Consuming Web APIs with Typescript 5

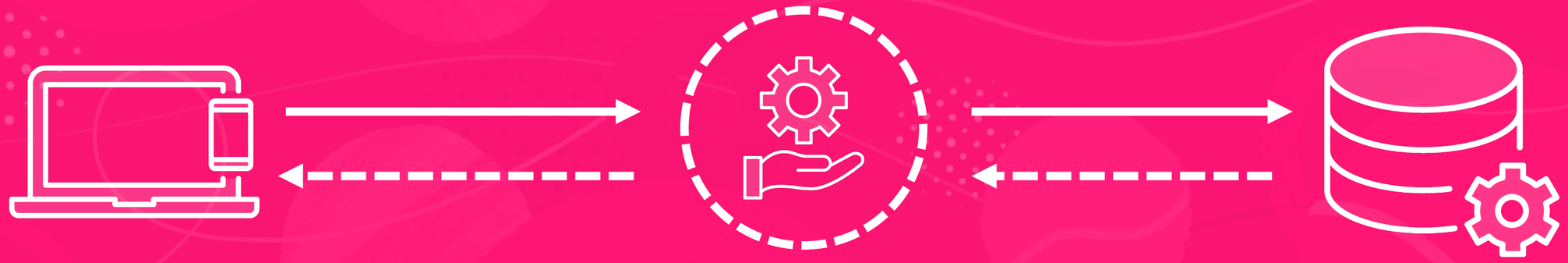
Understanding Web APIs and Their Importance

Allen O'Neill

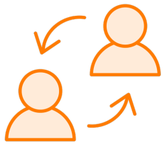
CTO / Senior Engineer, Microsoft Regional Director & MVP

@DataBytesAI | www.datalabs.io

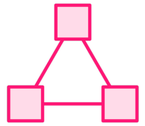
What are Web APIs?



Importance of Web APIs



Efficient Data Exchange



Interoperability

Feature Enhancement

Scalability

Ecosystem Expansion

Maximizing Web API Potential with TypeScript 5

Improvements in TypeScript 5

Decorators

const Type Parameters

Support for Export Type

Improved Error Messages

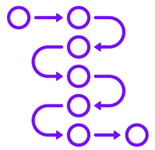
Smaller Size

Typescript 5 Versus It’s Older Versions

Web API Consumption in TypeScript 5 and Its Advantages

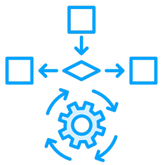
New HTTP client API

Improved Type Inference for Web APIs



Support for Asynchronous Iterators

New Security Features



Improved Support for Modules

Setting Up the Development Environment

Setting Up the Development Environment in Visual Studio

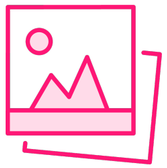
Install Visual Studio

Download and Install Node.js (if not already installed)

Install TypeScript

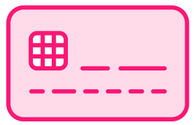
Installing Visual Studio Code

Free and Open Source



Cross-Platform

Lightweight and Fast



Extensible

https://code.visualstudio.com

Visual Studio Code Docs Updates Blog API Extensions FAQ Learn Search Docs Download

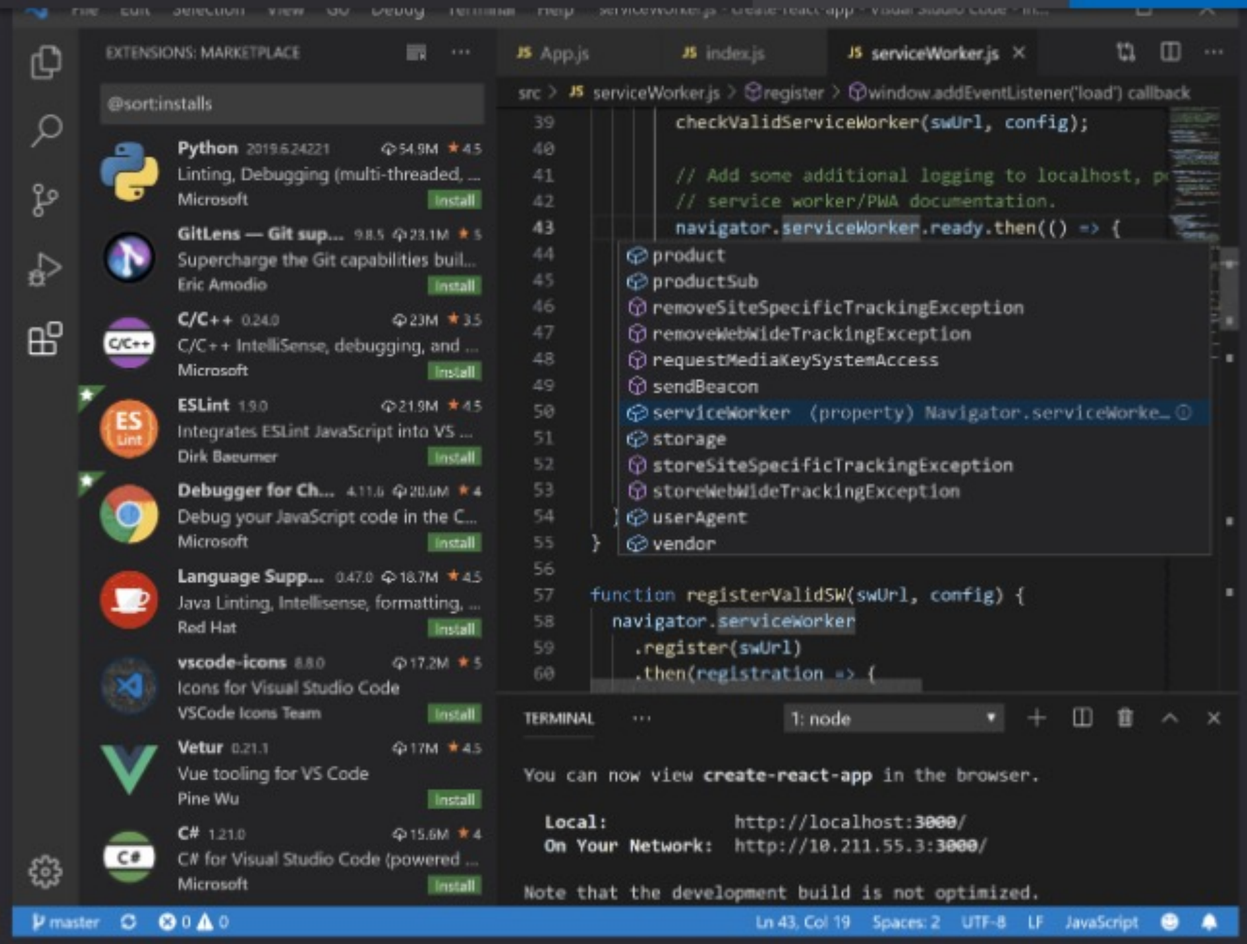
Code editing. Redefined.

