

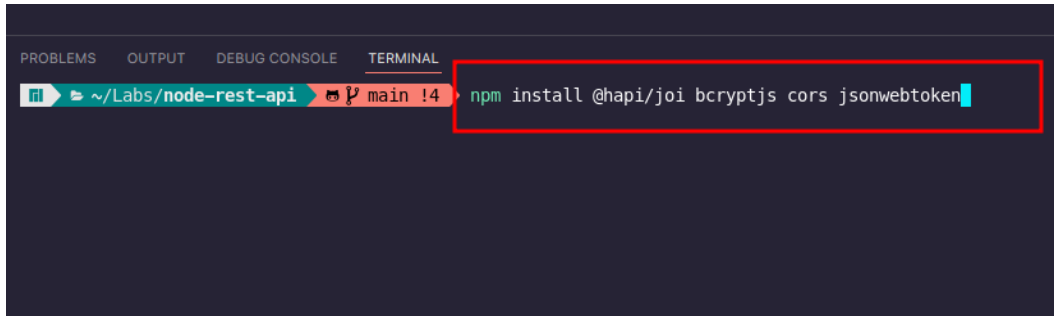


PCC CLASS

Membuat RESTful API dengan Node.js dan MongoDB

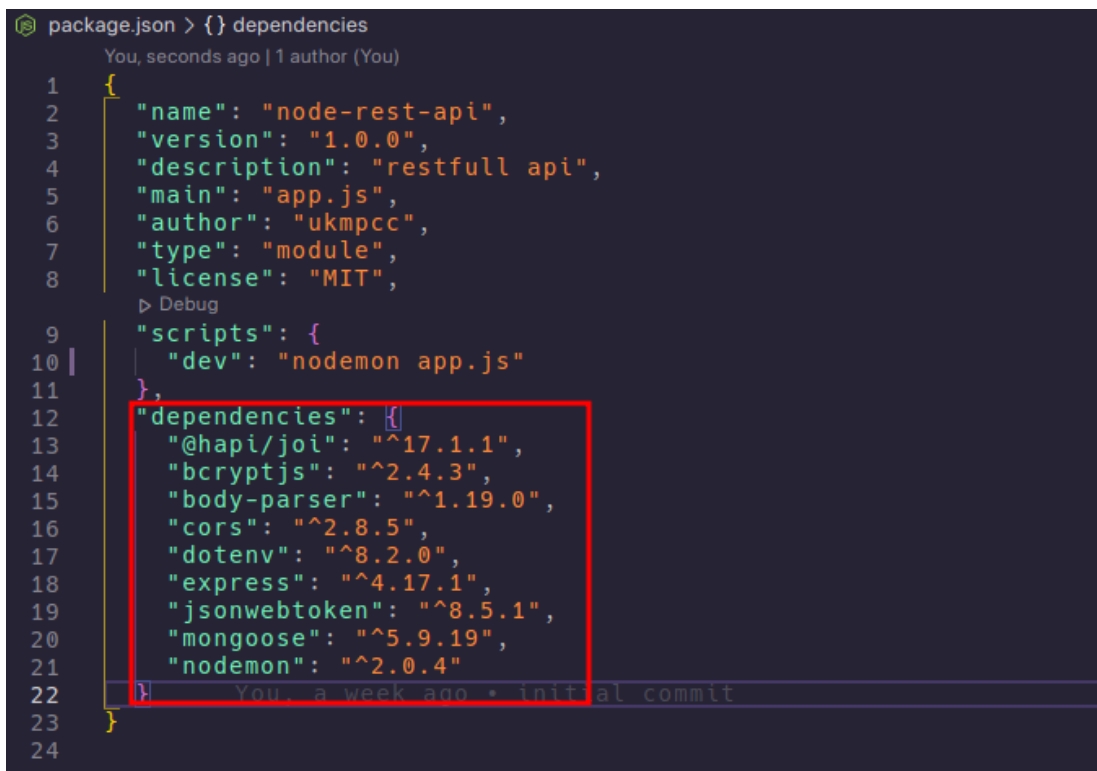


1. Buka folder bernama rest api yang telah dibuat sebelumnya di visual studio code
2. Buka terminal
3. Install beberapa dependency yang digunakan dengan mengetikkan perintah
npm install @hapi/joi bcryptjs cors jsonwebtoken



