INT 161

Basic Backend Development

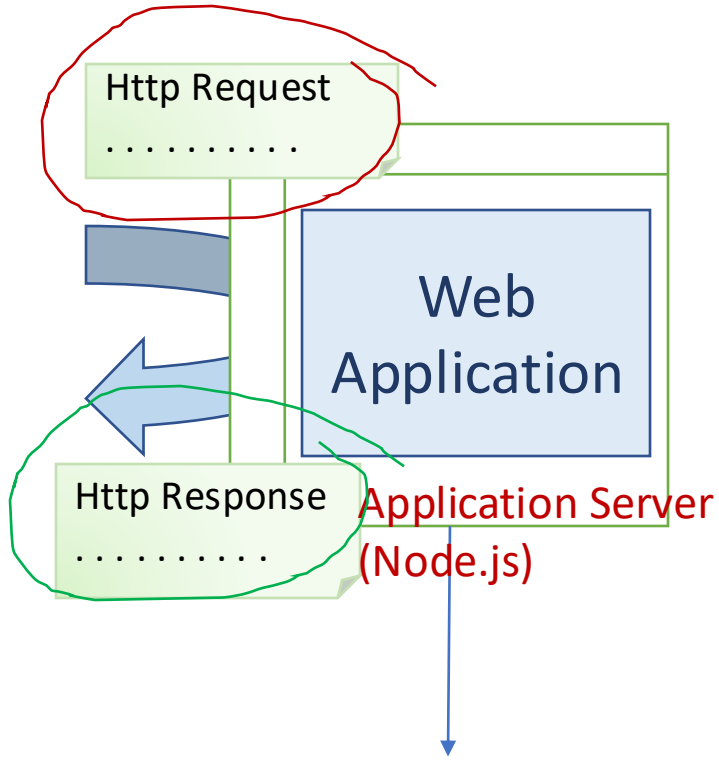
Week-02

HTTP Programing Basics

Unit Objectives

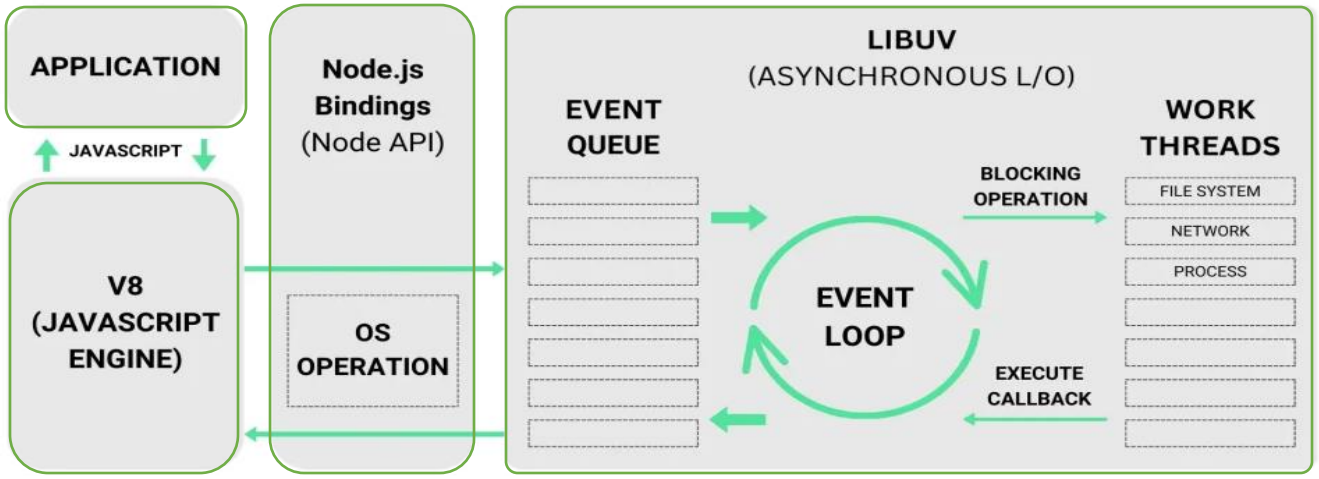
- After completing this unit, you should be able to:
  - Explain the HTTP protocol for Web Programming
  - Created HTTP Server using Node.js
  - Manage Request & Response objects
  - Read Request parameter/Request body and path variable
  - Created Express.js project

