CIS 371-01 Web Application Programming

ExpressJS

HTTP server in (Java|Type)Script

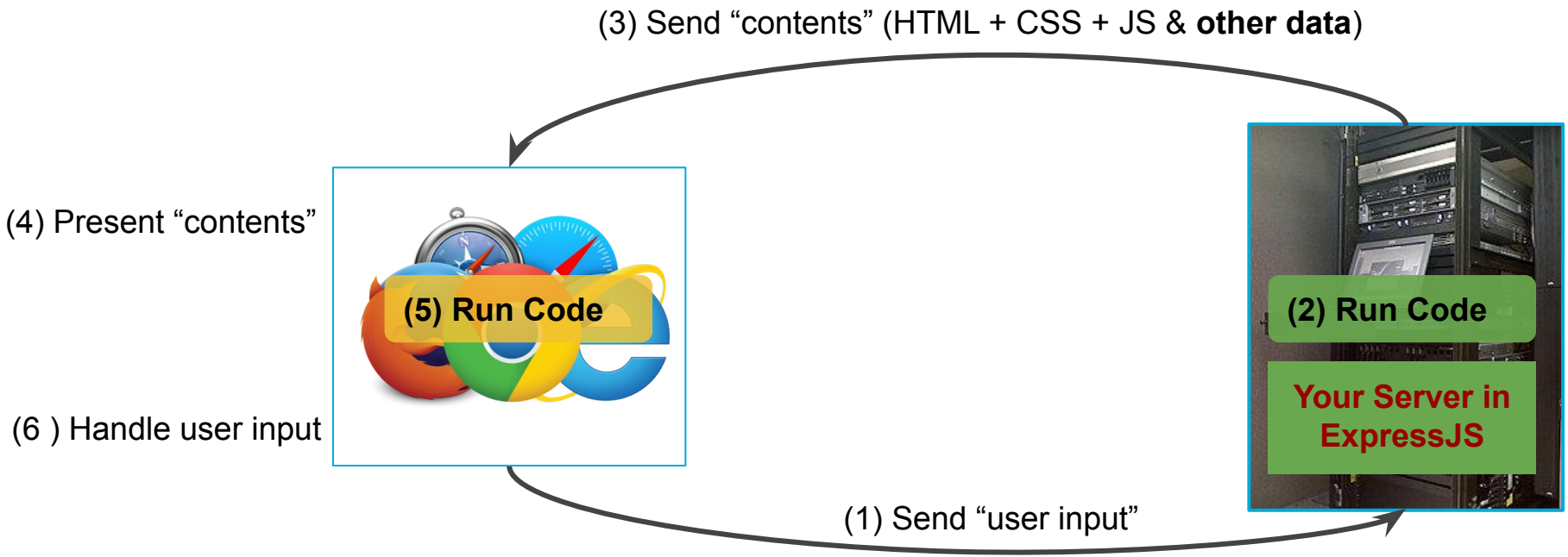
Lecturer: Dr. Yong Zhuang

Based on the original version by Professor Hans Dulimarta.

Topics

- What is ExpressJS
- Why need an app server
- Features provided
  - Router for HTTP methods (GET, POST, PUT, DELETE, …)
  - Middleware
  - Query Parameter parsing
  - Payload parsing (via middleware)
- Prerequisite
  - HTTP Protocol: Request & Response

Web Client/Server Architecture



Why Use an App Server

- Enhanced Computing Power:
  - Client Side: Limited computing capabilities on user machines.
  - App Server: Typically hosted on robust external machines offering greater computing power.
- Security Considerations:
  - Code Visibility: Code sent to a client's browser can be inspected by the user, possibly revealing proprietary information.
  - Company Secrets: Using an app server can protect sensitive logic and data from direct client access.
- Be aware of extra network transport overhead
- Potential Use Cases:
  - Proxy Server: Bypass CORS restrictions when accessing certain web services.
  - Data Aggregation: Consolidate data from various sources like web clients and mobile clients.
  - High-End Computations: Ideal for tasks that require heavy computation, such as machine learning, computational simulations, and data mining.