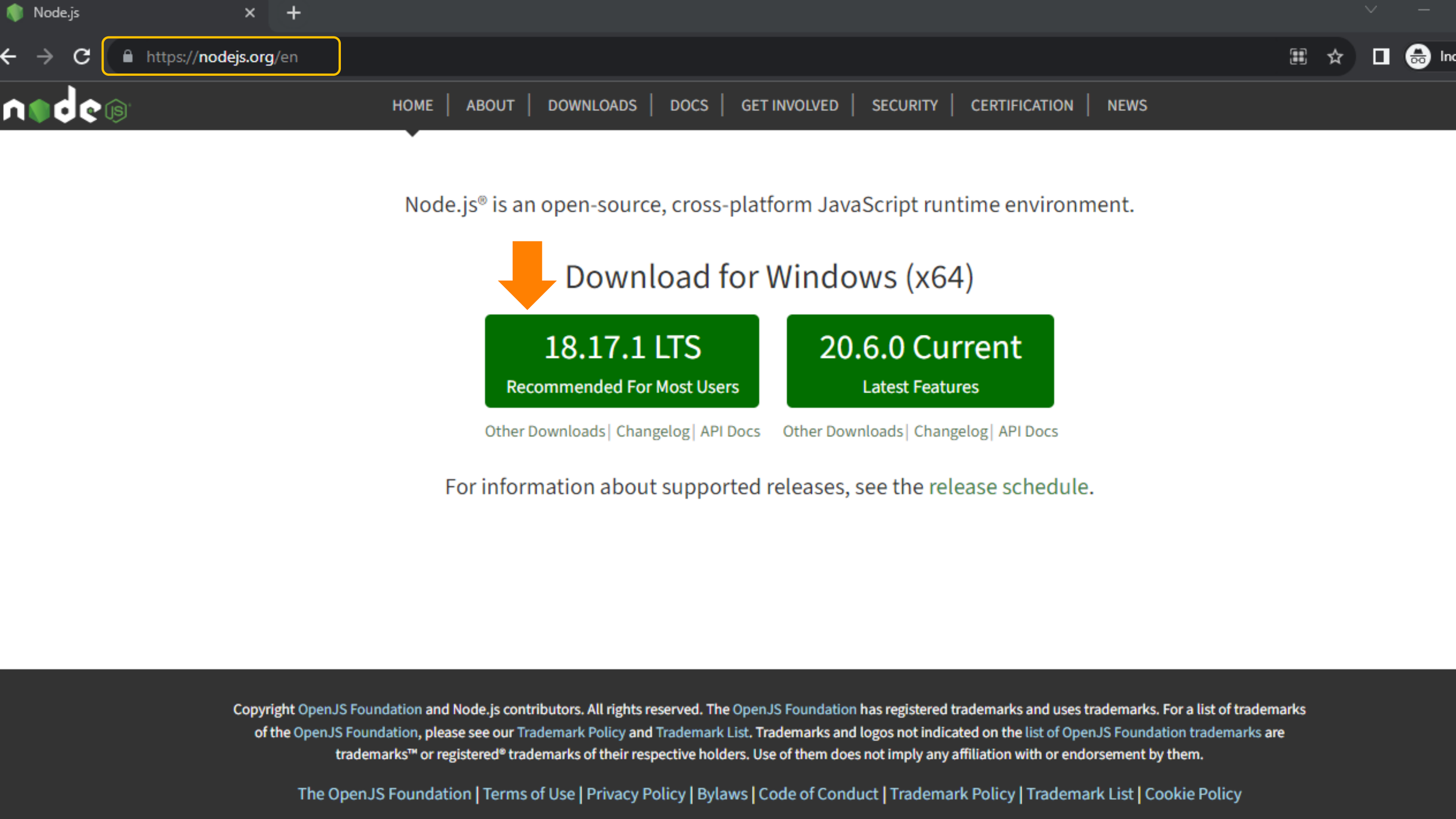
Free. Built on open source. Runs everywhere.

Download for Windows
Stable Build

		Stable	Insiders
macOS	Universal	↓	↓
Windows x64	User Installer	↓	↓
Linux x64	.deb	↓	↓
	.rpm	↓	↓

