Practical-5 &6

Task-1

Aim: Develop an ExpressJS API to perform CRUD operations using In Memory Database & Test it.

Theoretical Background:

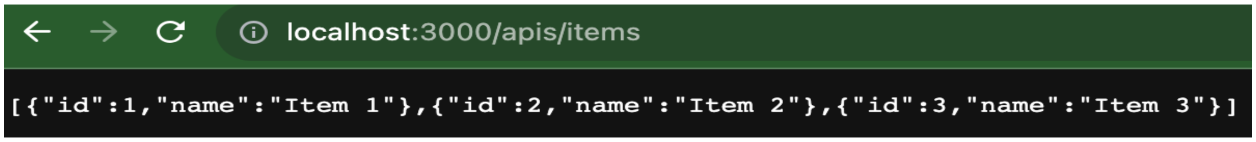
➔ ExpressJS: It's a fast, unopinionated web framework for Node.js, providing a robust set of features for building APIs and web applications.

➔ In-memory database: It's a database that stores data in the server's memory, typically using data structures like arrays or objects, rather than writing it to a physical storage medium like a disk. Since data is volatile in memory, it's used for testing and prototyping.

Source Code:

```
const express = require('express');
const app = express();
const PORT = process.env.PORT || 3000;
app.use(express.json());
// In-memory database (replace with an actual database for production)
let data = [];
// CRUD operations
// 1. Create (POST)
app.post('/apis/items', (req, res) => {
 const {id,name} = req.body;
 const newItem={id,name}
 data.push(newItem);
 res.json(newItem);
});
// 2. Read (GET all items)
app.get('/apis/items', (req, res) => {
 res.json(data);
});
// 3. Read (GET single item by ID)
app.get('/apis/items/:id', (req, res) => {
 const id = parseInt(req.params.id);
 const item = data.find(item => item.id === id);
 if (item) {
   res.json(item);
 } else {
   res.status(404).json({ message: 'Item not found' });
 }
});
// 4. Update (PUT)
app.put('/apis/items/:id', (req, res) => {
 const id = parseInt(req.params.id);
 const updatedItem = req.body;
 const index = data.findIndex(item => item.id === id);
 if (index !== -1) {
   data[index] = { ...data[index], ...updatedItem };
   res.json(data[index]);
 } else {
   res.status(404).json({ message: 'Item not found' });
 }
});

// 5. Delete (DELETE)
app.delete('/apis/items/:id', (req, res) => {
 const id = parseInt(req.params.id);
 const index = data.findIndex(item => item.id === id);
 if (index !== -1) {
   data.splice(index, 1);
   res.json({ message: 'Item deleted successfully' });
 } else {
   res.status(404).json({ message: 'Item not found' });
 }
});

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

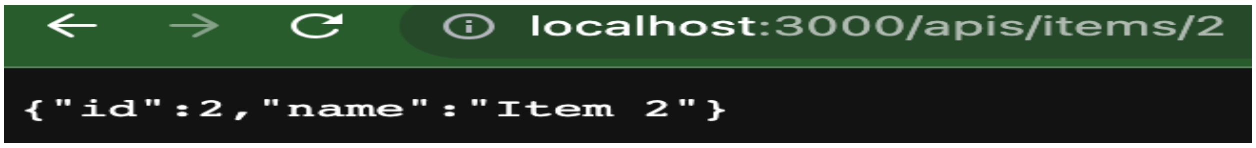
Output: //post-1,2,3



localhost:3000/apis/items

```
[{"id":1,"name":"Item 1"},{"id":2,"name":"Item 2"},{"id":3,"name":"Item 3"}]
```

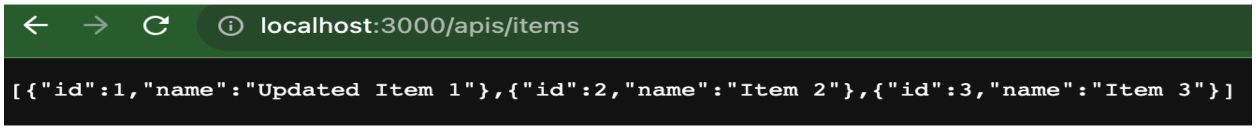
//get-2



localhost:3000/apis/items/2

