ENVIRONMENTAL MONITORING

IOT-PHASE4

INTRODUCTION

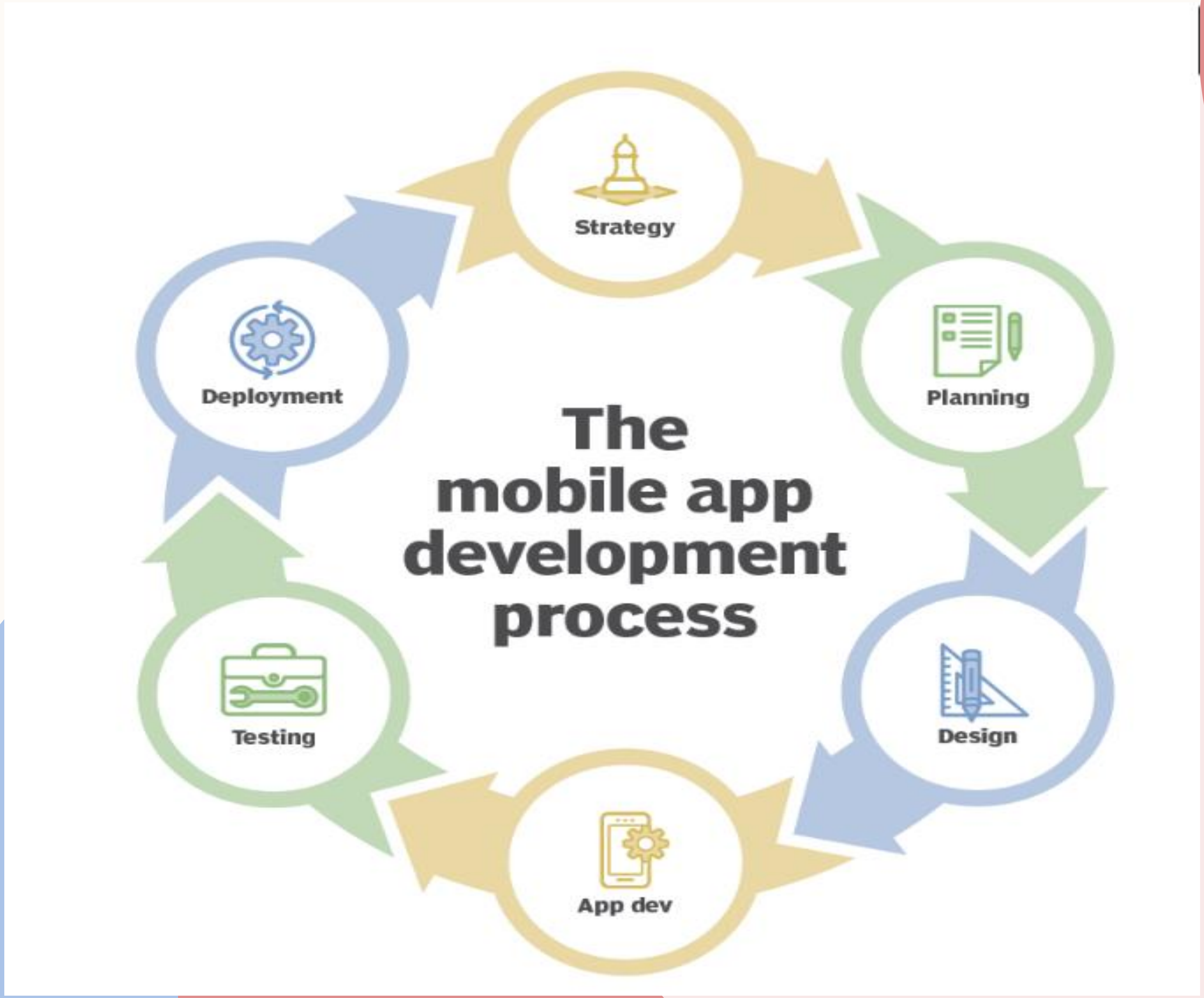
A project is a set of tasks that must be completed within a defined timeline to accomplish a specific set of goals. These tasks are completed by a group of people known as the project team, which is led by a project manager, who oversees the planning, scheduling, tracking and successful completion of projects.