Other downloads or open on web

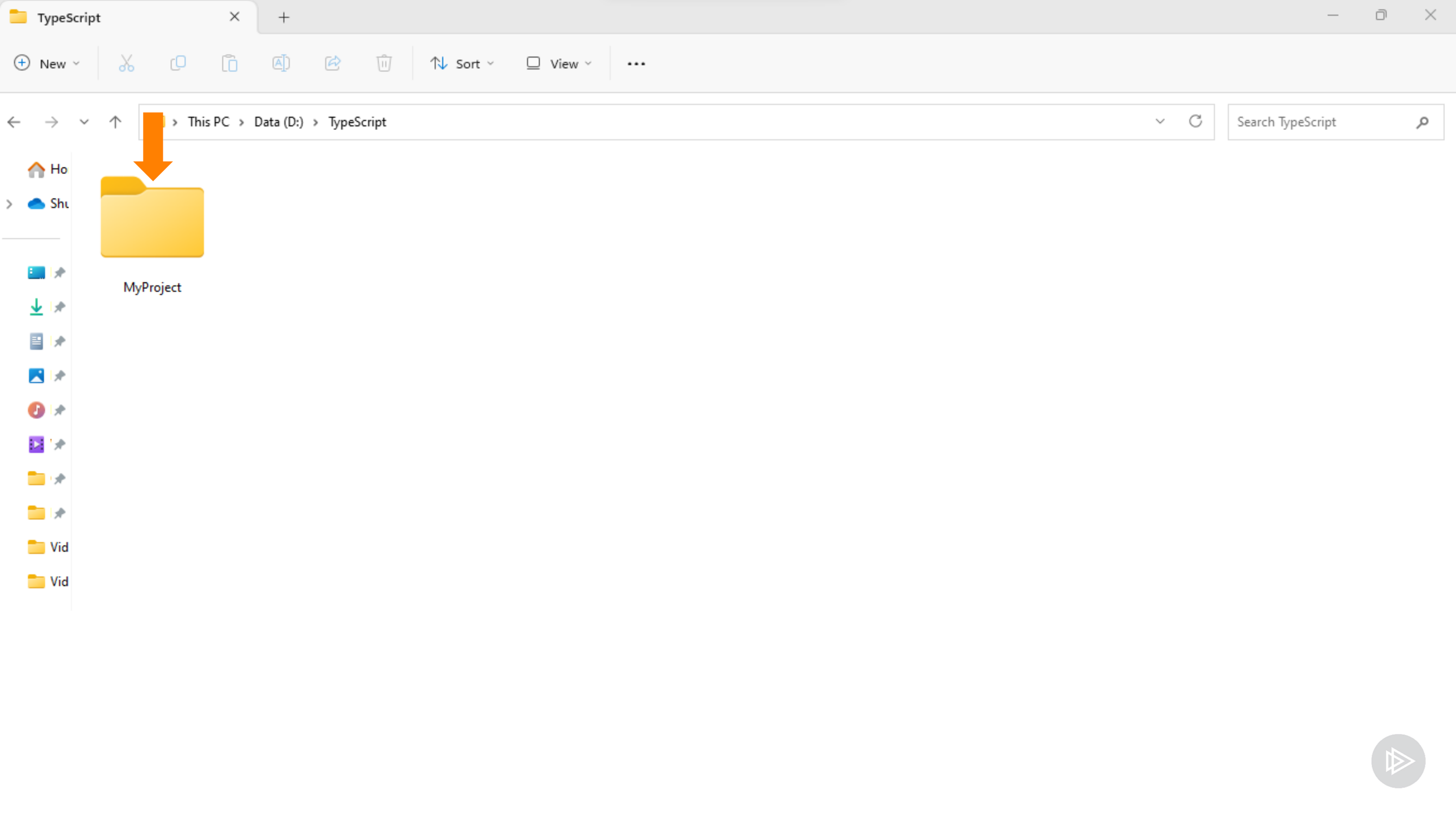


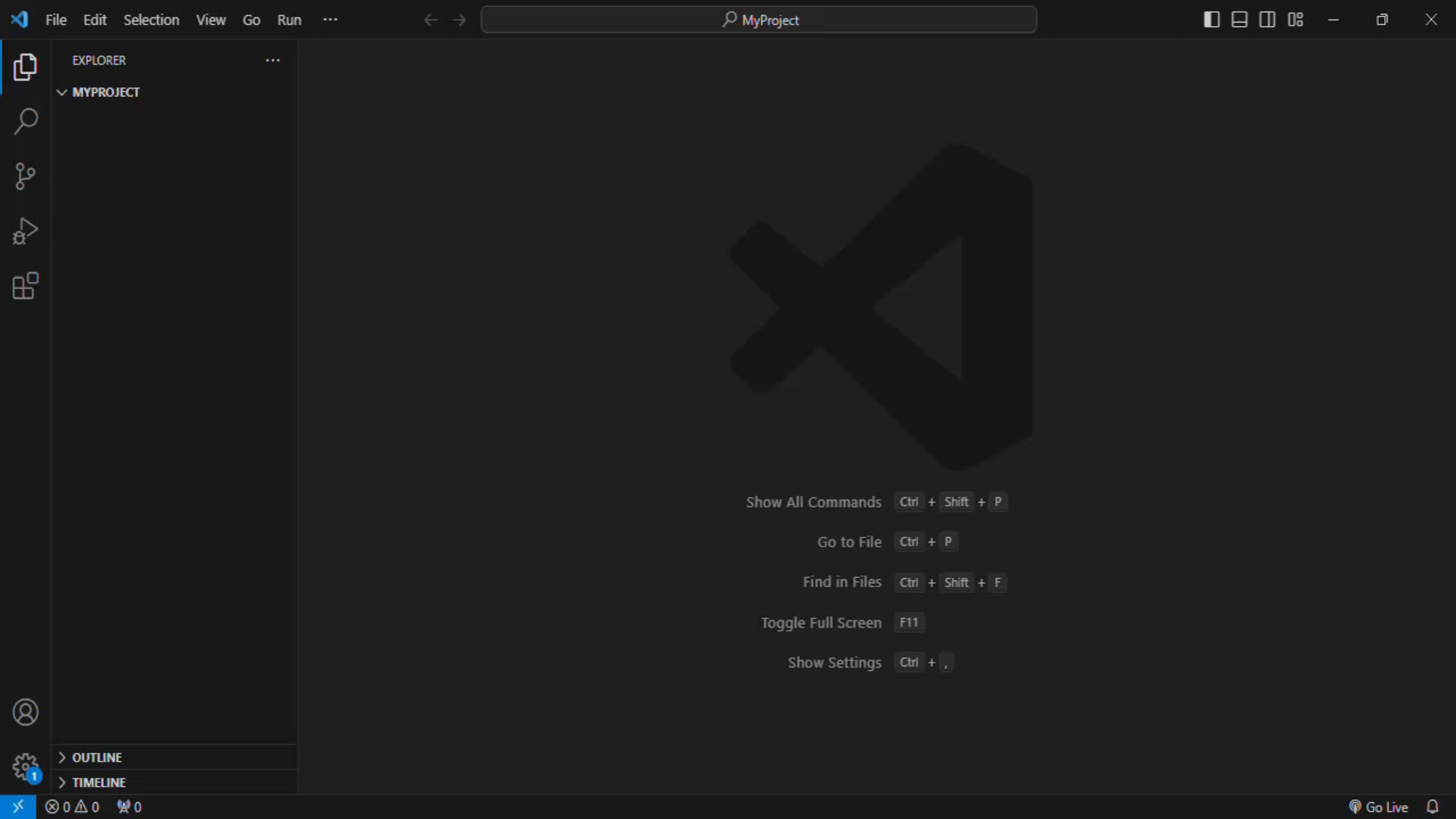

Node.js
https://nodejs.org/en
HOME | ABOUT | DOWNLOADS | DOCS | GET INVOLVED | SECURITY | CERTIFICATION | NEWS
Node.js® is an open-source, cross-platform JavaScript runtime environment.
Download for Windows (x64)
18.17.1 LTS
Recommended For Most Users
20.6.0 Current
Latest Features
Other Downloads | Changelog | API Docs
Other Downloads | Changelog | API Docs
For information about supported releases, see the release schedule.
Copyright OpenJS Foundation and Node.js contributors. All rights reserved. The OpenJS Foundation has registered trademarks and uses trademarks. For a list of trademarks of the OpenJS Foundation, please see our Trademark Policy and Trademark List. Trademarks and logos not indicated on the list of OpenJS Foundation trademarks are trademarks™ or registered® trademarks of their respective holders. Use of them does not imply any affiliation with or endorsement by them.
The OpenJS Foundation | Terms of Use | Privacy Policy | Bylaws | Code of Conduct | Trademark Policy | Trademark List | Cookie Policy

Node.js and NPM Installation Verification

Command Prompt

```
C:\Users\PC>node -v
v18.17.1

C:\Users\PC>npm -v
9.6.7
```


TypeScript
New
Sort
View
This PC
Data (D:)
TypeScript
Search TypeScript
MyProject


File
Edit
Selection
View
Go
Run
MyProject
EXPLORER
MYPROJECT
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Toggle Full Screen F11
Show Settings Ctrl + ,
OUTLINE
TIMELINE
Go Live

