Express.JS Quick Notes

Basic Server Code

```
const app = express();

// Define a port
const PORT = 3000;

// Basic route
app.get('/', (req, res) => {
  res.send('Hello, world!');
});

// Start the server
app.listen(PORT, () => {
  console.log(`Server running at http://localhost:${PORT}`);
});
```

Common Get Request

```

app.get('/user', (req, res) => {
  const user = {
    id: 1,
    name: 'Hyxal',
    email: 'hyxal@example.com'
  };

  // Send JSON response
  res.json(user);
});
```

Get Request

- Retrieve or `GET` data from the server.

Status	Meaning
200	OK
301	Moved Permanently
302	Found (Temporary Redirect)
304	Not Modified
400	Bad Request
401	Unauthorized

Status	Meaning
403	Forbidden
404	Not Found
500	Internal Server Error

```
app.get(path,(request,response)=>{
    response.send()
})
```

Route Parameters

```
app.get('/user/:id', (req: Request, res: Response) => {
  const userId = req.params.id;
  res.send(`User ID is: ${userId}`);
});
```

Query Parameters

```

app.get('/search', (req, res) => {
  const term = req.query.term;
  const sort = req.query.sort;

  res.send(`Search term: ${term}, Sort by: ${sort}`);
});
```

POST Request

- Create send or `POST` data to the server

Status	Meaning
201	Created
202	Accepted
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
409	Conflict

Status	Meaning
415	Unsupported Media Type
422	Unprocessable Entity
500	Internal Server Error

```
app.post('/submit', (req, res) => {
  const data = req.body;
  console.log('Received data:', data);

  // Send a response back
  res.status(200).json({ message: 'Data received successfully', data });
});
```

PUT Request

- `PUT` is used for replacing or updating existing resources entirely.

Status	Meaning
200	OK
201	Created
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
409	Conflict
415	Unsupported Media Type
422	Unprocessable Entity
500	Internal Server Error

```
app.put('/user', (req, res) => {
  const userData = req.body;
  res.send(`User updated with name: ${userData.name}`);
});
```

Send status

- instead of sending usual response one can send status as response

|

```
 app.get('/not-found', (req, res) => {
  res.status(404).json({ error: 'Resource not found' });
});
```

PATCH Request

Used to partially update a resource

Status	Meaning
200	OK
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
409	Conflict
415	Unsupported Media Type
422	Unprocessable Entity
500	Internal Server Error

```
 app.patch('/users/:id', (req, res) => {
  const userId = parseInt(req.params.id);
  const updates = req.body;

  const user = users.find(u => u.id === userId);
  if (!user) return res.status(404).send('User not found');

  Object.assign(user, updates); // partial update
  res.json(user);
});
```

DELETE Request

Used to delete and element usually from a database

Status	Meaning
200	OK
202	Accepted
204	No Content
400	Bad Request
401	Unauthorized
403	Forbidden
404	Not Found
500	Internal Server Error

```
app.delete('/users/:id', async (req, res) => {
  const result = await User.findByIdAndDelete(req.params.id);
  if (!result) return res.status(404).send('User not found');
  res.sendStatus(204);
});
```

Middleware

- Middleware functions are functions that have access to:
  - `req` (request)
  - `res` (response)
  - `next` (function to pass control to the next middleware)

```
function middleware(req, res, next) {
  // logic
  next();
}
```

Middleware Execution Flow

- Middleware functions are executed in order they are defined.
- You must call next() to move to the next middleware.

Types of Middleware

1. Application-Level Middleware

- Bound to an instance of the Express app.

```
app.use((req, res, next) => {
  console.log('App-level middleware');
  next();
});
```

2. Router-level Middleware

- Similar to application-level middleware but applied to an `express.Router()` instance.
- Useful for modular route handling.

```
const express = require('express');
const router = express.Router();

// Router-level middleware
router.use((req, res, next) => {
  console.log('Router middleware triggered');
  next();
});

// Example route
router.get('/', (req, res) => {
  res.send('Hello from the router!');
});

// Use the router in the app
app.use('/api', router);
```

3. Built-in Middleware

Express comes with several built-in middleware functions that help handle common tasks.

