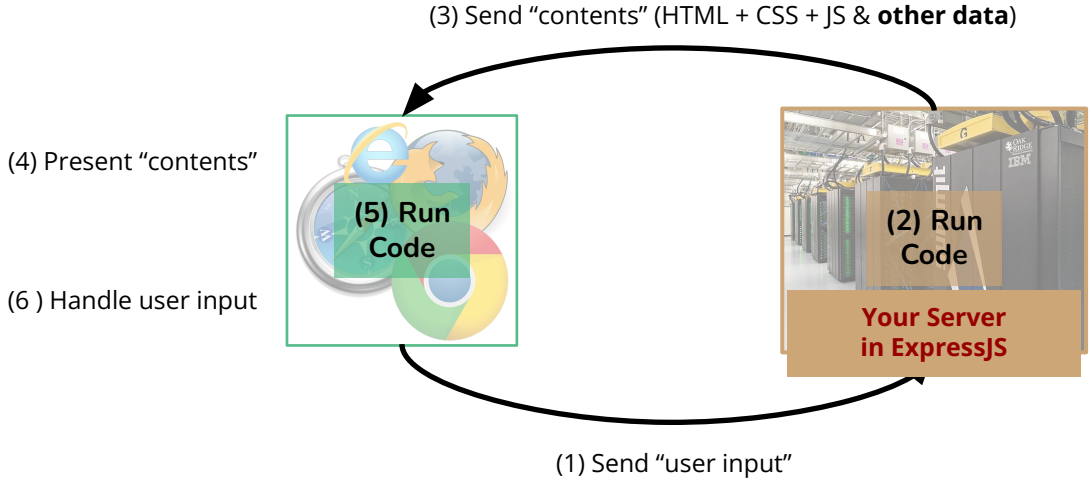
ExpressJS

HTTP server in (Java|Type)Script

Web Client/Server Architecture



Topics Covered

- 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

Why Use an App Server

- Computing power
  - Client code runs on your machine with *limited computing power*
  - App servers run on (external) machines with *more computing power*
- *Leaked company secret*: client code loaded to the user's browser can be viewed on the user's browser
- Be aware of *extra network transport overhead*
- Possible use cases
  - Proxy server to access web services which prevent CORS
  - Aggregate data from both web clients and mobile clients
  - Web apps that require results from heavy computation (machine learning, computational simulations, data mining, etc.)

Alternatives to ExpressJS

koa

fastify

django

Laravel

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

# Create tsconfig.json
```

tsconfig.json

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

Hello World

my-server.ts

```
import express, {Application, Request, Response} from “express”;

const app:Application = express();
const PORT = process.env.PORT ?? 8000;    // Allow dynamic PORT setting (Heroku)

// 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}`);
});
```

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

Browser localhost:8000

Defining More Routes/EndPoints

my-server.ts

```
// 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(“_____”);
});
```

Browser localhost:8000

Browser localhost:8000/about

Can’t invoke these
from browser omnibox

Sending Responses

Opt #1: file attachment

```
// 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 #2: plain text

```
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 #3: HTML string

```
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
})
```

Sending Responses

Opt #4: JSON

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

Opt #5: JSON from object

```
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);
});
```

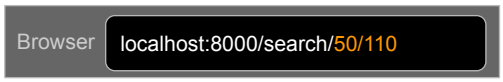
Opt #6: empty response

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

Parsing Dynamic Path/Route Names

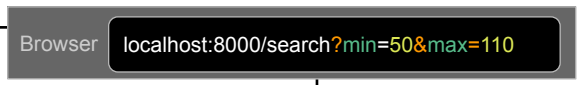
```
app.get("/user/:uname", (req:Request, res:Response) => {
  const who = req.params.uname;
  if (who == "___")
    res.send(`Option #1 here`);
  else
    res.send(`Option #2 here`);
})
```


```
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}`);
})
```

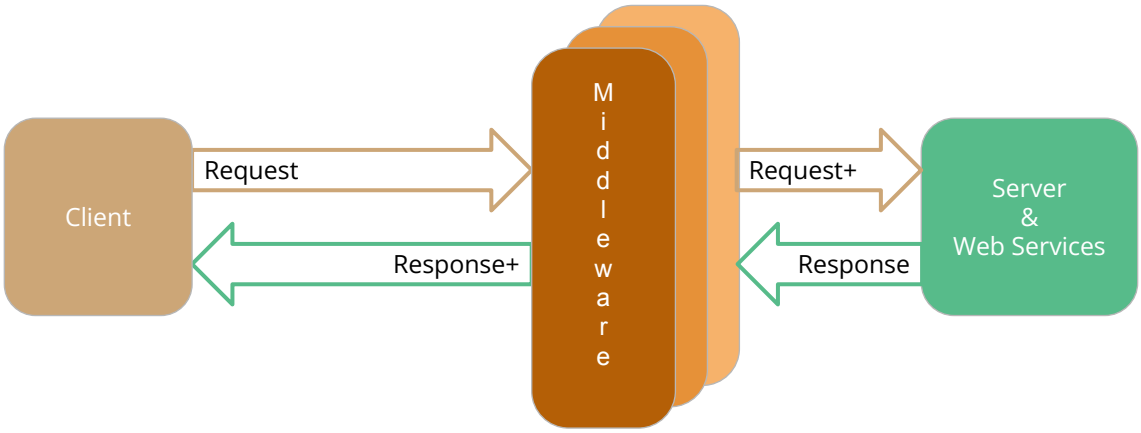


Parsing Query Parameters (TypeScript types)

