## Tạo được một HTTP Server sử dụng NodeJS

const http = require("http");

const server = http.createServer((req, res) => {

  const url = req.url;

  console.log(url);

  res.setHeader("Content-Type", "text/html");

  if (url === "/") {

    res.write("<h1>This is Home page</h1>");

    res.end();

  } else {

    res.end("<h2>404 - Page not found!</h2>");

  }

});

server.listen(5000);

## HIểu được một số khái niệm cơ bản trong Express: Middleware, Route, Router, Request

**Middleware**: Là một chuỗi những hàm mà một request sẽ đi qua thông qua hàm next() cho đến khi respone được gửi đi hoặc không tìm thấy hàm next(). Middleware có thể được dùng để xử lý lỗi, xác thực, chuyển đổi dữ liệu. Trong Express, toàn bộ đều là middleware.

**Route**: Route là một thành phần cực kỳ quan trọng của một website, nó giúp website biết được người dùng truy cập đến nơi nào của trang web, từ đó phản hồi lại một cách thích hợp.

**Router:** là một instance của middleware và route nó giống như một mini express app. Dùng để phân nhóm các route có liên quan với nhau một cách có hệ thống và dễ quản lý hơn.

**Request**: Là yêu cầu của người dùng được gửi về cho server từ browser, chứa các thông tin như Headers, Body, URL, Params, Query hay Cookies, …

## Sử dụng được một số Template Engine trong Express: Pug, Handlebar, EJS

## Vận dụng được mô hình MVC

**Model** là các đối tượng hoặc một phần của code chịu trách nhiệm biểu diễn cho dữ liệu trong code và cho phép bạn làm việc với dữ liệu. Những thao tác như lưu dữ liệu, tìm nạp dữ liệu vào hoặc từ một tệp, trong bộ nhớ, ... đều được xử lý bởi model.

**View** chịu trách nhiệm về những gì người dùng nhìn thấy cuối cùng. Ở đây là hiển thị nội dung phù hợp trong các tài liệu html. Vì vậy chúng được tách khỏi code ứng dụng của bạn và chỉ có một số tích hợp nhỏ liên quan đến dữ liệu mà chúng ta đưa vào công cụ tạo khuôn mẫu (templating engine) để tạo ra các view này.

**Controller** là điểm kết nối giữa các model và view. Các view không quan tâm đến logic ứng dụng và các model quan tâm đến cách để lưu và tìm nạp dữ liệu, ..., controller làm việc với các model, lưu dữ liệu đó hoặc kích hoạt quá trình lưu đó, truyền dữ liệu đã được tìm nạp vào view.

## Sử dụng được Dynamic Route để tạo các Route phức tạp hơn

app.get("/product/:params1?/:params2?", (req, res, next) => {

    const {params1, params2} = req.params

    const query = req.query.q

});

## Nắm được vai trò của cơ sở dữ liệu và sự khác biệt giữa SQL và NoSQL

Vai trò của cơ sở dữ liệu là dùng để lưu trữ dữ liệu, trích xuất và quản lý dữ liệu.

SQL (Structured Query Language) là một ngôn ngữ truy vấn được sử dụng để truy xuất dữ liệu từ một CSDL quan hệ (RDBMS). SQL được sử dụng trong các CSDL quan hệ như MySQL, Oracle, SQL Server, PostgreSQL, v.v. SQL cung cấp khả năng tương tác với các bảng dữ liệu, tìm kiếm, sắp xếp và xử lý dữ liệu. SQL được sử dụng phổ biến vì nó hỗ trợ các tính năng như giao dịch, quản lý dữ liệu và đảm bảo tính toàn vẹn của dữ liệu.

NoSQL (Not Only SQL) là một loại CSDL phi quan hệ được thiết kế để xử lý dữ liệu phi cấu trúc, không nhất thiết phải tuân theo quy tắc chuẩn quan hệ của SQL. NoSQL được sử dụng cho các loại dữ liệu phi cấu trúc như JSON, XML, v.v. NoSQL được phát triển để cung cấp tính linh hoạt cao hơn trong việc lưu trữ và truy xuất dữ liệu. NoSQL được sử dụng phổ biến trong các ứng dụng web và mobile, vì nó cung cấp khả năng mở rộng tốt hơn và có thể xử lý lượng dữ liệu lớn hơn.