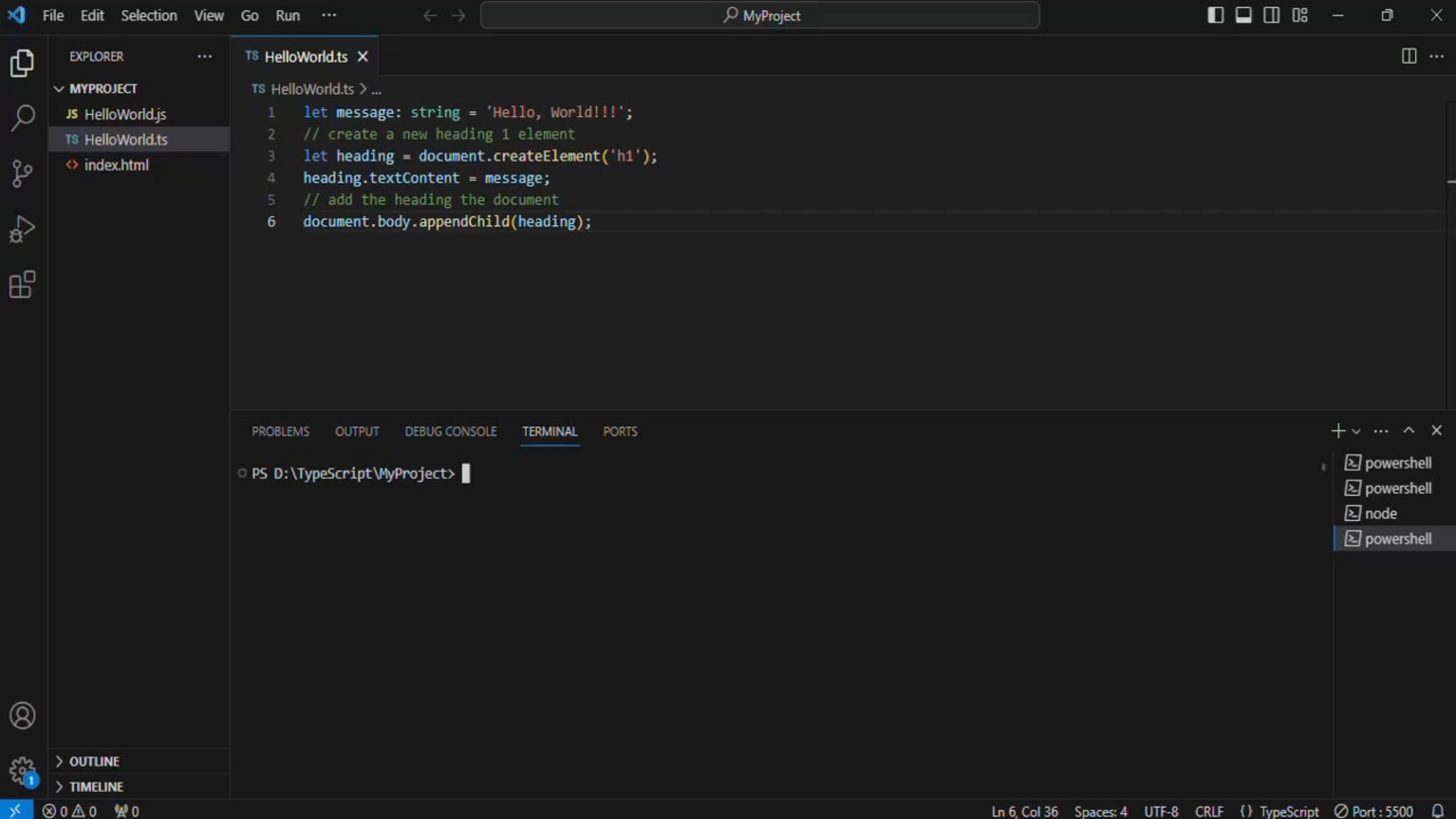
The Watch Parameter

Watch Parameter

What is Watch Mode?

Why We Need the Watch Mode?

How to Activate Watch Mode?


File Edit Selection View Go Run ···
MyProject
EXPLORER
MYPROJECT
HelloWorld.js
HelloWorld.ts
index.html
HelloWorld.ts
HelloWorld.ts > ...
1 let message: string = 'Hello, World!!!';
2 // create a new heading 1 element
3 let heading = document.createElement('h1');
4 heading.textContent = message;
5 // add the heading the document
6 document.body.appendChild(heading);
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\TypeScript\MyProject>
powershell
powershell
node
powershell
OUTLINE
TIMELINE
Ln 6, Col 36 Spaces: 4 UTF-8 CRLF {} TypeScript Port : 5500

