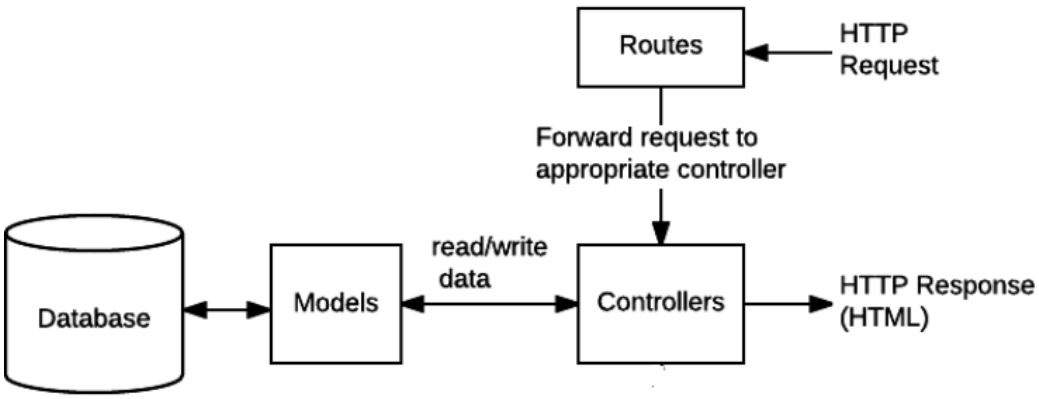
การเพิ่มการติดต่อกับฐานข้อมูล

(อ่านเพิ่มเติมได้ที่ https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes)

อาศัยหลักการพื้นฐานจาก MVC เราจะแบ่งโครงสร้างของการทำงานด้าน server เพื่อรองรับการทำ CRUD (Create Retrieve Update Delete) โดย



(แหล่งที่มาของภาพ https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes)

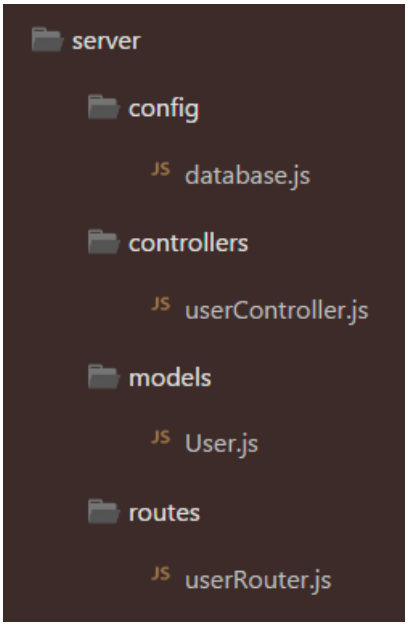
Models กำหนด Schema ของข้อมูลที่ต้องการเก็บในฐานข้อมูล

Controllers ทำหน้าที่ควบคุมการใช้โมเดล เช่นการคิวรี การบันทึก การแก้ไข

Routers ทำหน้าที่ route การร้องขอไปยัง controller ที่เหมาะสมกับการทำงาน

เริ่มต้นโดย

- ติดตั้ง mongoose ผ่าน npm เพื่อเชื่อมต่อกับฐานข้อมูล
- สร้างไดเร็กทอรีด้าน server แบ่งเป็น 4 โฟลเดอร์ย่อย คือ models, controllers, routers และ config



- กำหนด config ของ database เป็น (config/database.js)

```
module.exports =  {
  // do not forget to create user and grant role to this user
  database: 'mongodb://localhost:27017/products',
  userMongoClient: true,
}
```

- สร้างโมเดลสำหรับ user โดย Schema ของผู้ใช้ ประกอบด้วย ชื่อ, email, password (เข้ารหัส), สีที่ชอบ, isDeleted, วันที่ signUp