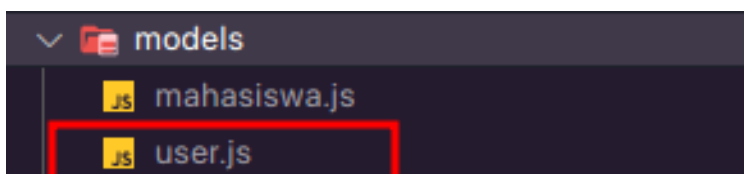
```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
~/Labs/node-rest-api main !4 npm install @hapi/joi bcryptjs cors jsonwebtoken
```

4. Jika sukses diinstall, maka pada file package.json akan menjadi seperti berikut



```
package.json > {} dependencies
You, seconds ago | 1 author (You)
1 {
2   "name": "node-rest-api",
3   "version": "1.0.0",
4   "description": "restfull api",
5   "main": "app.js",
6   "author": "ukmpcc",
7   "type": "module",
8   "license": "MIT",
9   "scripts": {
10    "dev": "nodemon app.js"
11  },
12  "dependencies": {
13    "@hapi/joi": "^17.1.1",
14    "bcryptjs": "^2.4.3",
15    "body-parser": "^1.19.0",
16    "cors": "^2.8.5",
17    "dotenv": "^8.2.0",
18    "express": "^4.17.1",
19    "jsonwebtoken": "^8.5.1",
20    "mongoose": "^5.9.19",
21    "nodemon": "^2.0.4"
22  }
23 }
24 You, a week ago • initial commit
```

5. Selanjutnya, buat file bernama **user.js** pada folder **models** untuk membuat schema collection user



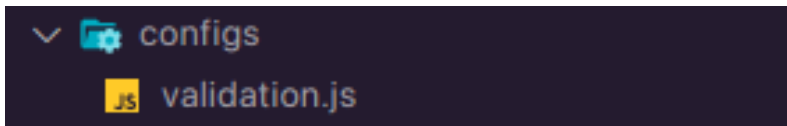
6. Ketik kode berikut pada file **user.js**

```
You, 5 days ago | 1 author (You)
import mongoose from "mongoose";

const userSchema = mongoose.Schema({
  username: {
    type: String,
    required: true,
    max: 255,
  },
  email: {
    type: String,
    required: true,
    max: 100,
  },
  password: {
    type: String,
    required: true,
    min: 8,
    max: 1024,
  },
  createdAt: {
    type: Date,
    default: Date.now,
  },
});

export default mongoose.model("User", userSchema);
```

7. Buat folder baru bernama **configs** dan didalamnya buat file bernama **validation.js**



8. Ketik kode berikut pada file **validation.js** untuk validasi input register dan login

```
import Joi from "@hapi/joi";

export const registerValidation = (data) => {
  const schema = Joi.object({
    username: Joi.string().required(),
    email: Joi.string().email().required(),
    password: Joi.string().min(8).required(),
  });

  return schema.validate(data);
};

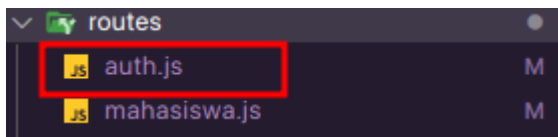
export const loginValidation = (data) => {
  const schema = Joi.object({
    email: Joi.string().email().required(),
    password: Joi.string().min(6).required(),
  });

  return schema.validate(data);
};
```

register

login

9. Buat file bernama **auth.js** pada folder **routes**



10. kemudian import beberapa packages, schema dan validation yang digunakan dengan menulis kode berikut

```
// import packages
import express from "express";
import bcrypt from "bcryptjs/dist/bcrypt.js";
import jwt from "jsonwebtoken";

//import validation
import { registerValidation } from "../configs/validation.js";

// import models
import User from "../models/user.js";

// inialisasi router express
const router = express.Router();
```