Tóm lại, SQL và NoSQL đều có vai trò quan trọng trong việc lưu trữ và truy xuất dữ liệu. SQL được sử dụng chủ yếu trong các ứng dụng doanh nghiệp, trong khi NoSQL được sử dụng phổ biến trong các ứng dụng web và mobile. SQL được sử dụng để lưu trữ dữ liệu có cấu trúc, trong khi NoSQL được sử dụng để lưu trữ dữ liệu phi cấu trúc.

## Kết nối và tạo Model với MongoDB, MySQL, sử dụng mongoose và sequelize

### **Mongoose:**

* Tạo model:

const mongoose = require("mongoose");

const Schema = mongoose.Schema;

const userSchema = Schema({

  email: { type: String, require: true },

  password: { type: String, require: true },

});

module.exports = mongoose.model("User", userSchema);

* Kết nối, thêm xoá sửa dữ liệu:

const express = require("express");

const mongoose = require("mongoose");

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

const app = express();

const DEFAULT\_ID = "640d12526c3ea406ce8654b7";

app.use(express.json());

app.get("/");

app.get("/create-user", (req, res, next) => {

  const newUser = new User({ email: "nghia@gmail.com", password: "password" });

  return newUser.save().then(() => {

    return res.send({ messgae: "Created User" });

  });

});

app.get("/delete-user/:id?", (req, res, next) => {

  User.findOneAndDelete({ email: "nghia@gmail.com" }).then((user) => {

    return res.send({ message: "Deleted User" });

  });

});

app.get("/edit/:id?", (req, res, next) => {

  User.findOneAndUpdate(

    { email: "nghia@gmail.com" },

    { $set: { password: "newpassword" } }

  ).then((user) => {

    return res.send({ message: "Updated User" });

  });

});

mongoose

  .connect(

    "mongodb+srv://owwibookstore:owwibookstore@cluster0.o5luvip.mongodb.net/Exam?retryWrites=true&w=majority"

  )

  .then(() => {

    console.log("Connected MongoDB!");

    app.listen(5000);

  });

### **MongoDB:**

**Kết nối (database.js):**

const mongoDB = require("mongodb");

const mongoClient = mongoDB.MongoClient;

let db;

const mongoConnect = (callback) => {

  mongoClient

    .connect(

      "mongodb+srv://owwibookstore:owwibookstore@cluster0.o5luvip.mongodb.net/Exam?retryWrites=true&w=majority"

    )

    .then((client) => {

      db = client.db();

      callback();

    });

};

const getDB = () => {

  if (db) {

    return db;

  } else {

    throw new Error("Not found database!");

  }

};

exports.mongoConnect = mongoConnect;

exports.getDB = getDB;

**Tạo Model (User.js):**

const mongoDB = require("mongodb");

const getDB = require("../utils/database").getDB;

class User {

  constructor(email, password, id) {

    this.email = email;

    this.password = password;

    this.\_id = id;

  }

  save() {

    const db = getDB();

    return db.collection("users").insertOne(this);

  }

  static deleteByEmail(email) {

    const db = getDB();

    return db.collection("users").deleteOne({ email: email });

  }

}

module.exports = User;

Thêm xoá:

const express = require("express");

const app = express();

const DEFAULT\_ID = "640d12526c3ea406ce8654b7";

const mongoConnect = require("./utils/database").mongoConnect;

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

app.use(express.json());

app.get("/");

app.get("/create-user", (req, res, next) => {

  const user = new User("nghia@gmail.com", "password");

  user.save().then(() => {

    res.send("Created User!");

  });

});

app.get("/delete-user/:email?", (req, res, next) => {

  const email = req.params.email;

  User.deleteByEmail(email).then(() => {

    res.send("Deleted User!");

  });

});

app.get("/edit/:id?", (req, res, next) => {

});

mongoConnect(() => {

  console.log("Database connected!");

  app.listen(5000);

});

### **Sequelize**:

**Kết nối:**

const Sequelize = require("sequelize");

const sequelize = new Sequelize("Exam", 'root', "Nghia089001",{

  host: "localhost",

  dialect: "mysql"