ไฟล์ models/User.js

```
const mongoose = require('mongoose');
const bcrypt = require('bcrypt');

const UserSchema = new mongoose.Schema({
  name: { type: String, trim: true, required: [true, 'Name is required']},
  email: { type: String, trim: true, unique: true, required: true},
  password: {type: String, required: true },
  color: {type: String},
  isDeleted: {type: Boolean, default: false },
  signUpDate: {type: Date, default: Date.now()}
});

const User = module.exports = mongoose.model('User', UserSchema, 'User');
```

- สร้างคอนโทรเลอร์สำหรับการทำงานกับโมเดล User (controllers/userController.js)

```
const User = require('../models/User.js');
const bcrypt = require('bcrypt');
const validator = require('validator');
const usrFieldProjection = {
    __v: false,
    //_id: false,
    password: false
  };

// Create and Save a new user
exports.create = (req, res) => {
  const {name, email, password, color} = req.body;

  // omit checking for correctness before save
  // if invalid content
  // return res.status(400).json({errors: {global: "Invalid User profile!"}});

  bcrypt.genSalt(10, (err, salt) => {
    bcrypt.hash(password, salt, (err, hash) => {
      if (err)
        res.status(400).json({errors: {global: 'Failed to register user: ' + err}});

      var newUser = new User({
        name: name,
        email: email,
        password: hash,
        color: color
      })
```

```
      // Save User in the database
      User.init()
      .then(function() { // avoid dup by wait until finish building index
        newUser.save()
          .then(user => {
            res.json({message: 'User Registered', user: user});
          }).catch(err => {
            res.status(400).json({errors: {global: 'Failed to register user: '+
err.errmsg}});
          });
      });
    });
  });
};

// check logon
exports.checkLogon = (req, res) => {
  const {email, password} = req.body;
  if (!email || !password) {
    res.status(404).json({errors: {global: "Invalid credential"}});
  }

  User.findOne({email: email}, (err, user) => {
    if (err || !user) {
      res.status(400).json({errors: {global: err.message||"Invalid email"}});
    }
    else {
      bcrypt.compare(password, user.password)
      .then(matched => {
        if (matched) {
          res.json({ user: {
            id: user._id,
            name: user.name,
            email: user.email,
            color: user.color
          }});
        }
        else res.status(404).json({errors: {global: "Invalid credential"}});
      })
      .catch(err => res.status(404).json({errors: {global: "Invalid credential!"}}));
    }
  });
}

// Retrieve and return all Users from the database.
exports.list = (req, res) => {
  var usersProjection = {
    __v: false,
    //_id: false,
    password: false
  };
  User.find({}, usersProjection)
//  User.find({isDeleted:false}, usersProjection)  // show only not mark as deleted
  .then(users => {
    res.json(users);
  }).catch(err => {
    res.status(500).send({
        errors: {global: err.message || "Some error occurred while retrieving Users."
    }});
```

```
  });
};

// Find a single User with an email
exports.findByEmail = (req, res) => {
  const email = req.params.email;
  User.findOne({email: email}, usrFieldProjection)
 .then(user => {
    if(!user) {
      return res.status(404).send({
          errors: {global: "User not found with email " + req.params.email
      }});
    }
    res.json(user);
  }).catch(err => {
    return res.status(500).send({
      errors: {global: "Error retrieving User with email " + req.params.email
    }});
  });
};

// Find a single User with a userId
exports.get = (req, res) => {
  const id = req.params.userId;
  User.findById(id, usrFieldProjection)
 .then(user => {
    if(!user) {
      return res.status(404).send({
          errors: {global: "User not found with id " + req.params.userId
      }});
    }
    res.json(User);
  }).catch(err => {
    if(err.kind === 'ObjectId') {
      return res.status(404).send({
          errors: {global: "User not found with id " + req.params.userId
      }});
    }
    return res.status(500).send({
      errors: {global: "Error retrieving User with id " + req.params.userId
    }});
  });
};

// Update a User identified by the userId in the request
exports.put = (req, res) => {
  // Validate Request
  const data = req.body || {};

  if (data.email && !validator.isEmail(data.email)) {
    return res.status(422).json({errors: {global: 'Invalid email address.'}});
  }

  if (data.name && !validator.isAlphanumeric(data.name)) {
    return res.status(422).send('name must be alphanumeric.');
  }

  // Find User and update it with the request body
  User.findByIdAndUpdate(req.params.userId,{$set:req.body}, {new: true})
```

```
  .then(user => {
    if(!user) {
      return res.status(404).send({
             errors: {global: "User not found with id " + req.params.userId
      }});
    }
    res.send(user);
  }).catch(err => {
    if(err.kind === 'ObjectId') {
      return res.status(404).send({
             errors: {global: "User not found with id " + req.params.userId
      }});
    }
    return res.status(500).send({
           errors: {global: "Error updating User with id " + req.params.userId
    }});
  });
};

// Updata isDeleted of the specified userId in the request
exports.delete = (req, res) => {
  User.findByIdAndUpdate(
    { _id: req.params.userId },
    { $set: { isDeleted: true }}
  )
    .then(user => {
      if (!user) {
        res.status(404).send({
           errors: {global: "User not found with id " + req.params.userId
         }});
      }
      res.send({message: "Successfully marked user as deleted!"});
    })
    .catch(err => {
        if(err.kind === 'ObjectId' || err.name === 'NotFound') {
            return res.status(404).send({
                errors: {global: "User not found with id " + req.params.userId
            }});
        }
        return res.status(500).send({
            errors: {global: "Could not delete User with id " + req.params.userId
        }});
    });
};

// Delete a User with the specified userId in the request
exports.deletePermanent = (req, res) => {
    User.findByIdAndRemove(req.params.userId)
    .then(user => {
        if(!user) {
            return res.status(404).send({
                errors: {global: "User not found with id " + req.params.userId
            }});
        }
        res.send({message: "User deleted successfully!"});
    }).catch(err => {
        if(err.kind === 'ObjectId' || err.name === 'NotFound') {
            return res.status(404).send({
                errors: {global: "User not found with id " + req.params.userId
```

```
            }});
        }
        return res.status(500).send({
            errors: {global: "Could not delete User with id " + req.params.userId
        }});
    });
};
```

