Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

CSU44000 Internet Applications

Week 3 Lecture 2

Conor Sheedy

More Complex Express Example

require, modules

- express
- path
    - To split up the url

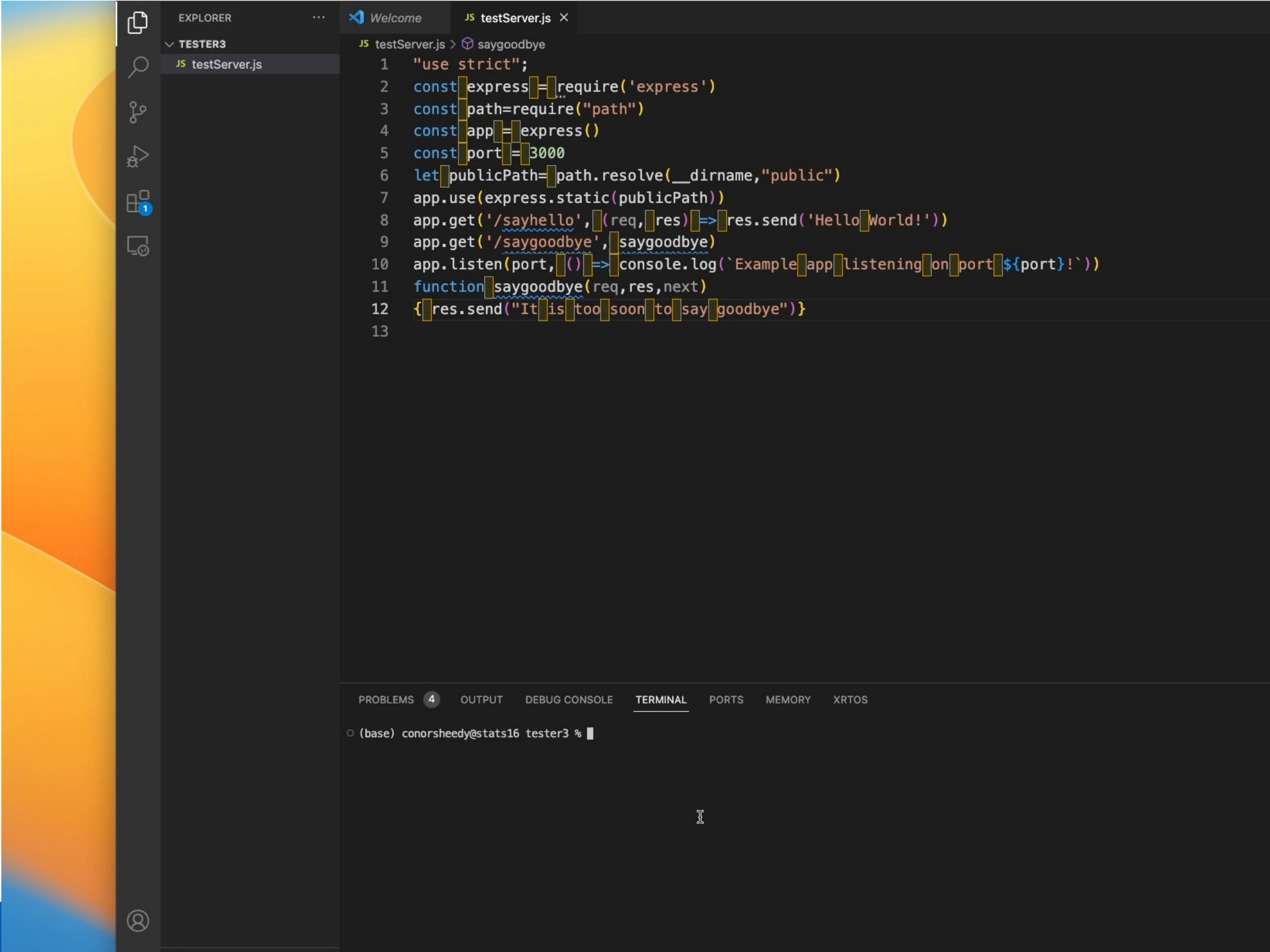
express()