Web Application



Database Server

Node.js Architecture



- Clients send requests to the webserver to interact with the web application. Requests can be non-blocking or blocking e.g Querying the data , updating the data or deleting the data .
- Node.js retrieves the incoming requests and adds those requests to the Event Queue.
- The requests are then passed one-by-one through the Event Loop. It checks if the requests are simple enough to not require any external resources.
- Event Loop processes simple requests (non-blocking operations), such as I/O Polling, and returns the responses to the corresponding clients.

HTTP Server (Core Module)

- node:http is the built-in HTTP module in Node.js, used to create HTTP servers and make HTTP client requests — basically, lets Node.js talk over the web without needing any external library.

```
const http = require('node:http');
const server = http.createServer((req, res) => {
  res.write('http version: ' + req.httpVersion + '\n');
  res.write('url: ' + req.url + '\n');
  res.write('rawHeaders: ' + req.rawHeaders + '\n---------\n\n');
  res.write('method: ' + req.method + '\n');
  res.write('headers.user-agent: ' + req.headers["user-agent"] + '\n');
  res.end('headers.host: ' + req.headers.host + '\n');
});
server.listen(3000, () => {
    console.log('Server running at http://localhost:3000');
});
```

Http Response

```
const http = require('node:http');
const server = http.createServer((req, res) => {
    res.writeHead(404, {'Content-Type': 'text/plain'});
    res.write('http version: ' + req.httpVersion + '\n');
     :
     :
    res.end('headers.host: ' + req.headers.host + '\n');
});

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000');
});
server.on('error', (err) => {
    console.log('Error: ' + err.message);
})
server.on('request', (req, res) => {
    console.log('Request received: ' + req.headers.host+ req.url
    + ', Method: ' + req.method);
});
```

HTTP Status Codes

- 1xx: Informational
- 2xx: Success (200 OK, 201 Created)
- 3xx: Redirection (301, 302)
- 4xx: Client Error (400, 404)
- 5xx: Server Error (500)

HTTP Methods

- **Method Purpose Example**
  - GET Retrieve data from resource /subjects /students
  - POST Create new resource /subjects /students (with data)
  - PUT Update existing data /subjects/{id} /students /{id} (with data)
  - DELETE Remove a resource /subjects/{id} /students /{id}

Request protocol, GET

- The request object encapsulates all of the information from the client request.
- The following methods are available to access parameters:
  - WHATWG URL (The URL Standard developed and maintained by the Web Hypertext Application Technology Working Group)

```
const urlObj = new URL(req.url,`http://${req.headers.host}`);
const queryString = urlObj.searchParams;

const name = queryString.get('name');
const subjects = queryString.getAll('favourite_subject');
```

Request protocol, GET - Example

```
const http = require('node:http');
const server = http.createServer((req, res) => {
    const urlObj = new URL(req.url, `http://${req.headers.host}`);
    params = urlObj.searchParams;
    name = params.get('name');
    subjects = params.getAll('favourite_subject');
    res.write('Name: ' + name + '\n');
    res.write('Favorite Subjects:\n');
    subjects.forEach(function(subject, index) {
            res.write('    ' + (index+1) + ', ' + subject + '\n');
    });
    res.end();
});
server.listen(3000, () => {
    console.log('Server running at http://localhost:3000');
});
```

