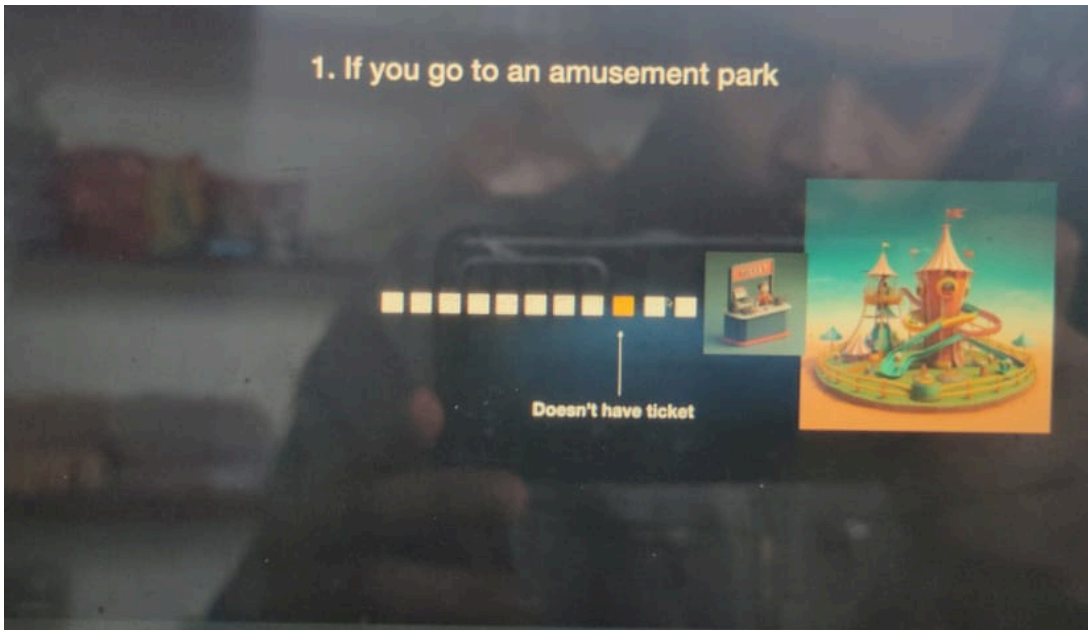
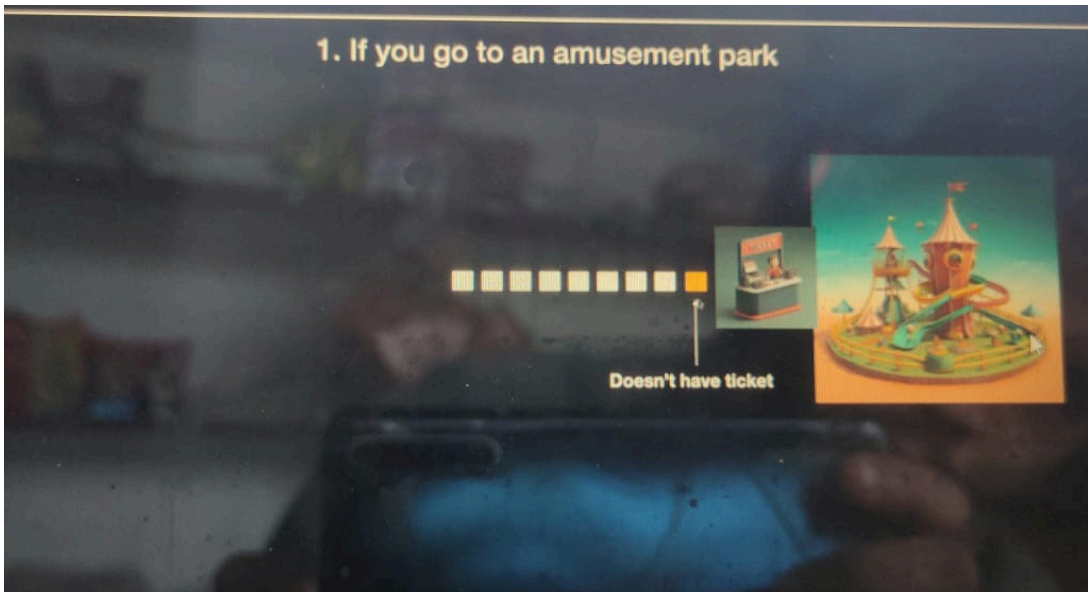
Middleware via examples