});

module.exports = sequelize

**Tạo model:**

const sequelize = require("../utils/database");

const Sequelize = require("sequelize");

const User = sequelize.define("user", {

  id: {

    type: Sequelize.INTEGER,

    allowNull: false,

    autoIncrement: true,

    primaryKey: true,

  },

  email: { type: Sequelize.STRING, allowNull: false },

  password: { type: Sequelize.STRING, allowNull: false },

});

module.exports = User

**Thêm:**

const express = require("express");

const app = express();

const DEFAULT\_ID = "640d12526c3ea406ce8654b7";

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

const sequelize = require("./utils/database");

app.use(express.json());

app.get("/");

app.get("/create-user", (req, res, next) => {

  User.create({ email: "nghia@gmail.com", password: "password" }).then(() => {

    return res.send("Created User");

  });

});

app.get("/delete-user/:email?", (req, res, next) => {});

app.get("/edit/:email?", (req, res, next) => {});

sequelize.sync({ force: true }).then(() => {

  console.log("databased connected!");

  app.listen(5000);

});

### **MySQL:**

**Kết nối:**

const mysql = require('mysql2')

const pool = mysql.createPool({

  host: "localhost",

  user: "root",

  database: "exam",

  password: "Nghia089001",

});

module.exports = pool.promise()

**Tạo model:**

const db = require('../utils/database')

module.exports = class Product {

  constructor(title, imageUrl, description, price) {

    this.title = title;

    this.imageUrl = imageUrl;

    this.description = description;

    this.price = price;

  }

  static findById(id){

    return db.execute('select \* from books where id = ?',[id])

  }

  static fetchAll(cb) {

    return db.execute('select \* from books')

  }

};

## Nắm được vai trò của Session & Cookie

Session và Cookie là hai công cụ được sử dụng để duy trì trạng thái và thông tin của người dùng khi họ truy cập vào một trang web.

Session là một phiên làm việc tạm thời được tạo ra khi người dùng truy cập vào trang web. Nó giúp lưu trữ thông tin về người dùng như tên đăng nhập, thông tin giỏ hàng, v.v. trong suốt quá trình duyệt web và giữ cho người dùng không phải đăng nhập lại mỗi khi họ thực hiện một thao tác mới trên trang web. Session được lưu trữ trên máy chủ và có thể chỉ được truy cập bởi người dùng trong phiên làm việc hiện tại.

Cookie là một tệp tin được lưu trữ trên máy tính của người dùng và chứa thông tin được lưu trữ bởi trang web. Cookie được sử dụng để nhận dạng người dùng, lưu trữ thông tin đăng nhập và lịch sử truy cập của người dùng trên trang web. Cookie có thể được thiết lập để tự động hết hạn sau một khoảng thời gian nhất định hoặc sau khi người dùng đăng xuất khỏi trang web.

Vai trò của Session và Cookie là giúp cho trang web có thể lưu trữ và duy trì thông tin người dùng, giúp người dùng truy cập trang web một cách tiện lợi và đáp ứng được nhu cầu của họ. Tuy nhiên, các trang web cần chú ý đến vấn đề bảo mật khi sử dụng Session và Cookie để tránh việc thông tin cá nhân của người dùng bị lộ ra ngoài.

Session và Cookie là hai công cụ được sử dụng để duy trì trạng thái và thông tin của người dùng khi họ truy cập vào một trang web. Tuy nhiên, có một số **khác biệt giữa Session và Cookie** như sau:

Vị trí lưu trữ: Session được lưu trữ trên máy chủ web, trong khi Cookie được lưu trữ trên máy tính của người dùng.

Thời gian sống: Session được thiết lập để tự động hết hạn sau khi người dùng đóng trình duyệt web hoặc sau một khoảng thời gian nhất định không có hoạt động nào từ người dùng. Cookie có thể được thiết lập để tự động hết hạn sau một khoảng thời gian nhất định hoặc sau khi người dùng đăng xuất khỏi trang web.

Thông tin được lưu trữ: Session được sử dụng để lưu trữ thông tin như tên đăng nhập, thông tin giỏ hàng, v.v. trong suốt quá trình duyệt web và giữ cho người dùng không phải đăng nhập lại mỗi khi họ thực hiện một thao tác mới trên trang web. Cookie được sử dụng để nhận dạng người dùng, lưu trữ thông tin đăng nhập và lịch sử truy cập của người dùng trên trang web.

