DESARROLLO DE APLICACIONES WEB AVANZADO

LABORATORIO N° 06

JWT

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

Criterio de Evaluación	*Excelente (4pts)*	*Bueno (3pts)*	*Requiere mejora (2pts)*	*No acept. (0pts)*	*Puntaje Logrado*
Entiende cómo funciona JWT					
Utiliza mongoose para conexión de base de datos					
Desarrolla aplicaciones web con express					
Realiza con éxito lo propuesto en la tarea.					
Es puntual y redacta el informe adecuadamente					

Laboratorio 06:
JWT

Objetivos:

Al finalizar el laboratorio el estudiante será capaz de:

- Entender el funcionamiento de Socket.io
- Desarrollar aplicaciones web con el framework Express
- Implementación correcta de mongoose

Seguridad:

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

Equipos y Materiales:

- Una computadora con:
  - Windows 7 o superior
  - VMware Workstation 10+ o VMware Player 7+
  - Conexión a la red del laboratorio
- Máquinas virtuales:
  - Windows 7 Pro 64bits Español - Plantilla
- Instalador de node.js

Procedimiento:

Lab Setup

1. **Configuración inicial del proyecto**
   1.1. Crearemos la carpeta **dawa_api** y ejecutaremos el comando **npm init** siguiendo la configuración propuesta por defecto.

   1.2. Esta acción, habrá creado un proyecto npm para poder gestionar. Ahora nuestros comandos de instalación de librerías variarán. Instale las siguientes librerías, con el argumento **--save**

```
$ npm install express --save
$ npm install body-parser --save
$ npm install mongoose --save
$ npm install --save bcrypt-nodejs
$ npm install --save jsonwebtoken
```

   1.3. Procederemos a crear nuestro archivo principal. Cree el archivo server.js con el siguiente contenido.

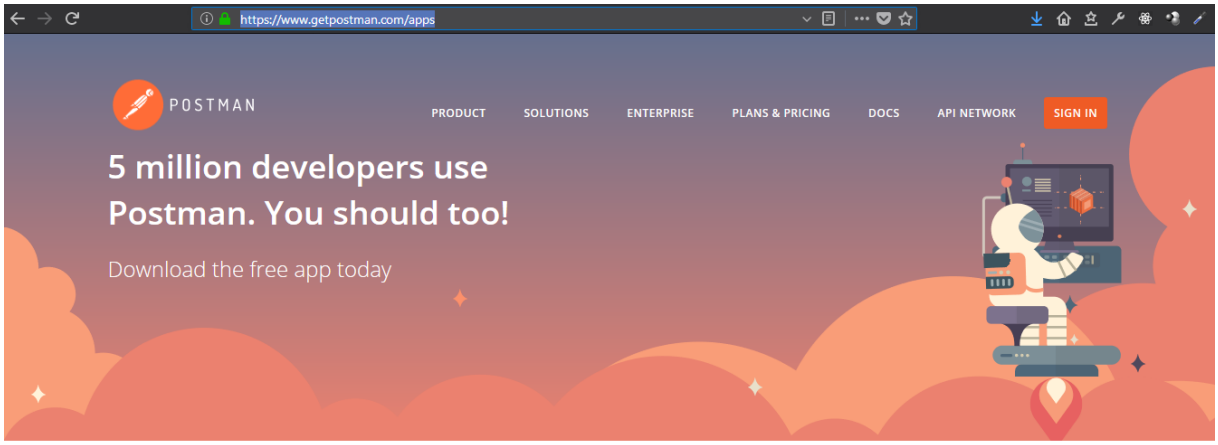
```
// importamos los paquetes necesarios
const express    = require('express');        // importamos express
const app        = express();                 // instanciamos una aplicación
const bodyParser = require('body-parser');

// configuramos nuestra app para usar bodyParser()
// el cual nos permitirá obtener data enviada por POST
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());

const port = process.env.PORT || 5000;        // configuramos nuestro puerto

app.use("/", function(req, res, next){
    res.header('Access-Control-Allow-Origin', '*');
    res.header('Access-Control-Allow-Methods', 'GET,PUT,POST,DELETE,OPTIONS');
    res.header('Access-Control-Allow-Headers', 'Content-Type, Authorization, Content-Length, X-Requested-With');
    next();
});

app.options("/*", function(req, res, next){
    res.sendStatus(200);
});

// RUTAS PARA NUESTRA API
// =============================================================================
const router = express.Router();              // obtenemos una instancia del enrutador de express

// ruta de prueba para ver si todo funciona (accesible por GET http://localhost:5000/api)
router.get('/', function(req, res) {
    res.json({ message: 'genial! bienvenido a nuestra api!' });
});

// REGISTRAMOS NUESTRAS RUTAS ------------------------------
// todas las rutas tendrán el prefijo /api
app.use('/api', router);

// Nos conectamos a nuestra base de datos
const mongoose    = require('mongoose');
mongoose.connect('mongodb://localhost:27017/dawa_blog');
mongoose.Promise = global.Promise;

// INICIAMOS EL SERVIDOR
// =============================================================================
app.listen(port);
console.log('La magia sucede en el puerto ' + port);
```