- To instantiate the module

path.resolve

- to resolve a sequence of path-segments to an absolute path

```
"use strict";

const express = require('express')
const path=require("path")
const app = express()
const port = 3000
let publicPath= path.resolve(__dirname,"public")

app.use(express.static(publicPath))

app.get('/sayhello', (req, res) => res.send('Hello World!'))

app.get('/saygoodbye', saygoodbye)

app.listen(port, () => console.log(`Example app listening on port ${port}!`))


function saygoodbye(req,res,next)
{ res.send("It is too soon to say goodbye")}
```



```
"use strict";
const express = require('express')
const path=require("path")
const app = express()
const port = 3000
let publicPath= path.resolve(__dirname,"public")
app.use(express.static(publicPath))
app.get('/sayhello', (req, res) => res.send('Hello World!'))
app.get('/saygoodbye', saygoodbye)
app.listen(port, () => console.log(`Example app listening on port ${port}!`))
function saygoodbye(req,res,next)
{ res.send("It is too soon to say goodbye")}

```

```
(base) conorsheedy@stats16 tester3 %
```

Continued…

express.static

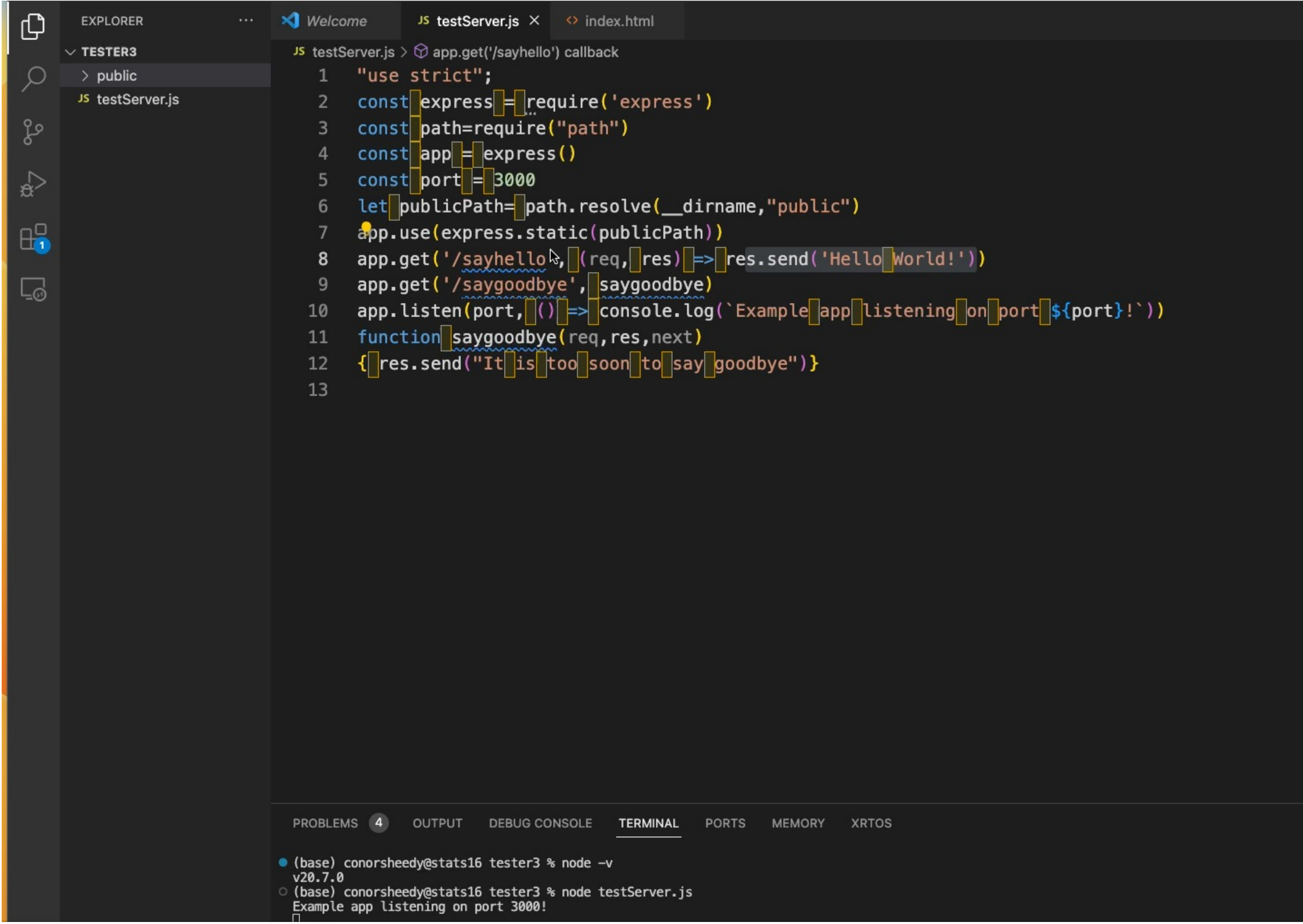
- matches the URL with filenames in a specified directory and serves them
- If control ‘falls through’ the stack of express handlers an error page is generated