```
{"id":2,"name":"Item 2"}
```

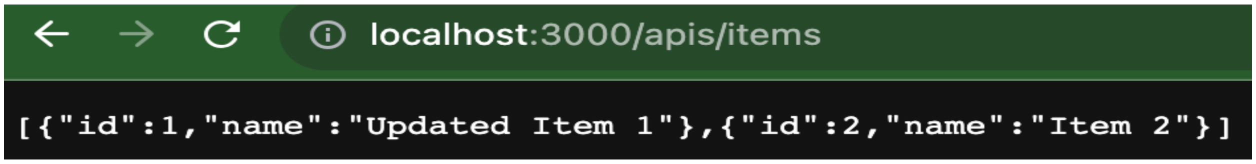
//update-1



localhost:3000/apis/items

```
[{"id":1,"name":"Updated Item 1"},{"id":2,"name":"Item 2"},{"id":3,"name":"Item 3"}]
```

//delete-3



localhost:3000/apis/items

```
[{"id":1,"name":"Updated Item 1"},{"id":2,"name":"Item 2"}]
```

Task-2

Aim: Validate API-Form/Data at server-side using validator/joi module

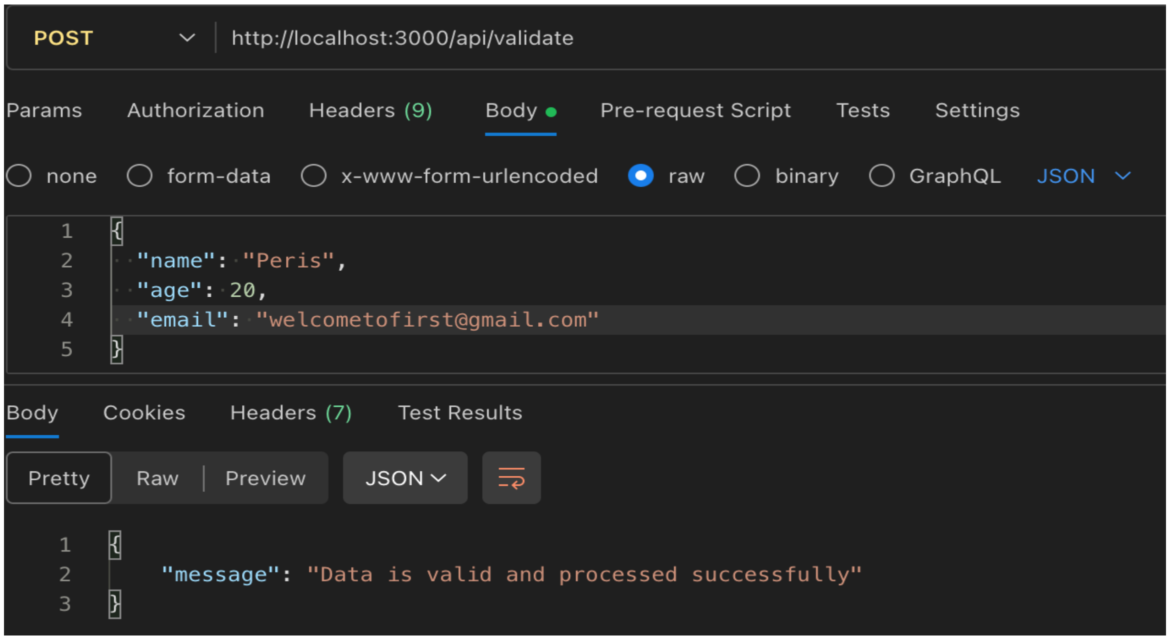
Theoretical Background:

➔ Joi is a powerful validation library for Node.js that allows you to define schemas to validate API form/data at the server-side. It provides a straightforward and expressive way to specify validation rules for various data types, ensuring that incoming data meets specific requirements before processing, which helps improve the reliability and security of your applications.

Source Code:

```
const express = require('express');
const Joi = require('joi');
const app = express();
// Middleware to parse JSON data in the request body
app.use(express.json());

// Define a Joi schema for validation
const schema = Joi.object({
 name: Joi.string().required(),
 age: Joi.number().integer().min(18).max(99).required(),
 email: Joi.string().email().required(),
});

// Define your API endpoint
app.post('/api/validate', (req, res) => {
 // Validate the data using the Joi schema
 const { error } = schema.validate(req.body);

 if (error) {
   // If validation fails, send an error response
   return res.status(400).json({ error: error.details[0].message });
 } else {
   // If validation passes, process the data
   // ... Your code to process the data goes here ...
   return res.status(200).json({ message: "Data is valid and processed successfully" });
 }
});

