



Authentication & Authorization

Gold - chapter 6 - Topic 3

**Selamat datang di Chapter 6 Topik 3 online
course Fullstack Web dari Binar Academy!**

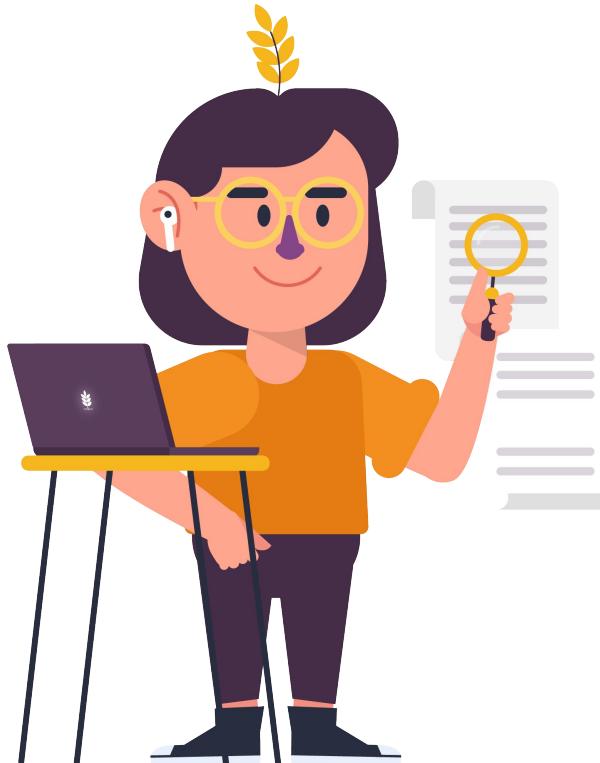




Buah kedondong buah markisa, selamat datang di Topik 3 ~

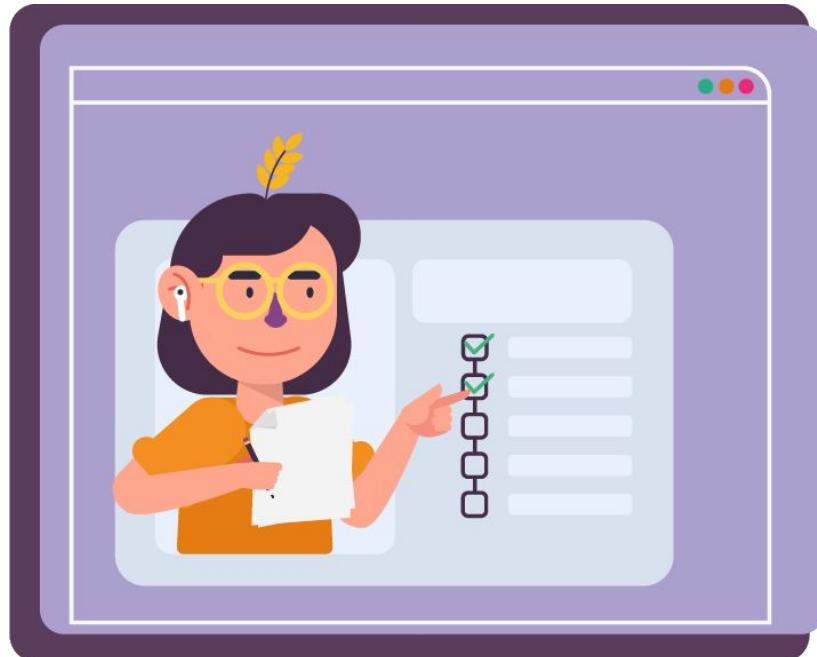
Di topik sebelumnya pada chapter ini, kamu sudah belajar mengenai design pattern beserta implementasinya. Selanjutnya kamu juga udah belajar proses asynchronus untuk memproses perintah di komputer.

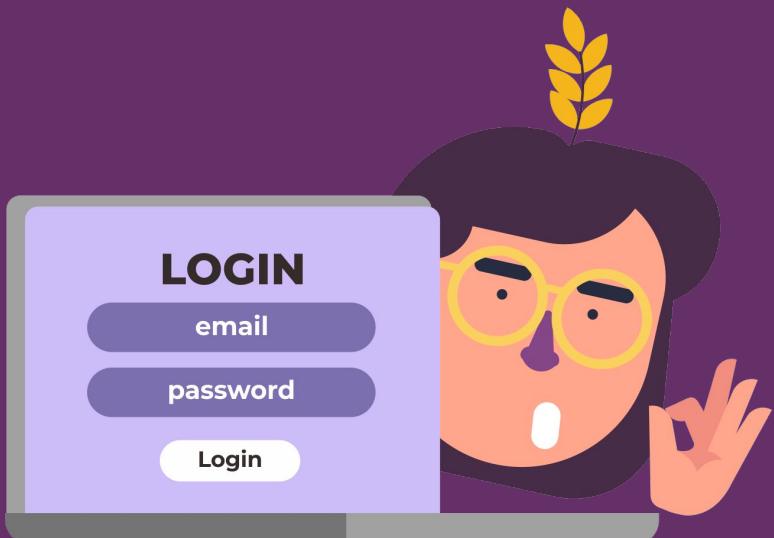
Nah, di topik ke 3 ini, kita akan bahas tentang **Authentication & Authorization** yang akan membantu kita memasuki laman web yang sudah dibuat, serta bagaimana cara mengimplementasikannya



Detailnya, kita bakal bahas hal-hal berikut ini:

- Mengenal konsep Authentication
- Menerapkan Authentication dengan menggunakan encryption
- Mengenal konsep Authorization
- Menerapkan Authentication dengan menggunakan Session Based Authentication
- Menerapkan Authentication dengan menggunakan Token Based Authentication (JWT)





Ketika kamu buka suatu website, tidak jarang kamu diminta untuk login dulu sebelum kamu mengakses lebih dalam web itu.