app.get

- Request
- Response
  - Handler

app.listen hand over control

```
"use strict";

const express = require('express')
const path=require("path")
const app = express()
const port = 3000
let publicPath= path.resolve(__dirname,"public")

app.use(express.static(publicPath))

app.get('/sayhello', (req, res) => res.send('Hello World!'))

app.get('/saygoodbye', saygoodbye)

app.listen(port, () => console.log(`Example app listening on port ${port}!`))


function saygoodbye(req,res,next)
{ res.send("It is too soon to say goodbye")}
```



```javascript
"use strict";
const express = require('express')
const path=require("path")
const app = express()
const port = 3000
let publicPath= path.resolve(__dirname,"public")
app.use(express.static(publicPath))
app.get('/sayhello', (req, res) => res.send('Hello World!'))
app.get('/saygoodbye', saygoodbye)
app.listen(port, () => console.log(`Example app listening on port ${port}!`))
function saygoodbye(req,res,next)
{ res.send("It is too soon to say goodbye")}
```

```
(base) conorsheedy@stats16 tester3 % node -v
v20.7.0
(base) conorsheedy@stats16 tester3 % node testServer.js
Example app listening on port 3000!
```

APIs: Application Programming Interface

Using Express, it is easy to code actions that respond to requests like:

http://www.example.com/v1.1/racewinner?raceno=56?time=0430

- The server will treat the incoming values in the URL as a request for something
- The request has parameters
- The server will parse the url, do something with the parameters and return something

These act like a remote Application Programming Interface (API)

Web API's have become very popular

RESTful APIs

Representational State Transfer (REST) is a style of API

- defined by Roy Fielding (developer of Http 1.1)
- A style of API that is ‘compatible with the HTTP object model’
- Proposed in his 2000 PhD thesis
  - An architectural style
  - That constrains the interface

– Client-Server Architecture
  - separate client and server concerns

– Statelessness
  - The server shouldn’t maintain state information for each client
  - The same request should get the same response

– Cachability
  - responses should where possible be cachable

– Layered System
  - client cannot distinguish between real server and a proxy

– Code on demand (optional)
  - Servers can extend client functionality(download code)

RESTful APIs

4 constraints are:

- Identification of resources
  - Typically a one to one mapping of URI to resource
- Manipulation of resources
  - CRUD
- Self-descriptive messages
- Hypermedia as the engine of application state

– Uniform Interface

- Resource id in request
- Resource manipulation by client via representations received
  - if the client has an object, it can delete or modify that object
  - Received representations contain enough information for manipulation of resource
- Self-descriptive messages
  - each message contains all info to process the message
    - » not dependent on context
- Hypermedia as the engine of application state
  - client can use server-provided links for directions

Typical Structure of APIs

A RESTful API will typically give access to individually identified resources e.g. GET /photos/415

