WEB PROGRAMMING

MODUL PRAKTIKUM Autentikasi (Passport.JS & JWT)

DAFTAR ISI

[1. Persiapan 2](#_Toc57120609)

[2. Passport JS 3](#_Toc57120610)

[2.1 Pendahuluan PassportJS 3](#_Toc57120611)

[2.2 Konfigurasi Passport.JS 4](#_Toc57120612)

[3. JsonWebToken 13](#_Toc57120613)

[3.1 Pendahuluan Apa itu JWT? 13](#_Toc57120614)

[3.2 Cara Kerja JWT 14](#_Toc57120615)

[3.3 Konfigurasi JWT 16](#_Toc57120616)

[3.4 Kode Full JWT 23](#_Toc57120617)

# Persiapan

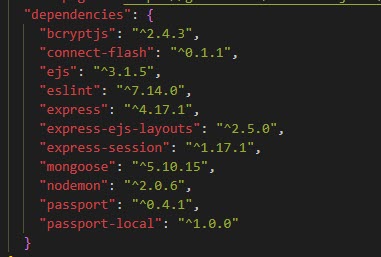
Pada pertemuan sebelumnya Kita sudah belajar menggunakan template engine EJS dan database noSQL menggunakan MongoDb. 2 modul pada pertemuan sebelumnya dapat dilanjutkan untuk menambahkan penggunaan passportjs dan JSON Web Token (JWT) pada praktikum ini.

Untuk melanjutkan praktikum sebelumnya file praktikum dapat diunduh pada tautan berikut :

<https://github.com/irvanbelajarit/passportjsnjwt/tree/databasemongoose>

pada tautan diatas merupakan file yang akan ditambahkan autentikasi pada praktikum saat ini.

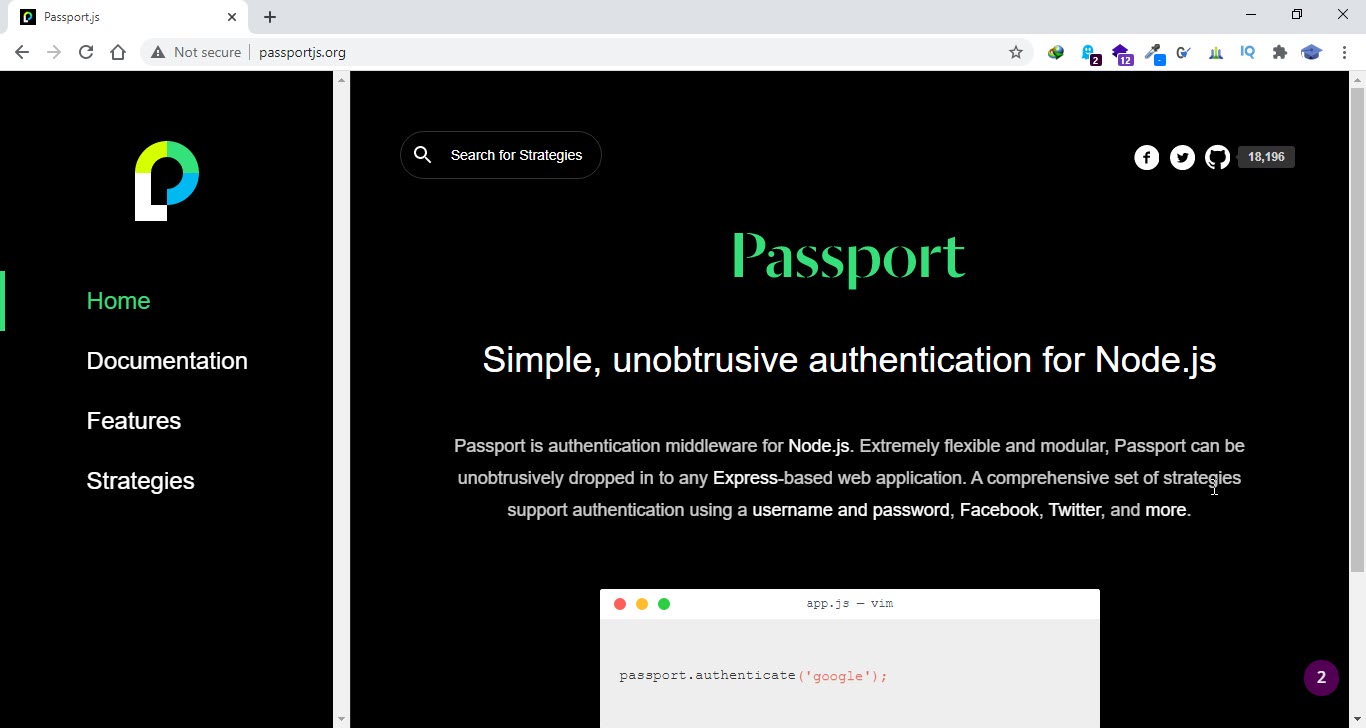
sebelum menggunakan file praktikum diatas Kita harus menginstall dependensi yang ada sebagai berikut dapat dilihat pada package.json didalam file tersebut.



# Passport JS

## Pendahuluan PassportJS

**Passport JS** adalah otentikasi middleware untuk Node.js. Sangat fleksibel dan modular, Passport dapat dengan mudah digunakan untuk aplikasi web berbasis framework Express JS. Seperangkat strategi komprehensif mendukung otentikasi menggunakan Username dan Password, Facebook, Twitter, dan lainnya. Kita tidak perlu membuat sebuah otentikasi dari dasar dengan memanfaatkan strategi - strategi yang disediakan oleh passportjs.

