAuthenticated) {
 this.setState({ isAuthenticated: false, user: {} });
 sessionStorage.removeItem('token');
 }
 },
 checkToken: async (token) => {
 if (token !== null) {
 if (Object.keys(this.state.user).length === 0) {
 return api.user.validateToken(token)
 .then((resp) => {
 const json = resp.json();
 if (resp.status >= 200 && resp.status < 300) {
 json.then(usr =>
 this.setState({ isAuthenticated: true, user: usr }));
 } else this.parentHandler.resetToken();
 })
 .catch(() => {
 this.parentHandler.resetToken();
 });
 }
 }
 return null;
 },
 },
}

```



```

render() {
 const { state } = this;
 return (
 <div className="ui container">
 <header>
 <NavMenu
 handler={this.parentHandler}
 isAuthenticated={state.isAuthenticated}
 user={state.user}
 />
 </header>
 <section>
 <Route path="/" exact component={HomePage} />
 <Route
 path="/login"
 exact
 render={() =>
 (<LoginPage
 handler={this.parentHandler}
 isAuthenticated={state.isAuthenticated}
 />)
 }
 />
 <Route path="/register" exact render={() =>
 (<RegisterPage
 handler={this.parentHandler}
 isAuthenticated={state.isAuthenticated}
 user={state.user}
 />)
 }
 />
 <PrivateRoute
 path="/secretPage"
 component={SecretPage}
 user={state.user}
 redirectTo="/login"
 isAuthenticated={state.isAuthenticated}
 />
 </section>
 </div>
);
}
}

export default App;

```

ปรับปรุงให้ตรวจสอบ Token ว่าเป็น token ที่ถูกต้องหรือไม่

ข้าง client ตรวจสอบว่ามี token โดยไม่ได้ผ่านการยืนยันตัวตน (isAuthenticated)

```

import React from 'react';
import { Route } from 'react-router-dom';
import HomePage from './components/pages/HomePage';
import LoginPage from './components/pages/LoginPage';
import RegisterPage from './components/pages/RegisterPage';
import SecretPage from './components/pages/SecretPage';

```

```

import GuestPage from './components/pages/GuestPage';
import NavMenu from './components/pages/NavMenu';
import { PrivateRoute, GuestRoute } from './PrivateRoute';
import api from './api';

class App extends React.Component {
 state = { isAuthenticated: false, user: {} };

 componentDidMount() {
 const token = sessionStorage.getItem('token');
 if (token !== null && !this.state.isAuthenticated) {
 this.fetchToken = this.parentHandler.checkToken(token);
 }
 }

 parentHandler = {
 setAuth: (data) => {
 const { isAuthenticated } = this.state;
 if (!isAuthenticated) {
 this.setState({ isAuthenticated: true, user: data.user });
 sessionStorage.setItem('token', data.token);
 }
 },
 resetToken: () => {
 const { isAuthenticated } = this.state;
 if (isAuthenticated) {
 this.setState({ isAuthenticated: false, user: {} });
 sessionStorage.removeItem('token');
 }
 },
 register: () => {
 const { isAuthenticated } = this.state;
 if (isAuthenticated) {
 this.setState({ isAuthenticated: false, user: {} });
 sessionStorage.removeItem('token');
 }
 },
 checkToken: (token) => {
 if (token !== null) {
 if (Object.keys(this.state.user).length === 0) {
 return api.user.validateToken(token)
 .then((resp) => {
 const json = resp.json();
 if (resp.status >= 200 && resp.status < 300) {
 json.then(usr =>
 this.setState({ isAuthenticated: true, user: usr }));
 } else this.parentHandler.resetToken();
 })
 .catch(() => {
 this.parentHandler.resetToken();
 });
 }
 }
 return null;
 },
 },
 render() { /* ล่ะจ้าจึ๊จึ๊ */ }
}

```

ปรับปรุงด้าน server เพื่อให้รองรับการขึ้นทะเบียน

ไฟล์ jwt-passport

```
// jwt-passport.js
const passport = require("passport");
const JwtStrategy = require('passport-jwt').Strategy;
const ExtractJwt = require('passport-jwt').ExtractJwt;
const User = require('../models/User');
const config = require('../config/jwtConfig');

module.exports = function () {
 let params = {
 jwtFromRequest: ExtractJwt.fromAuthHeaderWithScheme("JWT"),
 secretOrKey: config.jwtSecret,
 }
 var strategy = new JwtStrategy(params, (jwt_payload, done) => {
 User.findOne({email: jwt_payload.email}).select('id email name')
 .then(user => {
 if(user) {
 return done(null, user);
 }
 return done(null, false);
 })
 .catch(err => {
 return done(err, false);
 })
 });
 passport.use(strategy);
 return {
 initialize: function() {
 return passport.initialize();
 },
 authenticate: function() {
 return passport.authenticate("jwt", config.jwtSession);
 },
 };
};
}
```

ไฟล์ UserController.js ให้เพิ่มเมทอด validate token โดยต้องตัดสายอักขระ 4 ตัวแรก (JWT) ออกก่อนทำ verify

```
exports.validateToken = (req, res) => {
 const token = req.headers['authorization'].substring(4);
 try {
 var decode = jwt.verify(token, config.jwtSecret);
 res.json(decode);
 }
 catch (err) {
 res.status(401).json({err});
 }
};
```