ENVIRONMENT SETUP

In software, web and mobile application development, the development environment is a workspace with a set of processes and programming tools used to develop the source code for an application or software product.

Development environments enable developers to create and innovate without breaking something in a live environment.

IMAGE


Strategy
Planning
Design
App dev
Testing
Deployment
The mobile app development process

HTML STRUCTURE

Overvie

HTML (HyperText Markup Language) files are, basically, just simple text files that you could create in any text editor. But to be displayed correctly on the World Wide Web, an HTML document must be structured correctly.

Any variation from this structure will cause many web browsers to show the content incorrectly or not at all. Also, all HTML documents must have a suffix of "html" for the HTML code to be viewed correctly by a web browser.

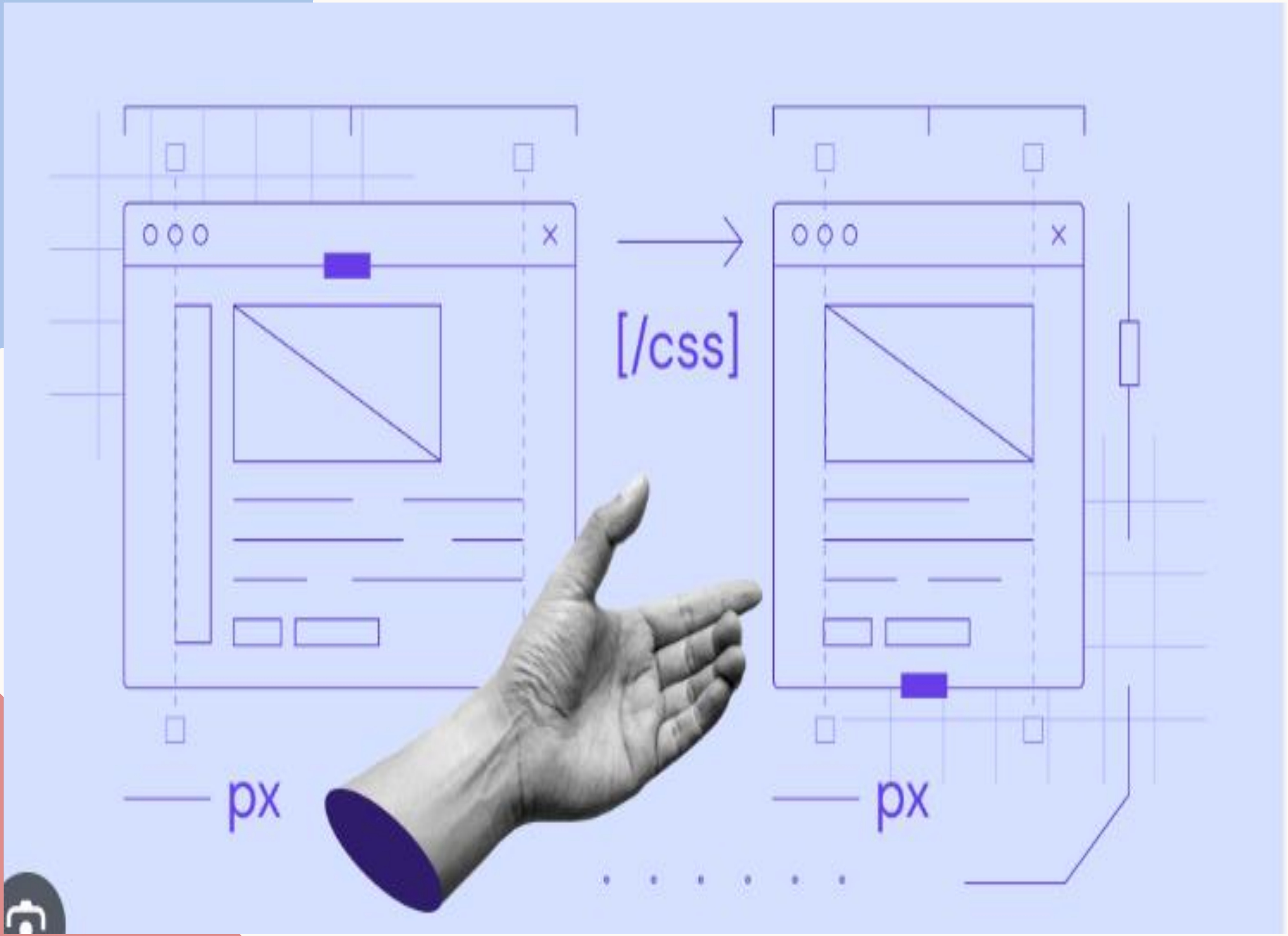
The rule-making body of the Web, the people who determine what this stucture is, is the World Wide Web Consortium (W3C).
Web browsers are supposed to follow these standards as close as possible.
For the most part they all do well, but some browsers fall quite short of this goal (Internet Explorer being the worst culprit). Nonetheless, it is best when you are creating a web page to follow the current web standards. This will ensure your site is viewable in as many browsers as possible. All HTML instruction on this ETS site follows the W3C standards.