11. Buat endpoint baru menggunakan router dengan method **post** dan endpoint **/register** dan ketikkan kode berikut

```
// Register
router.post("/register", async (req, res) => {
  // validation input
  const { error } = registerValidation(req.body);
  if (error) {
    return res.status(400).json({
      status: res.statusCode,
      message: error.details[0].message,
    });
  }

  // if email exist
  const emailExist = await User.findOne({ email: req.body.email });
  if (emailExist) {
    return res.status(400).json({
      status: res.statusCode,
      message: "Email Sudah digunakan !",
    });
  }

  // Hash Password
  const salt = await bcrypt.genSalt(10);
  const hashPassword = await bcrypt.hash(req.body.password, salt);

  const user = new User({
    username: req.body.username,
    email: req.body.email,
    password: hashPassword,
  });

  //create user
  try {
    const saveUser = await user.save();
    res.json(saveUser);
  } catch (err) {
    res.status(400).json({
      status: res.statusCode,
      message: "Gagal Membuat user baru",
    });
  }
});
```

12. Export route tadi menggunakan perintah berikut

```

//create user
try {
  const saveUser = await user.save();
  res.json(saveUser);
} catch (err) {
  res.status(400).json({
    status: res.statusCode,
    message: "Gagal Membuat user baru",
  });
}
});

export default router;

```

13. Kemudian import file **auth.js** yang sudah dibuat sebelumnya ke dalam file **app.js**

```

// Import Routes
import mahasiswaRoutes from "../routes/mahasiswa.js";
import userRoutes from "../routes/auth.js";

```

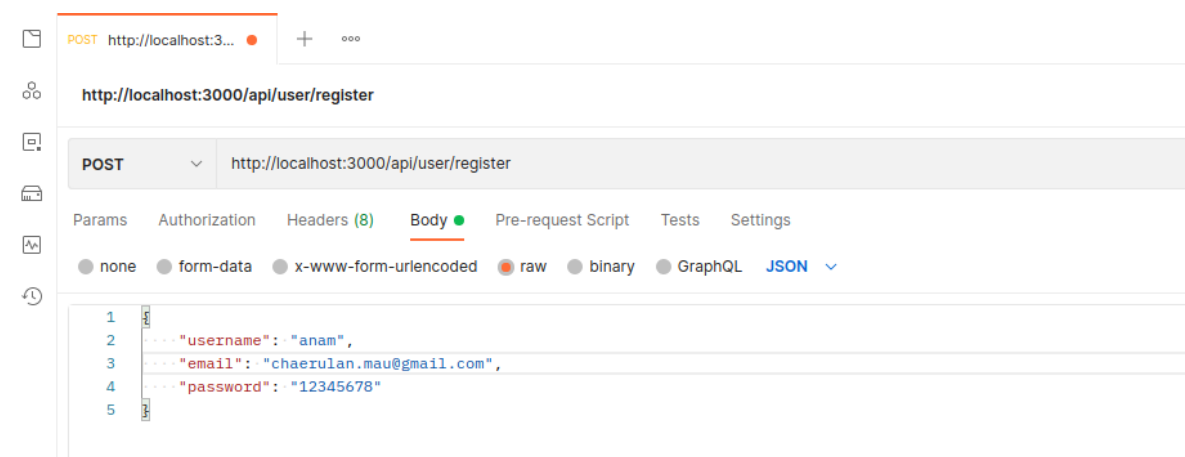
14. Lalu tambahkan route **auth.js** ke dalam file app.js menggunakan kode berikut

```

// routes example
app.use("/api/mahasiswa", mahasiswaRoutes);
app.use("/api/user", userRoutes);