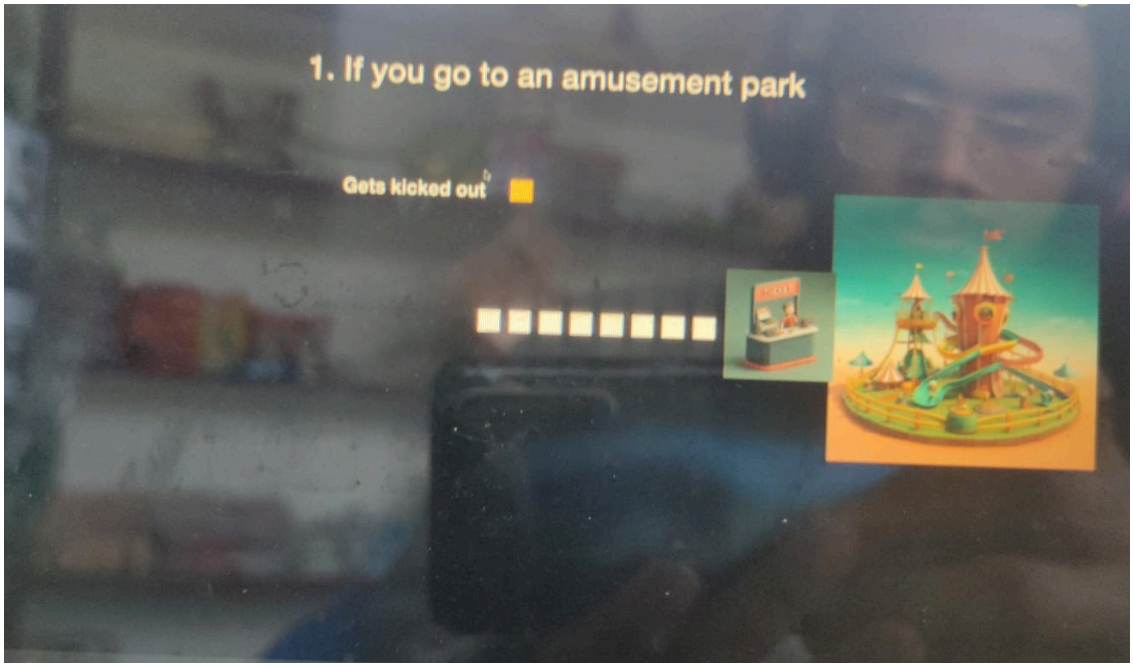
There is an amusement park and their is a wonderful ride there and there is long line in front of that ride and also their is a ticket checker in front of ride, so that nobody is allowed to pass without the ticket



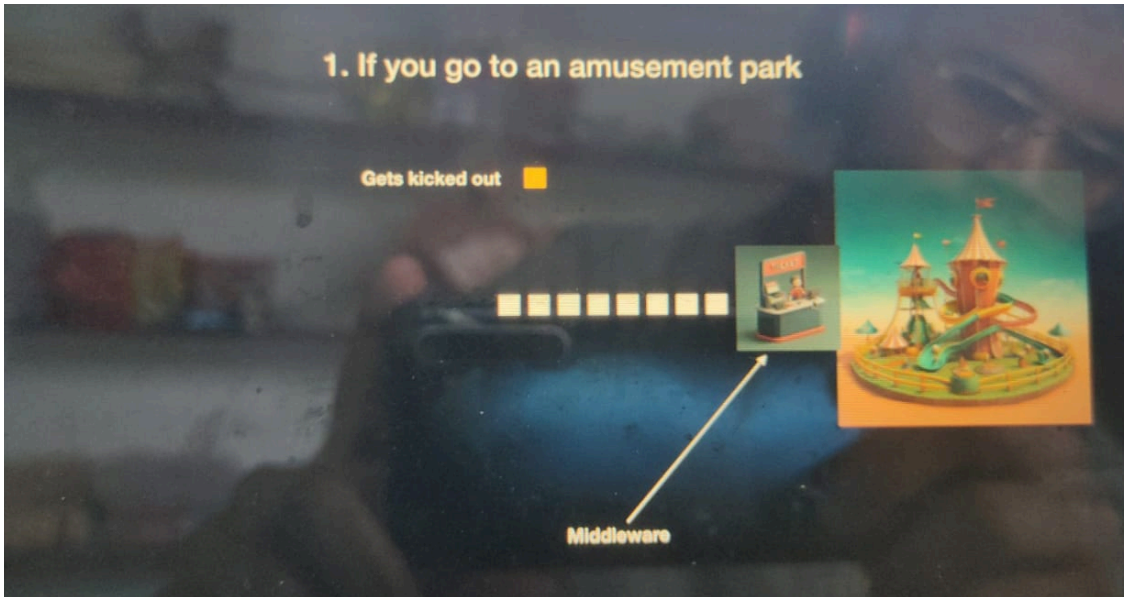
Suppose there is a guy without a ticket



Ticket Checker will kick the guy out



The the rest of people will continue going to the ride . This Ticket Checker is the middleware



A person sitting in the middle of the end client and the final ride you want to access doing certain checks making sure only that people with legitimate access can go through

Using middleware

Express is a routing and middleware web framework that has minimal functionality of its own. An Express application is essentially **a series of middleware function calls.**

Middleware functions have access to the **request object** (req), the **response object(**res**),** and the next middleware function in the application's request-response cycle. The next middleware function is commonly denoted by a variable named next.

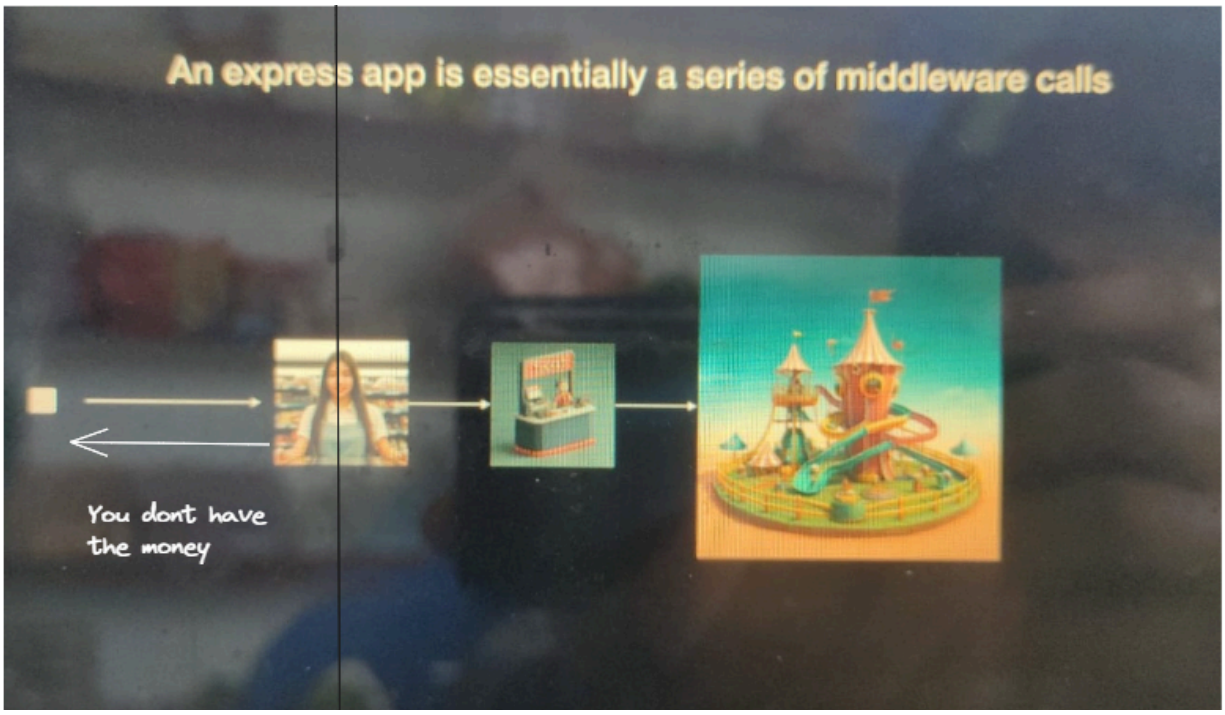
Middleware functions can perform the following tasks:

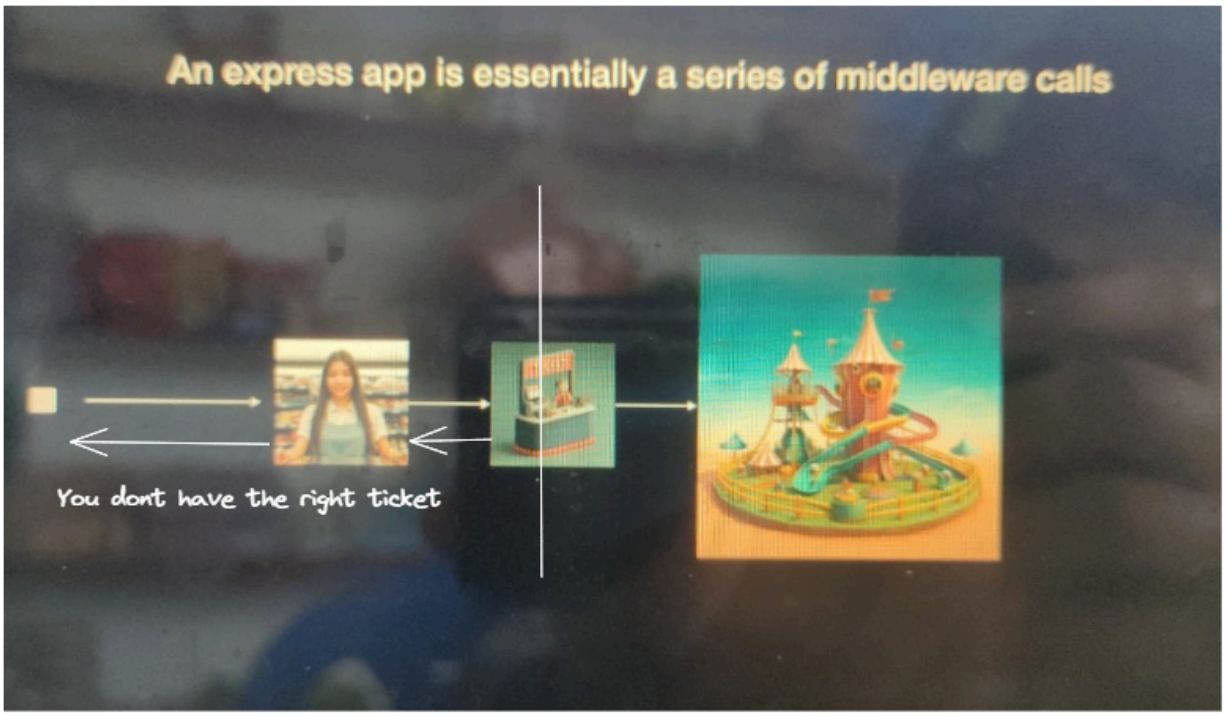
- Execute any code.
- Make changes to the request and the response objects.
- End the request-response cycle.
- Call the next middleware function in the task