Hmmm... kira-kira gimana ya buatnya? Kenapa emang harus dibuat login dulu?

Nah pertanyaan-pertanyaan tadi akan kita jawab pada topik **authentication** dan **authorization** ini, yuk mariiii !



Authentication

Oke, apa sih authentication itu?

Kita coba pakai analogi sekolahnya ya! Apa yang membuat Sabrina bisa masuk ke dalam lingkungan sekolah Binar? Mostly, yang paling mudah adalah **karena Sabrina pakai seragam Binar dan juga punya kartu perlajar sekolah Binar.**

Jadi kalau ada satpam yang menjaga gerbang masuk sekolah Binar, dia akan mengizinkan atau tidak mengizinkan Sabrina masuk dengan melihat identitas dan juga seragamnya.

(Karena pasti satpam ga inget nama / wajah satu persatu siswa sekolah Binar kan? Hehe)





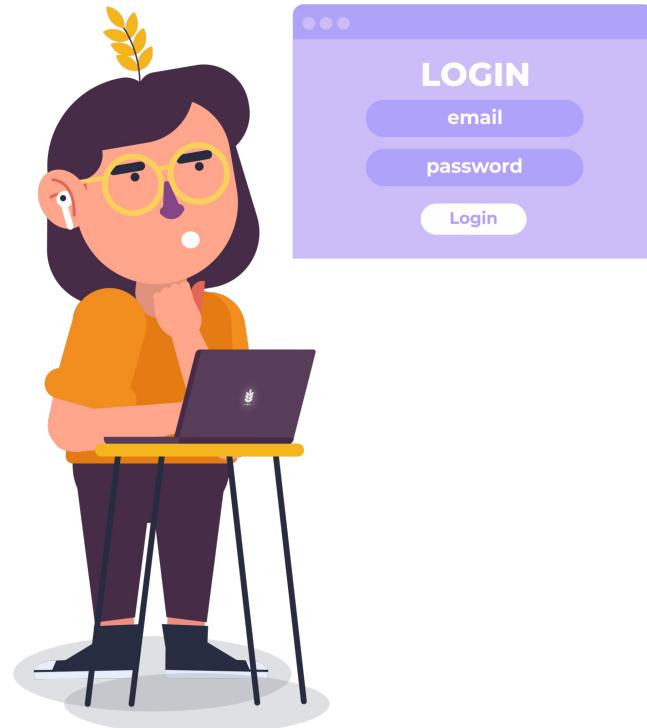
Proses pembuktian identitas inilah yang biasa disebut sebagai **Authentication**. Authentication membutuhkan satu elemen di dalamnya, yaitu credential. Dimana dari analogi sekolah tadi, credentialnya adalah seragam, kartu pelajar, dan lain-lain.





Nah... implikasinya dalam sebuah website, ketika kalian ingin melakukan beberapa aksi seperti komen, membuat sebuah postingan, dan mengupload video, website tersebut membutuhkan identitas dari user.

Salah satu cara paling efektif untuk **mendapatkan identitas dari user adalah dengan fitur authentication** ini. Biasanya fitur authentication ini merupakan proses berupa aksi **login** dan **register**.





Registration

Kalau tadi untuk melakukan authentication di sekolah bisa menggunakan "seragam" dan "kartu pelajar", maka pasti sebelumnya Sabrina perlu mendaftarkan diri menjadi siswa Binar untuk mendapatkan semua barang itu kan ya? Proses pendaftaran inilah yang biasa disebut sebagai Registration.

Jadi, registration itu merupakan **sebuah proses pendaftaran diri sebagai seorang user di dalam sebuah website.**





Sebagai contoh, ketika kalian melakukan **registrasi di facebook, kalian sebenarnya sedang mendaftarkan diri sebagai user di dalam facebook. Nah data user ini akan disimpan ke dalam sebuah server milik facebook**, sama seperti kalian mendaftarkan diri ke dalam sebuah sekolah, nama kalian akan tercatat di dalam database sekolah tersebut.

Biasanya ada 2 informasi yang kita berikan untuk proses pendaftaran ini, yaitu :

- Email
- Password





Dua informasi yang sudah kita berikan saat registrasi akan digunakan kembali ketika kita sudah logout dari sebuah website dan ingin login lagi.

Kok bisa ya web ini inget siapa aja yang sudah registrasi?

Soalnya, data-data registrasi akan disimpan dalam database website, dan database itu bisa diakses kapanpun oleh pemilik website.





Lah kalau database bisa diakses oleh owner website jadi bocor dong datanya?

Makanya tidak bijak kalau kita menyimpan **password di dalam database sebagai plain text aja. Data password harus disimpan dalam bentuk encrypted message.**

Artinya, siapapun yang punya akses ke database tersebut tidak akan tahu password asli dari pemilik akun, melainkan mereka hanya bisa melihat bentuk password tersebut dalam bentuk encrypted message.