Request protocol, POST

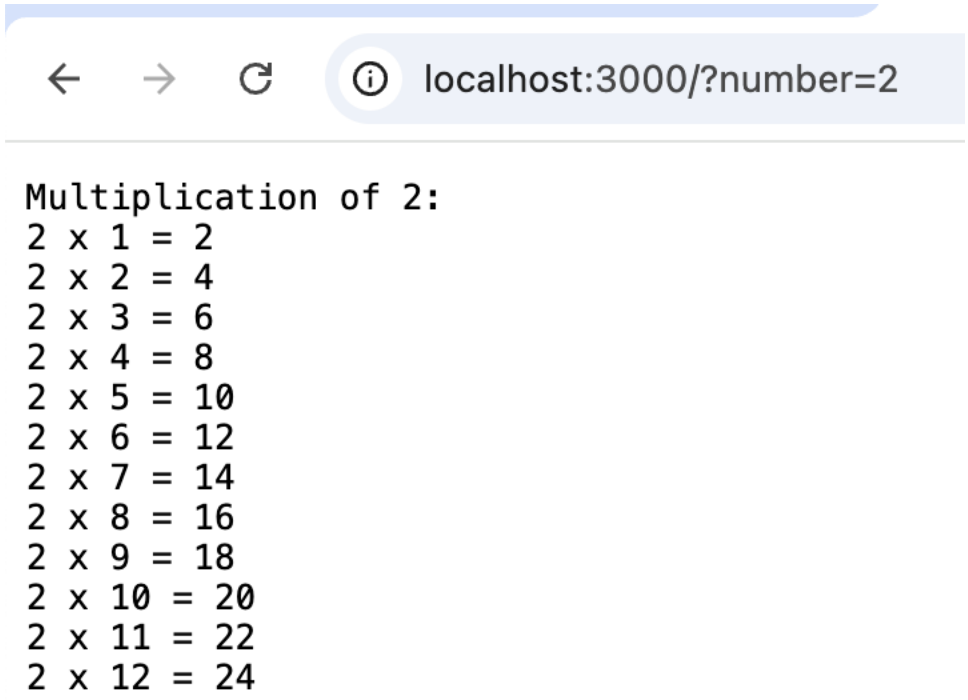
```
const http = require('node:http');
const server = http.createServer((req, res) => {
    console.log("content-type: "+ req.headers['content-type']);
    if(req.method != 'POST') return res.end("Please use POST method");
    let body = '';
    req.on('data', chunk => {body += chunk;});
    req.on('end', chunk => {
    const params = new URLSearchParams(body);
    name = params.get('name');
    subjects = params.getAll('favourite_subject');
    res.write('Name: ' + name + '\n');
    res.write('Favorite Subject:\n');
    subjects.forEach(function(subject, index) {
        res.write('    ' + (index+1) + ', ' + subject + '\n');
    });
    res.end();
  });
});
```

Request protocol, Read Json from request body

```
const http = require('node:http');
const server = http.createServer((req, res) => {
        console.log("content-type: "+ req.headers['content-type']);
        if(req.method != 'POST') return res.end("Please use POST method");
        let body = '';
        req.on('data', chunk => {
            body += chunk;
        });
        req.on('end', chunk => {
            jsonObj = JSON.parse(body);
            res.write('Request body :\n');
            res.end(JSON.stringify(jsonObj));
        });
});
server.listen(3000, () => {
    console.log('Server running at http://localhost:3000');
});
```

Exercise - Multiplication table

- Receive the request parameter 'number’
- returns a multiplication table for that number
- Example of response body



Request protocol, Path Variable + Example

```
const http = require('node:http');
const server = http.createServer((req, res) => {
    const urlObj = new URL(req.url, `http://${req.headers.host}`);
    const pathVar = urlObj.pathname.split('/').filter(Boolean);
    number = pathVar[1];
    res.write(`Multiplication of ${number}:\n`);
    for (let i = 1; i <= 12; i++) {
        res.write(number + ' x ' + i + ' = ' + i * number + '\n');
    }
    res.end();
});
server.listen(3000, () => {
    console.log('Server running at http://localhost:3000');
});
```

pathname.split('/') → gives an array like ["", "users", "123"] for /users/123
filter(Boolean) → runs Boolean(item) on each element:
filter() keeps only elements where the callback returns true, so empty strings are removed.