✅ Common Built-in Middleware

```
// Parses incoming JSON payloads (Content-Type: application/json)
app.use(express.json());

// Parses URL-encoded data (from HTML form submissions)
app.use(express.urlencoded({ extended: true }));

// Serves static files from a directory
app.use(express.static('public'));
```

4. Error-handling Middleware

- Defined with four parameters: (err, req, res, next)
- Used to catch and respond to errors.
- Must be placed after all other middleware and routes.

```
app.use((err, req, res, next) => {
  console.error(err.stack);
  res.status(500).json({ error: 'Something went wrong!' });
});
```

5. Third-party Middleware:

- Third-party middleware are external modules installed via npm, and used via `require()` and `app.use()`.
- `npm install cors`

```
const cors = require('cors');
app.use(cors());  // Allow all origins by default
```

Validation

`npm install express-validator`

Example Usage

```
const { body, validationResult } = require('express-validator');

app.post('/register',
  // Middleware array of validators
  [
    body('email').isEmail(),
    body('password').isLength({ min: 6 }),
    body('username').notEmpty()
  ],
  (req, res) => {
    // Check validation result
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
      return res.status(400).json({ errors: errors.array() });
    }

    res.send('User is valid!');
  }
);
```

Important Validators:

Validator	Description
`isEmail()`	Must be a valid email
`isLength({ min: n })`	Minimum string length
`notEmpty()`	Field must not be empty
`isNumeric()`	Only numbers

Validator	Description
`isInt()`	Integer only
`isAlphanumeric()`	Letters and numbers only
`isIn(['a', 'b'])`	Must be one of the allowed values
`isURL()`	Must be a valid URL
`isBoolean()`	Must be true or false
`isStrongPassword()`	Strong password (uppercase, symbols)

Custom Validators

```
body('age').custom(value => {
  if (value < 18) {
    throw new Error('Must be at least 18');
  }
  return true;
})
```

Validator as Middlware:

```
// Validator.js
const { body } = require('express-validator');

const validateUser = [
  body('email').isEmail().withMessage('Invalid email'),
  body('password').isLength({ min: 8 }).withMessage('Too short')
];

module.exports = validateUser;

// In route file
const validateUser = require('./validateUser');

app.post('/signup', validateUser, handler);
```

Express Routers

```
const express = require('express');
const router = express.Router();

// Route definition
router.get('/', (req, res) => {
  res.send('Hello from the router!');
});

module.exports = router;
```

```
const userRoutes = require('./routes/user');
app.use('/users', userRoutes);  // Now all routes are prefixed with /users
```

Router Level Middlware:

```
router.use((req, res, next) => {
  console.log('User router middleware');
  next();
});
```

Nested Routers:

```
const express = require('express');
const userRouter = express.Router();
const postRouter = express.Router({ mergeParams: true });

userRouter.use('/:userId/posts', postRouter);

postRouter.get('/', (req, res) => {
  res.send(`Posts for user ${req.params.userId}`);
});

module.exports = userRouter;
```

Cookies:

- Cookies are small pieces of data stored on the client-side and sent with every HTTP request. Express uses the cookie-parser middleware to handle cookies easily.
- `npm install cookie-parser`

Example

```
app.get('/set', (req, res) => {
  res.cookie('username', 'hyxal', { maxAge: 900000, httpOnly: true });
  res.send('Cookie set!');
});
```

Reading Cookies

```
app.get('/get', (req, res) => {
  const user = req.cookies.username;
  res.send(`Hello ${user}`);
});
```

Signed Cookies:

- Cookies Defined to stop tampering

```
// Set a signed cookie
res.cookie('session', 'abc123', { signed: true });

// Access signed cookie
const session = req.signedCookies.session;
```

Cookie Options

Option	Description
`maxAge`	Milliseconds until cookie expires
`expires`	Exact date when cookie should expire
`httpOnly`	Not accessible via JavaScript (`document.cookie`)
`secure`	Only sent over HTTPS
`path`	URL path where the cookie is valid
`domain`	Domain where the cookie is accessible
`signed`	Sign the cookie to detect tampering
`sameSite`	Control cross-site cookie sending (`'strict'`, `'lax'`, `'none'`)

Clearing Cookies:

```
res.clearCookie('username');
res.send('Cookie cleared');
```

Sessions

- Sessions allow you to store user data on the server across multiple HTTP requests. Unlike cookies (stored on the client), session data stays on the server, and a unique session ID is sent to the client via a cookie.
- `npm install express-session`

```
const session = require('express-session');

app.use(session({
  secret: 'your_secret_key',
  resave: false,
  saveUninitialized: false,
  cookie: { secure: false } // Set to true in production with HTTPS
}));
```

Storing Data in Session

```
app.get('/login', (req, res) => {
  req.session.user = 'hyxal';
  res.send('User stored in session');
});
```

Accessing Session Data

```
app.get('/profile', (req, res) => {
  const user = req.session.user;
  res.send(`Welcome back, ${user}`);
});
```

Destorying as session cookie

```
app.get('/logout', (req, res) => {
  req.session.destroy(err => {
    if (err) {
      return res.send('Error');
    }
    res.clearCookie('connect.sid'); // default session cookie name
    res.send('Logged out');
  });
});
```