**Multi-Channel Authentication by Using React and Node**

**in a 3-Layered Architecture**

In this tutorial, we will learn how to create a multi-channel authentication service by using React as front end and Node JS backend. We will also how to achieve he multi-channel authentication by using dependency injection for our various authentication services.

Create Node JS Express App

1.1 Environment setup

1. Setup Node JS

If you have not installed node yet, please go to <https://nodejs.org/en/> to download it. Then right click the downloaded file such as “node-v12.16.3-x64.msi” and select install. Accept the license terms, follow the instruction and complete the installation. After the installation completes, open a cmd line console and type in “node -v” then “npm -v”. If version displays, this means installation is successful.

1. Setup Mongo DB

If Mongo DB is not installed yet, please go to <https://www.mongodb.com/download-center> to download it. Then extract the downloaded compressed file into the location you want the Mongo DB to be installed and add the directory “…/bin” into system “Path” variable.

1. Setup Testing Environment

In this tutorial, we will use VS Code as our code editor. You can download it at <https://code.visualstudio.com/download>. Select whichever version is correct for your device and follow the instructions to complete the installation.

1.2 Create Application Folder

Open a cmd console, change directory to drive C, type in

mkdir multi-channelAuthentication

to create the main project folder in C drive named "multi-channelAuthentication".

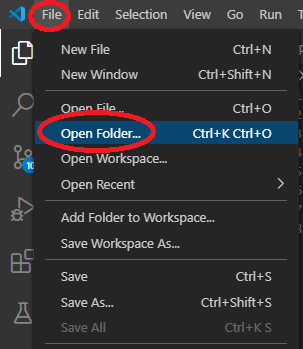
Change directory to the new directory just made, run the following commands:

npx express-generator api

cd api

npm install

This will create a new express app template in the folder api as well as install the default dependencies. Now, let’s open VS Code and open the folder:



1.3 Install all Required Backend Dependencies

We will start by installing all the dependencies that we will be using in our Node JS backend. These dependencies will all be used throughout the course of this tutorial. In the Command console, change directory to api. Type the following command:

npm install bcrypt-nodejs body-parser cors express-session mongoose passport passport-facebook passport-google-oauth passport-local

Among them,

bcrypt-nodejs: Will be used to hash out user passwords before storing to database

body-parser: Will be used to parse incoming request bodies before handlers

cors: Will be used to allow front end to communicate with backend

express-session: Will be used to store authentication sessions

mongoose: Will be used to help create models for database connection

passport: Will be used to manage all authentication methods

passport-facebook: Will be used for facebook authentication

passport-google-oauth: Will be used for google authentication

passport-local: Will be used for local authentication

1.4 Create Models

Within the folder “api” do the following:

1. Create new folders “models” in the api folder. The “models” will contain all the models. We will be using the mongoose node model to handle out models as well as the bcrypt-nodejs node model to hash out passwords.
2. Now create a new file named “OauthUser.js” in the newly created “models” folder. This file will define the model for our User. Here is the “OauthUser.js” file:

var mongoose = require('mongoose');

var bcrypt = require('bcrypt-nodejs');

var userSchema = mongoose.Schema({

    email: String,

    username: String,

    password: String,

    facebook: {

        id: String,

        token: String,

        name: String

    },

    google: {

        id: String,

        token: String,

        name: String

    },

    auto: Boolean

}, {

    collection: 'OauthUser',

    versionKey: false

});

userSchema.methods.generateHash = function (password) {

    return bcrypt.hashSync(password, bcrypt.genSaltSync(8), null);

};

userSchema.methods.validPassword = function (password) {

    return bcrypt.compareSync(password, this.password);

};

module.exports = mongoose.model('OauthUser', userSchema);

1.5 Authentication Handling with Passport

Within the folder “api” create a new folder called “configs”. This folder will contain all of the configurations, including our database and authentication ones. Within the “configs” folder do the following:

1. Create a new file named “database.js”. This file will have our database key that can be used to connect to the database. Here is the “database.js” file.

module.exports = {

    'url' : // YOUR DATABASE URL

};

Replace the commented text with your database connection string. If you are using your local mongodb database, use the link 'mongodb://localhost:27017/testDatabase.