Exercise

REST API – CRUD for Subject resource

SubjectRepository.js (1/3)

```
const subjects = [
    { "id": "INT 100", "name": "IT Fundamentals", "credits": 3 },
    { "id": "INT 101", "name": "Programming Fundamentals", "credits": 3 },
    { "id": "INT 102", "name": "Web Technology", "credits": 1 },
    { "id": "INT 114", "name": "Discrete Mathematics", "credits": 3 },
    { "id": "GEN 101", "name": "Physical Education", "credits": 1 },
    { "id": "GEN 111", "name": "Man and Ethics of Living", "credits": 3 },
    { "id": "LNG 120", "name": "General English", "credits": 3 },
    { "id": "LNG 220", "name": "Academic English", "credits": 3 },
    { "id": "INT 103", "name": "Advanced Programming", "credits": 3 },
    { "id": "INT 104", "name": "User Experience Design", "credits": 3 },
    { "id": "INT 105", "name": "Basic SQL", "credits": 1 },
    { "id": "INT 107", "name": "Computing Platforms Technology", "credits": 3 },
    { "id": "INT 200", "name": "Data Structures and Algorithms", "credits": 1 },
    { "id": "INT 201", "name": "Client-Side Programming I", "credits": 2 },
    { "id": "INT 202", "name": "Server-Side Programming I", "credits": 2 },
    { "id": "INT 205", "name": "Database Management System", "credits": 3 },
    { "id": "INT 207", "name": "Network I", "credits": 3 }
];
```

SubjectRepository.js (2/3)

```
function getSubjects() { return subjects; }
function getSubject(id) { return subjects.find(s => s.id === id); }
function addSubject(subject) { subjects.push(subject); }
function updateSubject(id, subject) {
    const idx = subjects.findIndex(s => s.id === id);
    if (idx === -1) return false;
    subjects[idx] = subject;
    return true;
}
function patchSubject(id, partial) {
    const idx = subjects.findIndex(s => s.id === id);
    if (idx === -1) return false;
    subjects[idx] = { ...subjects[idx], ...partial, id: subjects[idx].id };
    return true;
}
```

SubjectRepository.js (3/3)

```
function removeSubject(id) {
    const idx = subjects.findIndex(s => s.id === id);
    if (idx === -1) return false;
    subjects.splice(idx, 1);
    return true;
}

module.exports = {
    getSubjects,
    getSubject,
    addSubject,
    updateSubject,
    patchSubject,
    removeSubject
};
```

Utility.js (1/5)

```
// Utilities
function json(res, status, data) {
    res.writeHead(status, { 'Content-Type': 'application/json' });
    res.end(JSON.stringify(data));
}
function noContent(res) {
    res.writeHead(204);
    res.end();
}
function notFound(res, message = 'Not Found') {
    json(res, 404, { error: message });
}
function badRequest(res, message = 'Bad Request') {
    json(res, 400, { error: message });
}
```

Utility.js (2/5)

```
function conflict(res, message = 'Conflict') {
    json(res, 409, { error: message });
}
function methodNotAllowed(res, message = 'Method Not Allowed') {
    json(res, 405, { error: message });
}
function unsupportedMediaType(res, message = 'Unsupported Media Type')
{
    json(res, 415, { error: message });
}
```

Utility.js (3/5)

```
function readBody(req) {
    return new Promise((resolve, reject) => {
        let body = '';
        req.setEncoding('utf8');
        req.on('data', chunk => {
            body += chunk;
            if (body.length > 1e6) { // 1MB guard
                reject(new Error('Payload too large'));
                req.destroy();
            }
        });
        req.on('end', () => resolve(body));
        req.on('error', reject);
    });
}
```

Utility.js (4/5)

```
async function parseBody(req) {
  const ct = (req.headers['content-type'] ||
'').toLowerCase();
  if (ct.includes('application/json')) {
    const raw = await readBody(req);
    try {
      return { type: 'json', data: raw ? JSON.parse(raw) : {} };
    } catch {
      throw new Error('Invalid JSON');
    }
  }
  if (ct.includes('application/x-www-form-urlencoded')) {
    const raw = await readBody(req);
    const params = new URLSearchParams(raw || '');
    const obj = {};
    for (const [k, v] of params) {
      if (k in obj) obj[k] = Array.isArray(obj[k]) ?
[...obj[k], v] : [obj[k], v];
      else obj[k] = v;
    }
    return { type: 'form', data: obj };
  }
  if (!ct) {
    // ไม่มีบอดี้/ไม่มี content-type
    return { type: 'none', data: null };
  }
  throw new Error('UNSUPPORTED_CT');
}
```