https://expressjs.com/en/guide/using-middleware.html

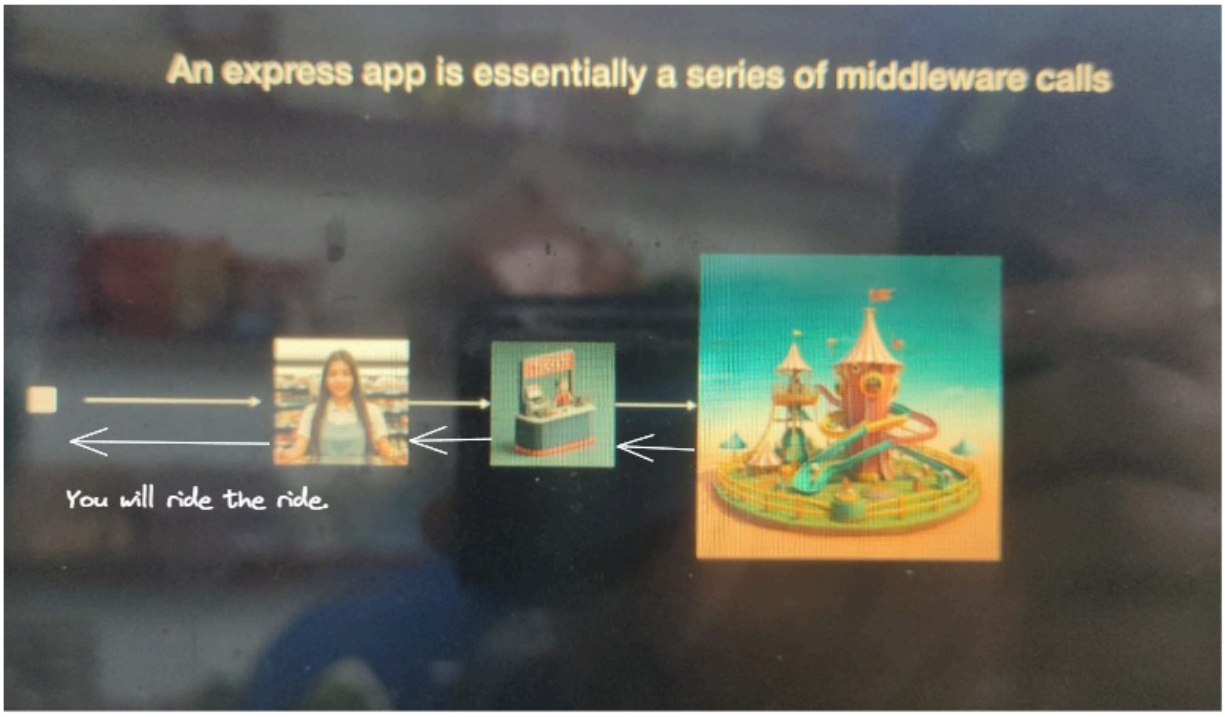
Express is just a bunch of middleware chained together.

A middleware can stop the request at any point



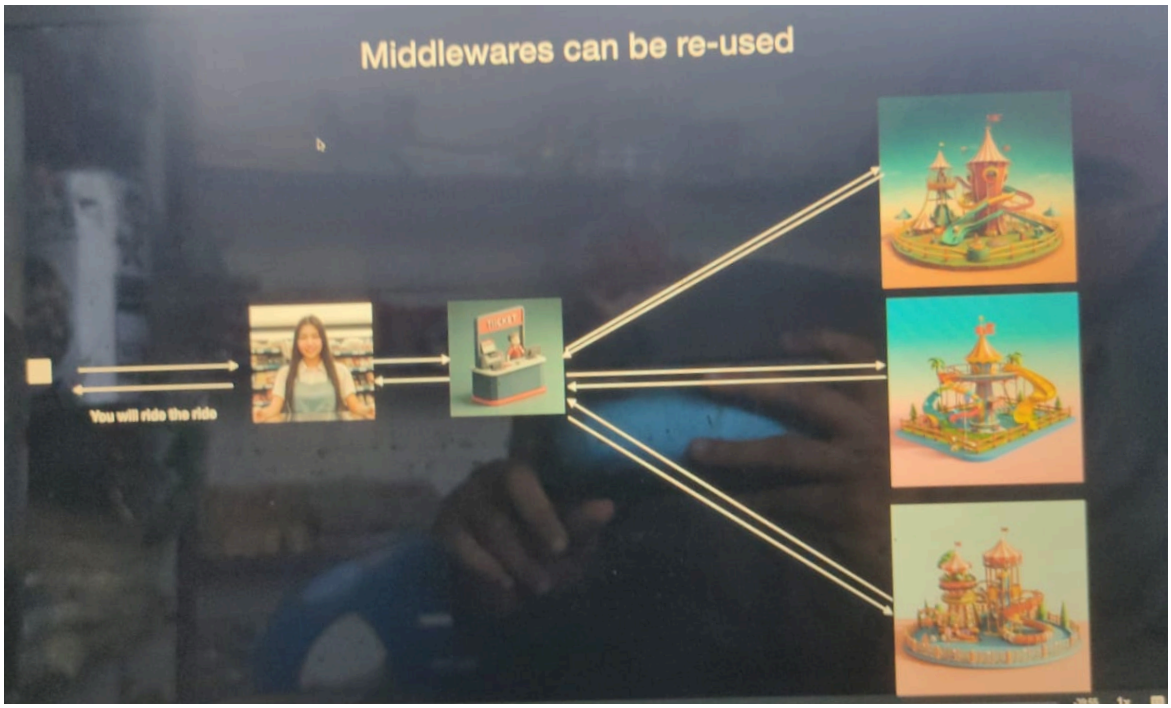


There can be multiple reasons for middleware to reject the request.