- GET is a verb
- ‘Resource’ is a noun
  - The REST architectural style
    - Makes use of standards
      - HTTP
      - URI
      - JSON

Typical Structure of APIs

A Common Pattern is called Create/Read/Update/Delete :CRUD

- The URL identifies the ‘Resource’
- POST Resource
  - Creates a resource e.g. upload a photo
- GET Resource
  - Reads the resource e.g returns Photo in JSON
- PUT Resource
  - Updates a resource e.g. changes a photo
    - Idempotent
      - Multiple calls will have the same result
- DELETE Resource
  - Deletes a resource e.g. deletes a photo from a library

RESTful APIs

Where does the name come from?

- representational state transfer
  - The server transfers a representation of the state of the resource
    - JSON (Javascript Object Notation) or other format
- Stateless
  - Server doesn't need to keep track of the clients state
    - Good for a many to one relationship
    - Good for cashing
    - Good for scaling out
  - The client transfer a representation of its own state with each request
    - If relevant

Why is it a good pattern?

- Scalability
- It leads to a loose or lightweight coupling between the client and the server
- Suitable for Internet-scale usage

RESTful APIs

Were there other popular styles of web API?

- SOAP
  - Simple Object Access Protocol
  - an official protocol, Standard based
    - Message format
    - Message processing models
    - Etc
  - XML based messages
  - REST
    - Guidelines
    - Lightweight
    - Popular

RESTful APIs vs GraphQL

Are there other popular styles of web API?

- GraphQL
  - Released by Facebook as an Open Source standard in 2015
  - GraphQL Foundation in 2018
  - declarative data fetching
  - a single endpoint
    - REST, multiple endpoints
  - defines types with fields
    - schema definition
  - Resolvers
    - functions that retrieve and map the data
  - Typically also returns json format data
  - Efficiency vs Complexity

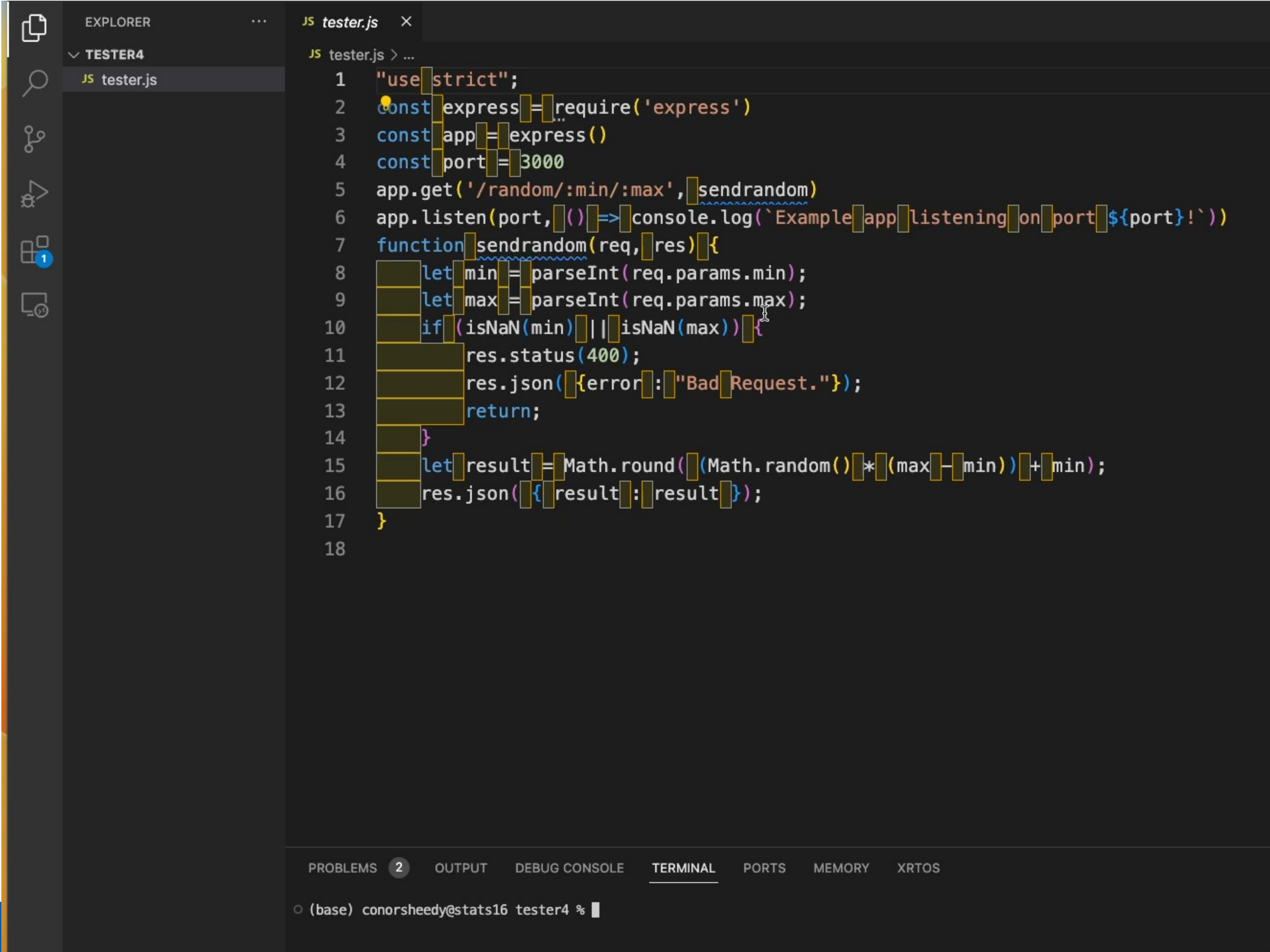
Example : A Web API to Generate Random Numbers