1.4. Al ejecutar **npm start** deberemos ver nuestro mensaje en la consola.

```
$ node server.js
```

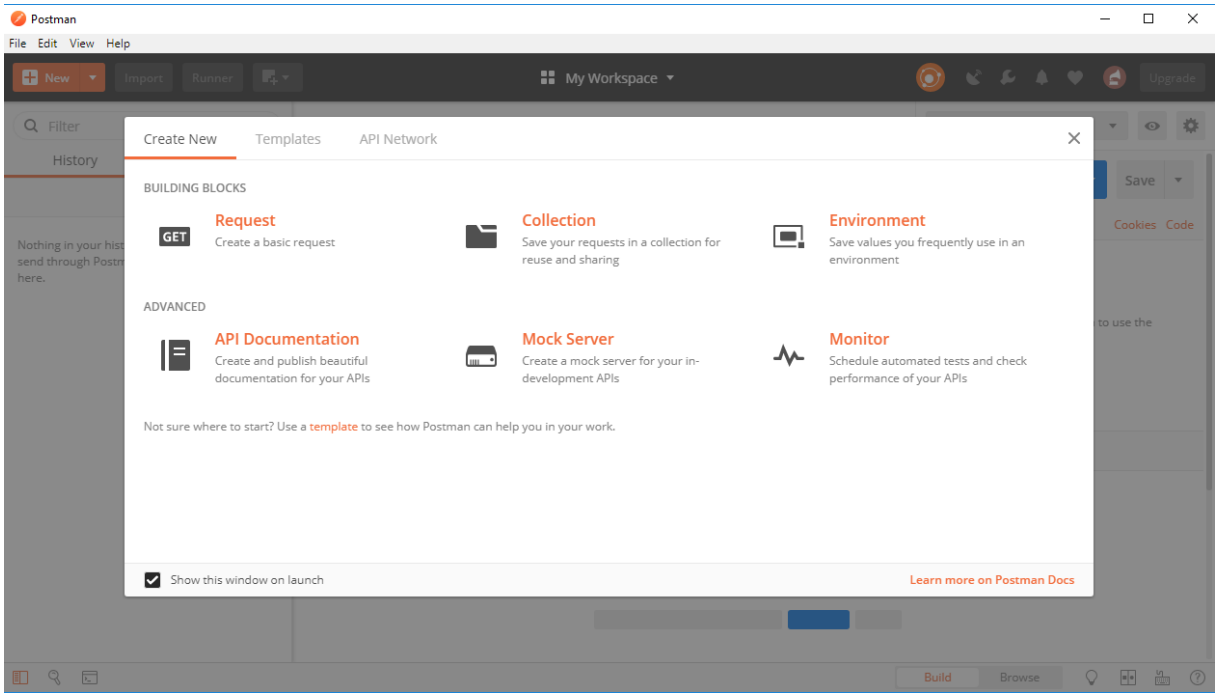
2. Instalación de Postman

2.1. Entre a la página web https://www.getpostman.com/apps y descargue la versión adecuada para su sistema operativo.

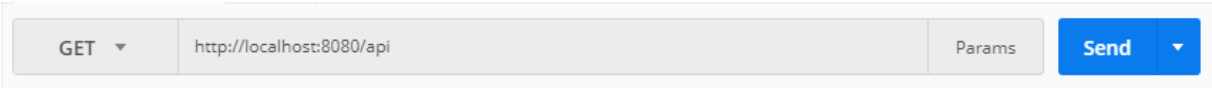


2.2. Instale como cualquier programa común con las opciones por defecto.

2.3. Después de iniciado el programa, veremos una pantalla parecida a la siguiente, la cual saltaremos para iniciar nuestra prueba.



2.4. En la parte del protocolo, lo dejaremos por defecto con GET y pondremos la siguiente URL para testear.



2.5. Haga click en Send y adjunte los resultados

3. Funciones de nuestra API

3.1. La forma de nuestra API será la siguiente:

Ruta	Acción HTTP	Descripción
/api/user	GET	Obtiene todos los usuarios
/api/user/:id	GET	Obtiene la data de un usuario

/api/user	POST	Crea un nuevo usuario
/api/user/:id	PUT	Actualiza un usuario
/api/user/:id	DELETE	Elimina un usuario

3.2. Cree la carpeta models y dentro de ella al archivo user.js con el siguiente contenido

```
const mongoose  = require('mongoose');
const Schema = mongoose.Schema;

const validateEmail = function(email) {
    const re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
    return re.test(email)
};

const userSchema   = new Schema({
    username: {
        type: String,
        trim: true,
        required: true,
        match: [/^[a-zA-Z0-9]+$/, 'username is invalid'],
        index: true,
        unique: true
    },
    name: {
        type: String,
        required: true
    },
    age: {
        type: Number,
        required: true
    },
    gender: {
        type: String,
        enum: ['F','M']
    },
    email: {
        type: String,
        trim: true,
        lowercase: true,
        required: true,
        unique: true,
        validate: [validateEmail, 'Please fill a valid email address']
    },
    password: {
        type: String,
        require: true,
        min: 6,
        max: 24
    }
}, {timestamps: true});

module.exports = userModel;
```

3.3. Cree la carpeta routes con el archivo user.js con el siguiente contenido

```
const express = require('express');
const router = express.Router();

const userController = require('../controllers/user');

router.get('/',userController.find);
router.get('/:id',userController.findOne);
router.post('/',userController.create);
router.put('/:id',userController.update);
router.delete('/:id',userController.delete);

module.exports = router;
```

3.4. Cree la carpeta controllers con el archivo user.js con el siguiente contenido

```
const User = require('../models/user.js');

const exposedFields = [
    'username',
    'name',
    'email'
];

module.exports = {
    create: (req,res,next)=>{
        var user = new User({