Utility.js (5/5)

```
module.exports = {
    json,
    noContent,
    notFound,
    badRequest,
    conflict,
    methodNotAllowed,
    unsupportedMediaType,
    readBody,
    parseBody
};
```

Simple-rest-api.js (1)

```
const http = require('node:http');

const {
    getSubjects,
    getSubject,
    addSubject,
    updateSubject,
    patchSubject,
    removeSubject
} = require('./SubjectRepository');
```

```
const {
    json,
    noContent,
    notFound,
    badRequest,
    conflict,
    methodNotAllowed,
    unsupportedMediaType,
    parseBody
} = require('./Utility');
```

Simple-rest-api.js (2)

```
// Router using WHATWG URL
function router(req, res) {
  const url = new URL(req.url, `http://${req.headers.host}`);
  const pathname = url.pathname;
  if (pathname === '/subjects') {
    if (req.method === 'GET') {
      return json(res, 200, getSubjects());
    }
    if (req.method === 'POST') {
      (async () => {
        try {
          const parsed = await parseBody(req);
          const data = parsed.data || {};
          if (!data || typeof data !== 'object') return badRequest(res, 'Body must be an object');
```

Simple-rest-api.js (3)

```
            const { id, name, credits } = data;
            if (!id || !name || typeof credits !== 'number') {
               return badRequest(res, 'Fields required: id(string), name(string), credits(number)');
            }
            if (getSubject(id)) return conflict(res, `Subject with id "${id}" already exists`);

            addSubject({ id, name, credits });
            return json(res, 201, { id, name, credits });
          } catch (e) {
            if (e && e.message === 'UNSUPPORTED_CT') return unsupportedMediaType(res);
            return badRequest(res, e.message || 'Bad Request');
          }
        })();
        return;
    }
    return methodNotAllowed(res);
}
```

Simple-rest-api.js - Test Router: GET, POST

```
const server = http.createServer(router);

server.listen(3000, () => {
    console.log('Server running at http://localhost:3000');
});
```

Simple-rest-api.js (4)

```
// /subjects/:id item routes
if (pathname.startsWith('/subjects/')) {
  const id = decodeURIComponent(pathname.slice('/subjects/'.length));
  if (!id) return notFound(res);

  if (req.method === 'GET') {
    const subject = getSubject(id);
    return subject ? json(res, 200, subject) : notFound(res, `Subject "${id}" not found`);
  }

  if (req.method === 'PUT') {
    (async () => {
      try {
        const parsed = await parseBody(req);
        const data = parsed.data || {};
        if (!data || typeof data !== 'object') return badRequest(res, 'Body must be an object');
```

Simple-rest-api.js (5)

```
            const { name, credits } = data;
            if (!name || typeof credits !== 'number') {
                return badRequest(res, 'Fields required: name(string), credits(number)');
            }
            const exists = getSubject(id);
            if (!exists) return notFound(res, `Subject "${id}" not found`);

            updateSubject(id, { id, name, credits });
            return json(res, 200, { id, name, credits });
        } catch (e) {
            if (e && e.message === 'UNSUPPORTED_CT') return unsupportedMediaType(res);
            return badRequest(res, e.message || 'Bad Request');
        }
    })();
    return;
}
```

Simple-rest-api.js (6)

```
    if (req.method === 'DELETE') {
        const ok = removeSubject(id);
        return ok ? noContent(res) : notFound(res, `Subject "${id}" not found`);
    }

    return methodNotAllowed(res);
}

// Default 404
notFound(res);
```

What is HTTP ?

- Application Layer Protocol for communicate between Client & Server
- Request
  - Method
  - URL
  - Header
  - Body
- Response
  - Status Code
  - Header
  - Body