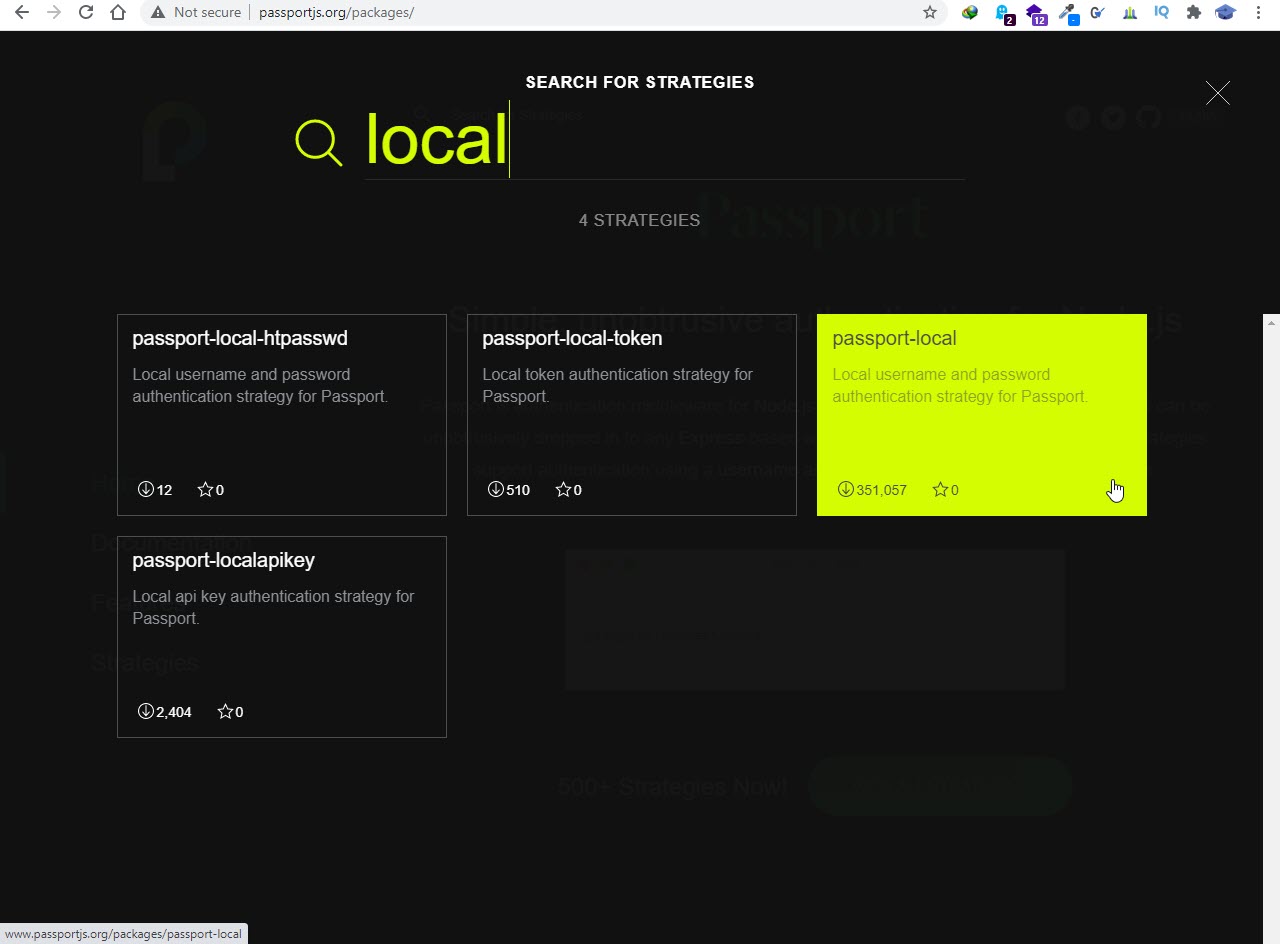


Banyak modul atau srategi yang bisa digunakan Passport JS, diantaranya yang banyak digunakan adalah: Modul Otentikasi Oauth 2.0, Modul Otentikasi Oauth Profile FB, Modul Otentikasi Oauth Twitter dan Modul Otentikasi Oauth Google. Otentikasi dapat digunakan secara praktis dan tentunya aman dan juga pengerjaan akan lebih cepat dengan adanya modul Otentikasi yang mudah dimengerti.

## Konfigurasi Passport.JS

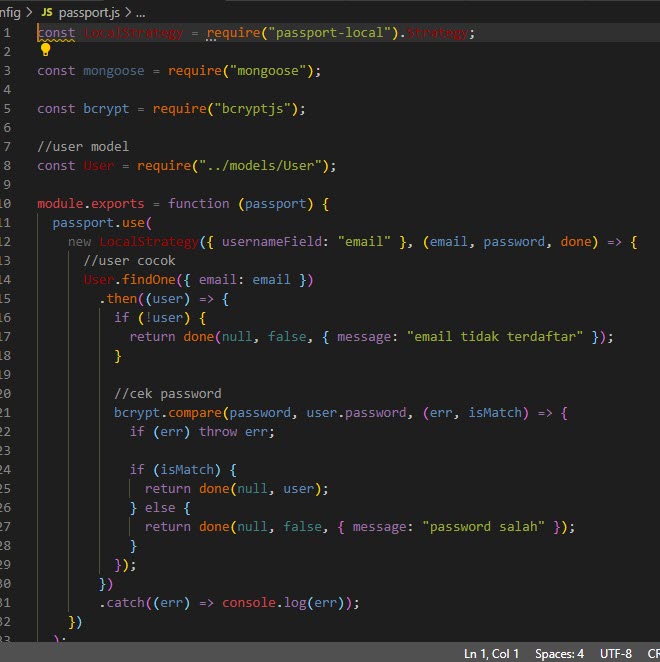
Passport js memiliki beberapa strategi untuk melakukan autentikasi. Informasi ini dapat Kita lihat pada website resmi passportjs <http://www.passportjs.org/>

Pada praktikum ini Kita akan menggunakan salah satu strategi passportjs yaitu Local strategy.



