Walchand College of Engineering, Sangli
Department of Computer Science and Engineering

Practical No. 7
Study and implementation of Express.js

Problem Statement 1: Basics of Express.js

1. **What is Express.js and how does it differ from Node.js?**
   Express.js is a minimal and flexible web application framework for Node.js that provides a robust set of features—such as an HTTP utility methods, routing, template engine integration, and middleware support—for building web and mobile applications Express. Node.js is a JavaScript runtime built on Chrome's V8 engine that allows JavaScript to run server-side and supplies low-level APIs (e.g., for HTTP, file system, streams), on which frameworks like Express are built AltexSoftMDN Web Docs.

2. **How do you create a simple Express.js server?**
   Install Express with `npm install express --save`, then in your main file write:

javascript
CopyEdit

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Hello World!')
})

app.listen(port, () => {
  console.log(`Server listening on port ${port}`)
})
```

This code imports Express, creates an app instance, defines a route for GET "/", and starts listening on port 3000 Express.

3. Explain the concept of routing in Express.js. How do you define routes?
   Routing in Express maps HTTP methods and URL paths to handler functions. You define routes using methods like `app.get(path, handler)`, `app.post(path, handler)`, etc. Paths can include static segments (e.g., `/about`), parameters (e.g., `/users/:id`), and patterns (e.g., `/ab?cd`) MDN Web Docs. A handler receives `(req, res)` and sends a response.

4. What is middleware in Express.js, and how does it work?
   Middleware are functions with signature `(req, res, next)` that execute in sequence for each incoming request. They can inspect or modify `req` and `res`, end the request-response cycle (by sending a response), or call `next()` to pass control to the next middleware. Common uses include body parsing, logging, authentication, and error handling Express.

5. How do you create and use custom middleware in an Express.js application?
   A custom middleware is any function taking `(req, res, next)`. For example:

javascript
CopyEdit

```
function myLogger(req, res, next) {
  console.log(`${req.method} ${req.url}`)
  next()
}
app.use(myLogger)
```

This logs each request's method and URL, then calls `next()` to continue processing Express.

6. What is the difference between application-level middleware and router-level middleware?
   Application-level middleware is bound to the entire app via `app.use()` or `app.METHOD()`, so it runs for every request (or matching path) in the application. Router-level middleware is bound to an `express.Router()`

instance via `router.use()` or `router.METHOD()`, so it only runs for requests routed through that particular router Cantech.

7. What are req and res in Express.js? Give examples of common properties and methods associated with each.
   `req` is the incoming HTTP request, with properties like `req.params` (route parameters), `req.query` (URL query string), `req.body` (parsed request body), and methods like `req.get(field)` to read headers Express.
   `res` is the HTTP response, with methods like `res.send(body)` to send a response, `res.json(obj)` to send JSON, `res.status(code)` to set the status code, and `res.redirect(url)` to redirect Express.

8. How would you extract query parameters from a URL in an Express.js route?
   Query parameters are available on `req.query`. For a URL such as `/search?term=node`, inside the route handler `const term = req.query.term;` gives `"node"` Express.

9. How does Express.js handle different HTTP methods (GET, POST, PUT, DELETE)?
   Express provides one method per HTTP verb on the app or router object, for example `app.get(path, handler)`, `app.post(path, handler)`, `app.put(path, handler)`, and `app.delete(path, handler)`. It matches incoming requests by comparing `req.method` and `req.url` to dispatch to the correct handler Express.

10. What are route parameters in Express.js? How do you use them in a route definition?
    Route parameters are named URL segments prefixed with `:` in the route path. For example, in `app.get('/users/:userId/books/:bookId', ...)`, `userId` and `bookId` become keys in `req.params` with the corresponding values from the URL (e.g., `/users/34/books/8989` yields `req.params = { userId: '34', bookId: '8989' }`) Express.

Problem Statement 2 : Basic Web Server with Express.js

```
const express = require('express')
const app = express()
const port = 3000

app.get('/', (req, res) => {
  res.send('Welcome to the Home Page')
})

app.get('/about', (req, res) => {
  res.send('This is the About Page')
})

app.get('/contact', (req, res) => {
  res.send('Contact us at: email@example.com')
})

// 404 handler
app.use((req, res) => {
  res.status(404).send('Page Not Found')
})

app.listen(port, () => {
  console.log(`Server listening on port ${port}`)
})
```

Welcome to the Home Page

This is the About Page

Contact us at: email@example.com

Problem Statement 3: Dynamic Route Parameters

```
const express = require('express')
const app = express()
const port = 3000

// Existing routes
app.get('/', (req, res) => {
  res.send('Welcome to the Home Page')
})

app.get('/about', (req, res) => {
  res.send('This is the About Page')
})

app.get('/contact', (req, res) => {
  res.send('Contact us at: email@example.com')
})

// New route: dynamic user ID
app.get('/users/:id', (req, res) => {
  const { id } = req.params
```

```
  res.send(`User ID: ${id}`)
})

// New route: dynamic product category and ID, responds with
JSON
app.get('/products/:category/:productId', (req, res) => {
  const { category, productId } = req.params
  res.json({ category, productId })
})

// 404 handler for any other routes
app.use((req, res) => {
  res.status(404).send('Page Not Found')
})

app.listen(port, () => {
  console.log(`Server listening on port ${port}`)
})
```

User ID: saurabh