Bảo mật: Session được xem là an toàn hơn Cookie vì thông tin được lưu trữ trên máy chủ web và không thể được truy cập bởi bất kỳ ai ngoài máy chủ. Cookie có thể bị lộ thông tin nếu máy tính của người dùng bị tấn công hoặc nếu người dùng chia sẻ máy tính với người khác.

Tóm lại, Session và Cookie đều có vai trò quan trọng trong việc lưu trữ và duy trì thông tin người dùng trên một trang web. Tuy nhiên, có sự khác biệt về vị trí lưu trữ, thời gian sống, thông tin được lưu trữ và bảo mật giữa hai công cụ này.

## Tạo được cơ chế xác thực cho Website

const express = require("express");

const session = require("express-session");

const app = express();

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

const cors = require("cors");

app.use(express.json({ type: ["application/json", "text/html"] }));

app.use(cors({ origin: true, credentials: true }));

app.use(

  session({

    secret: "SESSION\_SECRET",

    resave: false,

    saveUninitialized: false,

    cookie: {

      maxAge: 1000 \* 60 \* 60,

    },

  })

);

app.get("/check", (req, res, next) => {

  res.send({ isLoggedIn: req.session.isLoggedIn });

});

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

  const { email, password } = req.body;

  console.log(req.body);

  console.log(email, password);

  if (email === "email" && password === "password") {

    req.session.isLoggedIn = true;

    return res.send(req.session);

  } else {

    return res.send({ messgae: "Failed!" });

  }

});

app.listen(5000);

Lưu ý: Ở phía server, phải chỉnh cors và credential về true để tương tác với client. app.use(express.json()) để parse data từ body ở dạng json. Lỗi CORS policy thưởng xảy ra khi thiếu 2 thuộc tính này trong cors

Ở phía client, lúc fetch API phải chỉnh như sau để có thể đọc được data ở server:

const respone = await fetch("http://localhost:5000/login", {

        credentials: "include",

        method: "POST",

        headers: { "Content-Type": "application/json" },

        body: JSON.stringify({

          email: emailRef.current.value,

          password: passwordRef.current.value,

        }),

      });

## Nắm được các kỹ năng liên quan đến Email & Validation, Error Handling, Async Request

const express = require("express");

const session = require("express-session");

const app = express();

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

const cors = require("cors");

const multer = require("multer");

const { body, validationResult } = require("express-validator/check");

app.use(express.json({ type: ["application/json", "text/html"] }));

app.use(cors({ origin: true, credentials: true }));

app.use(

  session({

    secret: "SESSION\_SECRET",

    resave: false,

    saveUninitialized: false,

    cookie: {

      maxAge: 1000 \* 60 \* 60,

    },

  })

);

app.get("/check", (req, res, next) => {

  res.send({ isLoggedIn: req.session.isLoggedIn });

});

app.post(

  "/login",

  [

    body("email", "Email is not valid!").isEmail(),

    body("password").custom((value, { req }) => {

      if (req.body.password.length <= 5) {

        throw new Error("Password length must be greater than 5 characters!");

      }

    }),

  ],

  (req, res, next) => {

    const { email, password } = req.body;

    const errors = validationResult(req);

    console.log(errors.array());

    if (!errors.isEmpty()) {

      return res.send({ messgae: errors.array()[0].msg });

    }

    if (email === "email" && password === "password") {

      req.session.isLoggedIn = true;

      return res.send({ messgae: "Success!" });

    } else {

      return res.send({ messgae: "Failed!" });

    }

  }

);

app.listen(5000);

## Xử lý được các chức năng liên quan đến FIle, Upload, Download

## Hiểu và vận dụng được Rest API

Rest API là một phương thức giao tiếp giữa các ứng dụng phân tán trên mạng thông qua giao thức HTTP và các method như POST, PUT, GET, DELETE, PATCH,… để truyền tải dữ liệu giữa chúng

## Hiểu và vận dụng được SocketIO

Websocket là công nghệ hỗ trợ giao tiếp hai chiều giữa client và server bằng cách sử dụng một TCP socket để tạo một kết nối hiệu quả và nhanh chóng.

## Deploy được ứng dụng NodeJS lên online