Consuming Web APIs with Typescript 5

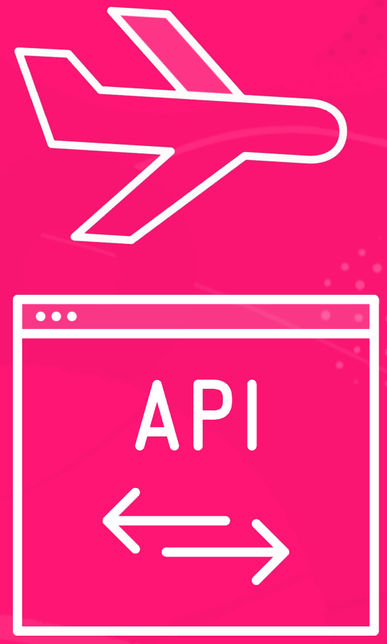
Defining Our Project

Allen O'Neill

CTO / Senior Engineer, Microsoft Regional Director & MVP

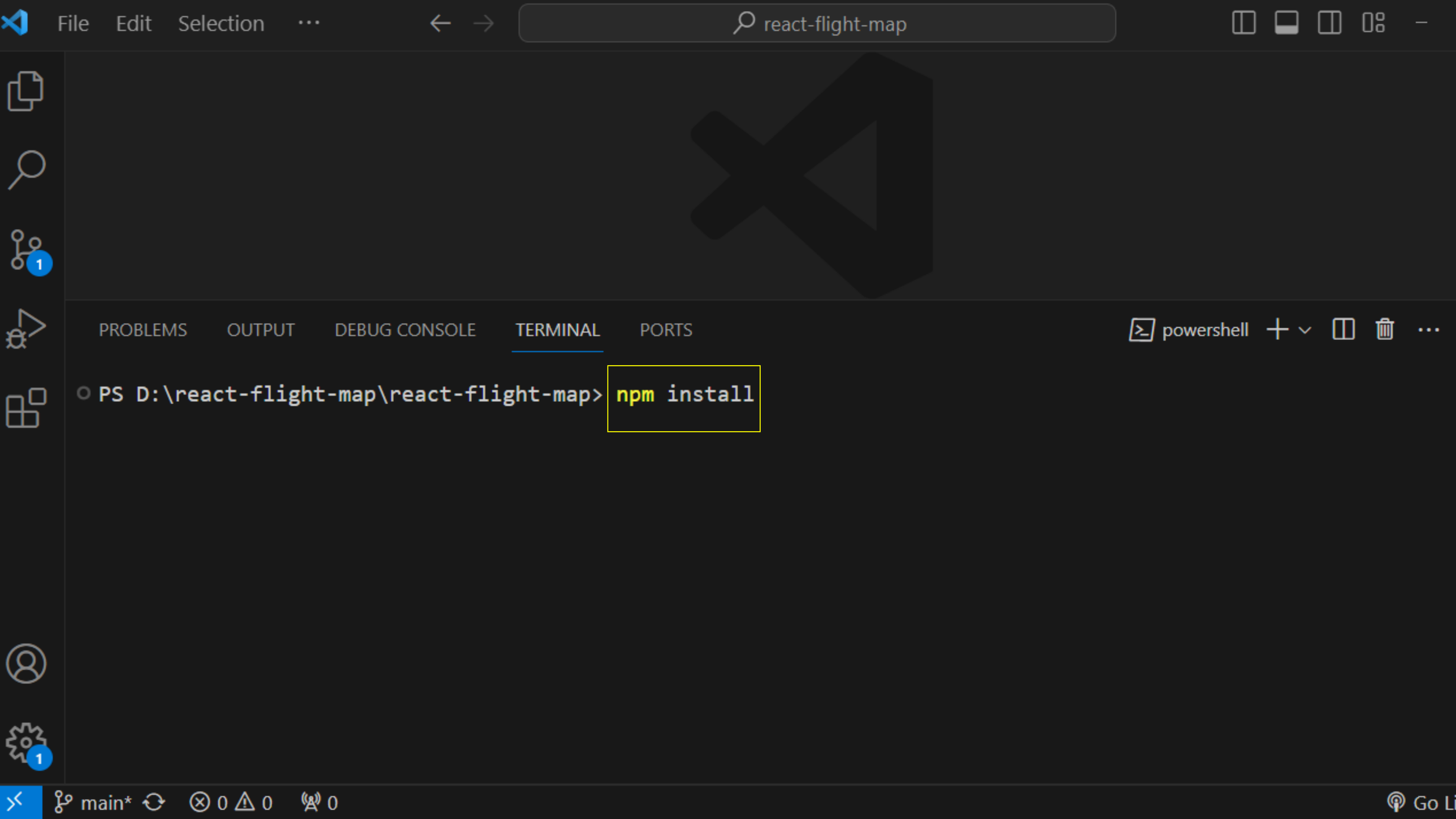
@DataBytesAI | www.datalabs.io

The Crucial Role of Web APIs in Our Project




React Flight App
localhost:3000
Geneva
Lyon
Milan
Venice
Slovenia
Croatia
Belgrade
Serbia
Bosnia and Herzegovina
Bordeaux
Monaco
San Marino
Marseille
Bilbao
Italy
Montenegro
Sofia
Vatican City
Skopje
Albania
Naples
Thessaloniki
Greece
Palermo
Athens
Catania
Annaba
Tunis
Constantine
Sousse
Malta
Gibraltar
Oujda
Fez
Casablanca
Tunisia
Zuara
Misrata
Benghazi
Ghardaia
Safi
Marrakesh
maptiler
© MapTiler © OpenStreetMap contributors

https://github.com/datalabs-io/pluralsight-webapi-typescript-v5-demo .

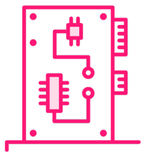

File
Edit
Selection
react-flight-map
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
PS D:\react-flight-map\react-flight-map> npm install
main*
0
0
0