            ...req.body
        });
        user
            .save()
            .then(result => {
                res.status(200).json({
                    message: 'User succesfully created!',
                    data: {
                        ...result['_doc']
                    }
                });
            })
            .catch(err => {
                console.log(err);
                res.status(500).json({
                    error:err
                });
            });
    },
    find: (req,res,next) => {
        User.find()
            .select(exposedFields.join(' '))
            .exec()
            .then(docs => {
```

```
            .exec()
            .then(docs => {
                const response = {
                    count: docs.length,
                    data: docs.map(doc => {
                        return {
                            ...doc['_doc']
                        };
                    })
                };
                res.status(200).json(response);
            })
            .catch(err=>{
                console.log(err);
                res.status(500).json({
                    error:err
                });
            });
    },
    findOne: (req,res,next)=>{
        const id = req.params.id;
        User.findById(id)
            .exec()
            .then(doc => {
                if(doc){
                    res.status(200).json({
                        data: doc['_doc'],
                    });
                } else {
                    res.status(404).json({message: 'No valid entry found for provided ID'});
                }
            })
            .catch(err =>{
                console.log(err);
                res.status(500).json({
                    error:err
                });
            });
    },
```

```
            });
    },
    update: (req,res,next) => {
        const id = req.params.id;
        let updateParams = {
            ...req.body
        };
        User.update({_id: id},{$set: updateParams})
            .exec()
            .then(result => {
                res.status(200).json({
                    message: 'User updated!',
                    data: result['_doc']
                });
            })
            .catch(err =>{
                console.log(err);
                res.status(500).json({
                    error: err
                });
            });
    },
    delete: (req,res,next)=>{
        const id = req.params.id;
        User.remove({_id: id})
            .exec()
            .then(result => {
                res.status(200).json({
                    message: 'User deleted!'
                });
            })
            .catch(err =>{
                console.log(err);
                res.status(500).json({
                    error: err
                });
            });
        });
    }
};
```