/random is the API resource

/:min/:max is a regular expression that picks up parameters

Return result is a JSON object

```
"use strict";

const express = require('express')
const app = express()
const port = 3000

app.get('/random/:min/:max', sendrandom)

app.listen(port, () => console.log(`Example app listening on port ${port}!`))

function sendrandom(req, res) {
   let min = parseInt(req.params.min);
   let max = parseInt(req.params.max);
   if (isNaN(min) || isNaN(max)) {
      res.status(400);
      res.json( {error : "Bad Request."});
      return;
   }
   let result = Math.round( (Math.random() * (max - min)) + min);

   res.json( { result : result });
}
```



```javascript
"use strict";
const express = require('express')
const app = express()
const port = 3000
app.get('/random/:min/:max', sendrandom)
app.listen(port, () => console.log(`Example app listening on port ${port}!`))
function sendrandom(req, res) {
  let min = parseInt(req.params.min);
  let max = parseInt(req.params.max);
  if (isNaN(min) || isNaN(max)) {
    res.status(400);
    res.json( {error : "Bad Request."});
    return;
  }
  let result = Math.round( (Math.random() * (max - min)) + min);
  res.json( { result : result });
}
```

```
○ (base) conorsheedy@stats16 tester4 % █
```

Widespread Availability of APIs

- **Very Many Web Providers (e.g. Facebook, Googlemaps, Twitter etc) provide Web APIs to allow access to their data and services**

- **While many are free (for low volumes), most require a sign-up; issue you a webtoken; this must be included with all requests**

- **They often offer a dashboard, where the volume/source of the requests can be tracked, paid for etc.**

- **Aggregators have appeared that simplify this e.g. RapidApi** https://rapidapi.com/blog/most-popular-api/

Example:Openweathermap will give information on weather in different parts of the world

Do a GET on

api.openweathermap.org/data/2.5/weather?q=Dublin,Ireland&APPID=3e2d927d4f28b456c6bc662f34350957

- URL contains parameters
    - City, country
- Response is in JSON format
- Containing useful weather information