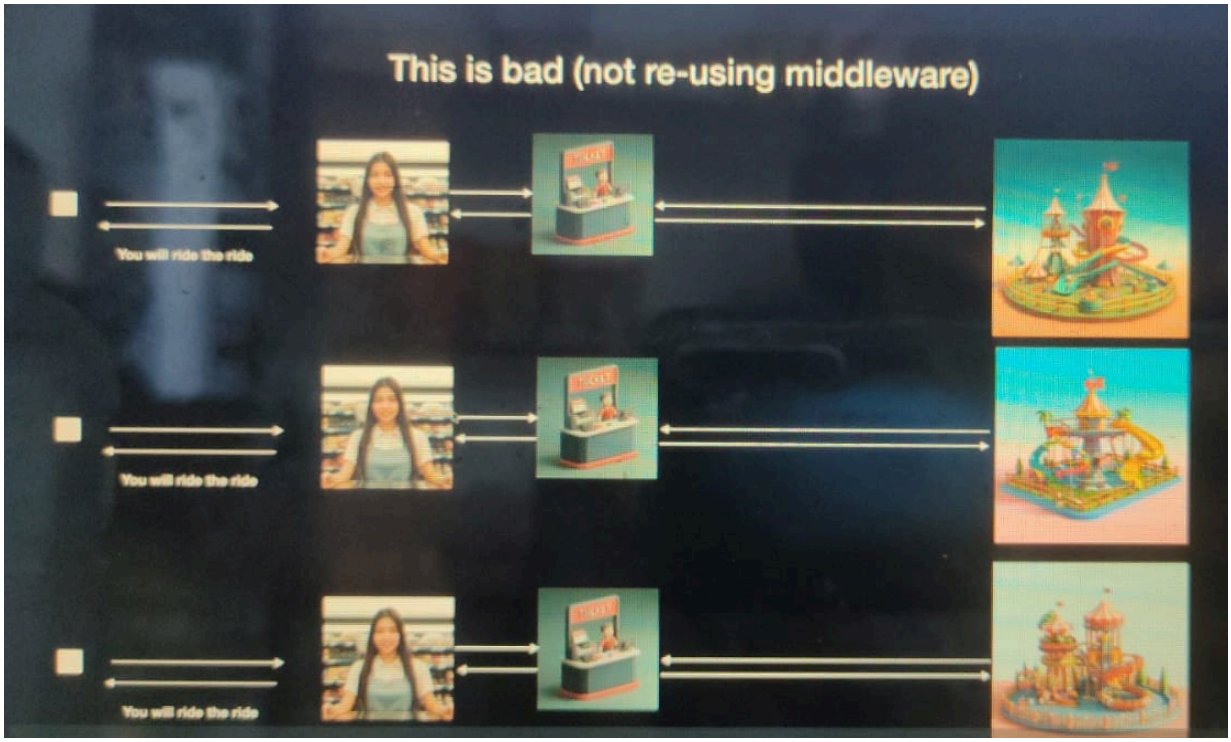


Why don’t I put all these middleware into a single middleware (where ticket counter, ticket checking, and ride all happen**)**
Why do we have to separate into 3 different middleware?

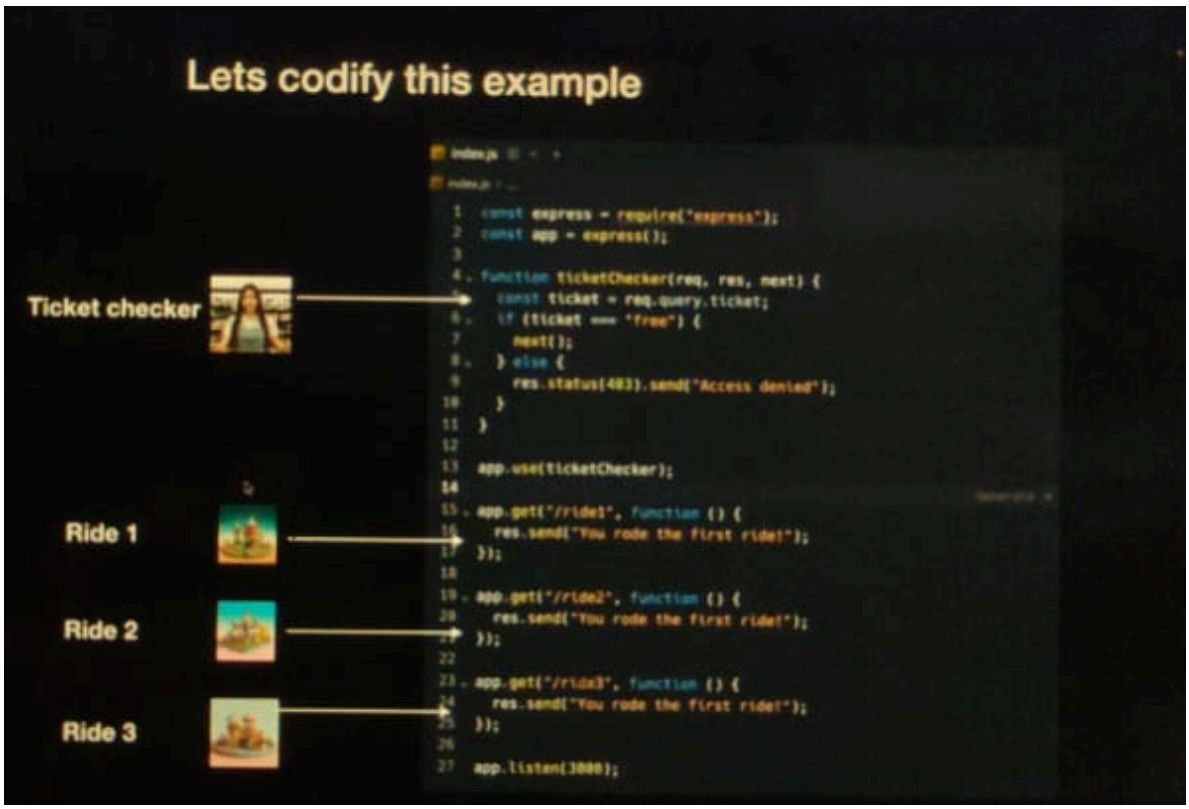
The main reason is that middleware can be re-used.
For example, in real amusement parks, we won't create a separate ticket counter and we may have different ticket checkers(14+ age and 5-foot height) for each ride.