- สร้าง router สำหรับการเลือกใช้ controllers ที่เหมาะสมกับการร้องขอ (routes/userRouter.js)

```
var express = require('express');
var router = express.Router();
var User_Ctrl = require('../controllers/userController');


router.post('/signup', User_Ctrl.checkLogon);
router.get('/users', User_Ctrl.list);
router.get('/user/email/:email', User_Ctrl.findByEmail);
router.get('/user/:userId', User_Ctrl.get);
router.put('/user/:userId', User_Ctrl.put);
router.post('/user/', User_Ctrl.create);
router.delete('/user/delete/:userId', User_Ctrl.deletePermanent);
router.delete('/user/:userId', User_Ctrl.delete);

module.exports = router;
```

กำหนดเส้นทาง routing ให้เหมาะกับการร้องขอ เช่น สำหรับการตรวจสอบ login ใช้ /signup จะส่งไปยัง controllers ในส่วนของ checkLogon เป็นต้น

แก้ไข index.js ไฟล์สำหรับการทำงานของ server

- เชื่อมโยงเข้ากับระบบจัดการฐานข้อมูล mongoose
- เปลี่ยนมาให้ใช้ router ที่สร้างขึ้นแทนการกำหนดเมท็อดเฉพาะแต่ละอัน

```
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

const mongoose = require("mongoose");
const config = require('./config/database');

mongoose.connect(config.database);
mongoose.connection.on('connected', () => {
  console.log('Connected to database '+config.database);
});
mongoose.connection.on('error', () => {
  console.log('Database error');
});
```

