Authentication

Menggunakan Bcrypt untuk Meng-hashing Password

- Menyimpan password dengan aman sangat penting untuk keamanan aplikasi.
- Bcrypt adalah sebuah pustaka yang populer untuk meng-hashing password dengan aman.

Mengapa Password Harus Aman?

- Password adalah informasi yang sangat sensitif.
- Menyimpan password dalam bentuk teks di database berisiko.
- Jika terjadi kebocoran database, password dapat dibaca oleh orang lain.
- Meng-hashing password membuatnya tidak dapat dibaca dan lebih aman.

Bcrypt

- Bcrypt adalah pustaka untuk meng-hashing password dengan aman.
- Menggunakan fungsi hash kriptografis satu arah.
- Memperlambat serangan brute-force dan rainbow table.

Menggunakan Bcrypt

Install Bcrypt dan Dotenv

```
npm install bcrypt dotenv
```

.env

```
BCRYPT_ROUND=10
```

Bcrypt Rounds

- Dalam peng-hashingan password menggunakan bcrypt, istilah "rounds" merujuk pada jumlah iterasi yang digunakan untuk meng-hash sebuah password.
- Bcrypt sengaja dirancang untuk memperlambat serangan brute-force dan rainbow table.
- Semakin tinggi jumlah rounds, semakin aman, tetapi juga lebih lambat proses peng-hashingan.

Mengapa Bcrypt Rounds Penting

- Bcrypt menggunakan faktor kerja (work factor) untuk menentukan jumlah rounds.
- Faktor kerja ini menentukan biaya komputasi dari peng-hashingan.
- Lebih banyak rounds meningkatkan keamanan, tetapi juga memerlukan lebih banyak waktu.
- Bcrypt rounds dapat disesuaikan berdasarkan kebutuhan keamanan Anda.

Mengatur Bcrypt Rounds

- Bcrypt rounds biasanya diatur ketika menghasilkan password yang di-hash.
- Jumlah rounds yang direkomendasikan adalah 12 hingga 14.

Menggunakan Bcrypt

Meng-hashing Password

```
const bcrypt = require("bcrypt")
require("dotenv").config()

const bcryptRound = process.env.BCRYPT_ROUND || 10
const password = "passwordAnda"

bcrypt.hash(password, bcryptRound, (err, hash) => {
  if (err) {
    // Tangani kesalahan
  }
  // Simpan 'hash' di database
})
```

Menggunakan Bcrypt

Membandingkan Password

```
const password = "passwordAnda"

// Ambil 'hash' dari database
bcrypt.compare(password, hash, (err, result) => {
  if (err) {
    // Tangani kesalahan
  }
  if (result) {
    // Password benar
  } else {
    // Password salah
  }
})
```

Express.js + Sequelize + Bcrypt

```
├── app
│   ├── console
│   ├── controllers
│   ├── middlewares
│   ├── models
│   ├── router.js
│   ├── support
│   └── validators
├── app.js
├── config
│   ├── auth.js
│   └── database.js
├── database
│   ├── migrations
│   └── seeders
├── nodemon.json
├── package.json
├── package-lock.json
└── server.js
```

Menggunakan Bcrypt dengan Sequelize

```
// app/models/user.js

User.beforeCreate((user, options) => {
  return bcrypt
    .hash(user.password, bcryptRound)
    .then((hash) => {
      user.password = hash
    })
    .catch((err) => {
      throw new Error()
    })
})
```

Authentication

- Authentication adalah proses untuk memverifikasi identitas pengguna.
- Biasanya dilakukan dengan membandingkan username dan password.
- Jika username dan password benar, pengguna dianggap otentik.
- Jika salah satu atau keduanya salah, pengguna dianggap tidak otentik.

REST API Authentication

- Mengamankan REST API Anda adalah sangat penting untuk melindungi data sensitif.
- Salah satu cara untuk mengamankan API Anda adalah dengan otentikasi berbasis token.
- Otentikasi berbasis token menggunakan token untuk identifikasi dan otentikasi pengguna.
- Server menghasilkan token unik untuk pengguna yang sudah diautentikasi, dan klien mengirimkan token ini bersama setiap permintaan.
- Token dapat dienkripsi dalam format Base64 untuk kesederhanaan.

Menggunakan Token dalam API

Langkah 1: Membuat Token

Setelah pengguna login, buat token unik untuk pengguna.

Langkah 2: Kirim Token ke Klien

Kirimkan token ke klien dan simpan dengan aman.

Langkah 3: Sertakan Token dalam Permintaan

Untuk setiap permintaan API, sertakan token dalam header permintaan.