Here, testDatabase if the name of the database being used.

1. Create a new file named “passport.js”. This file will be the beginning of all our authentication handlers. We will be using dependency injection in order to handle our various authentication methods. This way, we don’t have to explicitly create new routes, controllers and other classes we normally would have to if we want to add another form of authentication. Instead, all we must do it update our handler, and the rest will be taken care of. Here is the “passport.js” file.

var OauthUser = require('../models/OauthUser');

module.exports = function (passport) {

    passport.serializeUser(function (user, done) {

        done(null, user);

    });

    passport.deserializeUser(function (id, done) {

        OauthUser.findById(id, function (err, user) {

            done(err, user);

        });

    });

    require('./localConfig')(passport);

    require('./signupConfig')(passport);

    require('./autoConfig')(passport);

    require('./facebookConfig')(passport);

    require('./googleConfig')(passport);

};

The methods serializeUser and deserializeUser are methods that help with storing the authenticated user in the session when they are authenticated. Usually, we would put all our authentication handlers in the passport.js file. However, we will be separating all our authentication handlers. Within each of these individual handlers, a different strategy will be used to handle authentication.

1. Create a new file named “localConfig.js”. This file will handle our local authentication requests. Here is the “localConfig.js” file.

var LocalStrategy = require('passport-local').Strategy;

var OauthUser = require('../models/OauthUser');

module.exports = (passport) => {

    passport.use('local', new LocalStrategy({

        usernameField: 'email',

        passwordField: 'password',

        passReqToCallback: true

    },

        function (req, email, password, done) {

            if (email)

                email = email.toLowerCase();

            process.nextTick(function () {

                OauthUser.findOne({ 'email': email }, function (err, user) {

                    if (err)

                        return done(err);

                    if (!user)

                        return done(null, false);

                    if (!user.validPassword(password)) {

                        return done(null, false);

                    } else {

                        return done(null, user);

                    }

                });

            });

        }));

}

1. Create a new file named “signupConfig.js”. This file will handle our signup authentication requests. Here is the “signupConfig.js” file.

var LocalStrategy = require('passport-local').Strategy;

var OauthUser = require('../models/OauthUser');

module.exports = (passport) => {

    passport.use('signup', new LocalStrategy({

        usernameField: 'email',

        passwordField: 'password',

        passReqToCallback: true

    },

        function (req, email, password, done) {

            if (email)

                email = email.toLowerCase();

            process.nextTick(function () {

                OauthUser.findOne({ 'email': email }, function (err, user) {

                    if (err)

                        return done(err);

                    if (user) {

                        return done(null, false);

                    } else {

                        var newUser = new OauthUser();

                        newUser.email = email;

                        newUser.username = req.query.username

                        newUser.password = newUser.generateHash(password);

                        newUser.auto = req.query.auto

                        newUser.save(function (err) {

                            if (err) {

                                return done(err);

                            } else {

                                return done(null, newUser);

                            }

                        });

                    }

                });

            });

        }));

}

1. Create a new file named “autoConfig.js”. This file is will handle whether we want the user to be automatically logged in. The way this will work is that we will take out list of users and determine which of them have the auto login setting set to true. If of those users, one of them has the same username as the username of the current windows users, the user will automatically be authenticated without having to enter credentials Here is the “autoConfig.js” fine.

var CustomStrategy = require('passport-custom').Strategy

var os = require('os')

var OauthUser = require('../models/OauthUser');

module.exports = (passport) => {

    passport.use(

        'auto',

        new CustomStrategy(function (req, done) {

            process.nextTick(function () {

                const uname = os.userInfo().username;

                OauthUser.find({ auto: true }, function (err, user) {

                    user.forEach(obj => {

                        if (obj.username === uname) {

                            return done(null, obj);

                        }

                        return done(null, false, { errors: { user: 'is invalid' } });

                    });

                });

            });

        }),

    );