Pada folder config buat file baru dengan nama “passport.js”

isikan file tersebut seperti gambar dibawah ini :



Keterangan :

pada file ini Kita akan menggunakan local strategy yang artinya kita akan menggunakan database lokal untuk proses autentikasi menggunakan passport.js

const LocalStrategy = require("passport-local").Strategy;

Kemudian Kita akan memakai kriptografi bcrypt.js untuk membuat sebuah hash yang akan digunakan untuk menyimpan password kedalam database

const bcrypt = require("bcryptjs");

Pada script dibawah :

 passport.use(

    new LocalStrategy({ usernameField: "email" }, (email, password, done) => {

      //user cocok

      User.findOne({ email: email })

        .then((user) => {

          if (!user) {

            return done(null, false, { message: "email tidak terdaftar" });

          }

pada usernameField di hubungkan dengan data email agar tetap bersifat unik.

Jika email tidak ada pada database, maka akan muncul pesan “email tidak terdaftar”

Setelah mengecek email pada database selanjutnya akan di cocokan hash password yang diinputkan user dengan password yang ada pada database.

Jika password tidak sama maka akan mengeluarkan pesan “password salah”

 //cek password

          bcrypt.compare(password, user.password, (err, isMatch) => {

            if (err) throw err;

            if (isMatch) {

              return done(null, user);

            } else {

              return done(null, false, { message: "password salah" });

            }

          });

Selanjutnya Kita akan melakukan serialisasi dan deserialisasi data untuk dikirim atau di terima oleh User.

  passport.serializeUser(function (user, done) {

    done(null, user.id);

  });

  passport.deserializeUser(function (id, done) {

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

      done(err, user);

    });

  });

pada file “app.js”

tambahkan script yang ada pada kotak berwarna merah sebagai berikut :

const passport = require("passport");

const app = express();

//passport config

require("./config/passport")(passport);

Untuk dapat dikenali oleh file app.js maka Kita harus mengimport config/passport dan modul passport terlebih dahulu.

Selanjutnya lakukan inisialisasi passport dan session pada file “app.js” :

// Passport middleware

app.use(passport.initialize());

app.use(passport.session());

//connect flash

app.use(flash());

Selanjutnya Kita akan membuat route untuk autentikasi login pada file “user.js”.

tambahkan pada bagian paling bawah script sebagai berikut :

//login handle

router.post("/login", (req, res, next) => {

  passport.authenticate("local", {

    successRedirect: "/dashboard",

    failureRedirect: "/users/login",

    failureFlash: true,

  })(req, res, next);

});

Jika autentikasi sukses maka akan dialihkan ke halaman dashboard, Jika gagal maka akan dialihkan ke halaman users/login dengan flash message yang sudah di tentukan sebelumnya.

Kemudian pada bagian atas tambahkan import modul passportnya.

const passport = require("passport");

//user model

const User = require("../models/User");

pada file app.js tambahkan script global vars untuk menambahkan informasi error :

// global var

app.use((req, res, next) => {

  res.locals.success\_msg = req.flash("success\_msg");

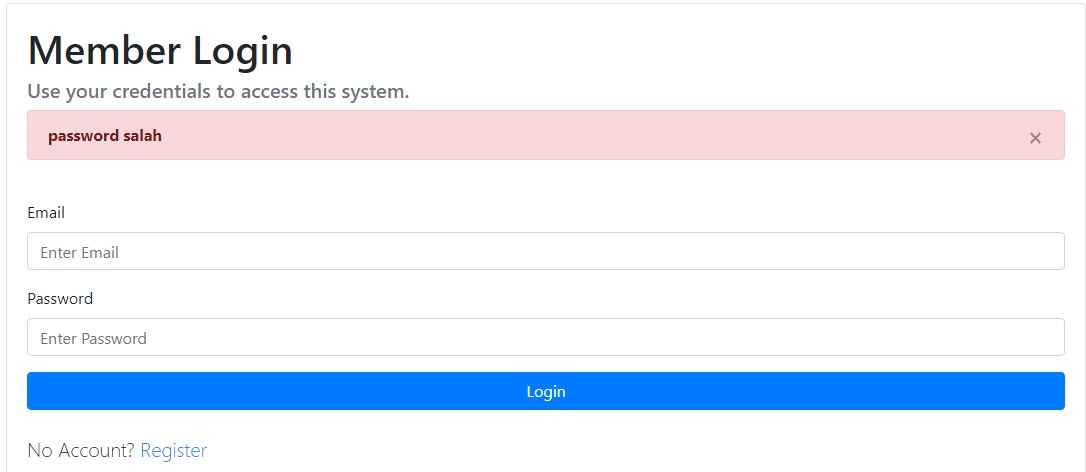
  res.locals.error\_msg = req.flash("error\_msg");