3.5. Modifique nuestro archivo server.js para que luzca de la siguiente manera

```
// ruta de prueba para ver si todo funciona (accesible por GET http://localhost:5000/api)
router.get('/', function(req, res) {
    res.json({ message: 'genial! bienvenido a nuestra api!' });
});

const userRouter = require('./routes/user');
router.use('/user', userRouter);

// REGISTRAMOS NUESTRAS RUTAS ---------------------------------
// todas las rutas tendrán el prefijo /api
app.use('/api', router);
```

4. Encriptación de contraseña

4.1. Modificaremos el inicio de nuestro modelo user.js

```
const mongoose  = require('mongoose');
const bcrypt = require('bcrypt-nodejs');
const Schema = mongoose.Schema;

const SALT_WORK_FACTOR = 10;
const validateEmail = function(email) {
    const re = /^\w+([\.-]?\w+)*@\w+([\.-]?\w+)*(\.\w{2,3})+$/;
    return re.test(email)
};
```

4.2. Así mismo, modificaremos el final del mismo archivo para que luzca de la siguiente manera

```
}, {timestamps: true});

userSchema.pre('save', function(next) {
    var user = this;

    // isModified permite preguntar si un campo ha sido modificado
    //en caso no se haya modificado el password, le decimos que continue
    if (!user.isModified('password')) return next();
    // generamos nuestra encriptcion
    bcrypt.genSalt(SALT_WORK_FACTOR, function(err, salt) {
        if (err) return next(err);
        // una vez con nuestra clave, procedemos a cifrar el password
        bcrypt.hash(user.password, salt, null, function(err, hash) {
            if (err) return next(err);
            user.password = hash;
            next();
        });
    });
});

userSchema.methods.comparePassword = function(candidatePassword){
    return new Promise((resolve,reject) => {
        bcrypt.compare(candidatePassword, this.password, (err, isMatch) => {
            if(err) reject({error: true,message: 'Password required'});
            resolve(isMatch);
        });
    })
};

const userModel = mongoose.model('users', userSchema);

module.exports = userModel;
```

5. Creación de JWT
   5.1. Crearemos la carpeta lib con el archivo utils.js

```
const jwt = require('jsonwebtoken');

module.exports = {
    generateToken: user => {
        //1. Dont use password and other sensitive fields
        //2. Use fields that are useful in other parts of the
        //app/collections/models
        const u = {
            _id: user._id,
            name: user.name,
            username: user.username,
            email: user.email
        };
        return token = jwt.sign(u, process.env.JWT_SECRET, {
            expiresIn: 60 * 60 * 24 // expires in 24 hours
        });
    },
    verifyToken: token => {
        return new Promise((resolve, reject) => {
            jwt.verify(token, process.env.JWT_SECRET, (err,user) => {
                if (err){
                    reject(err);
                }
                //return user using the id from w/in JWTToken
                resolve(user);
            });
        });
    },
    getCleanUser: user => {
        const { password, age, createdAt, updatedAt, __v, ...exposedData } = user;
        return exposedData;
    }
};
```

5.2. Agregaremos las siguientes rutas dentro de nuestro controlador. Es importante que estas sean colocadas antes de las rutas anteriormente declaradas (sobre todo para que funcione el middleware de verificación de token “verifyToken”)

```
router.post('/signup',userController.signup);
router.post('/signin',userController.signin);
router.post('/refresh',userController.refreshToken);

router.use(userController.verifyToken);
```

