## **LAB 14 – LẬP TRÌNH HƯỚNG SỰ KIỆN**

### ****Tóm tắt lý thuyết****

* **Lập trình hướng sự kiện (Event-driven programming)** là một mô hình lập trình mà luồng thực thi được điều khiển bởi các sự kiện (event), thay vì thực hiện tuần tự từ trên xuống dưới.
* Các sự kiện có thể là: tương tác người dùng (click, nhập phím), các sự kiện hệ thống (đọc file, request HTTP), timeout...
* **Node.js** là môi trường chạy JavaScript phía server, được xây dựng dựa trên lập trình hướng sự kiện và **non-blocking I/O**.

### ****Các thành phần chính trong mô hình Event-driven Node.js****

1. **EventEmitter**: Bộ quản lý sự kiện. Dùng để đăng ký và phát sinh sự kiện.
   * on(eventName, callback): Đăng ký lắng nghe sự kiện.
   * emit(eventName, args): Phát sinh sự kiện.
2. **Event**: Là sự kiện phát sinh trong ứng dụng.
3. **Event Loop**: Vòng lặp sự kiện, liên tục kiểm tra hàng đợi sự kiện và thực thi callback.
4. **Event Handler**: Hàm xử lý được gọi khi sự kiện xảy ra.

### ****Bài tập 14.1 – Viết lại các đoạn mã trong bài học****

#### **[1] Cách viết với hàm thường**

'use strict'

const express = require('express')

const app = express()

const port = process.env.PORT || 9000

const events = require('events')

const eventEmitter = new events.EventEmitter()

eventEmitter.on('hetGio', xuLyThongBao)

function xuLyThongBao(thongBao) {

console.log(thongBao)

}

setTimeout(() => {

eventEmitter.emit('hetGio', 'Hết giờ học rồi, về thôi!!!!!!!')

}, 2000)

app.listen(port, () => {

console.log(`server dang chay tren cong ${port}`)

})

#### **[2] Cách viết với arrow function**

'use strict'

const express = require('express')

const app = express()

const port = process.env.PORT || 9000

const events = require('events')

const eventEmitter = new events.EventEmitter()

eventEmitter.on('hetGio', (thongBao) => {

console.log(thongBao)

})

setTimeout(() => {

eventEmitter.emit('hetGio', 'Hết giờ học rồi, về thôi!!!!!!!')

}, 2000)

app.listen(port, () => {

console.log(`server dang chay tren cong ${port}`)

})

### ****Bài tập 14.2 – Viết chương trình với sự kiện vaoLop****

#### **[1] Viết bằng hàm thường**

'use strict'

const express = require('express')

const app = express()

const port = process.env.PORT || 9000

const events = require('events')

const eventEmitter = new events.EventEmitter()

eventEmitter.on('vaoLop', xuLyVaoLop)

function xuLyVaoLop(tb) {

console.log(tb)

}

setTimeout(() => {

eventEmitter.emit('vaoLop', 'Đã đến giờ học')

}, 5000)

app.listen(port, () => {

console.log(`Server đang chạy trên cổng ${port}`)

})

#### **[2] Viết bằng arrow function**

'use strict'

const express = require('express')

const app = express()

const port = process.env.PORT || 9000

const events = require('events')

const eventEmitter = new events.EventEmitter()

eventEmitter.on('vaoLop', (thongBao) => {

console.log(thongBao)

})

setTimeout(() => {

eventEmitter.emit('vaoLop', 'Đã đến giờ học')

}, 5000)

app.listen(port, () => {

console.log(`Server đang chạy trên cổng ${port}`)

})

### ****Câu hỏi 14.3 – Phát biểu KHÔNG đúng là:****

**A. Event Emitter liên tục kiểm tra hàng đợi sự kiện và thực thi các hàm callback khi có sự kiện xảy ra.**

**Đáp án đúng: A**  
 Vì **Event Loop** mới là thành phần chịu trách nhiệm kiểm tra hàng đợi và xử lý sự kiện, **không phải Event Emitter**.