Kode disamping ini adalah salah satu contoh implementasi fitur login yang ditulis di dalam Express.Js dalam bentuk REST API.

```
● ● ●

function encryptPassword(password) {
  // Pura-puranya ngeenkripsi password
  return password;
}

app.post("/api/v1/register", async (req, res) => {
  const email = req.body.email;
  const encryptedPassword = await encryptPassword(req.body.password);
  const user = await User.create({ email,
    encryptedPassword });
  res.status(201).json({
    id: user.id,
    email: user.email,
    createdAt: user.createdAt,
    updatedAt: user.updatedAt,
  });
})
```



Login

Okay, setelah tadi Sabrina melakukan registrasi / pendaftaran di sekolah Binar, Sabrina sudah bisa masuk sekolah. Dengan catatan, Sabrina harus menggunakan seragam dan membawa kartu pelajar.

Kalau Sabrina tidak menggunakan kedua identitas itu maka Sabrina akan dilarang untuk memasuki sekolah Blnar. Karena seragam tersebut adalah cara membuktikan bahwa Sabrina adalah siswa Binar. Seragam inilah yang biasa kita sebut sebagai credentials.



Credentials

Di dalam sebuah website, credentials tersebut adalah email dan password. Dua informasi ini akan di-input oleh user sebagai pembuktian bahwa user yang sedang menggunakan website tersebut adalah user yang benar-benar memiliki akun tersebut.

Proses login ini diperlukan ketika sebuah akses di dalam website memerlukan identitas dari user



Sebagai contoh, Sabrina ingin meminjam buku di perpustakaan Binar, tentu saja Sabrina harus masuk terlebih dahulu ke dalam sekolah Binar menggunakan seragam dan pergi perpustakaan. Nah, ketika Sabrina ingin meminjam buku, maka petugas perpustakaan yang melayani Sabrina akan meminta Kartu Pelajar Sabrina sebagai pembuktian bahwa Sabrina ini benar murid dari Binar dan yang hendak meminjam buku tersebut benar-benar Sabrina.





Berikut contoh kode yang mengimplementasikan fitur login menggunakan express.js dalam bentuk REST API.

```
● ● ●

function checkPassword(encryptedPassword, password) {
  // Pura-puranya ngecompare
  // password yang dienkripsi,
  // sama password yang dikirim user
  return encryptedPassword === password;
}

app.post("/api/v1/login", async (req, res) => {
  const email = req.body.email.toLowerCase(); // Biar case insensitive
  const password = req.body.password;

  const user = await User.findOne({
    where: { email }
  });

  if (!user) {
    res.status(404).json({ message: "Email tidak ditemukan" })
    return;
  }

  const isPasswordCorrect = await checkPassword(
    user.encryptedPassword,
    password
  );

  if (!isPasswordCorrect) {
    res.status(401).json({ message: "Password salah!" })
    return;
  }

  res.status(201).json({
    id: user.id,
    email: user.email,
    token: "iniToken", // Kita bakal ngomongin ini lagi nanti.
    createdAt: user.createdAt,
    updatedAt: user.updatedAt,
  })
})
```



Encryption

Seperti yang sudah disebutkan di bagian registration tadi, Encryption sangat penting dalam proses authentication. Dimana proses ini **bertujuan untuk menyembunyikan nilai asli dari sebuah data**, dalam kasus authentication data yang disembunyikan adalah data password.

Kita tidak ingin menyimpan data password secara mentah-mentah, karena akan sangat bahaya dan melanggar privasi.

Maka dari itu, **kita perlu simpan passwordnya dalam bentuk yang jauh dari bentuk aslinya, tapi tetap dapat dibaca oleh komputer ketika kita ingin melakukan pengecekan**. Data yang sudah dienkripsi biasanya disebut sebagai hash.





Nah, untuk membuat hash, ada banyak algoritma yang bisa kita gunakan, yaitu:

- MD5 (Udah kuno, jangan dipakai)
- SHA256 (Recommended)
- Bcrypt (Recommended and popular for password hashing)
- dan sebagai-nya

Nah, dalam materi ini, kita hanya akan membahas **bcrypt** saja, dan algoritma ini sudah cukup untuk kalian implementasikan dalam website yang professional.

Yuk kita langsung latihan untuk implementasinya ~





1. Buatlah project Node.Js pada umumnya

Seperti biasa, karena kita membuat website menggunakan express.js, maka dari itu kita perlu membuat project Node.js.



```
npm init -y
```



2. Install library bernama bcryptjs

Nah, kita tidak akan mengimplementasikan algoritma bcrypt dari nol kok, udah ada library yang bisa kita pakai dalam melakukan hashing password, yaitu `bcryptjs`



```
npm install bcryptjs
```



3. Buat fungsi encrypt dan checkPassword

Nah, contoh kode sebelumnya tidak mengimplementasikan proses encryption dalam penyimpanan dan komparasi password pada endpoint login dan register, maka dari itu, kode disamping terdapat 2 fungsi yang akan digunakan pada dua endpoint tersebut. Lihat dokumentasi [bcryptjs](#).

```
const bcrypt = require("bcryptjs");
const salt = 10;

function encryptPassword(password) {
  return new Promise((resolve, reject) => {
    bcrypt.hash(password, salt, (err, encryptedPassword) => {
      if (!!err) {
        reject(err);
        return;
      }

      resolve(encryptedPassword);
    })
  })
}

function checkPassword(encryptedPassword, password) {
  return new Promise((resolve, reject) => {
    bcrypt.compare(
      password,
      encryptedPassword,
      (err, isPasswordCorrect) => {
        if (!!err) {
          reject(err);
          return;
        }

        resolve(isPasswordCorrect);
      }
    )
  })
}
```



Hasil dari kode diatas dapat dilihat di [repository ini](#) ya teman-teman, jangan lupa di cek!