This approach is good but let's see a bad approach that will lead us to have several employees

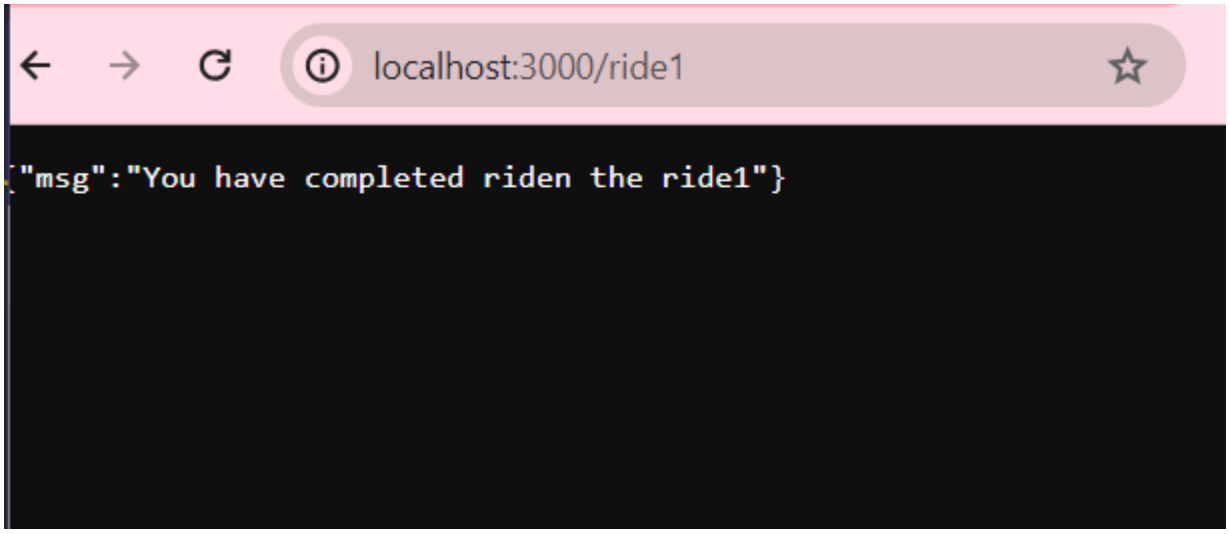


Lets codify this example



```
const express = require("express");
const app = express();

app.get("/ride1",function(req,res){
    res.json({
        msg:"You have completed riden the ride1"
    })
})

app.listen(3000);
```

Website:



But the problem is that we haven't had any checks before allowing to ride, or any ticket counter

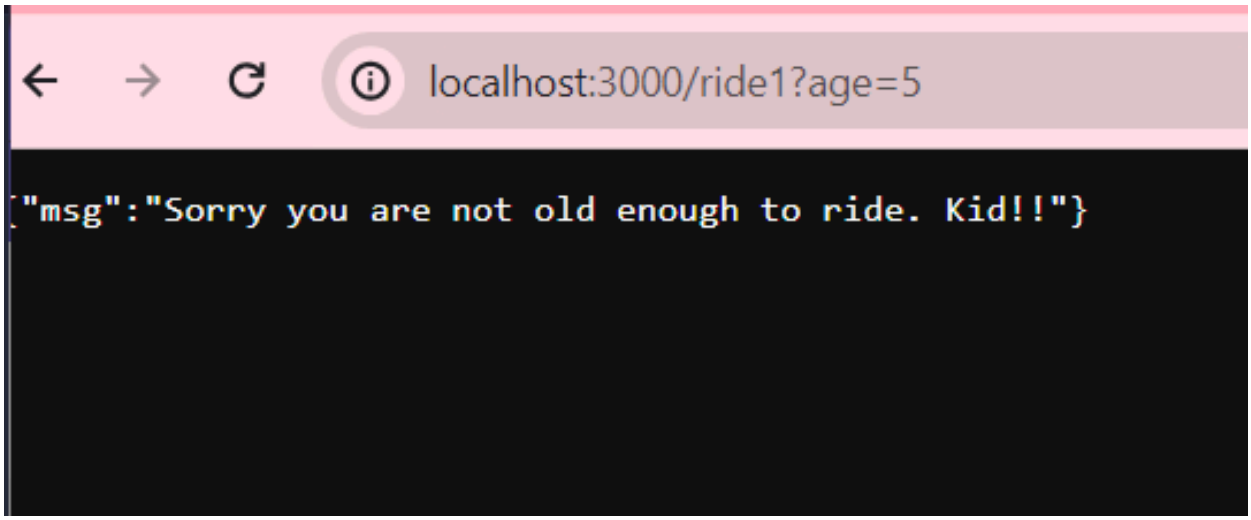
Lets introduce a function which returns boolean. If age is greater than 14 (ride requirement)

```
const express = require("express");
const app = express();

// function that returns a boolean if the age of the person is more than
14 or not
function isOldEnough(age){
    if(age>=14){
        return true;
    }else{
```

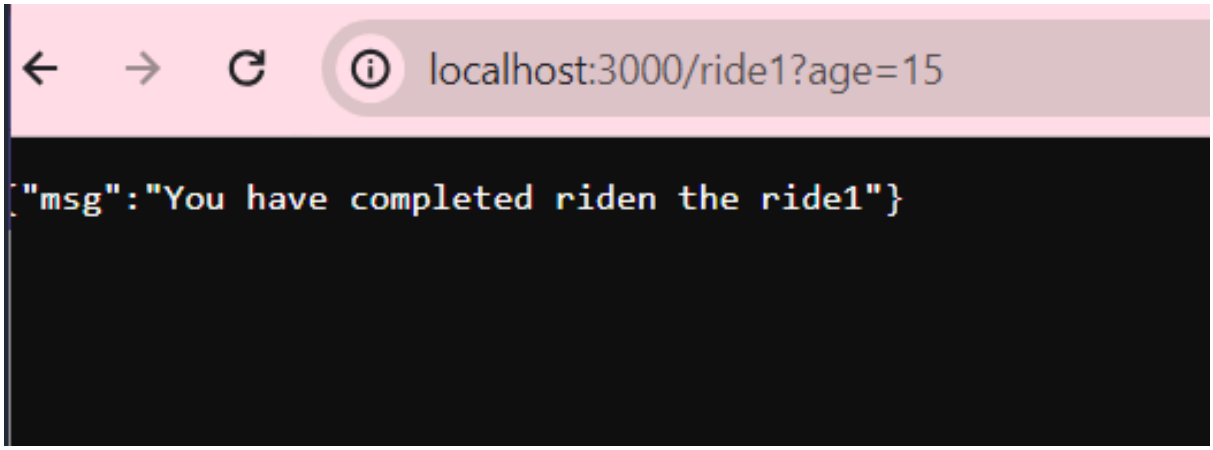
```
        return false;
    }
}

app.get("/ride1",function(req,res){
    if(isOldEnough(req.query.age)){
        return res.json({
            msg:"You have completed riden the ride1"
        })
    }else{
        res.status(411).json({
            msg:"Sorry you are not old enough to ride. Kid!!"
        })
    }
})

app.listen(3000);
```