EXAMPLE

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset=utf-8">
<title>A Simple HTML Example</title>
<style>
p {text-align:center;}
</style>
</head>
<body>
<h1>This is a header</h1>
<p>This is a paragraph, centered on the page.</p>
<p>And this is the second paragraph with <strong>bold
text</strong>.</p>
</body>
</html>
```

CSS STYLING

Cascading Style Sheets (**CSS**) is a stylesheet language used to describe the presentation of a document written in HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS describes how elements should be rendered on screen, on paper, in speech, or on other media.


[/css]
px
px

JAVASCRIPT FOR REAL-TIME DATA

There was a time when `XMLHttpRequest` was used to make API requests.

It didn't include Promises, and it didn't make for clean JavaScript code. Using jQuery, you could use the cleaner syntax of `jQuery.ajax()`.

Now, JavaScript has its own built-in way to make API requests. This is the Fetch API, a new standard to make server requests with Promises, but which also includes additional features.
In this tutorial, you will create both GET and POST requests using the Fetch API.

TO COMPLETE THIS TUTORIAL, YOU WILL NEED THE FOLLOWING:

- A LOCAL DEVELOPMENT ENVIRONMENT FOR NODE.JS. FOLLOW HOW TO INSTALL NODE.JS AND CREATE A LOCAL DEVELOPMENT ENVIRONMENT.
- A BASIC UNDERSTANDING OF CODING IN JAVASCRIPT, WHICH YOU CAN LEARN MORE ABOUT FROM THE HOW TO CODE IN JAVASCRIPT SERIES.
- AN UNDERSTANDING OF PROMISES IN JAVASCRIPT. READ THE PROMISES SECTION OF THIS ARTICLE ON THE EVENT LOOP, CALLBACKS, PROMISES, AND ASYNC/AWAIT IN JAVASCRIPT.

```
fetch(url).
fetch(url)
  .then(function() {
    // handle the response
  }).
fetch(url)
  .then(function() {
    // handle the response
  })
  .catch(function() {
    // handle the error
  });.
<h1>Authors</h1>
<ul id="authors"></ul>.
<h1>Authors</h1>
<ul id="authors"></ul>