https://github.com/datalabs-io/pluralsight-webapi-typescript-v5-demo-opensky

Interactive Flight-Map Application

Environment Variables

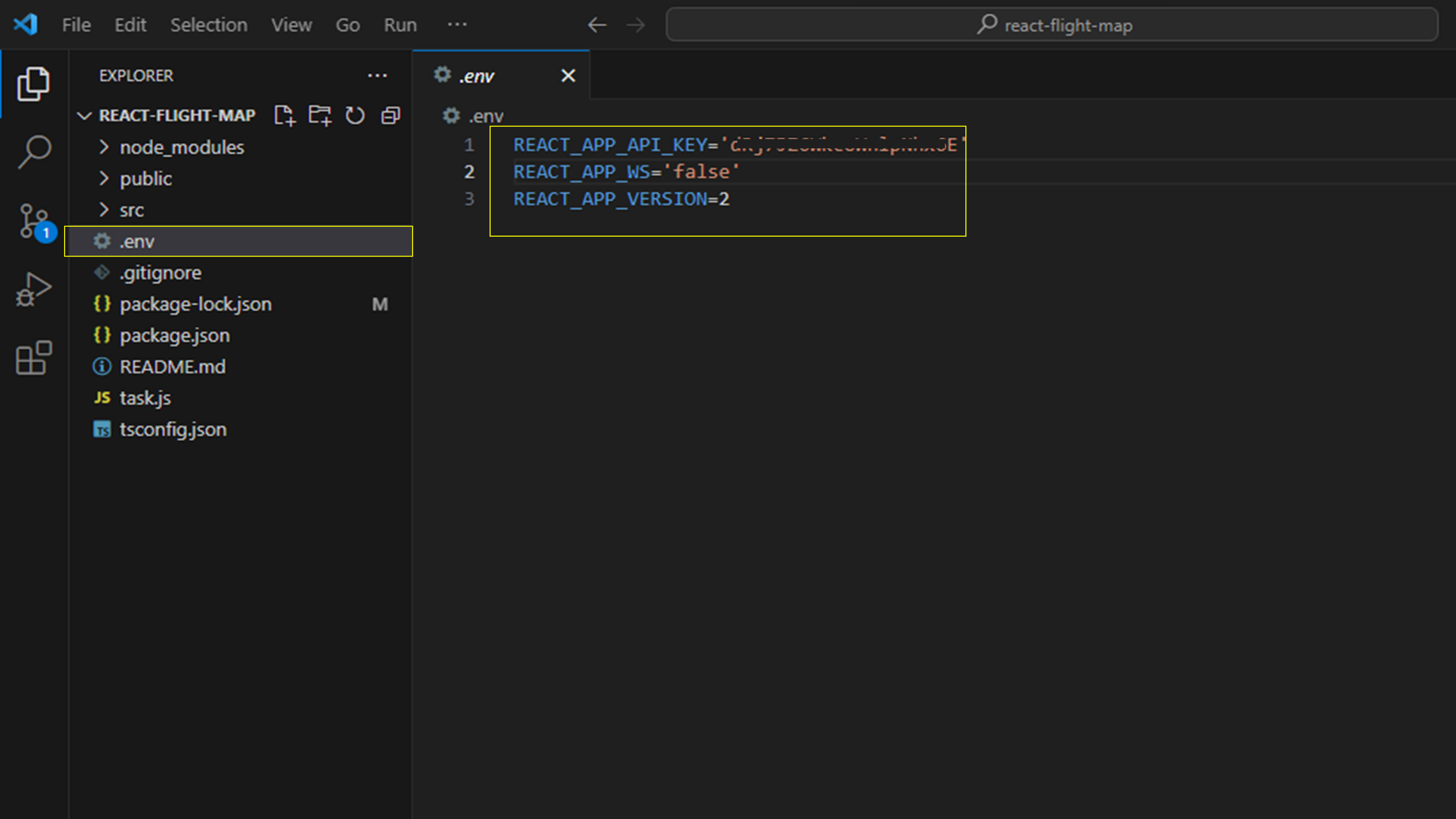
HTTP Requests

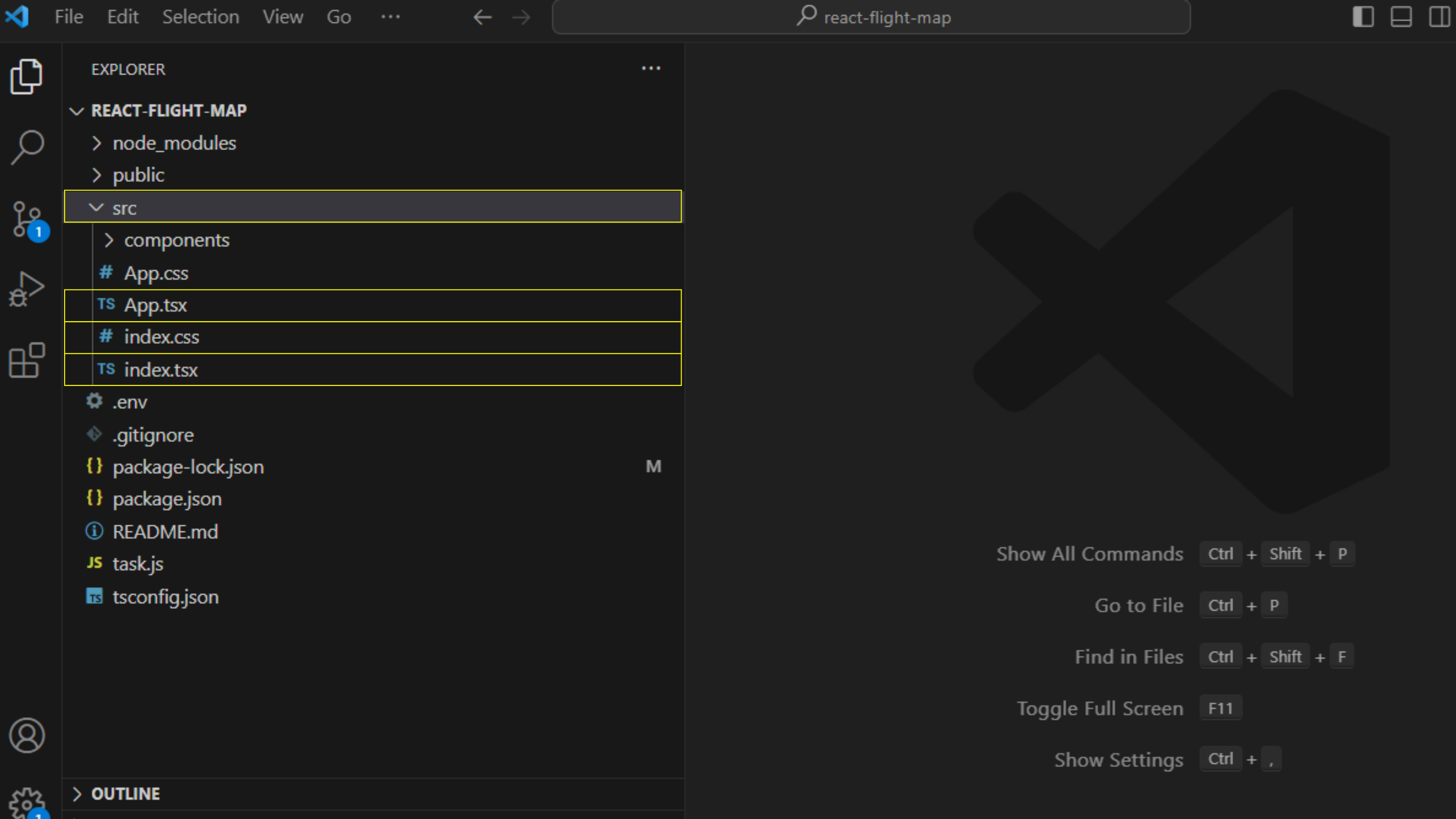


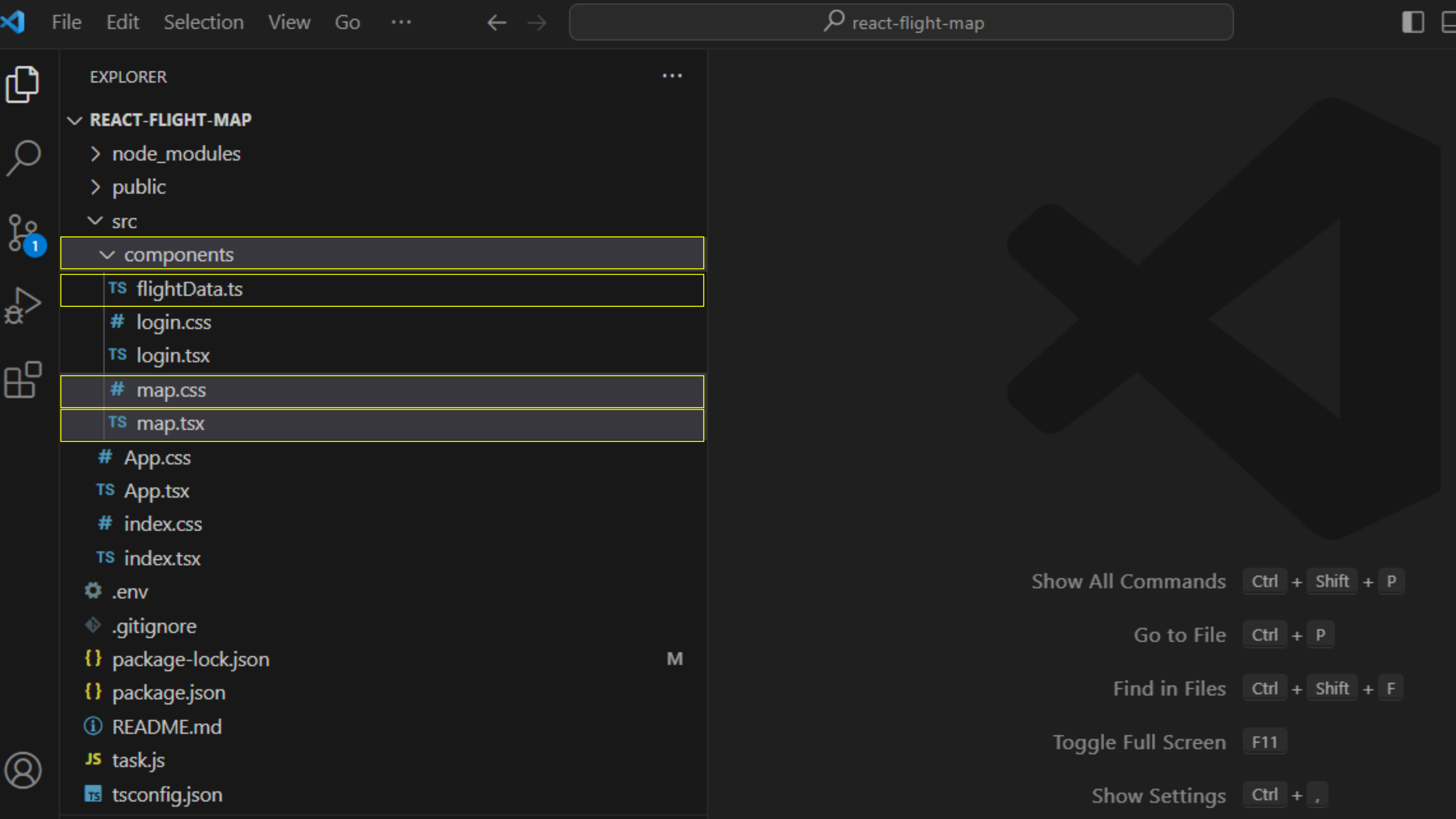
WebSocket

REACT_APP_API_KEY

REACT_APP_WS


File
Edit
Selection
View
Go
Run
react-flight-map
EXPLORER
REACT-FLIGHT-MAP
node_modules
public
src
.env
.gitignore
package-lock.json
M
package.json
README.md
task.js
tsconfig.json
.env
.env
1 REACT_APP_API_KEY='
2 REACT_APP_WS='false'
3 REACT_APP_VERSION=2