// Start the server
const PORT = 3000;
app.listen(PORT, () => {
 console.log(`Server started on http://localhost:${PORT}`);
});
```

Output:


POST http://localhost:3000/api/validate

Params | Authorization | Headers (9) | Body ● | Pre-request Script | Tests | Settings

none | form-data | x-www-form-urlencoded | [✓ raw] | binary | GraphQL | JSON

```
{
  "name": "Peris",
  "age": 20,
  "email": "welcometofirst@gmail.com"
}
```

Body | Cookies | Headers (7) | Test Results

Pretty | Raw | Preview | JSON

```
{
    "message": "Data is valid and processed successfully"
}
```

Task-3

Aim: Create API to upload file & Test it.

Theoretical Background:

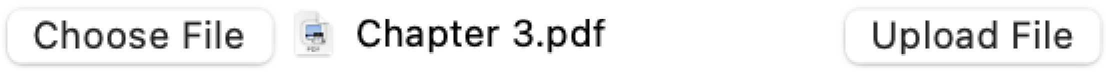
➔ Uploading files via API is a common requirement in web applications. Express, a popular Node.js framework, simplifies file upload handling with middleware like 'express-fileupload.' This middleware allows users to send files as part of their HTTP request, and it automatically parses and stores the files in a designated location. The server can then process, save, or manipulate the uploaded files as needed.

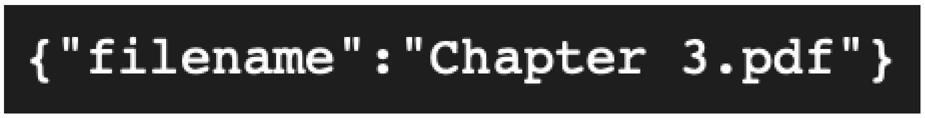
➔ The 'express-fileupload' middleware provides a convenient way to handle file uploads and simplifies the process of retrieving uploaded files from the request object. It exposes the uploaded files as a property of the `req.files` object, allowing easy access to the file data and metadata (e.g., file name, size, MIME type).

Source Code:

```
const express = require('express');
const fileUpload = require('express-fileupload');
const app = express();
const PORT = 3000; // You can change the port as needed

// Middleware to handle file uploads
app.use(fileUpload());

// API endpoint for file upload
app.post('/upload', (req, res) => {
 if (!req.files || Object.keys(req.files).length === 0) {
  return res.status(400).json({ error: 'No files were uploaded.' });
 }

 // Access the uploaded file using req.files.your_file_input_name
 const uploadedFile = req.files.your_file_input_name;
 console.log(uploadedFile);

 // Process the file as needed (e.g., save it to a specific location)
 // For demonstration purposes, we'll just respond with the file name
 const filename = uploadedFile.name;
 res.json({ filename: filename });
});

// Start the server
app.listen(PORT, () => {
 console.log(`Server started on port ${PORT}`);
});

//pr5_3_form.html
<form action="http://localhost:3000/upload" method="post" enctype= "multipart/form-data">
  <input type="file" name="your_file_input_name" />
  <input type="submit" value="Upload File" />
 </form>
```

Output:





```
{"filename":"Chapter 3.pdf"}
```

Task-4

Aim: Create Contact us API to send emails to predefined admins using the NodeMailer module

Theoretical Background:

➔ The nodemailer module is a popular library for sending emails using Node.js. It provides a way to programmatically send emails via various email providers (SMTP, Gmail, etc.). Using it ensures reliable and secure communication between users and administrators.

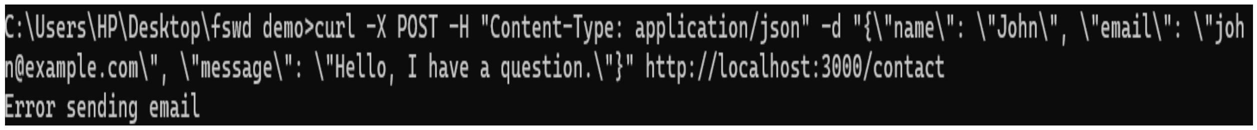
➔ API Routing: Set up a specific API route, like /contact, to handle incoming requests from users who want to contact the admin.

➔ User Input: The user provides their information, such as name, email, and message, through the request body or query parameters.
➔ Predefined Admin: A predefined admin email address is stored in your application's configuration or database. This is where the contact emails will be sent.
➔ Email Composition: Use the nodemailer library to create an email. This includes specifying the sender, recipient (admin), subject, and message content.
➔ Sending the Email: Utilize the configured nodemailer transport to send the email to the admin's email address.

Source Code:

```
const express = require('express');
const bodyParser = require('body-parser');
const nodemailer = require('nodemailer');
const app = express();
app.use(bodyParser.json());
// Configure nodemailer
const transporter = nodemailer.createTransport({
   service: 'Gmail',
   auth: {
      user: 'demo@gmail.com',
      pass: 'your-password'
   }
});
// Define a route for the Contact Us API
app.post('/contact', (req, res) => {
   const { name, email, message } = req.body;
   const mailOptions = {
      from: 'demo@gmail.com',
      to: 'admin@example.com',
      subject: 'New Contact Us Message',
      text: `Message from: ${name}\nEmail: ${email}\n\n${message}`
   };

   transporter.sendMail(mailOptions, (error, info) => {
      if (error) {
         console.error('Error sending email:', error);
         res.status(500).send('Error sending email');
      } else {
         console.log('Email sent:', info.response);
         res.status(200).send('Email sent successfully');
      }
   });
});
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
   console.log(`Server is running on port ${PORT}`);
});
```

Output:



```
C:\Users\HP\Desktop\fswd demo>curl -X POST -H "Content-Type: application/json" -d "{\"name\": \"John\", \"email\": \"joh
n@example.com\", \"message\": \"Hello, I have a question.\"}" http://localhost:3000/contact
Error sending email
```

Task-5

Aim: Create a program that uses the os module to display all the environment variables on the system.

Theoretical Background:

➔ An environment variable is a dynamically named value that can affect the way running processes behave on a computer. They are part of the environment in which a process runs, providing configuration and runtime information to programs and scripts.

➔ Environment variables are used to store various pieces of data that need to be accessed by different programs or scripts without hardcoding them directly into the code. This separation of configuration data from code helps improve maintainability and portability.

➔ Configuration Management: Environment variables allow you to separate configuration from code. This makes it easier to change settings without modifying the code itself.

➔ Security: Sensitive information like API keys, passwords, and credentials should not be hardcoded into source code. Storing them as environment variables adds a layer of security.

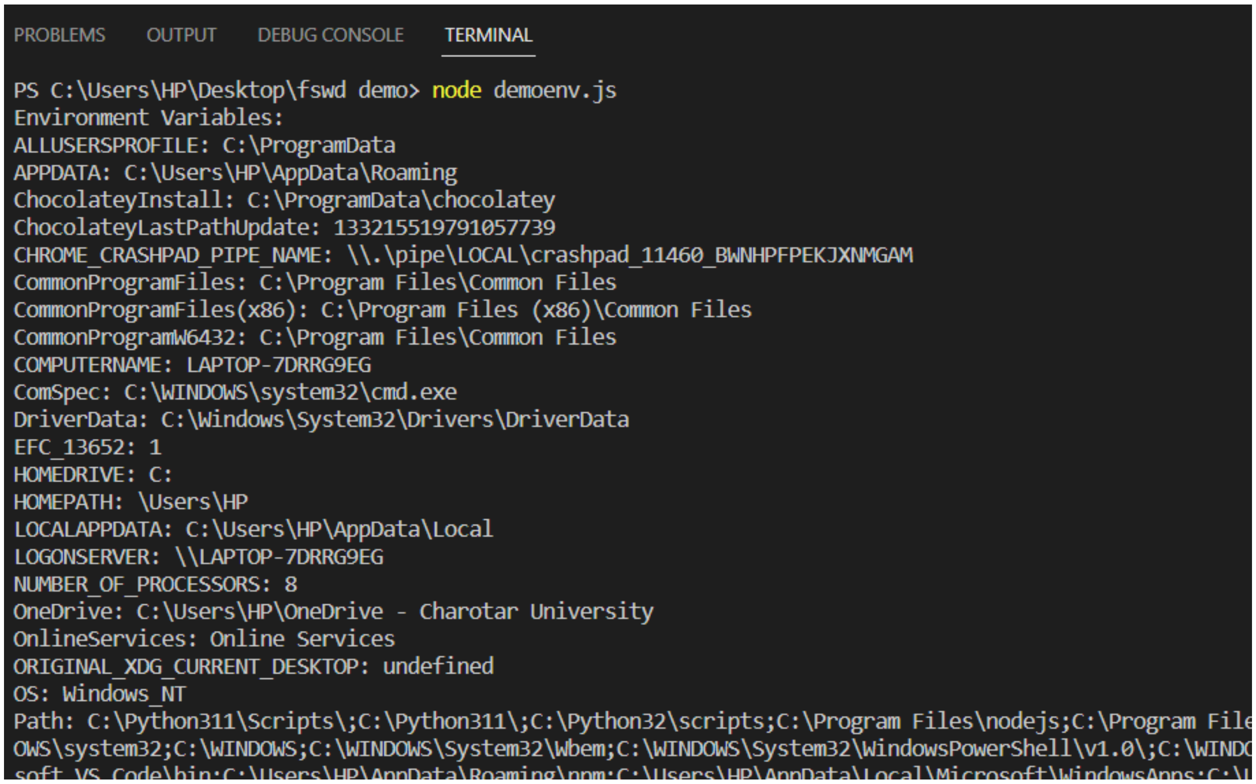
➔ Portability: Code can be deployed across different environments (development, testing, production) without modifications, as configuration changes can be made by adjusting environment variables.

➔ Collaboration: Teams can collaborate on projects without sharing sensitive information directly, as environment variables can be set differently for each user or environment.

Source Code:

```
// Iterate through all environment variables and display them
function displayEnvironmentVariables() {
   console.log("Environment Variables:");
   for (const key in process.env) {
     if (process.env.hasOwnProperty(key)) {
       console.log(`${key}: ${process.env[key]}`);
     }
   }
 }
 // Call the function to display environment variables
 displayEnvironmentVariables();
```

Output:



```
PS C:\Users\HP\Desktop\fswd demo> node demoenv.js
Environment Variables:
ALLUSERSPROFILE: C:\ProgramData
APPDATA: C:\Users\HP\AppData\Roaming
ChocolateyInstall: C:\ProgramData\chocolatey
ChocolateyLastPathUpdate: 133215519791057739
CHROME_CRASHPAD_PIPE_NAME: \\.\pipe\LOCAL\crashpad_11460_BWNHPFPEKJXNMGAM
CommonProgramFiles: C:\Program Files\Common Files
CommonProgramFiles(x86): C:\Program Files (x86)\Common Files
CommonProgramW6432: C:\Program Files\Common Files
COMPUTERNAME: LAPTOP-7DRRG9EG
ComSpec: C:\WINDOWS\system32\cmd.exe
DriverData: C:\Windows\System32\Drivers\DriverData
EFC_13652: 1
HOMEDRIVE: C:
HOMEPATH: \Users\HP
LOCALAPPDATA: C:\Users\HP\AppData\Local
LOGONSERVER: \\LAPTOP-7DRRG9EG
NUMBER_OF_PROCESSORS: 8
OneDrive: C:\Users\HP\OneDrive - Charotar University
OnlineServices: Online Services
ORIGINAL_XDG_CURRENT_DESKTOP: undefined
OS: Windows_NT
Path: C:\Python311\Scripts\;C:\Python311\;C:\Python32\scripts;C:\Program Files\nodejs;C:\Program File
OWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDO
soft VS Code\bin;C:\Users\HP\AppData\Roaming\npm;C:\Users\HP\AppData\Local\Microsoft\WindowsApps;C:\U
```

Task-6

Aim: Implement a function that takes an environment variable name as input and uses the process.env object to display its corresponding value.

Theoretical Background:

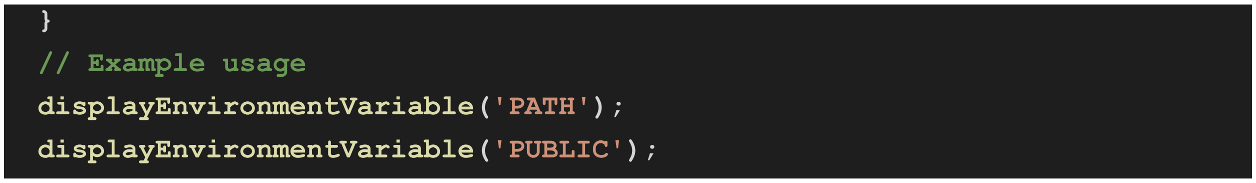
➔ process.env is a built-in object in Node.js that provides access to the environment variables of the operating system in which the Node.js process is running. It's a JavaScript object where each environment variable is represented as a property with the variable's name as the key and its value as the value.

➔ Use process.env to access configuration data, such as database connection strings, API keys, and URLs. This allows you to change settings without modifying the code.

➔ By using process.env, you can prevent sensitive information from being hardcoded into your source code. This improves security by keeping secrets out of version control.

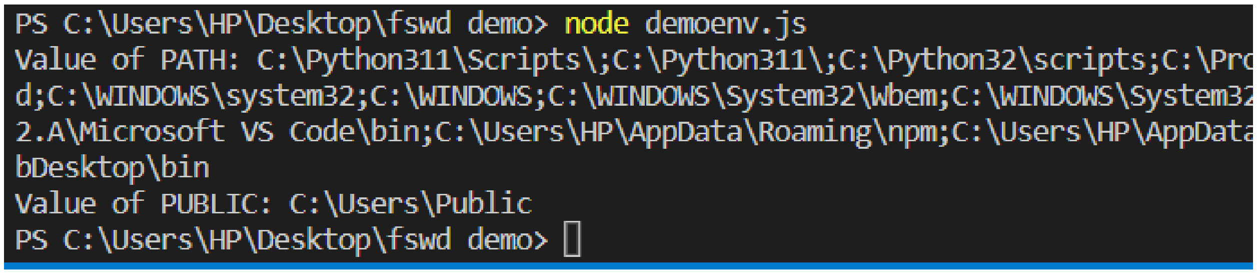
Source Code:

```
function displayEnvironmentVariable(variableName) {
    if (process.env.hasOwnProperty(variableName)) {
    console.log(`Value of ${variableName}: ${process.env[variableName]}`);
    } else {
      console.log(`Environment variable ${variableName} not found.`);
    }
```



```
}
// Example usage
displayEnvironmentVariable('PATH');
displayEnvironmentVariable('PUBLIC');
```

Output:



```
PS C:\Users\HP\Desktop\fswd demo> node demoenv.js
Value of PATH: C:\Python311\Scripts\;C:\Python311\;C:\Python32\scripts;C:\Pro
d;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32
2.A\Microsoft VS Code\bin;C:\Users\HP\AppData\Roaming\npm;C:\Users\HP\AppData
bDesktop\bin
Value of PUBLIC: C:\Users\Public
PS C:\Users\HP\Desktop\fswd demo> 
```

Task-7

Aim:experiments with the dotenv external module and set diff. Credentials for testing, UAT and production environments.

Theoretical Background:

➔ The dotenv module is a tool commonly used in software development to manage environment variables. Environment variables store configuration data and sensitive information outside of the application code, improving security, portability, and maintainability. The dotenv module simplifies the process of loading environment variables from a file into your application.

➔ Benefits of Using dotenv:
Separation of Concerns: Configuration data is kept separate from code, making it easier to change settings without modifying the application logic.
Security: Sensitive information like API keys, database credentials, and passwords are not hard-coded into your codebase, reducing the risk of exposure.
Environment Consistency: dotenv ensures that every team member uses the same environment-specific configurations, avoiding inconsistencies.

Source Code:

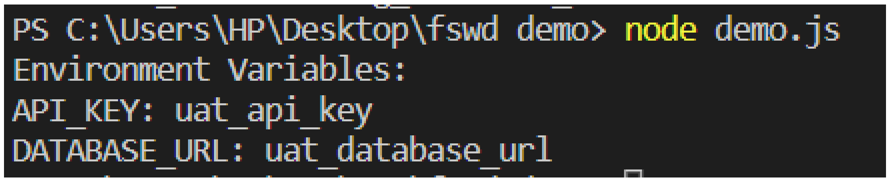
//demo.js

```
const dotenv = require('dotenv');
dotenv.config(); // Load variables from .env file
function displayEnvironmentVariables() {
  console.log("Environment Variables:");
  console.log(`API_KEY: ${process.env.API_KEY}`);
  console.log(`DATABASE_URL: ${process.env.DATABASE_URL}`);
}
displayEnvironmentVariables();
```

//.env

```
# .env for testing
API_KEY=testing_api_key
DATABASE_URL=testing_database_url

# .env for UAT
API_KEY=uat_api_key
DATABASE_URL=uat_database_url

# .env for production
API_KEY=production_api_key
DATABASE_URL=production_database_url
```

Output:



```
PS C:\Users\HP\Desktop\fswd demo> node demo.js
Environment Variables:
API_KEY: uat_api_key
DATABASE_URL: uat_database_url
```

Learning Outcome:

➔ CO1: Understand various technologies and trends impacting single-page web applications.

➔ CO4: Demonstrate the use of JavaScript to fulfill the essentials of front-end development to back-end development