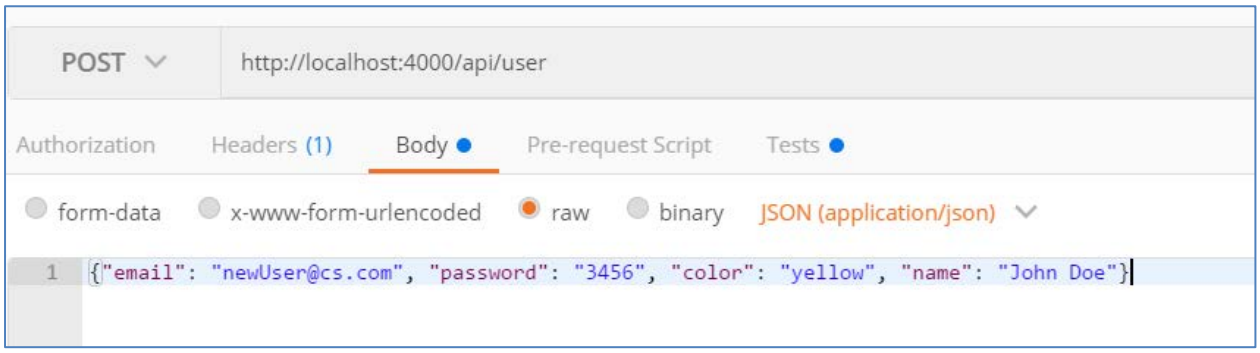
```
var users = require('./routes/userRouter');

// REST for users
app.use('/api', users);

app.get('/', (req, res) => {
  res.send('Invalid Endport');
});

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

var port = 4000;

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

การทดสอบการทำงาน

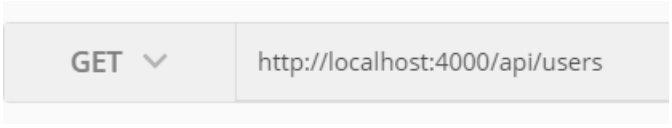
- start MongoDB server ที่ port 27017: **`mongod --port 27017 -dbpath`** *`databasePath`*
- start Node.js server (*`npm run server`*)

ทดสอบการทำงานของ server โดย Postman ตาม route ที่กำหนด เช่น

สร้างผู้ใช้ใหม่



หรือค้นหาผู้ใช้ทั้งหมด



นอกจากนี้สามารถทดสอบการทำงานกับ React Client โดยใช้ login ที่สร้างขึ้นจากการทดสอบ จะเห็นว่าสามารถผ่านการยืนยันตัวตนได้

การสร้าง Session ด้วย JWT (JSON Web Token)

JWT เป็น JSON-based open standard (RFC 7519) ใช้เพื่อการสร้าง tokens สำหรับการเข้าถึง ช่วยในการทำ session โดย server สร้าง token ที่ถูกเข้ารหัสด้วยคีย์ของ Server ส่งไปให้ client ซึ่งก็สามารถตรวจสอบรหัสว่าเป็นของ server ได้ โดย client จะใช้ token นี้ส่งกลับไปให้ server พร้อมกับการร้องขอในคราวต่อๆ ไป

การสร้าง Middleware เพื่อเป็นฟังก์ชันสำหรับการทำงานกับ JWT

- ติดตั้ง passport, passport-jwt
- สร้างไฟล์ config/jwtConfig เพื่อเก็บคีย์ของการเข้ารหัสสำหรับ JWT

```
module.exports = {
  jwtSecret: 'yoursecret',
  jwtSession: {
      session: false
  }
};
```

- สร้างฟังก์ชัน สำหรับการทำงานของ JWT (ไฟล์ util/jwt-passport.js)