Nah, fitur authentication ini tidak bisa berjalan dengan sendirinya, fitur authentication ini harus diikuti dengan fitur pendamping, yaitu authorization. Apa itu authorization?





Authorization



Nah, ternyata fitur authentication yang tadi sudah kita pelajari tidak bisa berjalan dengan sendirinya, fitur authentication ini harus diikuti dengan fitur pendamping, yaitu **authorization**.

Wah, benda apa itu bestie?



Authorization

Authorization adalah sebuah **proses perijinan untuk melakukan sebuah aksi**.

Sebagai contoh, ketika Sabrina terdaftar di sekolah Binar, pastinya Sabrina akan memiliki Kartu Pelajar Binar. Nah, kartu pelajar ini digunakan dalam proses administrasi di dalam sekolah. Misalnya, untuk meminjam buku di perpustakaan. Tentu saja Sabrina harus menunjukkan kartu pelajar untuk mendapat perijinan meminjam buku perpustakaan.

Sama seperti website, ketika kalian ingin mengakses sesuatu yang membutuhkan identitas, kalian akan menunjukkan sesuatu ke server yang membuktikan bahwa yang melakukan aksi tersebut benar-benar kalian (orang yang tepat).





Sama seperti website, ketika kalian ingin mengakses sesuatu yang membutuhkan identitas, kalian akan menunjukan sesuatu ke server yang membuktikan bahwa yang melakukan aksi tersebut benar-benar kalian (orang yang tepat).



Penggunaan authorization ini akan sangat bergantung dengan tipe authentication yang kita gunakan. Jadi tipe authentication yang akan kita bahas pada materi kali ini ada 2, yaitu:

1. **Session Based Authentication**
2. **Token Based Authentication (JWT)**

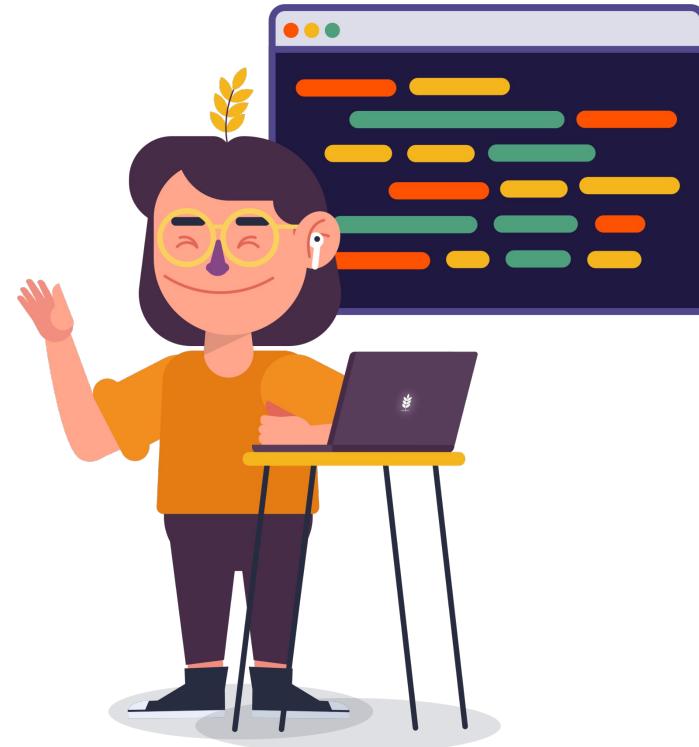




1. Session Based Authentication

Tipe authentication ini akan sangat **cocok digunakan ketika kita membuat sebuah website yang mengandalkan View Engine untuk membuat UI nya.** Dimana Frontend dan Backend-nya akan berada satu server yang sama.

Karena FE dan BE akan berada di satu server yang sama, maka dari itu kita bisa **mengandalkan Cookie dan Session untuk menyimpan data dari client (Browser).** Nah, salah satu kegunaannya adalah untuk Session Based Authentication.





Hmmm... jadi apa sih session based authentication itu?

Basically, session based **authentication** adalah **authentication yang status login atau tidaknya seorang user akan disimpan dan diingat oleh server menggunakan session**. Bagaimana server bisa tau session A milik user mana? Server akan membaca Cookie yang dikirim melalui request, yang mana ketika user berhasil login, server juga akan memodifikasi Cookie user untuk menyimpan yang namanya Session ID.

Cookie/Session Based Authentication





Ketika kita menggunakan session based authentication, di saat user berhasil login, maka server akan menyimpan informasi tersebut di dalam server. Maka dari itu request berikutnya server akan langsung mengetahui bahwa siapa yang mengakses server kita dari segi data user.

Untuk mengimplementasikan session based authentication, kita bisa menggunakan salah satu dari dua library berikut:

[Express Session](#)

[Passport.js](#)

Tapi, pada materi ini, kita hanya akan membahas express-session saja.





Ayo mulai implementasinya !