Alternatives to ExpressJS

Setup

```
npm init -y
npm i express # version 4.x
```

```
// Additional libraries for TypeScript dev
npm i --save-dev typescript ts-node nodemon

// Add type declaration files
npm i --save-dev @types/express @types/node

// Creates tsconfig.json
tsc --init
```

tsconfig.json

```
{
  "compilerOptions":{
  "target": "es6",
  "module": "commonjs",
  "rootDir": "./",
  "outDir": "./build",
  "esModuleInterop": true,
  "strict": true
  }
 };
```

Hello World

```ts
import express, { Application, Request, Response } from "express";

const app: Application = express();
const PORT = process.env.PORT ?? 8000;

// Define GET endpoint(s)
app.get("/", (req: Request, res: Response) => {
  res.send("Hello World!");
});

app.listen(PORT, () => {
  console.log(`Server is listening to port ${PORT}`);
});
```

my-server.ts

```
// Launch the server
npx nodemon my-server.ts
```

Browser http://localhost:8000/

Defining More Routes/EndPoints

```
// import lines not shown
app.get("/", (req: Request, res: Response) => {
  res.send("Hello World!");
});
app.get("/about", (req: Request, res: Response) => {
  res.send("Just a simple Express server");
});
app.post("/xyz", (req: Request, res: Response) => {
  res.send("Just a simple Express server");
});
app.put("/order/cancel/", (req: Request, res: Response) => {
  res.send("_____");
});
app.delete("/account", (req: Request, res: Response) => {
  res.send("_____");
});
```

my-server.ts

Browser http://localhost:8000/

Browser http://localhost:8000/about

Can't invoke these from browser omnibox

Sending Responses

```
// METHOD: get, post, put, delete, and so on
app.METHOD("/path/goes/here", (req: Request, res: Response) => {
  // any file type, its content will be included in the response body
  res.download("file-name"); // Without MIME type
  res.type("image/jpg").download("mycat.jpg"); // OR with MIME type
});
```

Opt #1: file attachment

```
app.METHOD("/path/goes/here", (req: Request, res: Response) => {
  res.send("some text here"); // Opt-2: send text response
  res.type("text/plain").send("some text here"); // Opt-2a: with Content-Type
});
```

Opt #2: plain text

```
app.METHOD("/path/goes/here", (req: Request, res: Response) => {
  res.send("<P>Sample HTML response</p>"); // Opt-3: send HTML response
  res.type("text/html").send("<P>Sample HTML response</p>"); // Opt-3a: with Content-Type
});
```

Opt #3: HTML string

Sending Responses

```
app.METHOD("/path/goes/here", (req: Request, res: Response) => {
  res.send({ shippingCost: 5.16, sameDay: false }); // Opt-4: send JSON response
});
```

Opt #4: JSON

```
app.METHOD("/path/goes/here", (req: Request, res: Response) => {
  // Opt-5: send JSON from a JS object
  const my_resp_obj = { msg: "Hello World", hasEmoji: false };
  res.json(my_resp_obj);
});
```

Opt #5: JSON from object

```
app.METHOD("/path/goes/here", (req: Request, res: Response) => {
  res.status(401).end(); // Opt-6: set HTTP status & an empty response
});
```

Opt #6: empty response

Parsing Dynamic Path/Route Names

```
app.get("/user/:uname", (req: Request, res: Response) => {
  const who = req.params.uname;
  if (who == "___") res.send(`Option #1 here`);
  else res.send(`Option #2 here`);
});
```

Browser http://localhost:8000/user/**bob**

```
app.get("/search/:minPrice/:maxPrice", (req: Request, res: Response) => {
  const low = req.params.minPrice;
  const high = req.params.maxPrice;
  res.send(`Price range ${low}-${high}`);
});
```