```
// 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')
    .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);
    }
  };
}
```

- แก้ไขไฟล์ UserControllers ให้ฟังก์ชัน checkLogon เพิ่มส่วนการใช้ Token

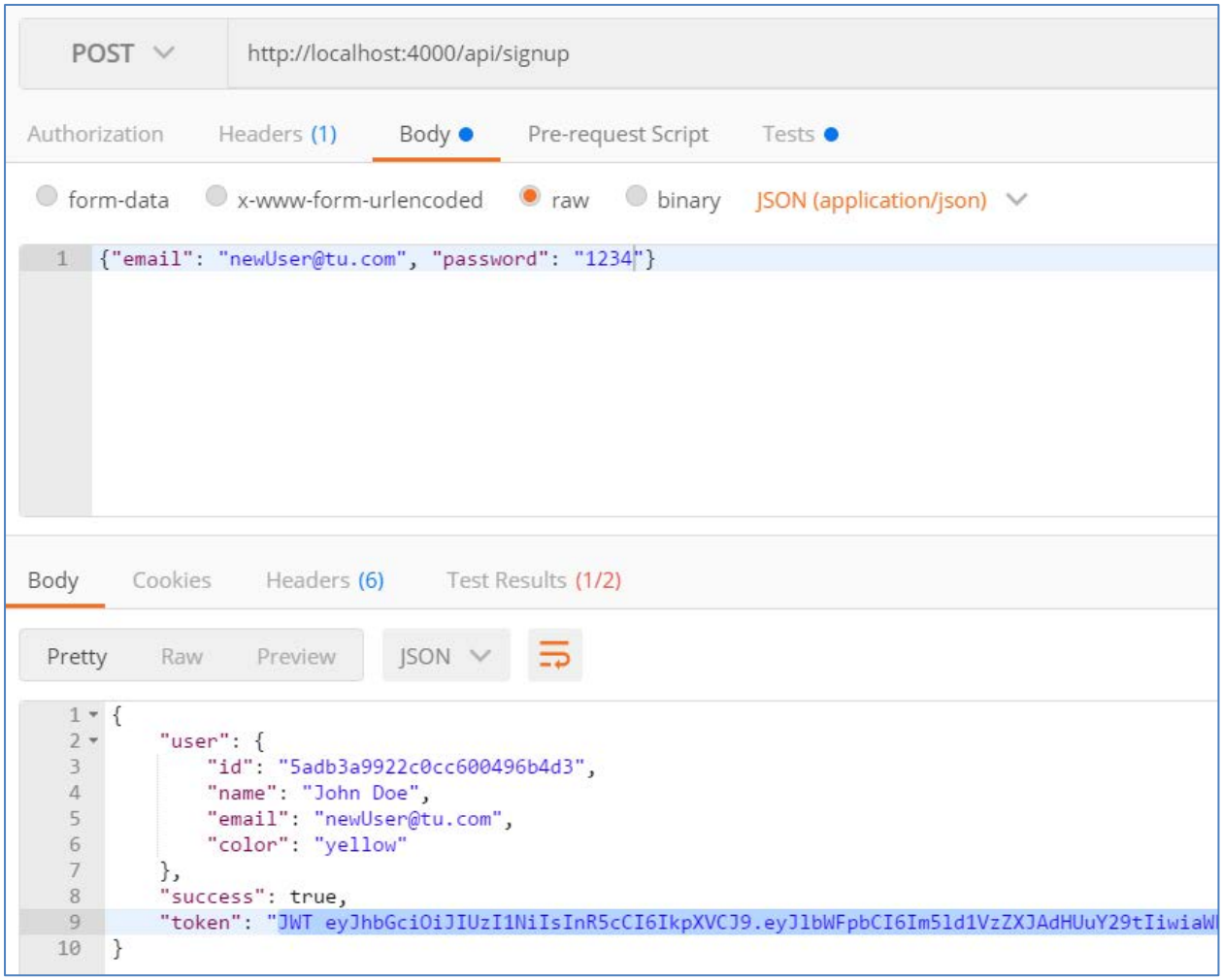
```
const jwt = require('jsonwebtoken');
const config = require("../config/jwtConfig");

exports.checkLogon = (req, res) => {
  const {email, password} = req.body;
  if (!email || !password) {
    res.status(404).json({errors: {global: "Invalid credential"}});
  }