5.3. Así mismo, modificaremos nuestro archivo user.js dentro de controllers para que ya contenga las funciones de las rutas que acabamos de agregar.

```
const utils = require('../lib/utils');

const User = require('../models/user.js');

const exposedFields = [
    'username',
    'name',
    'email'
];

module.exports = {
    signup: (req,res,next)=>{
        var user = new User({
            ...req.body
        });
        user
            .save()
            .then(result => {
                const token = utils.generateToken({
                    _id: result['_doc']['_id'],
                    name: result['_doc']['name'],
                    username: result['_doc']['username'],
                    email: result['_doc']['email']
                });
                const exposedData = utils.getCleanUser(result['_doc']);
                res.status(200).json({
                    message: 'User succesfully signup!',
                    data: exposedData,
                    token: token
                });
            })
            .catch(err => {
                console.log(err);
                res.status(500).json({
                    error:err
                });
            });
    },
    signin: (req,res,next)=>{
        User
            .findOne({username: req.body.username})    //<-- Check username
            .select(exposedFields.join(' ')+' password')
            .exec((err, user) => {
                if(err) res.status(500).json(err);
                if(!user){
                    return res.status(401).json({
                        error: true,
                        message: 'Username or Password is wrong'
                    });
                }
                user.comparePassword(req.body.password)
                    .then(valid => {
                        if(!valid){
                            return res.status(401).json({
                                error: true,
                                message: 'Username or Password is wrong'
                            });
                        }
                        res.json({
                            message: 'User succesfully logged!',
                            data: user,    //<--- Return both user and token
                            token: utils.generateToken(user)  //<-- Generate token
                        });
                    })
```

```
                .catch(err => {
                    res.status(500).json(err);
                });
        });
    },
    refreshToken: (req,res,next)=>{
        // check header or url parameters or post parameters for token
        var token = req.body.token || req.query.token;
        if (!token) {
            return res.status(401).json({message: 'Must pass token'});
        }

        // Check token that was passed by decoding token using secret
        utils.verifyToken(token)
            .then(user => {
                //search user after token info
                User.findById({
                    '_id': user._id
                }, function(err, user) {
                    if (err) throw err;

                    const exposedData = utils.getCleanUser(user['_doc']);
                    const newToken = utils.generateToken(exposedData);

                    res.status(200).json({
                        user: exposedData,
                        token: newToken
                    });
                });
            })
            .catch(err => {
                res.status(500).json(err);
            });
    },
    verifyToken: (req,res,next)=>{
        const token = req.headers['authorization'];
        if (!token) res.status(401).json({
            error: true,
            message: 'Please register Log in using a valid email to submit posts'
        });
        utils.verifyToken(token)
            .then(function(user){
                //return user using the id from w/in JWTToken
                req.user = user; //set the user to req so other routes can use it
                next();
            })
            .catch(function(err){
                console.log(err)
                res.status(500).json({
                    error: err
                });
            });
    },
```

6. **Creación de sitio en React**

6.1. Procederemos a crear el proyecto lab06 con react

```
$ create-react-app lab06
```

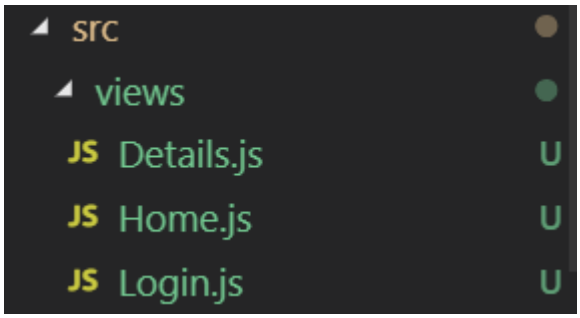
6.2. Iniciaremos el proyecto para probar su funcionamiento

```
$ cd lab06
/lab06$ npm start
```

6.3. Para este proyecto, instalaremos las librerías para navegación, en este caso, react-router

```
$ npm install --save react-dom react-router react-router-dom
```

6.4. Crearemos ahora la carpeta views dentro de src, con los archivos **Details.js**, **Home.js** y **Login.js**



6.5. Contenido de **Home.js**

```
import React from 'react';

const Home = () => {
    return (<div>
        <h1>Este es mi inicio</h1>
    </div>);
};

export default Home;
```