Website

- Age < 14



- Age >= 14



Introducing another ride name ride2

```
app.get("/ride2",function(req,res){
    if(isOldEnough(req.query.age)){
        return res.json({
            msg:"You have completed riden the ride2"
        })
    }else{
        res.status(411).json({
            msg:"Sorry you are not old enough to ride. Kid!!"
        })
    }
})
```

function like **isOldEnough** is one way to introduce a ticket checker another better way is to use middleware

Defining isOldEnough function as middleware

```
function isOldEnoughMiddleware(req, res, next){
    if(age>=14){
        next();
        // user propagate the next middleware.
    } else{
        res.json({
```

```
            msg:"Sorry you are not allowed to ride. Kid!!"
        })
    }
}
// In express the function isOldEnoughMiddleware , or ride1 and ride2
handler function all have access to a third argument called next.

// calling this next function takes control from this middleware to the
next middleware.

// we haven't written next here as most of the time we will not need it
here in the last route handler function.
```

Now since with the middleware, endpoints don't have to focus on the checking the age and print message accordingly now it will focus on the thing it had to do , which is to ride.

```
app.get("/ride1",function(req,res){
        res.json({
            msg:"You have completed riden the ride1"
        })
})

app.get("/ride2",function(req,res){
        res.json({
            msg:"You have completed riden the ride2"
        })
})
```

Verification checks happen before.

Express is the series of middleware calls now we will call the middleware.

```
app.get("/ride1",isOldEnoughMiddleware,function(req,res){
        res.json({
            msg:"You have completed riden the ride1"
```

```
        })
})

app.get("/ride2",isOldEnoughMiddleware,function(req,res){
        res.json({
            msg:"You have completed riden the ride2"
        })
})
```

If there is certain middleware which is used in all the endpoints then we will use **app.use(isOldEnoughMiddleware)**

Assignments:

Go the assignment directory

Then npm install

Assignment 1:

Creating a middleware for logging the number of requests on a server.

Initial code:

```
const request = require('supertest');
const assert = require('assert');
const express = require('express');

const app = express();
let requestCount = 0;

// You have been given an express server which has a few endpoints.
// Your task is to create a global middleware (app.use) which will
// maintain a count of the number of requests made to the server in the
global
// requestCount variable

app.get('/user', function(req, res) {
  res.status(200).json({ name: 'john' });
```

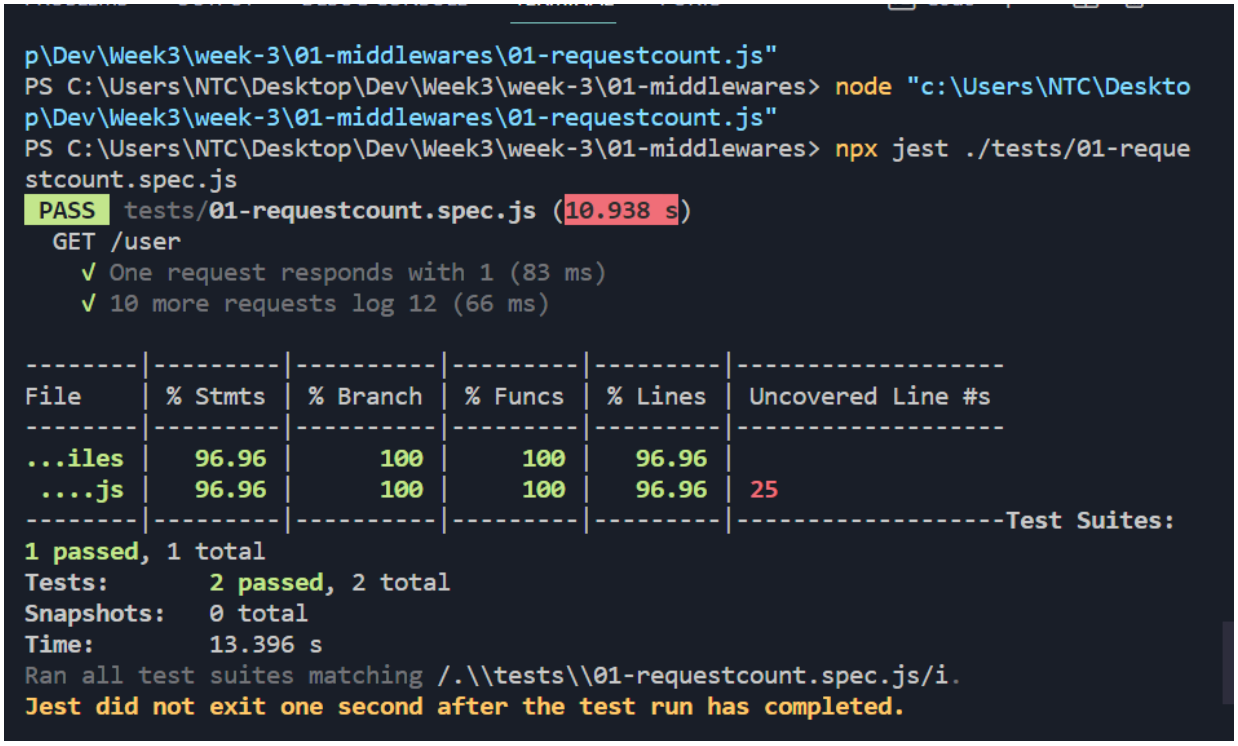
```
});

app.post('/user', function(req, res) {
  res.status(200).json({ msg: 'created dummy user' });
});

app.get('/requestCount', function(req, res) {
  res.status(200).json({ requestCount });
});

module.exports = app;
```