```
type Query2 = {
  min: number;
  max: number;
}
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**
compression	Zlib compression to HTTP responses
cors	Enable Cross-Origin Resource Sharing
morgan	Log incoming requests
multer	Parse multi-part form data in incoming requests

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
```

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.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}`);
});
```

Opt2: Allow CORS on specific 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}`);
});
```

Deployment

Deployment Options

- Where to host?
  - **Render.com (new)**
  - Heroku
  - Google Cloud Platform
  - Amazon Web Services
  - Microsoft Azure
  - Digital Ocean
  - Platform.sh
  - *Can't deploy to static hosting providers, look for NodeJS hosting providers*
- Deployment Architectures
  - Individual NodeJS App
  - Docker Containers (with Docker Compose)

Deploy to Render.com

Step 1: Prepare Your Files

package.json

```
{
  “script”: {
    “build”: “npm install typescript; tsc”,
    “serve”: “node dist/myserver.js”
  }
}
```

.gitignore

```
node_modules
dist
package-lock.json
yarn.lock
```

tsconfig.json

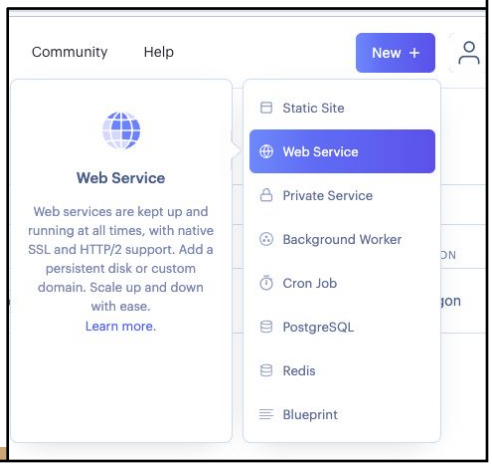
```
{
  “compilerOptions”: {
    “target”: “es6”,
    /* other options not shown */
    “outDir”: “dist”
  },
  “files”: [“myserver.ts”]
}
```

```
# Edit package.json, tsconfig.json
# test your app locally
npm install -g typescript
npm install

npm run build
npm run serve
```

Step 2: Create Repo and Link to Your Render Acct

1. Create a GitHub/GitLab repository for your project
2. Git-push your files to the GitHub/GitLab repo created
3. Sign in to Render.com
4. Create a new Web Service on Render.com
   a. Enter "." for Root Directory ⇒ "."
   b. Select Node Runtime
   c. Enter "npm run build" for Build Command
   d. Enter "npm run serve" for Start Command
   e. Click "Create Web Service"
   f. Connect your Render.com Web Service to your GitHub/GitLab repository



Deploy to Heroku

Prerequisites

- Use process.env.PORT for the HTTP port of your server
- Install Heroku CLI (download link)
- Install Git
- Heroku account
- [GitHub account]

Used to be FREE, but now requires Credit Card Setup

Deploy to Heroku

```
# Edit package.json, tsconfig.json, create Procfile
# test your app locally
npm install
heroku local web
# Login & Deploy
heroku login
# The following commands must be typed from a
# Git repo of your ExpressJS server
heroku create     # will print the project-name

git remote add heroku https://git.heroku.com/project-name.git
git commit        # commit local changes
git push heroku main

# The following will launch the app on a web browser
heroku open
# Check any deployment errors
heroku logs
```

package.json

```
{
  “script”: { “postinstall”: “tsc” },
  “engines”: { “node”: “14.5.1” }
}
```

Postinstall: run the TypeScript compiler (tsc) to compile your-server.ts into build/your-server.js

tsconfig.json

```
{
  “compilerOptions”: {
    “target”: “es6”,
    /* other options not shown */
    “outDir”: “build”
  },
  “files”: [“your-server.ts”]
}
```

Procfile

```
web: node build/your-server.js
```

Procfile: used by Heroku to run your server

Secure HTTP Servers

- Public/Private Key Pair
- SSL Certificate (X509 standard) which includes
  - The public key
  - Server Identity
  - Target Address
  - Expiration Date
  - Certificate Chain (Root CA to Intermediate CA to this certificate)
- Where to get SSL certificates
  - Paid vs Free services
  - Command Line
- Installation
  - Upload the SSL Certificate and the private key

Secure HTTP Server (Self-Signed Certificates)

```
openssl req -nodes -new -x509 -keyout SAMPLE.key -out SAMPLE.cert
# openssl req -new -key SAMPLE.key -out SAMPLE.cert

# You will be prompted several questions
```

```
import express
import fs from “fs”
import https, {createServer} from “https”
const app = express()
const option = { key: fs.readFileSync(“SAMPLE.key”),
                 cert: fs.readFileSync(“SAMPLE.cert”)}
createServer(option, app)
  .listen(9000, () => {
    consoleinfo(“Secure server on port 9000”)
  })
```