6.6. Contenido de **Details.js**

```
import React from 'react';

const Details = () => {
    return (<div>
        <h1>Este son los detalles</h1>
    </div>);
};

export default Details;
```

6.7. Contenido de **Login.js**

```
import React from 'react';

const Login = () => {
    return (<div>
        <h1>Este es el login</h1>
    </div>);
};

export default Login;
```

6.8. Ahora modificaremos App.js para declarar las rutas de nuestra aplicación web.

```
import React, { Component } from 'react';
import { BrowserRouter, Route, Switch} from 'react-router-dom';

import Home from './views/Home';
import Details from './views/Details';
import Login from './views/Login';

class App extends Component {
    render() {
        return (<BrowserRouter>
            <div>
                <h1>Sitio de Películas</h1>
                <Switch>
                    <Route path='/' exact component={Home} />
                    <Route path='/details' component={Details} />
                    <Route path='/login' component={Login} />
                </Switch>
            </div>
        </BrowserRouter>);
    }
}

export default App;
```

6.9. Modificaremos Home.js para ver el funcionamiento de la navegación. Fíjese en como la página cambia, pero no se recarga. Este es el efecto de utilizar React, lograr una **SINGLE PAGE APLICATION**

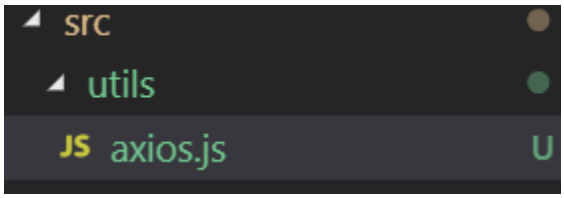
```
import React from 'react';
import {Link} from 'react-router-dom';

const Home = () => {
    return (<div>
        <h1>Este es mi inicio</h1>
        <Link to='login'>
            Iniciar sesión
        </Link>
    </div>);
};

export default Home;
```

7. Conectaremos nuestro proyecto con la API

7.1. Al igual que en el caso de React Native, instalaremos axios para comunicarnos con el backend

```
$ npm install --save axios
```

7.2. Crearemos un archivo de configuración (no es obligatorio, pero nos ayudará en el proyecto) el cual estará ubicado dentro de una nueva carpeta llamada **utils** dentro de **src**.



7.3. El contenido de **axios.js** será el siguiente:

```
import axios from 'axios';

const instance = axios.create({
    baseURL: 'http://localhost:5000/api/'
});

export default instance;
```

7.4. Modificaremos totalmente Login.js para que sea un formulario que envíe una consulta a la API

```
import React,{Component} from 'react';

import axios from '../utils/axios';

class Login extends Component{
    state = {
        usuario: '',
        password: ''
    }
    usuarioHandler = e => this.setState({usuario: e.target.value})
    passwordHandler = e => this.setState({password: e.target.value})
    submitHandler = e => {
        e.preventDefault();
        axios({
            method: 'post',
            url: 'user/signin',
            data: {
                username: this.state.usuario,
                password: this.state.password
            }
        }).then(response => {
            console.log('mi respuesta',response)
        }).catch(error => {
            console.log('hubo un error',error)
        })
    }
    render(){
```

```
    render(){
        return (<form onSubmit={this.submitHandler}>
            <div>
                <label>Usuario</label>
                <input
                    type='text'
                    placeholder='Usuario para sesión'
                    value={this.state.usuario}
                    onChange={this.usuarioHandler} />
            </div>
            <div>
                <label>Contraseña</label>
                <input
                    type='password'
                    placeholder='123456'
                    value={this.state.password}
                    onChange={this.passwordHandler} />
            </div>
            <button type='submit'>Iniciar sesión</button>
        </form>);
    }
}

export default Login;
```

8. Ejercicio propuesto
   8.1. Se debe realizar el mismo formulario (usuario y contraseña) pero en React Native.

9. **Finalizar la sesión**
   9.1. Apagar el equipo virtual
   9.2. Apagar el equipo

Conclusiones:

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