  User.findOne({email: email}, (err, user) => {
    if (err || !user) {
      res.status(400).json({errors: {global: err||"Invalid email"}});
    }
    else {
      bcrypt.compare(password, user.password)
      .then(matched => {
        if (matched) {
          const token = jwt.sign({ email: user.email },
                                 config.jwtSecret,
                                 { expiresIn: 3600 }); // 1 hour
          res.json({ user: {
              id: user._id,
              name: user.name,
              email: user.email,
              color: user.color
            },
            success: true,
            token: "JWT " + token,
          });
        }
        else res.status(404).json({errors: {global: "Invalid credential"}});
      })
      .catch(err => res.status(404).json({errors: {global: "Invalid credential!"}}));
    }
  });
}
```

- แก้ไขไฟล์ UserRouter ให้ router ขดงการลบต้องผ่าน authentication แล้วจึงใช้งานได้

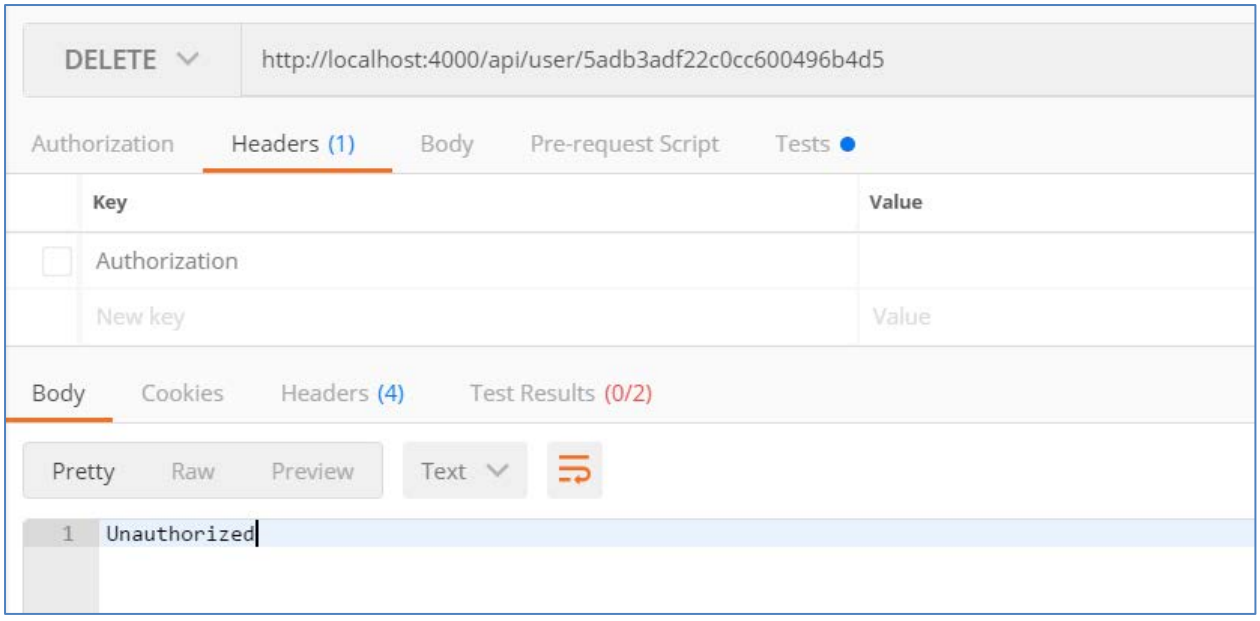
```
// look for more detail at
// https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/routes
var express = require('express');
var router = express.Router();
var User_Ctrl = require('../controllers/userController');

var auth = require("../utils/jwt-passport.js")();
router.use(auth.initialize());

router.post('/signup', User_Ctrl.checkLogon);
router.get('/users', User_Ctrl.list);
router.get('/user/email/:email', User_Ctrl.findByEmail);
router.get('/user/:userId', User_Ctrl.get);
router.put('/user/:userId', User_Ctrl.put);
router.post('/user/', User_Ctrl.create);
router.delete('/user/delete/:userId', User_Ctrl.deletePermanent);
router.delete('/user/:userId', auth.authenticate(), User_Ctrl.delete);
```