Untuk mulai belajar menggunakan express-session,
gunakan [repository berikut](#). Silahkan clone, dan lakukan
instalasi **express-session** dan **ejs**.



```
npm install express-session ejs
```



Lalu modifikasi file **app/index.js**, untuk melakukan setup view engine, dan initialisasi **express-session**.

```
/** Install Body Parser */
app.use(express.urlencoded({ extended: true }))

/** Install view engine */
app.set("views", viewsDir);
app.set('view engine')

/** Install express-session */
app.set('trust proxy', 1) // trust first proxy
app.use(session({
  secret: process.env.COOKIE_SECRET || 'Rahasia',
  resave: false,
  saveUninitialized: false
}))
```



Buatlah halaman Home dan Login

Pastikan halaman home, mengakses informasi user, seperti **Halo, \${Nama User}!**, dan juga untuk halaman login, hanya dapat diakses oleh user yang belum login saja. Namun, kita tidak perlu menghiraukan hal tersebut terlebih dahulu, yang penting kita bikin view-nya dulu aja.



```
<h1>Halo, <%= user.email %>!</h1>
```



Buatlah halaman Home dan Login

Pastikan halaman home, mengakses informasi user, seperti **Halo, \${Nama User}!**, dan juga untuk halaman login, hanya dapat diakses oleh user yang belum login saja. Namun, kita tidak perlu menghiraukan hal tersebut terlebih dahulu, yang penting kita bikin view-nya dulu aja.

Setelah membuat kedua view tersebut, sekarang kita tinggal implementasikan endpoint-nya.



app/views/index.ejs

```
<h1>Halo, <%= user.email %>!</h1>
```



app/views/login.ejs

```
<form method="POST"
action="/login">
  <input type="email" name="email"
placeholder="Enter your email">
  <input type="password"
name="password" placeholder="Enter
your password">
  <input type="submit"
value="Login">
</form>
```



Membuat endpoint login dan route untuk halaman Home dan Login

Implementasi loginnya sama dengan implementasi yang ada bagian **encryption**, bedanya, ketika kita berhasil login, kita akan menggunakan utility dari **express-session**, untuk membuat data session.

```
● ● ● app/controllers/mainController.js

module.exports = {
  index(req, res) {
    res.render("index.ejs", {
      user: req.session.user,
    })
  },
  login(req, res) {
    res.render("login.ejs")
  },
}
```

```
● ● ● app/controllers/authController.js

module.exports = {
  async login(req, res) {
    const email = req.body.email.toLowerCase(); // Biar case insensitive
    const password = req.body.password;

    const user = await User.findOne({
      where: { email },
    });

    if (!user) {
      res.status(404).send("Email tidak ditemukan");
      return;
    }

    const isPasswordCorrect = await checkPassword(
      user.encryptedPassword,
      password
    );

    if (!isPasswordCorrect) {
      res.status(401).send("Password salah!");
      return;
    }

    req.session.isAuthenticated = true;
    req.session.user = user;
    res.redirect("/");
  },
  async logout(req, res) {
    req.session.destroy()
    res.redirect("/login")
  },
};
```



config/routes.js

```
const appRouter = express.Router();

appRouter.get("/", controllers.mainController.index)
appRouter.get("/login", controllers.mainController.login)
appRouter.post("/login", controllers.authController.login)
appRouter.post("/logout", controllers.authController.logout)

module.exports = appRouter;
```



Oh iya, karena halaman home membutuhkan user untuk login terlebih dahulu, maka dari itu kita perlu menambahkan middleware untuk mencegah user yang belum login mengakses halaman home. Middleware tersebut akan berbentuk seperti ini.

```
● ● ● app/controllers/authController.js

module.exports = {
  ... authController,
  authorizedOnly(req, res, next) {
    if (!req.session.isAuthenticated) {
      res.redirect("/login");
      return;
    }

    next();
  },
  publicOnly(req, res, next) {
    if (!req.session.isAuthenticated) {
      next();
      return;
    }

    res.redirect("/");
  }
}
```



Lalu, kita append kedua fungsi tersebut ke routes.

```
● ● ● app/controllers/authController.js

const appRouter = express.Router();

appRouter.get("/", controllers.authController.authorizedOnly,
  controllers.mainController.index)
appRouter.get("/login", controllers.authController.publicOnly,
  controllers.mainController.login)
appRouter.post("/login", controllers.authController.publicOnly,
  controllers.authController.login)
appRouter.get("/logout", controllers.authController.authorizedOnly,
  controllers.authController.logout)

module.exports = appRouter;
```

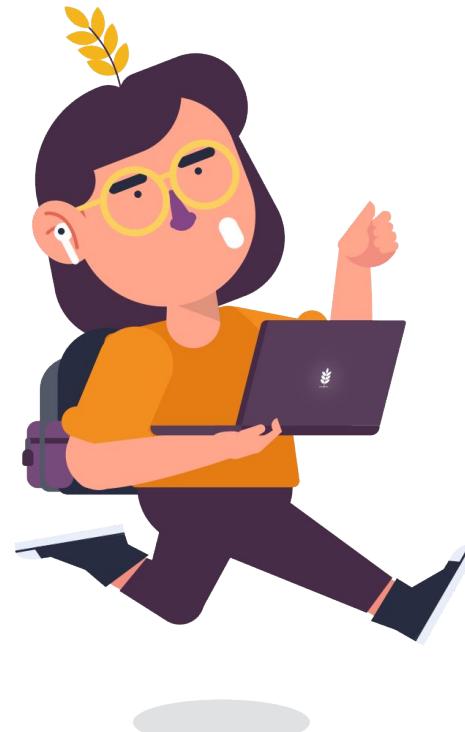


Source code dari session based authentication pada materi ini, dapat kalian lihat [disini](#).

Ada satu latihan lagi yang perlu kalian coba, yaitu mengimplementasi CRUD suatu resource, sebagai contoh Todo List, yang mana owner dari resource tersebut adalah user yang saat ini sedang aktif di dalam sessionnya, lalu prevent user lain menghapus todo list dari user lainnya.

Selamat mengeksplor ~

Sekarang kita lihat apa bedanya dengan Token Based Authentication (JWT), capcussss!