Browser http://localhost:8000/search/**50/110**

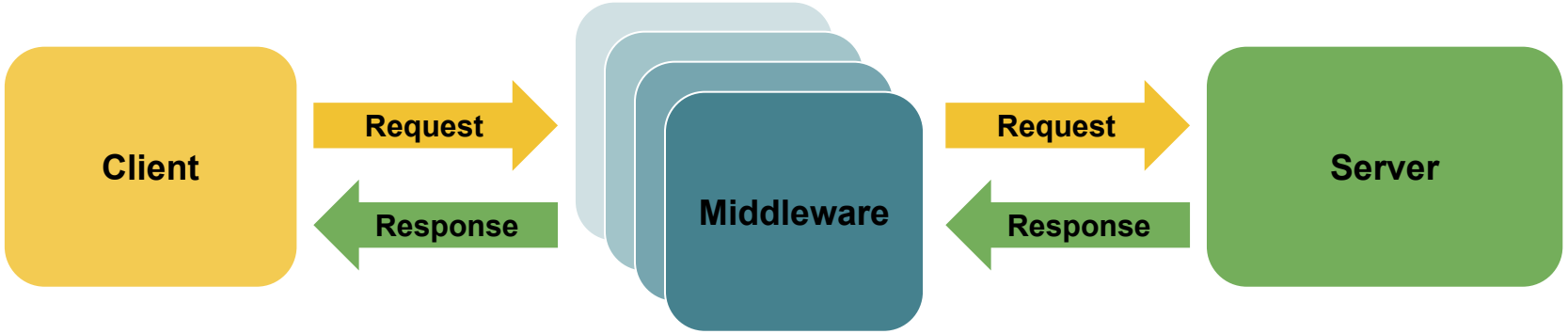
Parsing Query Parameters (TypeScript types)

```
type Query2 = {
  min: number;
  max: number;
};
```

Browser http://localhost:8000/search?min=50&max=110

```
app.get("/search", (req: Request<any, any, any, Query2>, res: Response) => {
  res.send(`Price range ${req.query.min}-${req.query.max}`);
});
```

ExpressJS Middleware



Middleware (running on the server)

- *Apply pre-processing logic incoming requests*
- *Apply post-processing logic to outgoing responses*

ExpressJS Middleware

Name	Functionality	Usage
compression	Zlib compression to HTTP responses	Reduces response size for faster transfers and lower bandwidth.
cors	Enable Cross-Origin Resource Sharing	Configures which domains can access server resources, enhancing security and flexibility.
morgan	Log incoming requests	Monitors server activity, aiding in debugging and tracking request patterns.
multer	Parse multi-part form data in incoming requests	Simplifies processing of multi-part content, especially for file uploads.

Parsing POST request payload

```
// Package for handling multipart form-data
npm i multer
```

```
import multer from "multer"
app.use(express.json()); // Enable handling of application/json
app.use(express.urlencoded()); // Enable handling of application/x-www-form-urlencoded
const multiPartParser = multer();
app.post("/path-for-this-end-point",
 multiPartParser.none(), // .none() do not parse files in multiform
 (req:Request, res:Response) => {
 const payload = req.body;
 // payload is an object whose props are name in the form-data
 res.send(YOUR_RESPONSE_HERE);
})
```

Browser Same Origin Resource Sharing Policy

- HTTP GET requests from a script are allowed to fetch only resources from the same origin as the script
- This restriction does not apply to URLs that you type directly in the browser omnibox (because the request does not originate from a script)

Same / Cross Origin

Script Origin	Resource URL	Same/Cross Origin
http://www.mysites.org/code/one.js	http://www.mysites.org/img/logo.png	Same origin
http://www.mysites.org/code/one.js	http://**mysites.org**/img/logo.png	Cross origin
http://www.mysites.org/code/one.js	http://www.mysites.org**:1234**/img/logo.png	Cross origin
http://www.mysites.org/code/one.js	**https**://www.mysites.org/img/logo.png	Cross origin

Allow CORS

```
// Cross-Origin Resource Sharing
npm i cors
npm i –save-dev @types/cors
```

```
import express, { Application, Request, Response } from "express";
import cors from "cors";
const PORT = process.env.PORT ?? 8000;
const app: Application = express();
app.use(cors()); // Allow CORS on ALL routes
app.get("/", (req: Request, res: Response) => {
  res.send("Hello World!");
});
app.listen(PORT, () => {
  console.log(`Listening to port ${PORT}`);
});
```

Opt1: Allow CORS on all routes

```
import express, { Application, Request, Response } from "express";
import cors from "cors";
const PORT = process.env.PORT ?? 8000;
const app: Application = express();
app.get("/", (req: Request, res: Response) => {
  res.send("Hello World!");
});
app.get("/app", cors(), (q: Request, p: Response) => {
  res.send("This route allows CORS");
});
app.listen(PORT, () => {
  console.log(`Listening to port ${PORT}`);
});
```

Opt2: Allow CORS on specific routes