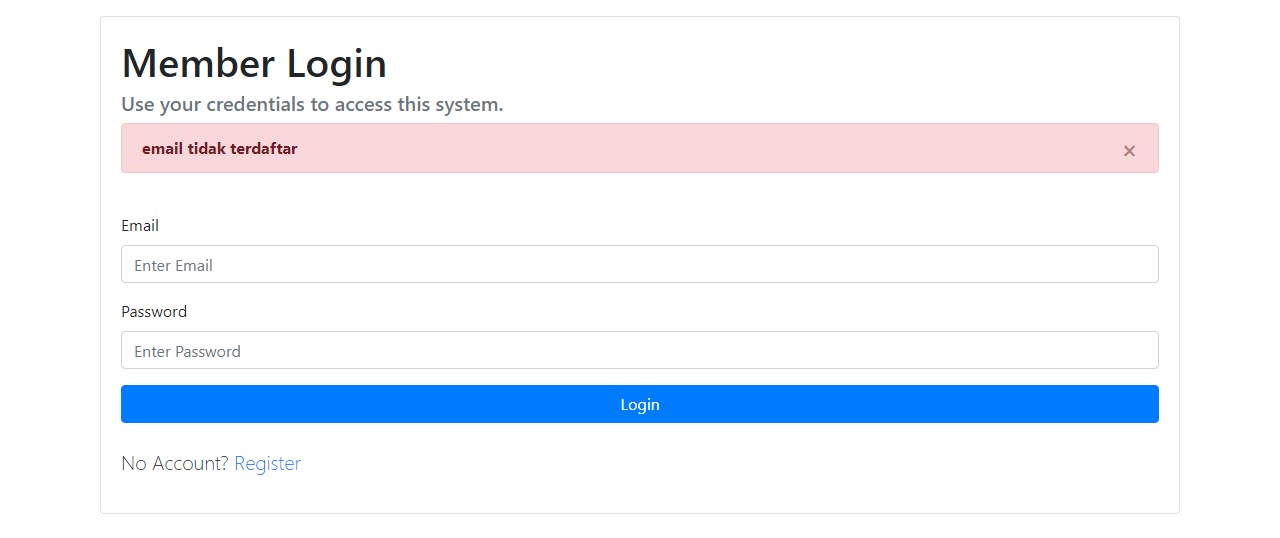
  res.locals.error = req.flash("error");

  next();

});

Maka akan muncul pesan error pada flash message seperti gambar berikut :





Dan jika berhasil maka akan dialihkan ke halaman dashboard. Sebelumnya Kita akan menambahkan halaman dashboard pada router agar dapat diakses di browser.

Pada file router/index.js tambahkan script dibawah ini.

const express = require("express");

const router = express.Router();

router.get("/", (req, res) => res.render("homepage"));

//dasboard

router.get("/dashboard", (req, res) =>

  res.render("dashboard")

);

module.exports = router;

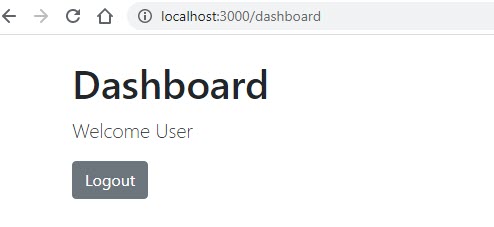
pada file views/dashboard.ejs tambahkan script sebagai berikut :

<h1 class="mt-4">Dashboard</h1>

<p class="lead mb-3">Welcome User</p>

<a href="/users/logout" class="btn btn-secondary">Logout</a>

Maka ketika dijalankan dan melakukan login dengan benar maka akan dialihkan ke halaman dashboard.



Selanjutnya Kita akan membuat fungsi logout pada file user.js

tambahkan pada bagian paling bawah sebelum script export pada file user.js

//logout handle

router.get("/logout", (req, res) => {

  req.logout();

  req.flash("success\_msg", "Anda berhasil Log out");

  res.redirect("/users/login");

});

module.exports = router;

saat ini Jika Kita mengakses langsung halaman dasboard tanpa login masih dapat dilakukan.

Untuk menangani hal tersebut diperlukan sebuah *ensureauthenticated* agar halaman dashboard hanya dapat diakses ketika Kita sudah melakukan login.

Buat sebuah file pada folder config dengan nama auth.js

isikan script dengan gambar diabawah ini :

module.exports = {

  ensureAuthenticated: function (req, res, next) {

    if (req.isAuthenticated()) {

      return next();

    }

    req.flash("error\_msg", "harap login untuk melihat halaman ini");

    res.redirect("/users/login");

  },

};

Kemudian pada file index.js tambahkan autentikasi ensureauthenticated dengan menambahkan script berikut :

const express = require("express");

const router = express.Router();

const { ensureAuthenticated } = require("../config/auth");

router.get("/", (req, res) => res.render("homepage"));