};

1. Create a new file named “facebookConfig.js”. This file will handle our facebook authentication requests. Here is the “facebookConfig.js” file.

var FacebookStrategy = require('passport-facebook').Strategy;

var OauthUser = require('../models/OauthUser');

var configAuth = require('./auth');

module.exports = (passport) => {

    var fbStrategy = configAuth.facebook;

    fbStrategy.passReqToCallback = true;

    passport.use(new FacebookStrategy(fbStrategy,

        function (req, token, refreshToken, profile, done) {

            process.nextTick(function () {

             OauthUser.findOne({ 'email': profile.emails[0].value.toLowerCase() }, function (err, user) {

                    if (err)

                        return done(err);

                    if (user) {

                        if (!user.facebook.token) {

                            user.facebook.id = profile.id;

                            user.facebook.token = token;

                            user.save(function (err) {

                                if (err) {

                                    return done(err);

                                }

                                else {

                                    return done(null, user);

                                }

                            });

                        } else {

                            return done(null, user);

                        }

                    } else {

                        done(null);

                    }

                });

            });

        }));

}

1. Create a new file named “googleConfig.js”. This file will handle our google authentication requests. Here is the “googleConfig.js” file

var GoogleStrategy = require('passport-google-oauth').OAuth2Strategy;

var OauthUser = require('../models/OauthUser');

var configAuth = require('./auth');

module.exports = (passport) => {

    passport.use(new GoogleStrategy({

        clientID: configAuth.google.clientID,

        clientSecret: configAuth.google.clientSecret,

        callbackURL: configAuth.google.callbackURL,

        passReqToCallback: true

    },

        function (req, token, refreshToken, profile, done) {

            process.nextTick(function () {

                OauthUser.findOne({ 'email': profile.emails[0].value.toLowerCase() }, function (err, user) {

                    if (err) {

                        return done(err);

                    }

                    if (user) {

                        if (!user.google.token) {

                            user.google.id = profile.id;

                            user.google.token = token;

                            user.save(function (err) {

                                if (err) {

                                    return done(err);

                                } else {

                                    return done(null, user);

                                }

                            });

                        } else {

                            return done(null, user);

                        }

                    } else {

                        done(null);

                    }

                });

            });

        }));

}

1. Create a new file named “auth.js”. This file will contain the clientID’s and clientSecret’s for google and facebook authentication. Here is the “auth.js” file

module.exports = {

    'facebook': {

        'clientID': // YOUR clientID HERE,

        'clientSecret': // YOUR clientSecret HERE,

        'callbackURL': 'http://localhost:3000/auth/facebook/callback',

        'profileURL': 'https://graph.facebook.com/v2.5/me?fields=first\_name,last\_name,email',

        'profileFields': ['id', 'email', 'name']

    },

    'google': {

        'clientID': // YOUR clientID HERE,

        'clientSecret': // YOUR clientSecret HERE,

        'callbackURL': 'http://localhost:3000/auth/google/callback'

    }

};

The next step will tell you how to find your clientID and clientSecret

1.6 Setting up Facebook and Google OAuth

Even though we have already configured passport to handle google and Facebook authentication, they won’t work unless we set up authentication from Google’s and Facebook’s end.

1.6.1 Setting up Facebook Oauth

Open a new browsing window and type the following url:

<https://developers.facebook.com/tools/>

Next, do the following:

1. You will need to have a valid Facebook account. If you don’t then create one.
2. After logging in, click MyApps in the top navigation bar.
3. Click Add a New App
4. Select For Everything Else
5. Give your App a name
6. Complete the security check
7. On the side dashboard, go into settings and Basic
   1. The App ID is your ClientID
   2. The App Secret is your clientSecret
8. In App Domains type localhost
9. Scroll to the bottom of the screen and select Add Platform
10. Select Website
11. For the site type <http://localhost:3000/>

With that, you have set up Facebook Oauth

1.6.2 Setting up Google OAuth

Open a new browsing window and type the following url:

<https://console.developers.google.com/>

Next, do the following:

1. You will need to have a valid google account. If you don’t then create one.
2. After login, click Credentials in the left navigation bar.
3. At the top of the screen, click Create Credentials
4. Select Oauth client ID
5. Select Web Application
6. Give your App a name “Multi-ChannelAuthWebApp”
7. Click Add URI under Authorized redirect URIs and type in <http://localhost:3000/auth/google/callback>
8. Click Create
9. Immediately upon creating your App, you will be shown your google clientID and clientSecret

With that, you have set up google Oauth

1.7 Create Routes and Controllers

Within the folder “api” do the following:

1. Create a new file named “authRouter.js” in the routes folder. In this example, we are handling authentication requests as well as a logout request. Here is the “authRouter.js” file:

module.exports = (app, passport) => {

    var pageController = require('../controllers/authController')(passport);

    // LOGOUT

    app.get('/logout', pageController .logout);

    // AUTHENTICATION

    app.get('/auth/:method', pageController .authenticate);

    app.get('/auth/:method/callback', pageController.authenticateCallback)

    return app;

};

Note that we are requiring a file named pageController. We will implement that in the next step. You can also delete the default files in the routes folder as we will not be using them.

1. Create a new folder called “controllers” in the api folder. This folder will contain all the controllers that we will be using. Next, create a new file named “authController.js” in the newly created controllers’ folder. Here is the “authController.js” file:

module.exports = (passport) => {

    const auth = {};

    auth.authenticate = (req, res) => {

        var method = req.params.method

        passport.authenticate(method, { scope: 'email'})(req, res);

    }

    auth.authenticateCallback = (req, res) => {

        var method = req.params.method

        passport.authenticate(method, {

            successRedirect: 'http://localhost:3003/auth/home',

            failureRedirect: 'http://localhost:3003/auth/error'

        })(req, res)

    }

    auth.logout = (req, res) => {

        req.logout();

        res.redirect('http://localhost:3003/');

    }

    return auth;

}

While these redirects won’t do anything right now because they don’t exist, we will be creating them when we create our front end. It is also important to note that depending on the url we give (eg auth/local or auth/facebook) they will still do to the same method. In this sense we are injecting what type of authentication we want to use.

1.7 Updating app.js

Now that we have all the backend configured, all that’s left is to make sure everything is added to app.js so that it can be used. Update the app.js file to the following:

var createError = require('http-errors');

var express = require('express');

var path = require('path');

var cookieParser = require('cookie-parser');

var logger = require('morgan');

var mongoose = require('mongoose');

var passport = require('passport');

var flash = require('connect-flash');

var cors = require("cors");

var morgan = require('morgan');

var cookieParser = require('cookie-parser');

var bodyParser = require('body-parser');

var session = require('express-session');

var app = express();

var config = require('./configs/database');

var url = config.url;

var connect = mongoose.connect(url, {

  useNewUrlParser: true

});

connect.then((db) => {

  console.log("Connected correctly to server");

}, (err) => { console.log(err); });

require('./configs/passport')(passport);

app.use(session({

  secret: 'verysecretkey',

  resave: false,

  saveUninitialized: false,

  cookie: { maxAge: 3600000 },

  user: {}

}));

app.use(morgan('dev'));

app.use(cookieParser());

app.use(bodyParser.json());

app.use(bodyParser.urlencoded({ extended: true }));

app.use(cors());

app.use(passport.initialize());

app.use(passport.session());

app.use(flash());

require('./routes/authRouter.js')(app, passport);

// view engine setup

app.set('views', path.join(\_\_dirname, 'views'));

app.set('view engine', 'jade');

app.use(logger('dev'));

app.use(express.json());

app.use(express.urlencoded({ extended: false }));

app.use(cookieParser());

app.use(express.static(path.join(\_\_dirname, 'public')));

// catch 404 and forward to error handler

app.use(function(req, res, next) {

  next(createError(404));

});

// error handler

app.use(function(err, req, res, next) {

  // set locals, only providing error in development

  res.locals.message = err.message;

  res.locals.error = req.app.get('env') === 'development' ? err : {};

  // render the error page

  res.status(err.status || 500);

  res.render('error');

});

module.exports = app;

Open a new browser window and type in <http://localhost:3000/auth/facebook>. Put in your Facebook credentials, and notice how you are redirected to the page specified in our controller. It’s okay that you get a page not found error as we haven’t created the page yet. In the next stage, we will be creating our frontend.