<script>
  const ul = document.getElementById('authors');
```

SERVER SETUP (IF REQUIRED)

In this article, we will create a Node.js proxy that forwards requests to different servers or endpoints.

Proxy server: Proxy servers act as intermediaries between the end-user and the website or API they wish to access.

Why Proxy Server?

In the standard method, the client sends a request directly to the API endpoint to fetch the desired data.

In this case, the node.js proxy will act as an intermediary between the user and the API.

Thus, the user makes a request to the proxy, which is then forwarded to the API endpoint.

The user requests to the proxy endpoint.

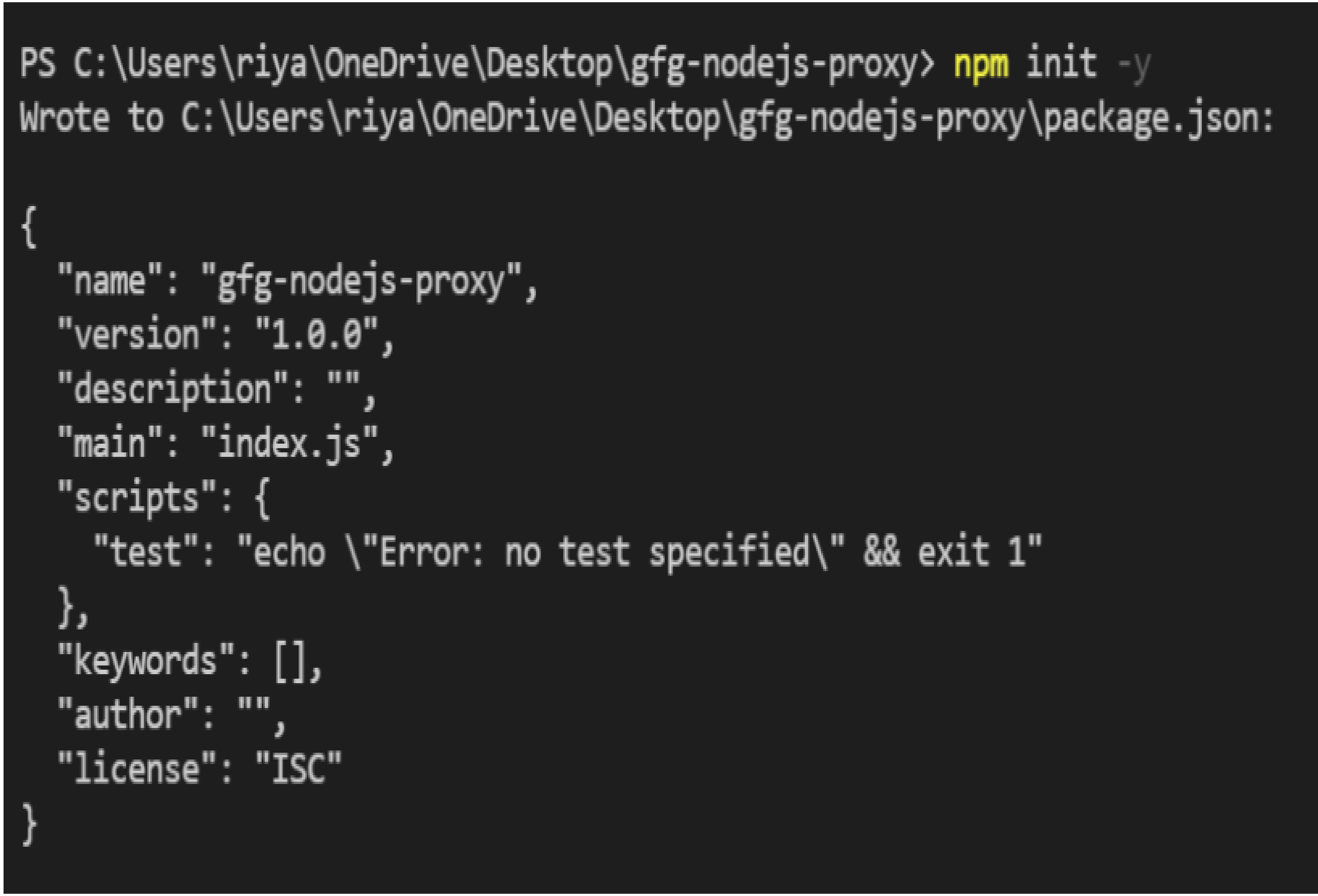
The benefits of using a proxy is to:

- Allow and restrict certain resources of the API.
- Improves the network performance.
- Load Balancing.

Example: Let’s build a node.js proxy server.

Prerequisites: Make sure you have the following applications on your device.

- Node.js (install).
- Visual studio code or any other code editor.
- Postman to test API requests.



```
PS C:\Users\riya\OneDrive\Desktop\gfg-nodejs-proxy> npm init -y
Wrote to C:\Users\riya\OneDrive\Desktop\gfg-nodejs-proxy\package.json:

{
  "name": "gfg-nodejs-proxy",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

STEP 2: INSTALL THE REQUIRED DEPENDENCIES

We need a few packages in our project listed below:

- **express:** It is a node.js framework.
- **http-proxy-middleware:** It is a proxy framework.
- **dotenv:** Loads environment variables.
- **morgan:** Logs the requests.