Final code

```
const express = require('express');

const app = express();
let requestCount = 0;

app.use(function(req, res, next){
  // when control reaches here means new request has come to the server
  // Now I can keep track of the no. of request that have been send to the
server.
  // We should count no. of requests so that our backend doesn't get
overloaded.

  requestCount++;
  next();
})

app.get('/user', function(req, res) {
  res.status(200).json({ name: 'john' });
});

app.post('/user', function(req, res) {
  res.status(200).json({ msg: 'created dummy user' });
});

app.get('/requestCount', function(req, res) {
  res.status(200).json({ requestCount });
```

```
});

app.listen(3000)
module.exports = app;
```

We can also test it in our system since it is an assignment



Assignment 2:

Rate limiter

(They are used when someone try to bombard our system for example sending request using for loop , then we can limit the rate of request. Example 5 request in second)

Initial Code:

```
const request = require('supertest');
const assert = require('assert');
const express = require('express');
const app = express();
```

```
// You have been given an express server which has a few endpoints.
// Your task is to create a global middleware (app.use) which will
// rate limit the requests from a user to only 5 request per second
// If a user sends more than 5 requests in a single second, the server
// should block them with a 404.
// User will be sending in their user id in the header as 'user-id'
// You have been given a numberOfRequestsForUser object to start off with
which
// clears every one second

let numberOfRequestsForUser = {};
setInterval(() => {
    numberOfRequestsForUser = {};
}, 1000)

app.get('/user', function(req, res) {
  res.status(200).json({ name: 'john' });
});

app.post('/user', function(req, res) {
  res.status(200).json({ msg: 'created dummy user' });
});

module.exports = app;
```

Final code:

```
const express = require('express');
const { use } = require('./01-requestcount');
const app = express();
// You have been given an express server which has a few endpoints.
// Your task is to create a global middleware (app.use) which will
// rate limit the requests from a user to only 5 request per second
// If a user sends more than 5 requests in a single second, the server
// should block them with a 404.
// User will be sending in their user id in the header as 'user-id'
// You have been given a numberOfRequestsForUser object to start off with
which
// clears every one second
```

```
let numberOfRequestsForUser = {};
setInterval(() => {
  numberOfRequestsForUser = {};
}, 1000) //reinitializing to empty object on every 1 second


app.use(function(req, res, next){
  const userId = req.headers['user-id'];
  if(numberOfRequestsForUser[userId]){
    numberOfRequestsForUser[userId]++;
    if(numberOfRequestsForUser[userId] > 5){
      res.status(404).json({msg: 'Too many requests'})
    }else{
      next();
    }
  }else{
    numberOfRequestsForUser[userId] = 1;
    next();
    // allow the user to flow thorough
  }
})
app.get('/user', function(req, res) {
  res.status(200).json({ name: 'john' });
});

app.post('/user', function(req, res) {
  res.status(200).json({ msg: 'created dummy user' });
});

module.exports = app;
```

This is generally done through ip6 and block the user ip

Assignment 3:

Error count

```
app.get('/user',function(req,res){
    let a;//undefined
```

```
    a.length();// Js throw an error and stop execution of program write
here
    // This is not good for express app as it will stop the server and we
can also see the error message in detail with too much information
    //
    res.status(200).json({name:'john'});

})
```

Initial code:

```
const request = require('supertest');
const assert = require('assert');
const express = require('express');

const app = express();
let errorCount = 0;

// You have been given an express server which has a few endpoints.
// Your task is to
// 1. Ensure that if there is ever an exception, the end user sees a
status code of 404
// 2. Maintain the errorCount variable whose value should go up every time
there is an exception in any endpoint

app.get('/user', function(req, res) {
  throw new Error("User not found");
  res.status(200).json({ name: 'john' });
});

app.post('/user', function(req, res) {
  res.status(200).json({ msg: 'created dummy user' });
});

app.get('/errorCount', function(req, res) {
  res.status(200).json({ errorCount });
});

module.exports = app;
```

Final solution :

```
const request = require('supertest');
const assert = require('assert');
const express = require('express');

const app = express();
let errorCount = 0;

// You have been given an express server which has a few endpoints.
// Your task is to
// 1. Ensure that if there is ever an exception, the end user sees a
status code of 404
// 2. Maintain the errorCount variable whose value should go up every time
there is an exception in any endpoint

app.get('/user', function(req, res) {
  throw new Error("User not found");
  res.status(200).json({ name: 'john' });
});

app.post('/user', function(req, res) {
  res.status(200).json({ msg: 'created dummy user' });
});

app.get('/errorCount', function(req, res) {
  res.status(200).json({ errorCount });
});

app.listen(3000)

// Express by default send 500 status code
app.use(function(err,req, res, next){
  res.status(404).send({});
  errorCount++;
})

module.exports = app;
```