Create React App

2.1 Install react

Now we will create our frontend form. First go back to the command console, change directory to the project root “multi-channelAuthentication”. Type in the following command:

npx create-react-app client

cd client

npm install

This will create a new React template under the directory “client”.

2.2 Install all Required Frontend Dependencies

We will start by installing all the dependencies that we will be using in our React frontend. These dependencies will all be used throughout the course of this tutorial. In the Command console, change directory to client. Type the following command:

npm install @fortawesome/fontawesome-svg-core @fortawesome/free-brands-svg-icons @fortawesome/react-fontawesome bootstrap bootstrap-social font-awesome react-router-dom reactstrap

@fortawesome/fontawesome-svg-core: Will be used for Icons

@fortawesome/free-brands-svg-icons: Will be used for Icons

@fortawesome/react-fontawesome: Will be used for Icons

bootstrap: Will be used for bootstrap libraries

bootstrap-social: Will be used for bootstrap icons

font-awesome: Will be used for font-awesome icons

react-router-dom: Will be used for page routing

reactstrap: Will be used for form elements and responsive web pages

2.3 Update App.js

1. Change App.js from a functional component to a class component

Now let’s start making our form. In VS Code, navigate to “client/src/”, and open App.js. Change App.js from a functional component to a class component, it should look like this:

import React, { Component } from 'react';

import logo from './logo.svg';

import './App.css';

class App extends Component {

render() {

return (

<div></div>

);

}

}

export default App;

2.4 Create Components

Now let’s create the frontend pages using Components. In the src folder, create a new folder called views. In the views folder, create the following files:

1. authIndex.js

The authIndex.js file will be the base login page. This component will be the page that first renders when the user tries to use our authentication service. In order to deal with the option of autologin, we will use the object lifecycle method componentDidMount. Here is the authIndex.js file.

import React, { Component } from 'react';

import { Button, Form, FormGroup, Label, Input, Col, Alert } from 'reactstrap';

import { NavLink } from 'react-router-dom';

const validEmailRegex = /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/;

const validateForm = (errors) => {

    let valid = true;

    Object.values(errors).forEach(

        (val) => val.length > 0 && (valid = false)

    );

    return valid;

}

class indexComponent extends Component {

    constructor(props) {

        super(props);

        this.state = {

            email: '',

            password: '',

            errors: {

                email: ''

            }

        };

        this.handleInputChange = this.handleInputChange.bind(this);

        this.handleSubmit = this.handleSubmit.bind(this);

    }

    handleInputChange = event => {

        const { name, value } = event.target;

        let errors = this.state.errors;

        switch (name) {

            case 'email':

                errors.email = !validEmailRegex.test(value) ? 'Not a Valid Email' : '';

                break;

        }

        this.setState({ errors, [name]: value })

        this.setState({

            [name]: value,

        });

    };

    handleSubmit = event => {

        event.preventDefault();

        if (validateForm(this.state.errors)) {

            window.location.replace(`http://localhost:3000/auth/local/callback?email=${this.state.email}&password=${this.state.password}`);

        }

    };

    componentDidMount(){

        window.location.replace(`http://localhost:3000/auth/auto/callback`);

    }

    render() {

        return (

            <div className="row row-container">

                <div className="col-12">

                    <h1><span className="fa fa-sign-in"></span>Login</h1>

                </div>

                <div className="col-sm-6 col-sm-offset-3">

                    <Form onSubmit = {this.handleSubmit}>

                        <FormGroup row>

                            <Label htmlFor="email" md={2}>Email</Label>

                            <Col md={7}>

                                <Input type="email" id="email" name="email"