//dasboard

router.get("/dashboard", ensureAuthenticated, (req, res) =>

  res.render("dashboard", {

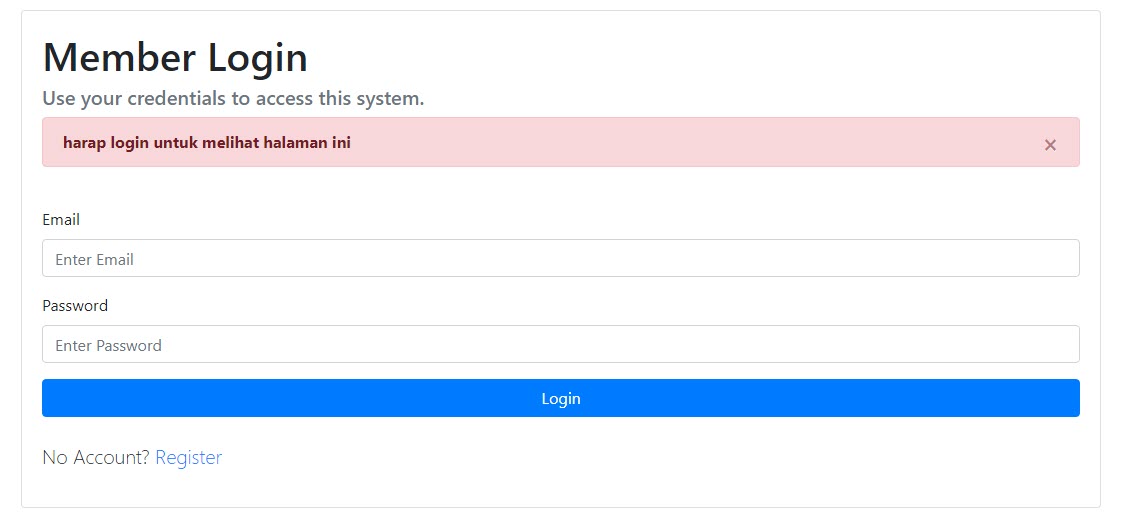
    name: req.user.name,

  })

);

module.exports = router;

fungsi ensureAuthenticated akan mengalihkan halaman yang diautentikasi ke halaman login jika belum ada autentikasi yang diterima. dan akan menampilkan informasi flash message error “harap login untuk melihat halaman ini”.



“name:req.user.name” akan digunakan pada dashboard.ejs untuk memberikan nama user pada setiap user yang berhasil login.

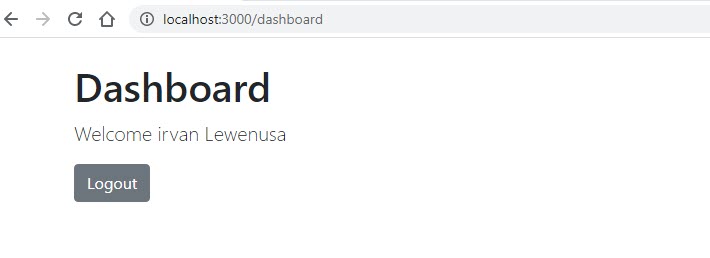
pada file dashboard.ejs tambahkan script sebagai berikut :

<h1 class="mt-4">Dashboard</h1>

<p class="lead mb-3">Welcome <%= name %></p>

<a href="/users/logout" class="btn btn-secondary">Logout</a>

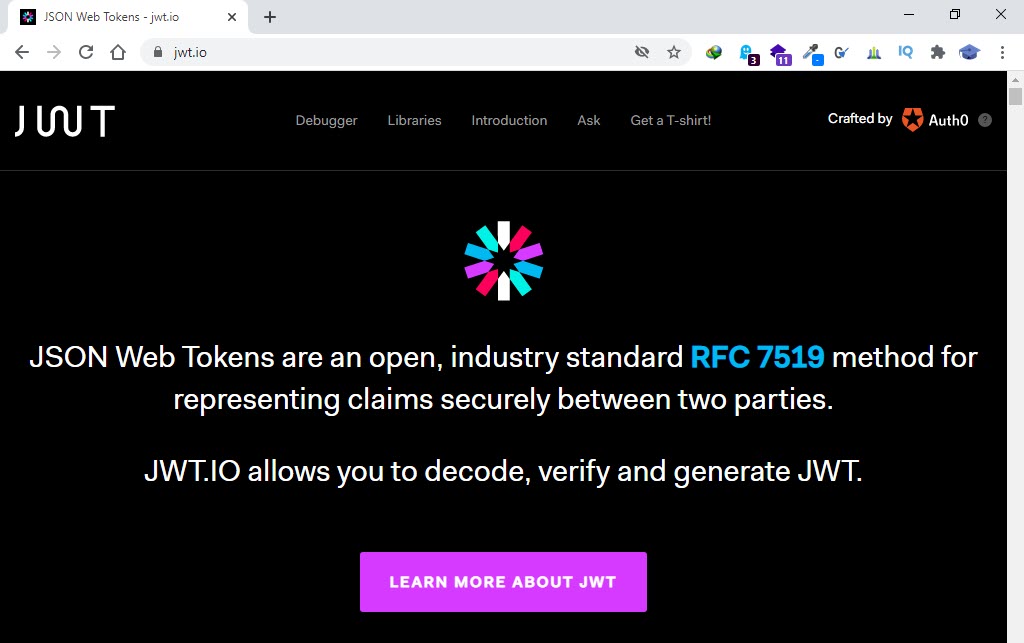
maka pada browser akan terlihat sebagai berikut.



# JsonWebToken

## Pendahuluan Apa itu JWT?

JSON Web Token atau lebih dikenal dengan JWT adalah sebuah token berbentuk string panjang yang sangat random yang gunanya sendiri untuk melakukan sistem **Autentikasi** dan **Pertukaran Informasi**. Umumnya untuk melakukan login tidak seperti pada aplikasi website biasa dimana kita menggunakan session untuk mengingat siapa yang sedang Login. Tapi didalam API sendiri kita menggunakan konsep JWT. Website resminya ada di **jwt.io.**

****

## Cara Kerja JWT

JWT atau Token ini seperti password jadi ketika users berhasil melakukan Login maka server akan memberikan sebuah Token. Nanti Token tersebut akan disimpan oleh users pada Local Storage atau Cookies Browser dan bila users ingin mengakses halaman halaman tertentu maka harus menyertakan token tersebut.  Untuk itu users akan mengirim balik token yang dikasih diawal tadi sebagai bukti bila user ini, sudah melakukan login. Sekarang kita akan lihat struktur dasarnya Tokennya dimana terdiri dari tiga bagian yaitu yang pertama **header** lalu kedua bagian **payloadnya** atau datanya dan yang ketiga adalah bagian **verify signature**.