```

15. Selanjutnya kita akan melakukan ujicoba untuk melakukan register dengan menggunakan postman



16. Jika sukses akan menerima respond berikut



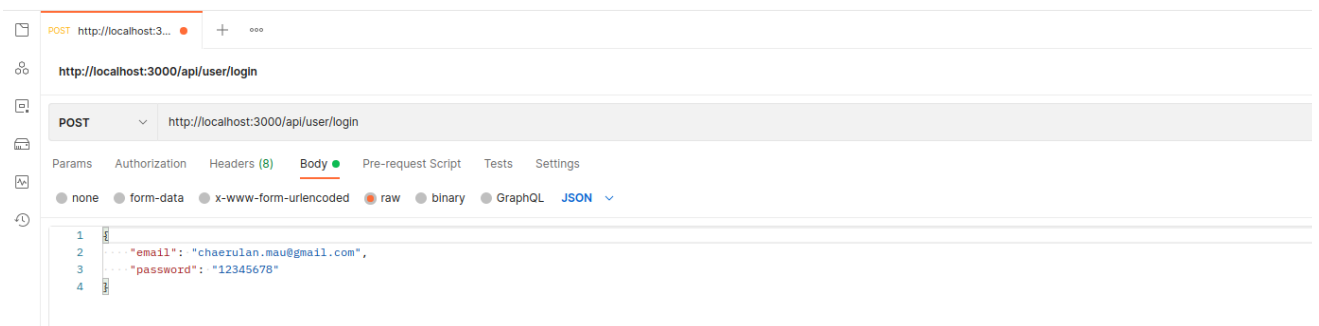
17. Buat endpoint baru menggunakan router dengan method **post** dan endpoint **/login** dan ketikkan kode berikut dibawah endpoint **/register**

```
// Login
router.post("/login", async (req, res) => {
  // if email exist
  const user = await User.findOne({ email: req.body.email });
  if (!user)
    return res.status(400).json({
      status: res.statusCode,
      message: "Email Anda Salah!",
    });

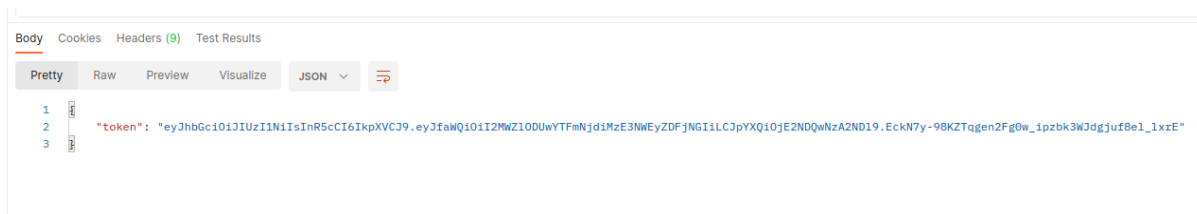
  // check password
  const validPwd = await bcrypt.compare(req.body.password, user.password);
  if (!validPwd)
    return res.status(400).json({
      status: res.statusCode,
      message: "Password Anda Salah!",
    });

  // membuat token menggunakan JWT
  const token = jwt.sign({ _id: user._id }, process.env.SECRET_KEY);
  res.header("auth-token", token).json({
    token: token,
  });
});
```

18. Selanjutnya kita akan melakukan ujicoba untuk melakukan login dengan menggunakan postman



Jika sukses, maka akan mendapatkan respon berupa token, yang nantinya akan kita gunakan untuk mengamankan api kita, sehingga hanya yang memiliki token saja yang bisa mengakses api kita



19. Selanjutnya kita akan membuat **verifytoken** untuk api kita, buat file bernama **verifytoken.js** pada folder **routes**
20. Ketikkan kode berikut

```
import jwt from "jsonwebtoken";

const verifyToken = (req, res, next) => {
  const token = req.header("auth-token");
  if (!token)
    return res.status(400).json({
      status: res.statusCode,
      message: "Access Denied!",
    });
  try {
    const verified = jwt.verify(token, "pccclass");
    req.user = verified;
    next();
  } catch (err) {
    res.status(400).json({
      status: res.statusCode,
      message: "Invalid Token!",
    });
  }
};

export default verifyToken;
```