File
Edit
Selection
View
Go
react-flight-map
EXPLORER
REACT-FLIGHT-MAP
node_modules
public
src
components
App.css
App.tsx
index.css
index.tsx
.env
.gitignore
package-lock.json
M
package.json
README.md
task.js
tsconfig.json
OUTLINE
Show All Commands
Ctrl + Shift + P
Go to File
Ctrl + P
Find in Files
Ctrl + Shift + F
Toggle Full Screen
F11
Show Settings
Ctrl + ,

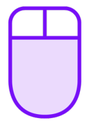

File
Edit
Selection
View
Go
react-flight-map
EXPLORER
REACT-FLIGHT-MAP
node_modules
public
src
components
flightData.ts
login.css
login.tsx
map.css
map.tsx
App.css
App.tsx
index.css
index.tsx
.env
.gitignore
package-lock.json
M
package.json
README.md
task.js
tsconfig.json
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Toggle Full Screen F11
Show Settings Ctrl + ,

The Map Component

Delivers the Interactive Map

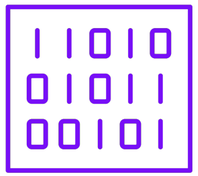
MapLibre GL Library

Functions for customizing marker styling and rendering flight data

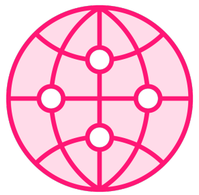


Manages user interactions, allows left and right clicks

WebSocket Integration



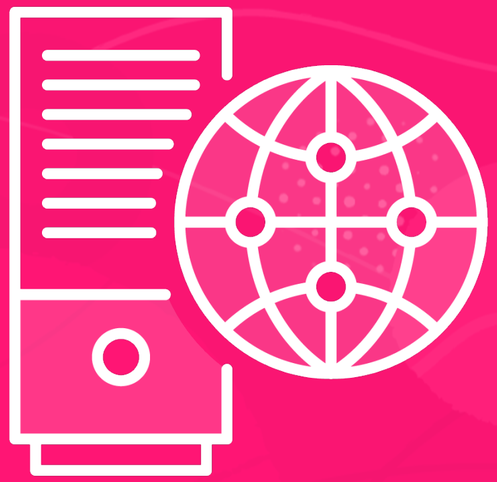
Display real-time flight data



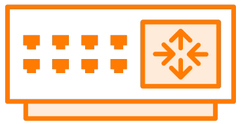
Establish WebSocket Connection

Creation of a WebSocket Server

Creating the Server



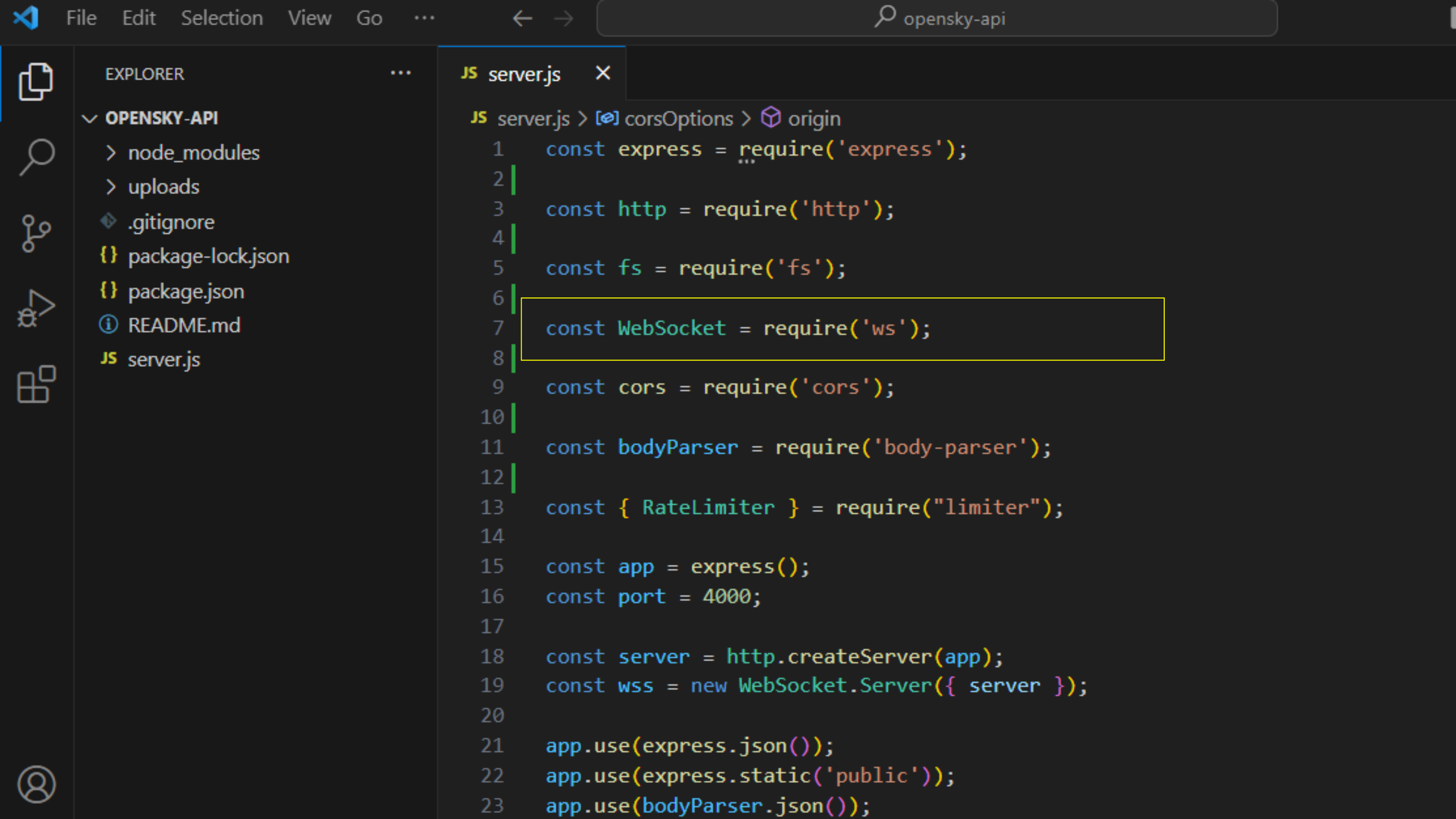
Server Setup



The server is set up to listen on port 4000

CORS is enabled to allow cross-origin requests

A WebSocket server is created for real-time data communication


File Edit Selection View Go ···
opensky-api
EXPLORER
OPENSKY-API
node_modules
uploads
.gitignore
package-lock.json
package.json
README.md
server.js
server.js
server.js > corsOptions > origin
1 const express = require('express');
2
3 const http = require('http');
4
5 const fs = require('fs');
6
7 const WebSocket = require('ws');
8
9 const cors = require('cors');
10
11 const bodyParser = require('body-parser');
12
13 const { RateLimiter } = require("limiter");
14
15 const app = express();
16 const port = 4000;
17
18 const server = http.createServer(app);
19 const wss = new WebSocket.Server({ server });
20
21 app.use(express.json());
22 app.use(express.static('public'));
23 app.use(bodyParser.json());

Restful APIs

/opensky-local

/history

File Edit Selection View Go ··· opensky-api

EXPLORER

OPENSKY-API

- node_modules
- uploads
- .gitignore
- package-lock.json
- package.json
- README.md
- server.js M

OUTLINE

TIMELINE

server.js M

server.js > ...