* **Headernya** sendiri hanya terdiri dari Algoritma HS256 yang kita gunakan dan Typenya jwt sebagai defaultnya.

****

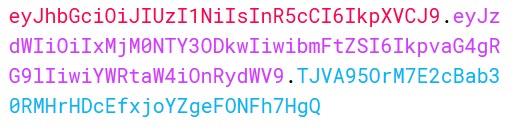
* **Payloadnya** sebagai infomasi atau data yang ingin kita kirimkan untuk users misalnya \_id usernya atau tanggal expirednya dan lain lain.



* **Verify Signature** adalah hasil dari Hash atau gabungan dari isi encode Header dan Payloadnya lalu ditambahkan kode secretnya.



Dan hasil ketiga bagian tersebut akan digabung dan otomatis di encode menjadi Token string random panjang seperti berikut :



Dan apabila isi Header atau Payload dirubah maka isi Signature menjadi tidak valid. Satu hal lagi JWT ini tidak tergantung sama bahasa program tertentu jadi kita bisa mengimplementasikan di Laravel Codeigniter Node JS dan yang lainnya.

## Konfigurasi JWT

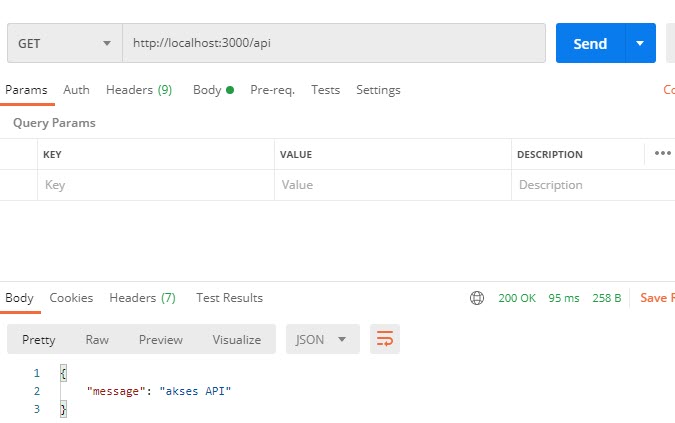
Pada bagian ini Kita akan mencoba membuat sebuah token autentikasi JWT untuk mengakses halaman tertentu pada bagian web yang sudah dibuat. Kita akan menggunakan aplikasi postman untuk mencoba menjalankan JWT.

langkah pertama Kita akan membuat sebuah api sederhana untuk dapat dijalankan pada aplikasi postman sebagai berikut :

* Lakukan perintah npm init untuk inisialisasi aplikasi nodejs kita dan isikan sesuai dengan kebutuhan. isikan index.js insialisasi menjadi app.js untuk menjalankan aplikasi pertama kali.
* Setelah selesai inisialisasi install dependensi modul yang dibutuhkan dengan perintah npm install nodemon express jsonwebtoken
* Buat file baru dengan nama “app.js” dan inputkan script dibawah untuk menjalankan aplikasi pada browser.



Jalankan nodemon pada terminal untuk menjalankan webserver dan inputkan URL <http://localhost:3000/api> dengan method GET pada aplikasi Postman



* Selanjutnya Kita akan membuat sebuah route untuk di proteksi yang hanya dapat dibuka dengan token yang dibuat menggunakan JWT.
* pada app.js tambahkan script berikut untuk membuat sebuah route baru.

app.post("/api/posts", (req, res) => {

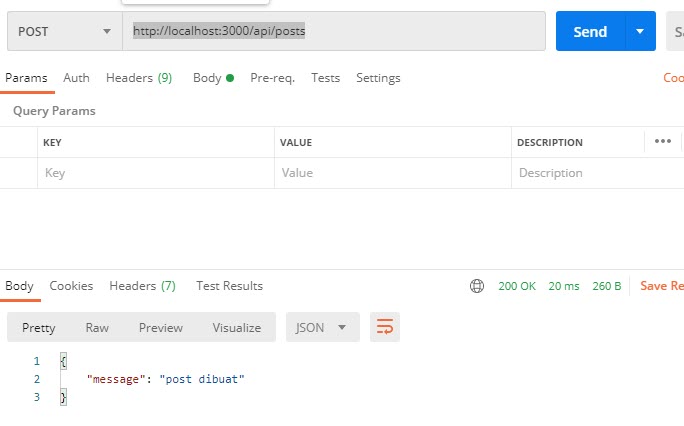
  res.json({

    message: "post dibuat",

  });

});

* Akses route pada postman dengan URL <http://localhost:3000/api/posts> Method POST



* Selanjutnya Kita akan mencoba mengimplementasikan JWT, sebelumnya buatlah sebuah route login untuk membuat token JWT.