2. Token Based Authentication (JWT)

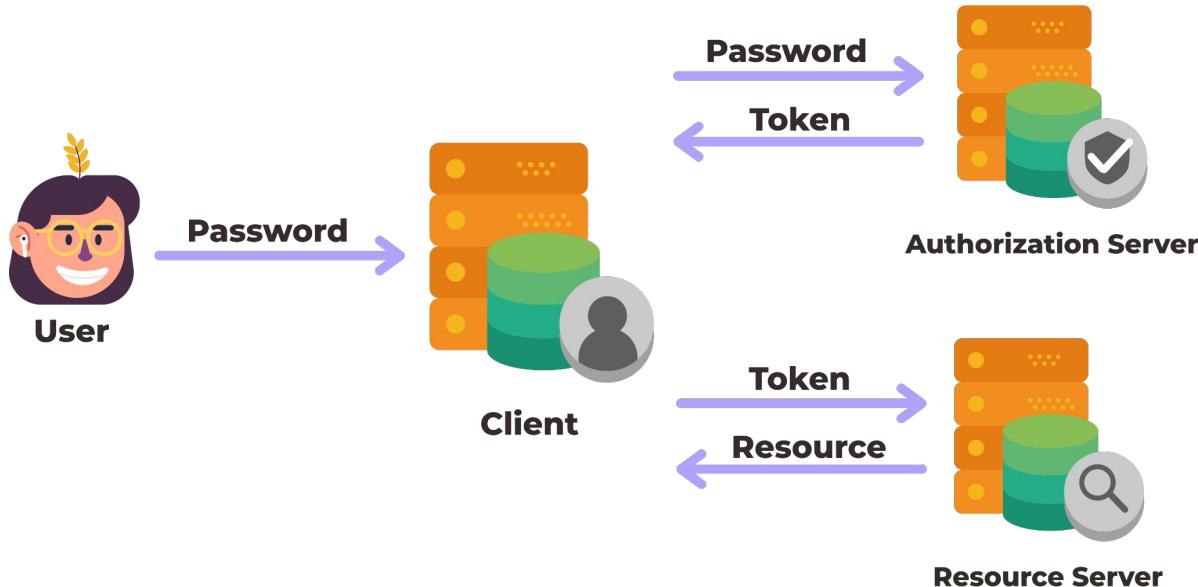
Berbeda dengan session based authentication, token based authentication ini **digunakan khusus untuk RESTful API dan tipe API stateless lainnya.**

Token Based Authentication ini ibarat naik kereta. Kalau Sabrina mau naik kereta dia pasti membeli tiket terlebih dahulu agar dapat naik kereta. Nah tiket ini yang biasa kita sebut sebagai credential, dalam konteks Authentication.

Dimana etika naik kereta, kita perlu menuju ke stasiun terlebih dahulu, dan petugas akan meminta tiket yang sudah kita beli untuk ditukar dengan boarding pass. Boarding pass inilah yang kita sebut sebagai Token, karena dengan Boarding Pass, kita dapat mengakses fasilitas-fasilitas yang tersedia untuk penumpang, entah itu di Kereta, maupun di Stasiun.



Salah satu jenis **token yang umum digunakan dalam pengembangan website adalah JSON Web Token (JWT dibaca Jot).** User akan mendapatkan JWT ini ketika mereka berhasil login, dan karena JWT ini hanya digunakan di dalam REST-like API, maka dari itu, token tersebut akan disimpan oleh Client Side Application untuk digunakan di request berikutnya.





Kalau kamu mau tahu contoh JSON Web Token, kurang lebih kaya gambar disamping ini..

Biasanya, token ini akan dikirim oleh client melalui Request Header bernama Authorization. Yang mana, dalam pengiriman token, ada banyak jenisnya, tapi disini kita hanya akan bahas standar Bearer Token saja.

Yaitu ketika kita mengirim token di dalam request, token tersebut disimpan di request header bernama **Authorization**, dengan prefix **Bearer** diikuti dengan token.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJd  
WIi0iIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9  
lIiwiWF0IjoxNTE2MjM5MDIyfQ.SflKxwRJSMeKKF  
2QT4fwpMeJf36P0k6yJV_adQssw5c
```



Kode di samping adalah contoh request yang menggunakan Token Based Authentication.

Oke, tanpa basa-basi lagi, langsung kita coba implementasikan saja. Gunakan [repository ini](#) sebagai dasar dari implementasinya.

Oh iya, sebenarnya ada jalan pintas buat implementasi ini lho! Yaitu dengan menggunakan Passport.js, tapi cenderung terlalu abstrak. Jadi untuk pertama kali, dianjurkan untuk mengikuti tutorial di slide setelah ini ya ~

```
curl -X GET \
-H 'Authorization: Bearer
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9 ..
.' \
http://localhost:8080/api/v1/user/profile
```



Ayo kita mulai implementasi !

Pertama-tama, setelah clone repository tadi, kalian harus **instal beberapa dependency** yang dibutuhkan, yaitu cuma jsonwebtoken, karena bcrypt dan lain-lainnya udah ada di repo tadi.



```
npm install jsonwebtoken
```



Modifikasi endpoint Login untuk merespon dengan Token di dalamnya

Untuk melakukan hal ini, buka file **app/controllers/api/v1/authController.js** dan modifikasi bagian login menjadi seperti kode di bawah ini.

Perhatikan fungsi `createToken`, di dalam fungsi tersebut kita melemparkan sesuatu yang bernama payload. **Payload ini adalah data yang akan dienkripsi bersamaan dengan token**, jadi ketika token tersebut di-decode, data payload yang kita masukkan akan bisa terbaca dan digunakan untuk kebutuhan request. Payload token biasanya berisi informasi user.