                                    value={this.state.email}

                                    onChange={this.handleInputChange} />

                            </Col>

                            <Col md={3}>

                                {this.state.errors.email.length > 0 &&

                                    <Alert color="danger">

                                        {this.state.errors.email}

                                    </Alert>}

                            </Col>

                        </FormGroup>

                        <FormGroup row>

                            <Label htmlFor="password" md={2}>Password</Label>

                            <Col md={7}>

                                <Input type="password" id="password" name="password"

                                    value={this.state.password}

                                    onChange={this.handleInputChange} />

                            </Col>

                        </FormGroup>

                        <FormGroup row>

                            <Col md={{ size: 10, offset: 2 }}>

                                <Button type = 'submit' color="warning">

                                    Login

                                </Button>

                                <Button href="http://localhost:3000/auth/facebook/" color="primary"><span className="fa fa-facebook"></span>Facebook Login</Button>

                                <Button href="http://localhost:3000/auth/google/" color="danger"><span className="fa fa-google"></span>Google Login</Button>

                            </Col>

                        </FormGroup>

                    </Form>

                    <Col md={{ size: 10, offset: 2 }}>

                        <NavLink to="/auth/signup">Signup</NavLink>

                    </Col>

                </div>

            </div>

        );

    }

}

export default indexComponent;

1. authSignup.js

The authSignup.js file will be the signup page. This component will be the page that renders when the user wants to create an account within our App. Here is the authSignup.js file.

import React, { Component } from 'react';

import { Button, Form, FormGroup, Label, Input, Col, Alert } from 'reactstrap';

import { NavLink } from 'react-router-dom';

const validEmailRegex = /^\w+([\.-]?\w+)\*@\w+([\.-]?\w+)\*(\.\w{2,3})+$/;

const validateForm = (errors) => {

    let valid = true;

    Object.values(errors).forEach(

        (val) => val.length > 0 && (valid = false)

    );

    return valid;

}

class signupComponent extends Component {

    constructor(props) {

        super(props);

        this.state = {

            email: '',

            password: '',

            username: '',

            auto: false,

            errors: {

                email: ''

            }

        };

        this.handleInputChange = this.handleInputChange.bind(this);

        this.handleSubmit = this.handleSubmit.bind(this);

    }

    handleInputChange = event => {

        const target = event.target;

        const value = target.type === 'checkbox' ? target.checked : target.value;

        const name = target.name;

        let errors = this.state.errors;

        switch (name) {

            case 'email':

                errors.email = !validEmailRegex.test(value) ? 'Not a Valid Email' : '';

                break;

        }

        this.setState({ errors, [name]: value })

        this.setState({

            [name]: value,

        });

    };

    handleSubmit = event => {

        event.preventDefault();

        if (validateForm(this.state.errors)) {

            window.location.replace(`http://localhost:3000/auth/signup/callback?email=${this.state.email}&username=${this.state.username}&password=${this.state.password}&auto=${this.state.auto}`);

        }

    };

    render() {

        return (

            <div className="row row-container">

                <div className="col-12">

                    <h1><span className="fa fa-sign-in"></span>Sign Up</h1>

                </div>

                <div className="col-sm-6 col-sm-offset-3">

                    <Form onSubmit={this.handleSubmit}>

                        <FormGroup row>

                            <Label htmlFor="email" md={2}>Email</Label>

                            <Col md={7}>

                                <Input type="email" id="email" name="email"

                                    value={this.state.email}

                                    onChange={this.handleInputChange} />

                            </Col>

                            <Col md={3}>

                                {this.state.errors.email.length > 0 &&

                                    <Alert color="danger">

                                        {this.state.errors.email}

                                    </Alert>}

                            </Col>

                        </FormGroup>

                        <FormGroup row>

                            <Label htmlFor="username" md={2}>Username</Label>

                            <Col md={7}>

                                <Input type="username" id="username" name="username"

                                    value={this.state.username}

                                    onChange={this.handleInputChange} />

                            </Col>

                            <Col md={3}>

                            </Col>

                        </FormGroup>

                        <FormGroup row>

                            <Label htmlFor="password" md={2}>Password</Label>

                            <Col md={7}>

                                <Input type="password" id="password" name="password"