app.post("/api/login", (req, res) => {

  //user data

  const user = {

    id: 1,

    username: "irvan",

    email: "irvan@belajarit.com",

  };

  jwt.sign({ user: user }, "rahasia", (err, token) => {

    res.json({

      token,

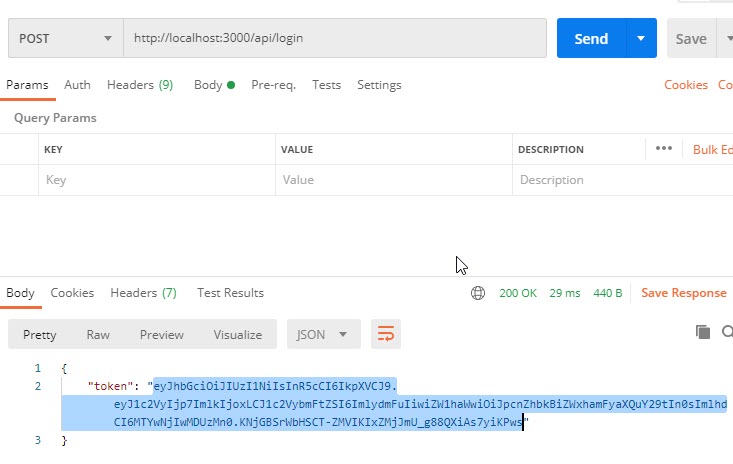
    });

  });

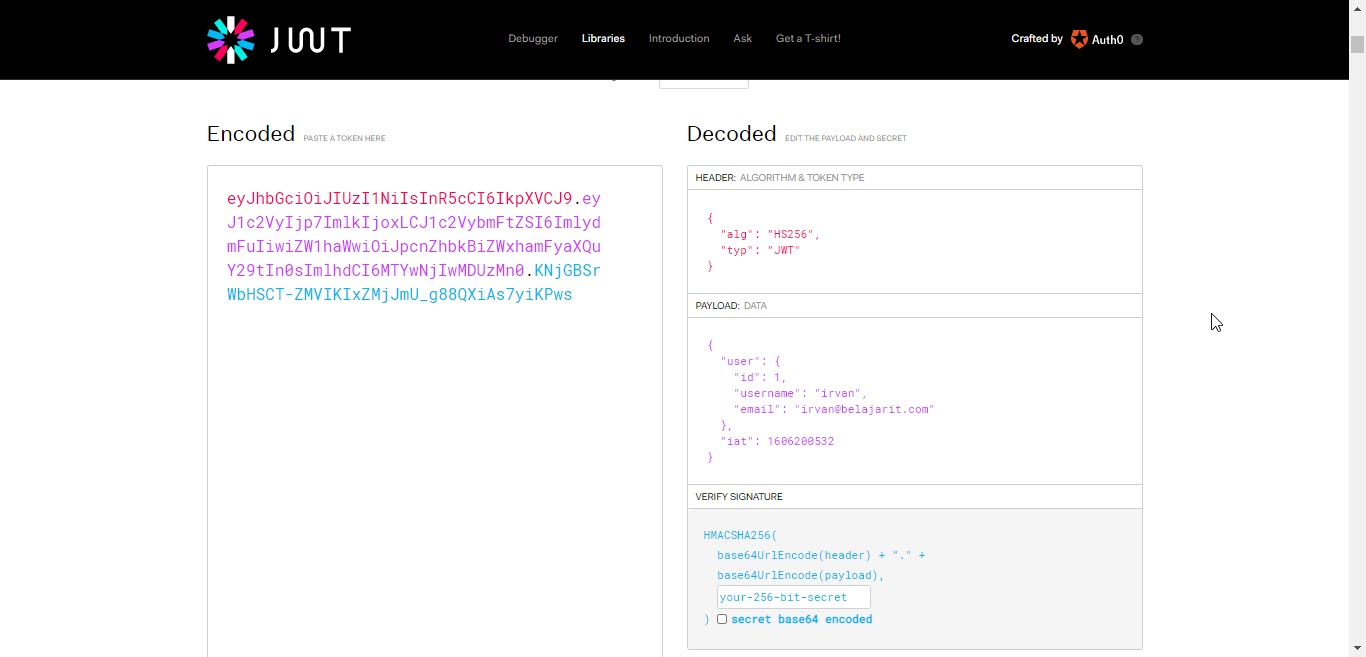
});

Kita akan menggunakan data user statis yaitu dengan const user yang sudah di tulis pada script diatas. Kemudian token akan dibuat dengan payload user yang sudah didefinisikan diatas.

Pada script diatas Kita menggunakan secretkeys : “rahasia”. jwt.sign akan membuat token yang siap digunakan.



* Kita dapat melihat isi dari token yang sudah dibuat pada web jwt.io debuger.



* Selanjutnya Kita akan membuat halaman yang akan diproteksi untuk dibukan menggunakan token JWT yang sudah dibuat.
* edit router.post (‘/api/posts’) dengan script dibawah ini :

app.post("/api/posts", verifyToken, (req, res) => {

  res.json({

    message: "post dibuat",

  });

});

verifyToken adalah sebuah fungsi yang akan Kita pakai untuk otentikasi.

* Tambahkan fungsi verifyToken pada bagian bawah file

//verifikasi fungsi

function verifyToken(req, res, next) {

  //get header

  const bearerHeader = req.headers["authorization"];

  if (typeof bearerHeader !== "undefined") {

  } else {

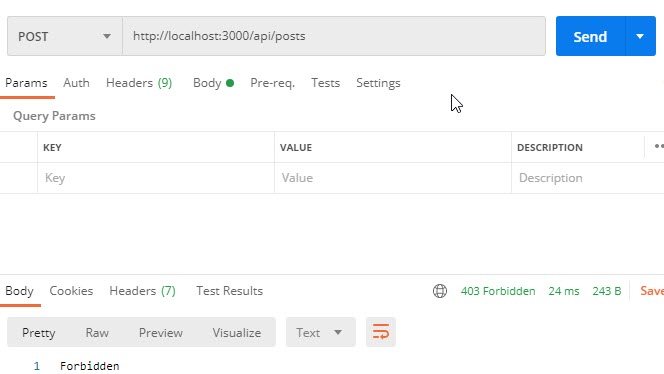
    //unauthorization

    res.sendStatus(403);

  }

}

Pada bagian ini Kita akan mencoba masuk url api/posts menggunakan postman dengan hasil status 403 (forbidden).