21. Buka file bernama **mahasiswa.js** pada folder routes
22. Import verifyToken menggunakan perintah berikut

```
import verifyToken from "../verifyToken.js";
```

23. Pada tiap endpoint, tambahkan satu parameter callback verifytoken

```
// CREATE
router.post("/", verifyToken, async (req, res) => {
  const reqMhs = new Mahasiswa({
    nama: req.body.nama,
    nim: req.body.nim,
    prodi: req.body.prodi,
    jurusan: req.body.jurusan,
  });

  try {
    const mahasiswa = await reqMhs.save();
    res.json(mahasiswa);
  } catch (err) {
    res.json({ message: err });
  }
});
```

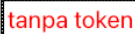
```
// READ
router.get("/", verifyToken, async (req, res) => {
  try {
    const mahasiswa = await Mahasiswa.find();
    res.json(mahasiswa);
  } catch (err) {
    res.json({ message: err });
  }
});

router.get("/:id", verifyToken, async (req, res) => {
  try {
    const mahasiswa = await Mahasiswa.findOne({ _id: req.params.id });
    res.json(mahasiswa);
  } catch (err) {
    res.json({ message: err });
  }
});
```

```
// UPDATE
router.put("/:id", verifyToken, async (req, res) => {
  try {
    const updateMhs = await Mahasiswa.updateOne(
      { _id: req.params.id },
      {
        nama: req.body.nama,
        nim: req.body.nim,
        prodi: req.body.prodi,
        jurusan: req.body.jurusan,
      }
    );
    res.json(updateMhs);
  } catch (err) {
    res.json({ message: err });
  }
});

// DELETE
router.delete("/:id", verifyToken, async (req, res) => {
  try {
    const deleteMhs = await Mahasiswa.deleteOne({
      _id: req.params.id,
    });
    res.json(deleteMhs);
  } catch (err) {
    res.json({ message: err });
  }
});
```

24. Selanjutnya kita akan melakukan ujicoba untuk kode diatas menggunakan postman



26. Buat file bernama `.env` pada root project, lalu ketikkan kode berikut

```

.env
1 DB_CONNECTION = mongodb+srv://namassist:hustt2453@cluster0.4lcd1.mongodb.net/mahasiswa
2 PORT = 3000
3 SECRET_KEY = pccclass

```

27. Buka file app.js

28. Import packages dotenv dan cors

```

import express from "express";
import mongoose from "mongoose";
import bodyParser from "body-parser";
import mahasiswaRoutes from "../routes/mahasiswa.js";
import userRoutes from "../routes/auth.js";
import cors from "cors";
import dotenv from "dotenv";
dotenv.config();

```

29. Tambahkan middlewares cors pada file **app.js**

```

// Middleware
app.use(bodyParser.json());
app.use(cors());

```

30. Selanjutnya ganti string connect to db mongodb dan port listen menjadi seperti berikut

```

// connect to DB
mongoose.connect(process.env.DB_CONNECTION, {
  useNewUrlParser: true,
  useUnifiedTopology: true,
});
let db = mongoose.connection;

db.on("error", console.error.bind(console, "Database connect Error!"));
db.once("open", () => {
  console.log("Database is Connected");
});

// listen
app.listen(process.env.PORT, () => {
  console.log(`Server running in port ${process.env.PORT}`);
});

```

31. Terakhir buka file verifyToken.js pada folder configs, dan ubah secret key jwt seperti berikut

```
import jwt from "jsonwebtoken";

const verifyToken = (req, res, next) => {
  const token = req.header("auth-token");
  if (!token)
    return res.status(400).json({
      status: res.statusCode,
      message: "Access Denied!",
    });
  try {
    const verified = jwt.verify(token, process.env.SECRET_KEY);
    req.user = verified;
    next();
  } catch (err) {
    res.status(400).json({
      status: res.statusCode,
      message: "Invalid Token!",
    });
  }
};

export default verifyToken;
```