```
app/controllers/api/v1/authController.js

const jwt = require("jsonwebtoken");

async function login(req, res) {
  const email = req.body.email.toLowerCase(); // Biar case insensitive
  const password = req.body.password;

  const user = await User.findOne({
    where: { email },
  });

  if (!user) {
    res.status(404).json({ message: "Email tidak ditemukan" });
    return;
  }

  const isPasswordCorrect = await checkPassword(
    user.encryptedPassword,
    password
  );

  if (!isPasswordCorrect) {
    res.status(401).json({ message: "Password salah!" });
    return;
  }

  const token = createToken({
    id: user.id,
    email: user.email,
    createdAt: user.createdAt,
    updatedAt: user.updatedAt,
  });

  res.status(201).json({
    id: user.id,
    email: user.email,
    token,
    createdAt: user.createdAt,
    updatedAt: user.updatedAt,
  });
}
```



Setelah endpoint tersebut termodifikasi, ketika kalian melakukan request ke endpoint **Login**, dan berhasil, akan mendapatkan response seperti ini.

Nah, nilai dari token tersebut dapat disimpan, dan digunakan pada request berikutnya, namun sebelum membuat endpoint yang mengakses data user, kita perlu membuat middleware untuk memverifikasi token tersebut valid atau tidak, layaknya session, token hanya merepresentasikan satu user saja.

```
● ● ●  
{  
  "id": 1,  
  "email": "sabrina@binar.co.id",  
  "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9... ",  
  "createdAt": "2022-04-19T02:24:02.179Z",  
  "updatedAt": "2022-04-19T02:24:02.179Z"  
}
```



Membuat middleware Authorize

Middleware ini digunakan untuk memvalidasi token, apakah token tersebut valid dan merepresentasikan user di dalam website kita.

Agar dapat dilakukan test middleware ini, kita perlu satu endpoint yang berguna untuk mengeluarkan data dari user berdasarkan token. Maka dari itu, kita bisa modifikasi file **app/controllers/api/v1/authController.js**, untuk penambahan fungsi baru.

```
● ● ● app/controllers/api/v1/authCor  
  
async function whoAmI(req, res) {  
  res.status(200).json(req.user);  
}
```



Lalu, kita bisa menambahkan satu fungsi lagi di file yang sama, yang akan kita gunakan sebagai middleware.

Middleware disamping digunakan untuk **mengekstrak token dari request header dari key Authorization**, dan setelah itu, token divalidasi menggunakan jwt.verify yang mana jika tokennya valid, kita akan mendapatkan payload yang sebelumnya kita masukkan waktu login tadi.

Artinya kita punya akses ke data user sebagai Object. Maka dari itu, kita perlu memvalidasinya sekali lagi dengan melakukan query ke tabel user, dan memasukkan hasilnya ke request object, agar dapat diakses fungsi berikutnya pada endpoint tersebut.

```
● ● ● app/controllers/api/v1/authController.js

async function authorize(req, res, next) {
  try {
    const bearerToken =
req.headers.authorization;
    const token =
bearerToken.split("Bearer ")[1];
    const tokenPayload =
jwt.verify(token,
process.env.JWT_SIGNATURE_KEY ||
"Rahasia")

    req.user = await
User.findByPk(tokenPayload.id);
    next();
  }

  catch(err) {
    res.status(401).json({
      message: "Unauthorized",
    })
  }
}
```



Setelah middleware-nya jadi, kita bisa langsung menempelkan middleware dan request handler ke dalam route.

```
config/routes.js

const express = require("express");
const controllers = require("../app/controllers");

const apiRouter = express.Router();

/**
 * Authentication Resource
 */
apiRouter.get("/api/v1/whoami",
  controllers.api.v1.authController.authorize,
  controllers.api.v1.authController.whoAmI)
apiRouter.post("/api/v1/login",
  controllers.api.v1.authController.login);
apiRouter.post("/api/v1/register",
  controllers.api.v1.authController.register);

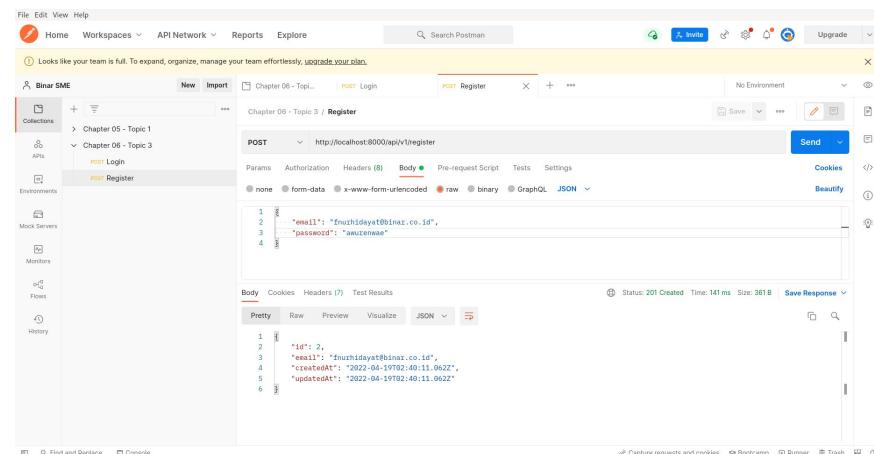
apiRouter.use(controllers.api.main.onLost);
apiRouter.use(controllers.api.main.onError);

module.exports = apiRouter;
```



Testing dengan Postman

Setelah semuanya terpasang dan servernya menyala, kita bisa **gunakan Postman untuk mengetest kedua fungsi tersebut**. Yang pertama harus kita lakukan adalah, melakukan **registrasi**, yaitu dengan melakukan request ke endpoint **POST /api/v1/register**.



The screenshot shows the Postman application interface. On the left, there's a sidebar with 'Collections' (Binar SME, Chapter 05 - Topic 1, Chapter 06 - Topic 3), 'APIs', 'Environments', 'Mock Servers', 'Monitors', 'Flows', and 'History'. In the main area, a 'Register' API is selected under 'Chapter 06 - Topic 3'. The 'Body' tab is active, showing a JSON payload:

```
1 {  
2   "email": "fruithidayat@binar.co.id",  
3   "password": "auzenemas"  
4 }
```

Below the body, the response status is 'Status: 201 Created' with a 'Save Response' button. The response body is also shown in JSON format:

```
1 {  
2   "id": 2,  
3   "email": "fruithidayat@binar.co.id",  
4   "createdAt": "2022-04-19T02:40:11.062Z",  
5   "updatedAt": "2022-04-19T02:40:11.062Z"  
6 }
```

Setelah berhasil melakukan registrasi, kita bisa mencoba untuk login. Dengan melakukan request ke **POST /api/v1/login**.

The screenshot shows the Postman application interface. The left sidebar includes sections for Home, Workspaces, API Network, Reports, and Explore. A notification at the top left says "Looks like your team is full. To expand, organize, manage your team effortlessly, upgrade your plan." The main workspace shows a collection named "Binar SME" containing two chapters: "Chapter 05 - Topic 1" and "Chapter 06 - Topic 3". Under "Chapter 06 - Topic 3", there are two requests: "Login" and "Register". The "Login" request is selected and is a POST request to "http://localhost:8000/api/v1/login". The "Body" tab is active, showing a JSON payload:

```
1 {  
2   "email": "fnuhidayat@binar.co.id",  
3   "password": "amurenae"  
4 }
```

The response status is "201 Created", with a response time of "92 ms" and a size of "637 B". The response body is a JSON object:

```
1 {  
2   "id": 2,  
3   "email": "fnuhidayat@binar.co.id",  
4   "token": "eyJhbGciOiJIUzI1NiJ9.eyJhdWxkX2F1dGhvcml0eT9pZC1sInYxZW52aR80dC1oIiI6IjM0Qm1tU2d1N0AqM1tMDYw5i1sLWaZGF9ZWR8c16IeyJpZCIjDQ1xixL2h1a01JehnyNxYyLhdB1aMhc15jby5pZC1sInYxZW52aR80dC1oIiI6IjM0Qm1tU2d1N0AqM1tMDYw5i1sLWaZGF9ZWR8c16I",  
5   "createdAt": "2022-04-19T02:40:11.062Z",  
6   "updatedAt": "2022-04-19T02:40:11.062Z"  
7 }
```



Setelah berhasil login, copy token tadi, dan gunakan untuk request berikutnya, yaitu **GET /api/v1/whoami**, untuk mendapatkan current user.

The screenshot shows the Postman application interface. On the left, there's a sidebar with sections like Collections, APIs, Environments, Mock Servers, Monitors, Flows, and History. The main area shows a collection named 'Chapter 05 - Topic 1' with a sub-section 'Chapter 06 - Topic 3'. Under 'Chapter 06 - Topic 3', there are three requests: 'POST Login', 'POST Register', and 'GET Whoami'. The 'GET Whoami' request is selected. The request details panel shows the method as 'GET', the URL as 'http://localhost:8000/api/v1/whoami', and the 'Headers' tab is active, containing an 'Authorization' header with the value 'Bearer eyJhbGciOiJIUzI1NiisInR5cCI6IkpXvCj0eyJpZ...'. Below the headers, the 'Body' tab shows a JSON response:

```
1 {  
2   "id": 3,  
3   "email": "fnuzhizdayat@binar.co.id",  
4   "encryptedPassword": "$2a$10$ewkxZrWfMT98.1K14hxFe9f9ZLvvzudybQjY48HCJVAu4oku94k",  
5   "createdAt": "2022-04-19T02:40:11.062Z",  
6   "updatedAt": "2022-04-19T02:40:11.062Z"  
7 }
```

The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 39 ms'. There are also buttons for 'Save Response', 'Find and Replace', 'Console', and other Postman-specific icons.



Yes ! Sudah selesai pembahasan Token Based Authentication. Semua kode pada tutorial ini dapat dilihat [disini](#). Untuk lebih mengasah kemampuan kamu bisa coba implementasikan Todo List yang mana tiap entry todo list tersebut akan memiliki atribut userId, yang diambil berdasarkan Authorization yang dilakukan. Dan juga, setiap user hanya dapat melihat todo list mereka saja.

Good luck!

Referensi buat tambahan kamu belajar ~

- <https://www.section.io/engineering-education/session-management-in-nodejs-using-expressjs-and-express-session/>
- <https://medium.com/weekly-webtips/how-to-create-a-simple-login-functionality-in-express-5274c44c20df>
- <https://www.youtube.com/watch?v=oYGhoHW7zqI>



Gimana nih sudah lebih paham belum tentang **Authentication** dan **Authorization**?

Kalau sudah yuk lanjut ke topik berikutnya tentang **Open API** ~

Bukan API yang panas ko, tapi API terkait database. Gaassss~



Terima Kasih!



Next Topic

loading...