* Kemudian Kita akan membuat script lanjutan ketika halaman dapat diakses.

//verifikasi fungsi

function verifyToken(req, res, next) {

  //get header

  const bearerHeader = req.headers["authorization"];

  if (typeof bearerHeader !== "undefined") {

    // split header

    const bearer = bearerHeader.split(" ");

    //get token

    const bearerToken = bearer[1];

    //set token

    req.token = bearerToken;

    //next middleware

    next();

  } else {

    //unauthorization

    res.sendStatus(403);

  }

}

const bearer akan memotong kata pada bearerHeader kedalam sebuah array dipisahkan dengan tanda spasi (“ “).

Kemudian token yang ada pada index ke 2 array (bearer[1]) array index dimulai dari angka 0, berisi token yang sudah dibuat pada login api.

req.token = bearerToken akan menampung token yang akan digunakan pada fungsi selanjutnya (next()).

* Pada script api/posts edit scritp sebagai berikut :

app.post("/api/posts", verifyToken, (req, res) => {

  jwt.verify(req.token, "rahasia", (err, authData) => {

    if (err) {

      res.sendStatus(403);

    } else {

      res.json({

        message: "post dibuat",

        authData,

      });

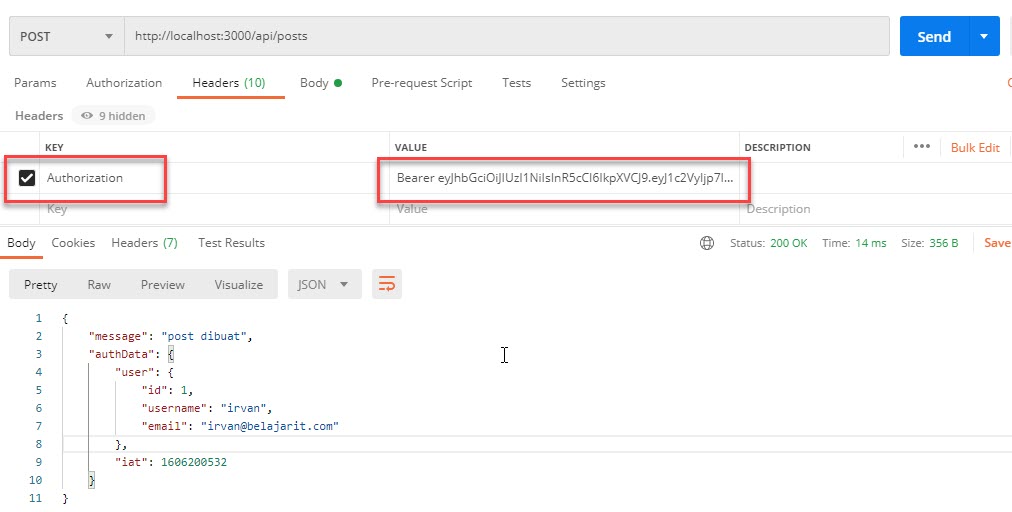
    }

  });

});

jwt.verify membutuhkan req.token yang didapat dari fungsi verifyToken sebelumnya, kemudian secretkeys yang digunakan pada saat jwt.sign yaitu “rahasia”.

Selanjutnya akan memanggil fungsi callback ketika status err maka akan mengirimkan status forbidden pada browser atau postman. Dan ketika status OK maka akan mengeluarkan pesan “post dibuat” dan authData.



Isian header dengan pilihan KEY **Authorization** dan value diawali dengan Kata “Bearer (spasi) token yang didapat”. Untuk dapat mengakses halaman api/posts.

* Kita juga dapat menambahkan waktu kadaluarsa pada token yang dibuat dengan menambahkan option expiresIn pada bagian jwt.sign

jwt.sign({ user: user }, "rahasia", { expiresIn: "7d" }, (err, token) =>

pada scritp diatas token akan kadaluarsa pada 7 hari kemudian.

## Kode Full JWT

const express = require("express");

const jwt = require("jsonwebtoken");

const app = express();

app.get("/api", (req, res) => {

  res.json({

    message: "akses API",

  });

});

app.post("/api/posts", verifyToken, (req, res) => {

  jwt.verify(req.token, "rahasia", (err, authData) => {

    if (err) {

      res.sendStatus(403);

    } else {

      res.json({

        message: "post dibuat",

        authData,

      });

    }

  });

});

app.post("/api/login", (req, res) => {

  //user data

  const user = {

    id: 1,

    username: "irvan",

    email: "irvan@belajarit.com",

  };

  jwt.sign({ user: user }, "rahasia", { expiresIn: "7d" }, (err, token) => {

    res.json({

      token,

    });

  });

});

//format token

//authorization : Bearer <access\_token>

//verifikasi fungsi

function verifyToken(req, res, next) {

  //get header

  const bearerHeader = req.headers["authorization"];

  if (typeof bearerHeader !== "undefined") {

    // split header

    const bearer = bearerHeader.split(" ");

    //get token

    const bearerToken = bearer[1];

    //set token

    req.token = bearerToken;

    //next middleware

    next();

  } else {

    //unauthorization

    res.sendStatus(403);

  }

}

app.listen(3000, () => console.log(`server berjalan pada port 3000`));