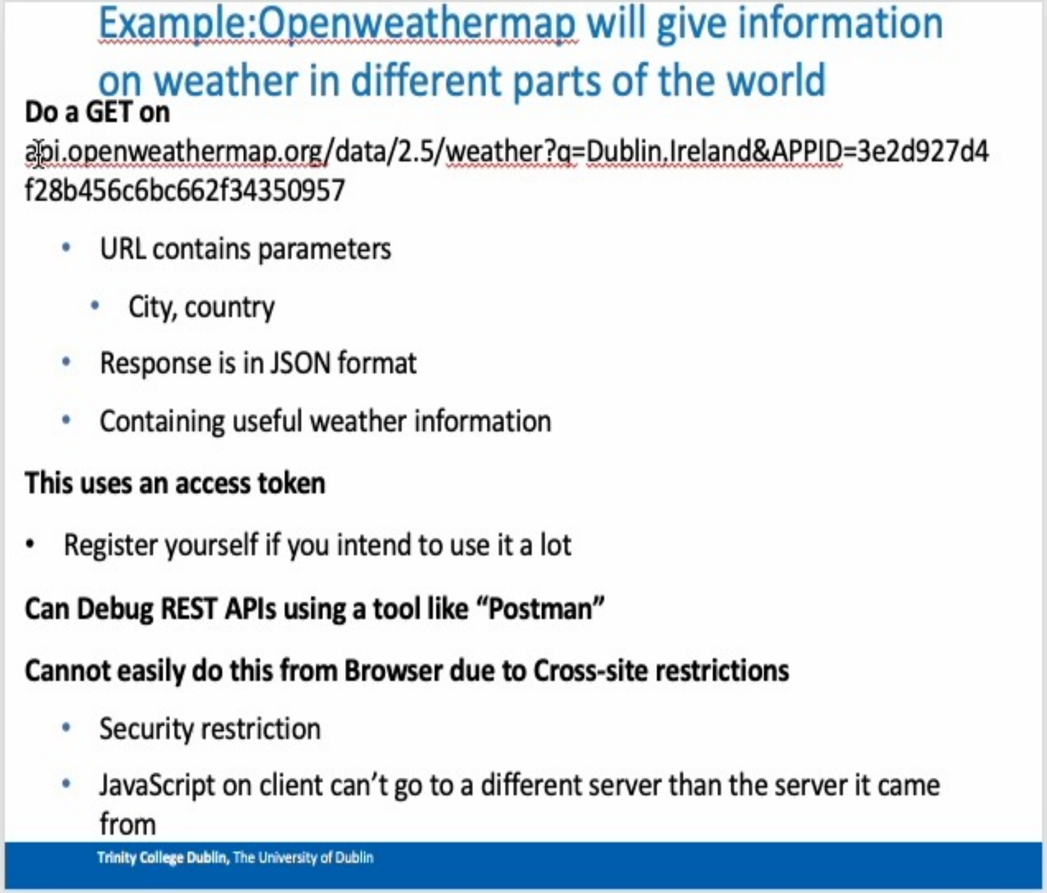
This uses an access token

- Register yourself if you intend to use it a lot

Can Debug REST APIs using a tool like "Postman"

Cannot easily do this from Browser due to Cross-site restrictions

- Security restriction
- JavaScript on client can't go to a different server than the server it came from



Example:Openweathermap will give information on weather in different parts of the world

Do a GET on

api.openweathermap.org/data/2.5/weather?q=Dublin.Ireland&APPID=3e2d927d4f28b456c6bc662f34350957

- URL contains parameters
  - City, country
- Response is in JSON format
- Containing useful weather information

This uses an access token

- Register yourself if you intend to use it a lot

Can Debug REST APIs using a tool like “Postman”

Cannot easily do this from Browser due to Cross-site restrictions

- Security restriction
- JavaScript on client can’t go to a different server than the server it came from

Trinity College Dublin, The University of Dublin

from

Summary of Server-Side

Server-side logic,

- database access can be coded at the server side
- popular choice for this right now is JavaScript code with NPM libraries

Express

- makes it very easy to implement a Web Server with Complex Routing
  - Routing – how you handle the different URLs

RESTful APIs

- are a popular style of server development

Often Web Applications are implemented with a combination of a

- Web API,
- Some Static pages
- Client front end (on the Browser)