1.Build a basic HTTP server that can handle different routes and HTTP methods (GET, POST, PUT, DELETE).

1. const http = require('http');

Imports the built-in http module, which provides the capability to create a basic HTTP server in Node.js.

2-6. function sendResponse(res, statusCode, message) { ... }

Defines a helper function sendResponse to simplify sending responses.

It accepts three parameters:

res: the response object.

statusCode: the HTTP status code to send with the response.

message: the message to include in the response.

The function sets the response header with res.writeHead(statusCode, { 'Content-Type': 'application/json' });, specifying the status code and content type as JSON.

It then ends the response with res.end(JSON.stringify({ message }));, sending a JSON object containing the message.

7-8. const server = http.createServer((req, res) => { ... });

Creates an HTTP server using http.createServer(), passing a callback function with two parameters:

req: the request object containing information about the incoming request.

res: the response object used to send a response back to the client.

9. const { method, url } = req;

Extracts the HTTP method and URL from the request object using destructuring. These will be used to determine how to handle each route.

10-15. Handling the root route (/)

```
if (url === '/') {
  if (method === 'GET') {
    sendResponse(res, 200, 'Hello, World! This is the root route.');
  } else {
    sendResponse(res, 405, `Method ${method} not allowed on this route.`);
  }
}
```

Checks if the URL is /, the root route of the server.

If the request method is GET, it sends a 200 OK response with a welcome message.

If the request method is not GET, it sends a 405 Method Not Allowed response, indicating that the method is not allowed for this route.

16-23. Handling the /items route

```
else if (url === '/items') {
  if (method === 'GET') {
    sendResponse(res, 200, 'Fetching all items...');
  } else if (method === 'POST') {
    sendResponse(res, 201, 'Creating a new item...');
  } else {
    sendResponse(res, 405, `Method ${method} not allowed on this route.`);
  }
}
```

Checks if the URL is /items.

If the method is GET, it sends a 200 OK response indicating that it is "Fetching all items..."

If the method is POST, it sends a 201 Created response indicating that it is "Creating a new item..."

For any other method, it responds with 405 Method Not Allowed.

24-35. Handling dynamic item routes (/items/:id)

```
else if (url.match(/^\/items\/\d+$/)) {
  const id = url.split('/')[2];
  if (method === 'GET') {
    sendResponse(res, 200, `Fetching item with ID: ${id}`);
  } else if (method === 'PUT') {
    sendResponse(res, 200, `Updating item with ID: ${id}`);
  } else if (method === 'DELETE') {
    sendResponse(res, 200, `Deleting item with ID: ${id}`);
  } else {
    sendResponse(res, 405, `Method ${method} not allowed on this route.`);
  }
}
```

Checks if the URL matches the pattern /items/:id, where :id is a placeholder for a numeric ID (using a simple regex).

Extracts the item ID from the URL by splitting it on / and getting the third part.

If the method is GET, PUT, or DELETE, it responds with a message indicating the action being taken on the item with the specified ID.

For any other method, it responds with 405 Method Not Allowed.

36-38. Handling unknown routes

```
else {
  sendResponse(res, 404, 'Route not found.');
}
```

If none of the defined routes are matched, it sends a 404 Not Found response, indicating that the requested route does not exist.

39-43. Starting the server

```
server.listen(3000, () => {
  console.log('Server is running on http://localhost:3000');
});
```

Tells the server to start listening on port 3000.

Once the server is up, it logs a message to the console with the URL where the server can be accessed.

The code implements a simple HTTP server with basic route handling, using pure Node.js without any additional frameworks. The server supports different routes (/, /items, and /items/:id) and handles various HTTP methods, responding accordingly.

2.Create a server that allows users to upload files and save them to the server's filesystem.

1. const http = require('http');

Imports the built-in http module to create an HTTP server.

2. const formidable = require('formidable');

Imports the formidable library, which is used for parsing form data, especially file uploads.

3. const fs = require('fs');

Imports the fs (file system) module to interact with the file system, allowing operations like reading and writing files.

4. const path = require('path');

Imports the path module, which provides utilities for working with file and directory paths.

5-6. const hostname = '127.0.0.1'; and const port = 3000;

Sets the server hostname to 127.0.0.1 (localhost).

Sets the server port to 3000.

7. const server = http.createServer((req, res) => { ... });

Creates an HTTP server that handles incoming requests with the provided callback function.

This function receives two parameters:

req: the request object, which contains information about the incoming request.

res: the response object, which is used to send a response back to the client.

8-17. Handling GET requests to render the HTML form

```
if (req.method.toLowerCase() === 'get') {
  res.writeHead(200, { 'Content-Type': 'text/html' });
  res.end(`
    <form action="/upload" enctype="multipart/form-data" method="post">
      <input type="file" name="fileupload"><br><br>
      <input type="submit" value="Upload File">
    </form>
  `);
}
```

Checks if the request method is a GET request.

If true, sets the response header with Content-Type as text/html and a 200 OK status code.

Sends an HTML form with a file input field for uploading files and a submit button. The form's action attribute is set to /upload, and it uses POST as the method and enctype="multipart/form-data" to handle file data.

18-41. Handling POST requests to the /upload route for file uploads

```
else if (req.url === '/upload' && req.method.toLowerCase() === 'post') {
  const form = new formidable.IncomingForm();
  form.parse(req, (err, fields, files) => {
    if (err) {
      console.error('Error parsing the form:', err);
      res.writeHead(500, { 'Content-Type': 'text/plain' });
      res.end('An error occurred during the file upload.');
      return;
    }
    const uploadedFile = files.fileupload;
    const oldPath = uploadedFile.filepath;
    const newPath = path.join(__dirname, 'uploads', uploadedFile.originalFilename);
    fs.rename(oldPath, newPath, (err) => {
      if (err) {
        console.error('Error saving the file:', err);
        res.writeHead(500, { 'Content-Type': 'text/plain' });
        res.end('File could not be saved.');
        return;
      }
      res.writeHead(200, { 'Content-Type': 'text/plain' });
      res.end('File uploaded and saved successfully!');
    });
  });
}
```

Checks if the request URL is /upload and the request method is POST.

Creates a new formidable.IncomingForm() instance, which will handle the form parsing.

Calls form.parse(req, (err, fields, files) => { ... }) to parse the incoming request:

If an error occurs, it logs the error and sends a 500 Internal Server Error response.

Extracts the uploaded file object from files.fileupload.

Defines the file's oldPath as the temporary file path and newPath as the destination path in the uploads directory.

Calls fs.rename(oldPath, newPath, (err) => { ... }) to move the file to the new path:

If an error occurs during the move, it logs the error and sends a 500 Internal Server Error response.

Otherwise, it sends a 200 OK response, confirming successful file upload.

42-46. Handling other routes with a 404 response

```
else {
  res.writeHead(404, { 'Content-Type': 'text/plain' });
  res.end('Route not found.');
}
```

If the request does not match any of the defined routes, sends a 404 Not Found response.

47-51. Creating the uploads directory if it doesn’t exist

```
const uploadDir = path.join(__dirname, 'uploads');
if (!fs.existsSync(uploadDir)) {
  fs.mkdirSync(uploadDir);
}
```

Defines uploadDir as the path to the uploads directory.

Checks if the directory exists with fs.existsSync. If not, it creates it using fs.mkdirSync.

52-55. Starting the server and listening on the specified hostname and port

```
server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Calls server.listen() to start the server on 127.0.0.1:3000.

Logs a message to the console confirming that the server is running and providing the URL.