                                    value={this.state.password}

                                    onChange={this.handleInputChange} />

                            </Col>

                        </FormGroup>

                        <FormGroup row>

                            <Label htmlFor="auto" md={2}>Auto?</Label>

                            <Col md={7}>

                                <Input type="checkbox" id="auto" name="auto"

                                    value={this.state.password}

                                    onChange={this.handleInputChange} />

                            </Col>

                        </FormGroup>

                        <FormGroup row>

                            <Col md={{ size: 10, offset: 2 }}>

                                <Button type='submit' color="warning">

                                    Sign Up

                                </Button>

                            </Col>

                        </FormGroup>

                    </Form>

                    <Col md={{ size: 10, offset: 2 }}>

                        <NavLink to="/auth">Login</NavLink>

                    </Col>

                </div>

            </div>

        );

    }

}

export default signupComponent;

1. authHome.js

The authHome.js file will be the home page. This component will be the page that renders if the user is successfully authenticated. Here is the authHome.js file.

import React from 'react';

import { Button } from 'reactstrap';

const home = () => {

    return (

        <div>

            <div>

                <h1>Home</h1>

                <p>Welcome! You have been authenticated</p>

            </div>

            <div>

                <Button href="http://localhost:3000/logout" color="warning">Logout</Button>

            </div>

        </div>

    );

}

export default home;

1. authError.js

The authError.js file will be the error page. This component will be the page that renders if the user is not successfully authenticated. Here is the authError.js file.

import React from 'react';

import { NavLink } from 'react-router-dom';

const home = () => {

    return (

        <div>

            <h1>401: Could not Authenticate</h1>

            <NavLink to="/">Go back Home</NavLink>

        </div>

    );

}

export default home;

2.5 Update App.js

We will be using App.js to handle our frontend routes and connect them to our components. Change the App.js file to look like this.

import React, { Component } from 'react';

import { BrowserRouter, Route, Switch, Redirect } from 'react-router-dom';

import logo from './logo.svg';

import './App.css';

import authError from './views/authError

import authIndex from './views/authIndex

import authSignup from './views/authSignup

import authHome from './views/authHome

class App extends Component {

  render() {

    return (

      <BrowserRouter>

        <div>

          <Switch>

            <Route

              exact

              path="/"

              component={(props) =>

                  <Redirect to="/auth" />

              }

            />

            <Route path="/auth/error" component={authError} />

            <Route path="/auth" component={authIndex} />

            <Route path="/auth/signup" component={authSignup} />

            <Route path="/auth/home" component={authHome} />

          </Switch>

        </div>

      </BrowserRouter>

    );

  }

}

export default App;

Testing

Let’s test to see if our authentication app works. Run npm start on both the api and client folders. When that’s done, navigate to <http://localhost:3003/> to start testing

Let’s start by trying to log in with invalid credentials. When pressing the redirect button, you should be redirected to a 401 could not authenticate page.

Now, let’s create an account. On the main login page, select Signup under the Login, facebook, and google buttons. Input and email and password pair (one that’s linked to a google or facebook account) and press signup. You should automatically be redirected to the Homepage.

Log out by pressing the logout button. This will take you back to the base login page. Enter the credentials that you previously used to sign in. This time notice that you are sent ot the homepage rather than the error page as the credentials are valid.

Finally, test the oauth methods. Try using the google and facebook button and try logging into a google or facebook account **that is linked to the account you signed up with**. If you don’t, then you will be redirected to the error page. If you do, then you will be authenticated.

Conclusion

With this, we have a fully functioning authentication service. To test if the code is working, we can do the following.

1. Ensure that mongo db is running in the background.
2. Navigate the the api and client folders on two separate command prompts and run npm start on both
3. Go to <http://localhost:3003/>

The full repository can be found at this link: <https://github.com/DarKnight0102/easy-node-authentication>. Remember to run npm install in both the api and client folders before running if you clone the repository.