```
module.exports = router;
```

ทดสอบโดยใช้ Postman โดยทำ signup เพื่อให้ได้ค่า Token



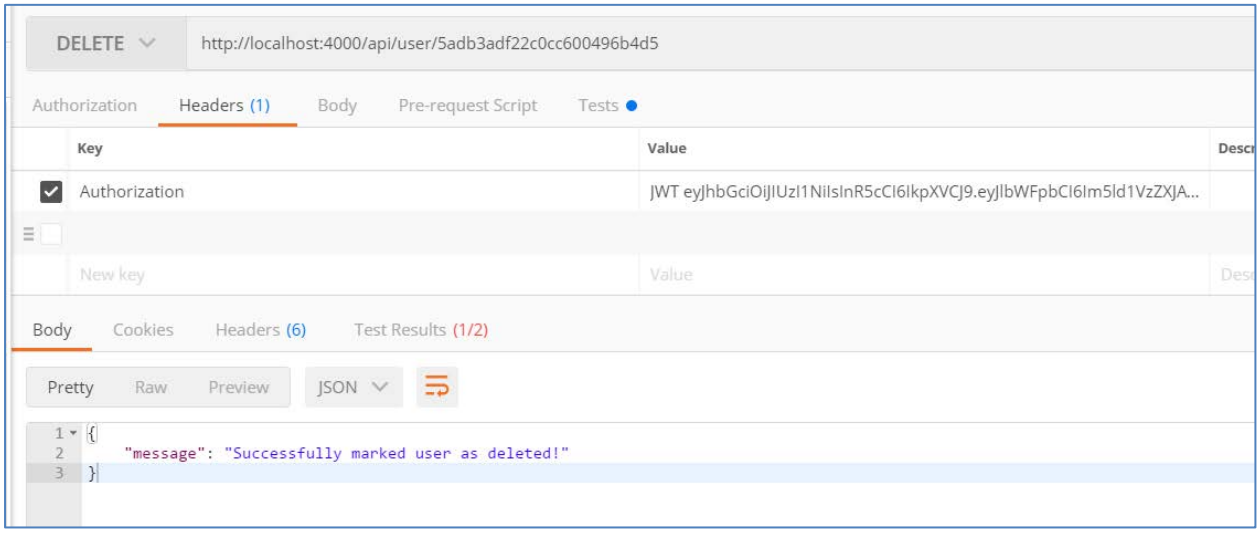
ทดสอบด้วยการร้องขอการลบรายการ



ระบบแสดงว่าไม่มีสิทธิ

ทดสอบใหม่อีกครั้ง โดยสำเนาค่า token ที่ได้รับ เมื่อตอน signUp แล้วร้องขอใหม่ โดยให้ค่า Header เป็น

Authorization: value ของ Token



จะเห็นว่าทำการลบรายการได้สำเร็จ

การแก้ไข client ให้เก็บค่า Token ไว้กับ sessionStorage ของ client

```
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: {}, token: '' };

  componentDidMount() {
    let prevState = sessionStorage.getItem('token');
    if (prevState !== null)
      this.setState({token: prevState, isAuthenticated: true});
  }

  setAuth = (data) => {
    this.setState({ isAuthenticated: true, user: data.user, token: data.token });
    sessionStorage.setItem('token', data.token);
  };

  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;
```

เมื่อทดสอบการทำงานพบว่าเมื่อ Login แล้วทำ reload หน้าใหม่อีกครั้ง ผู้ใช้ยังคงมีสถานะ isAuthorized อยู่ ต่อไปเราสามารถนำ Token ที่ได้รับนี้ ไปใช้ร้องขอการทำงานที่ต้องมีการยืนยันตัวตนกับ server ก่อนได้