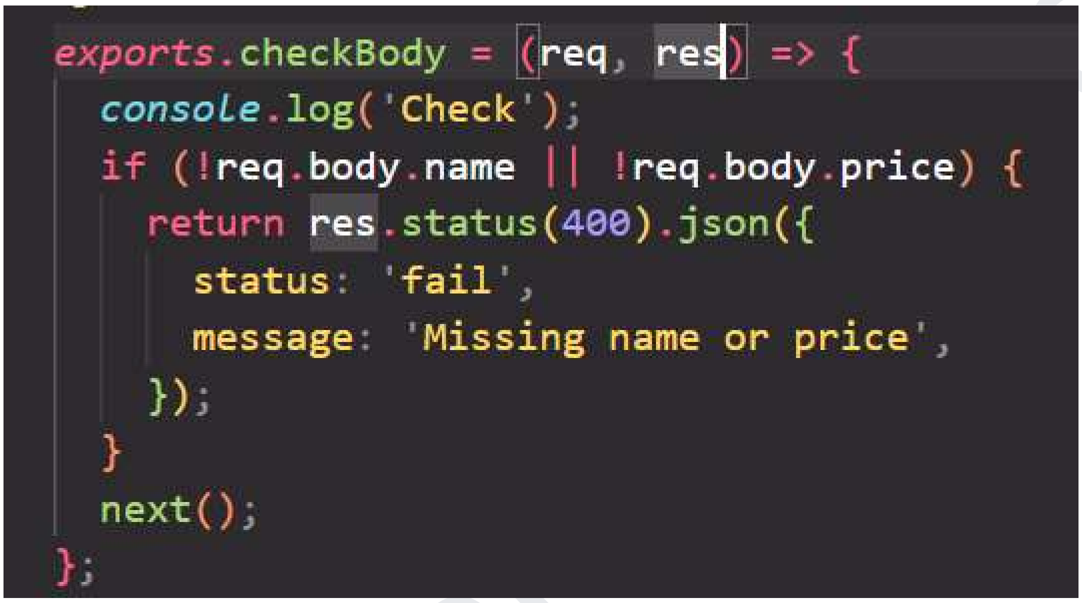
Abhishek Singh

How you know that function is middleware function

Access to `req` and `res` objects: The function has access to the `req` (request) and `res` (response) objects, which are essential for handling HTTP requests and sending responses.
Handling Requests: This function processes a request, in this case, it likely handles an HTTP `PUT` or `PATCH` request to update a "tour" resource.
Sending a Response: The function sends a JSON response back to the client with a status code of 200, indicating success.
Can be Part of a Middleware Stack: Although this function sends the response and ends the request-response cycle, if it didn't, it could call `next()` to pass control to the next middleware function in the stack.



```
exports.checkBody = (req, res) => {
  console.log('Check');
  if (!req.body.name || !req.body.price) {
    return res.status(400).json({
      status: 'fail',
      message: 'Missing name or price',
    });
  }
  next();
};
```

The middleware is designed to check if the `name` and `price` properties exist in the request body before proceeding to the next middleware or route handler.
Logging:

- `console.log('Check');` logs the string `'Check'` to the console every time this middleware is invoked.

Condition Check:

- The `if` statement checks if either `req.body.name` or `req.body.price` is missing (i.e., they are `undefined`, `null`, or `false`).
- If either is missing, the middleware sends a `400 Bad Request` status with a JSON response indicating that the request is missing the `name` or `price` fields.

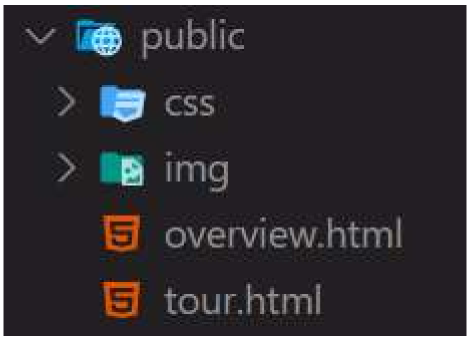
Returning the Response:

- The `return` statement ensures that once the response is sent, the function stops executing, and the `next()` function is not called. This is important because once a response is sent, you don't want to continue with the request processing.

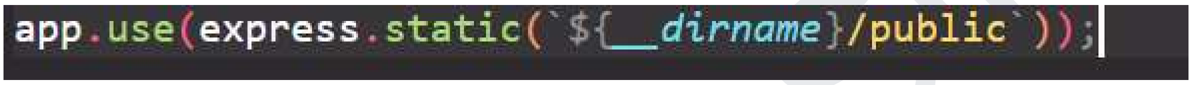
Abhishek Singh
Calling `next()`:

- `next();` is called if the `name` and `price` are both present. This allows the request to move on to the next middleware function or route handler.

Serving Static file

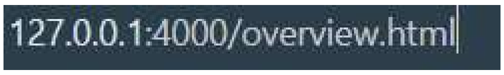


Here overview.html is a static file to access this file we use - express.static and pass the path of file here overview.html is in public folder



```
app.use(express.static(`${__dirname}/public`));
```

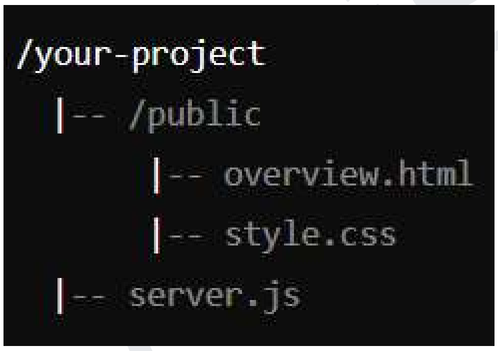
But we would be able to use it using without specifying public in path



```
127.0.0.1:4000/overview.html
```

The reason you're able to access `overview.html` directly via `http://127.0.0.1:4000/overview.html` without specifying the `public` path in the URL is because of how the `express.static` middleware works in Express.js.

If your project directory looks like this:



```
/your-project
  |-- /public
       |-- overview.html
       |-- style.css
  |-- server.js
```

With the `express.static` setup, `overview.html` can be accessed via `http://127.0.0.1:4000/overview.html`.
Similarly, if you had a `style.css` file in the `public` directory, it would be accessible via `http://127.0.0.1:4000/style.css`.

Why This Happens:

Express treats the directory passed to `express.static` as the root directory for static files. Therefore, all files inside the `public` directory are served as if they were in the root

URL, which simplifies the paths you need to use when accessing these resources from the browser.