Langkah 4: Verifikasi Token di Server

Di server, verifikasi keaslian token dan identitas pengguna.

Authentication

```
// app/controllers/auth.js
router.post("/", async (req, res) => {
  const { email, password } = req.body
  const user = await User.findOne({
    where: { email: email },
  })

  if (!user) {
    res.status(422).send({ error: "Invalid credentials" })
    return
  }

  const validPassword = await bcrypt.compare(password, user.password)
  if (!validPassword) {
    res.status(422).send({ error: "Invalid credentials" })
    return
  }

  const token = Buffer.from(randomString.generate()).toString("base64")
  await Token.create({ userId: user.id, token: token })

  res.send({ token: token })
})
```

Authentication

Generate token dengan `randomstring`:

```
npm install randomstring
```

```
const randomString = require("randomstring")

const token = Buffer.from(randomString.generate()).toString("base64")
```

Authentication: Middleware

```
// app/middlewares/token-auth.js
const tokenAuth = async (req, res, next) => {
  const authorizationToken = req.headers["authorization"]

  if (!authorizationToken) {
    res.status(401).send({ error: "No token provided" })
    return
  }

  const userToken = await Token.findOne({ where: { token: authorizationToken } })
  if (!userToken) {
    res.status(401).send({ error: "Invalid token" })
    return
  }

  const user = await User.findByPk(userToken.userId)
  req.user = user.toJSON()

  next()
}
```

Authentication: Middleware

```
// app/controllers/books.js
const express = require("express")
const tokenAuth = require("../middlewares/token-auth")
const router = express.Router()

router.use(tokenAuth)

router.get("/", async (req, res) => {
  // periksa data pengguna
  const user = await req.user
  console.log(user)

  res.send({ message: "Hello World from protected routes!" })
})
```

Advance: Separation of Concerns

Hash Class

```
// app/support/Hash.js

const bcrypt = require("bcrypt")
const auth = require("../../config/auth")

class Hash {
  static async create(password) {
    try {
      const hash = await bcrypt.hash(password, parseInt(auth.bcryptRound))
      return hash
    } catch (error) {
      throw error
    }
  }

  static async check(password, hash) {
    try {
      const isMatch = await bcrypt.compare(password, hash)
      return isMatch
    } catch (error) {
      throw error
    }
  }
}

module.exports = Hash
```

Hash Class Usage

```
// app/models/user.js
const Hash = require("../support/Hash")

User.beforeCreate(async (user, options) => {
  return Hash.create(user.password).then((hash) => {
    user.password = hash
  })
})
```

Hash Class Usage

```
// app/controllers/auth.js
const Hash = require("../support/Hash")

router.post("/", new AuthValidator().validate(), async (req, res) => {
  const { email, password } = req.body
  const user = await User.scope("withPassword").findOne({
    where: { email: email },
  })

  if (!user) {
    res.status(422).send({ error: "Invalid credentials" })
    return
  }

  if (!Hash.check(password, user.password)) {
    res.status(422).send({ error: "Invalid credentials" })
    return
  }

  const token = Buffer.from(randomString.generate()).toString("base64")
  await Token.create({ userId: user.id, token: token })

  res.send({ token: token })
})
```

TokenHandler Class

```
const { Token } = require("../../models")

class TokenHandler {
  constructor(tokenStrategy) {
    this.strategy = tokenStrategy
  }

  async create(user) {
    const token = this.strategy.create()
    await Token.create({ userId: user.id, token: token })
    return token
  }

  verify(token) {
    return this.strategy.verify(token)
  }
}
```

TokenHandler Class

```
const { User, Token } = require("../../models")
const randomString = require("randomstring")

class SimpleToken {
  create() {
    const token = Buffer.from(randomString.generate()).toString("base64")
    return token
  }

  async verify(authorizationToken) {
    const token = await Token.findOne({ where: { token: authorizationToken } })
    if (!token) {
      return null
    }

    const user = await User.findByPk(token.userId)

    return user
  }
}
```

TokenHandler Class

```
const jwt = require("jsonwebtoken")
const auth = require("../../../config/auth")
const { Buffer } = require("buffer")

class JWT {
  constructor(user) {
    this.user = user
  }

  create() {
    const data = Buffer.from(`${this.user.id}:${new Date().getTime()}`).toString("base64")
    const token = jwt.sign(data, auth.key)
    return token
  }

  verify(authorizationToken) {
    try {
      const decoded = jwt.verify(authorizationToken, auth.key)
      return decoded
    } catch (error) {
      return null // Token is invalid
    }
  }
}
```