Install the above packages by running the following command:

```
npm i express http-proxy-middleware
dotenv morgan
```

STEP 3: CREATING PROXY SERVER

CREATE AN APP.JS FILE AND WRITE THE CODE FOR THE PROXY SERVER.

FIRST, WE WILL IMPORT THE INSTALLED PACKAGES INTO OUR PROJECT AND CREATE AN EXPRESS SERVER.

```
CONST EXPRESS = REQUIRE('EXPRESS');
CONST MORGAN = REQUIRE("MORGAN");
CONST { CREATEPROXYMIDDLEWARE } = REQUIRE('HTTP-
PROXY-MIDDLEWARE');
REQUIRE('DOTENV').CONFIG()

// CREATING EXPRESS SERVER
CONST APP = EXPRESS();
```

TO USE THE OPEN WEATHER MAP API, YOU NEED AN API KEY. GO TO HTTPS://OPENWEATHERMAP.ORG/API SIGN IN OR CREATE A NEW ACCOUNT. CLICK ON API KEYS AND COPY THE KEY.

TESTING AND DEPLOYMENT

Testing Process

Before deploying your environmental monitoring platform to a web server or hosting service, it's crucial to thoroughly test the application to ensure it works as expected.

Testing can be broken down into several key steps:

Local Testing

- **Unit Testing:** Test individual components, functions, and modules to ensure they work correctly.
- **Integration Testing:** Verify that different components of your platform work together seamlessly. For example, test data retrieval, processing, and display.
- **End-to-End Testing:** Simulate user interactions with your platform to ensure that it behaves as expected from start to finish.

Data Simulation

If you are using a simulated IoT device, ensure that it generates realistic data for testing. This includes data generation within expected ranges for temperature and humidity.

7.1.3. Data Validation

Check the accuracy and consistency of the data retrieved from your IoT device. Verify that it matches the expected format and values.

Cross-Browser and Cross-Device Testing

Test your platform on various web browsers and devices to ensure compatibility and responsiveness. Check for any rendering issues and address them.

7.1.5. Security Testing

Ensure that the platform is secure. Check for vulnerabilities like cross-site scripting (XSS) and implement security measures as necessary.

7.1.6. Performance Testing

Assess the platform's performance, especially if it will handle a large volume of data. Optimize code, database queries, and resources to ensure responsiveness.

User Acceptance Testing (UAT)

Gather feedback from potential users or stakeholders to validate that the platform meets their expectations and requirements.

7.2. Deployment Options

Once you've completed testing and are confident that your environmental monitoring platform functions correctly, you can proceed with deployment. There are several deployment options to consider:

7.2.1. Self-Hosted Deployment

- **Web Server**: Deploy the platform on a web server of your choice (e.g., Apache, Nginx) if you have access to hosting resources.
- **Server Configuration**: Ensure that the server is configured correctly to support your platform, including necessary server-side technologies (e.g., Node.js, PHP).

7.2.2. Cloud Hosting

- **Platform as a Service (PaaS)**: Deploy your platform on a cloud-based PaaS, such as Heroku, AWS Elastic Beanstalk,or Google App Engine. These platforms simplify server management.
- **Infrastructure as a Service (IaaS)**: Set up virtual machines or containers on cloud providers like AWS, Azure, or Google Cloud and configure them for your platform.

7.2.3. Docker Containerization

- Use Docker to package your application and its dependencies into containers. This simplifies deployment across various platforms and environments.

Continuous Integration and Continuous Deployment (CI/CD)

Set up a CI/CD pipeline to automate the deployment process. Tools like Jenkins, Travis CI, or GitLab CI/CD can help streamline the deployment of code changes.

7.2.5. Monitor and Maintain

After deployment, regularly monitor your platform to ensure it remains operational. Implement error tracking and logging to identify and address issues promptly. Consider setting up automated alerts for critical events.

7.2.6. Scaling

If your platform experiences increased demand, plan for scaling options, such as load balancing, horizontal scaling, or autoscaling, to handle increased traffic and data.

7.2.7. Backup and Recovery

Establish a backup and recovery strategy to safeguard your data and ensure quick recovery in case of unexpected outages or data loss.

Deployment options may vary based on your specific requirements, resources, and constraints. Choose the approach that best suits your project's needs and capabilities.

THANK YOU !