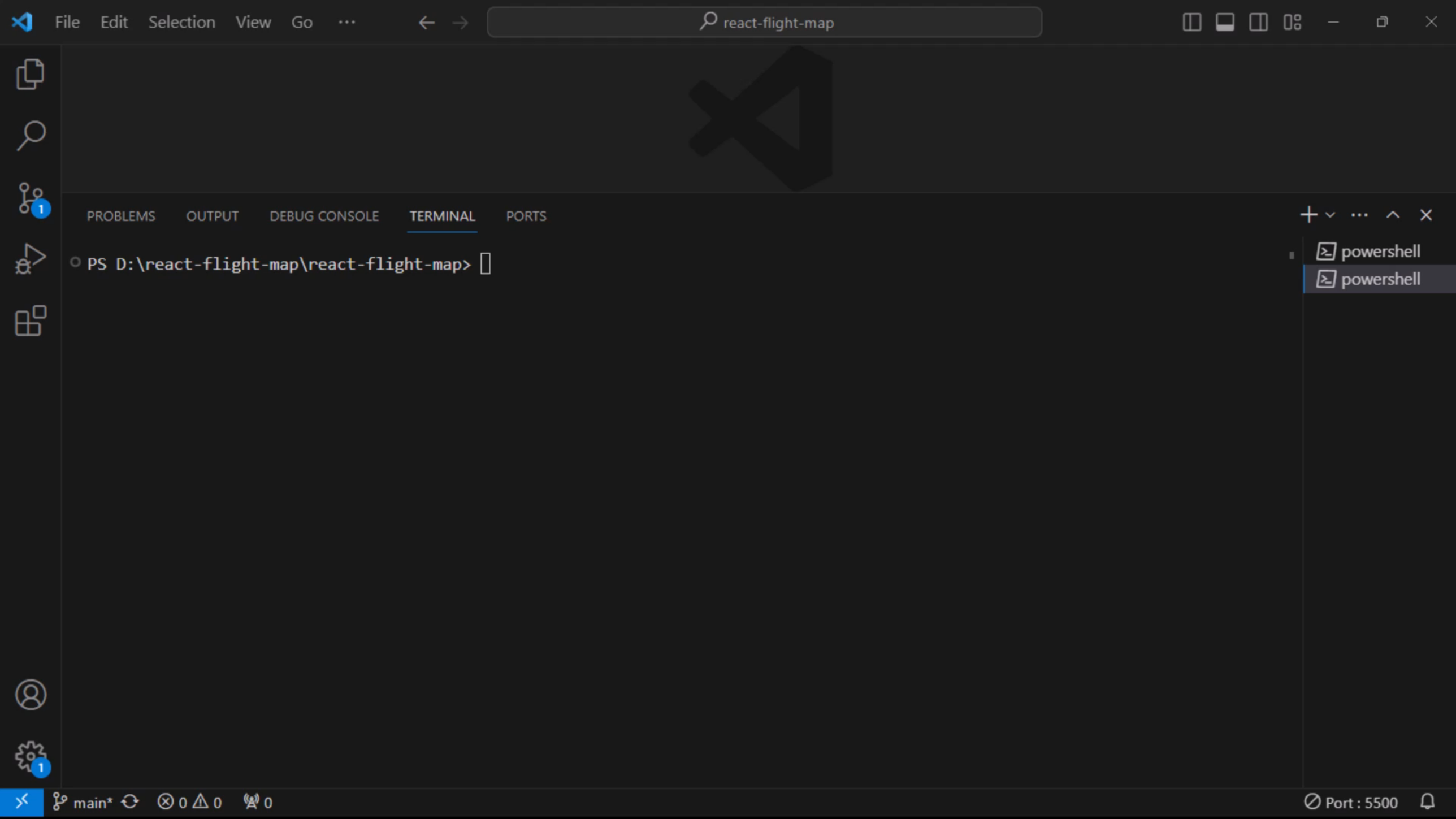
```
1  const express = require('express');
2
3  const http = require('http');
4
5  const fs = require('fs');
6
7  const WebSocket = require('ws');
8
9  const cors = require('cors');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\opensky-api\opensky-api>
```

powersh

powersh


File
Edit
Selection
View
Go
react-flight-map
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS D:\react-flight-map\react-flight-map>
powershell
powershell
main*
0 0
0
Port : 5500

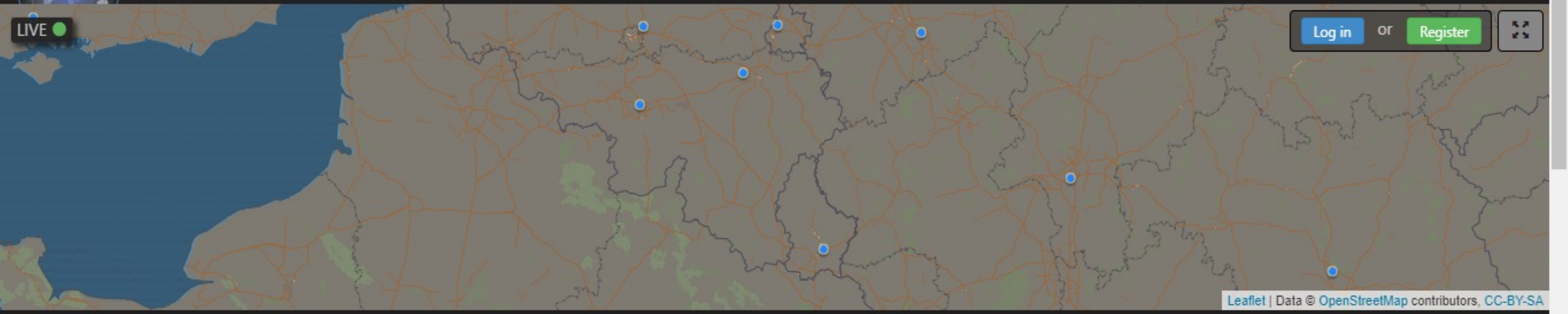
The OpenSky Network - Free AD

opensky-network.org

Network Community Data About Contribute My OpenSky

LIVE

Log in or Register



Open Air Traffic Data for Research.

×1000 MODE-S messages collected so far:
26,209,958,543

tracked a/c MODE-S:
64

MODE-S Messages / s:
53,939

# of seen aircraft:
506,048

FLARM messages collected so far:
6,776,346,006

tracked a/c FLARM:
0

FLARM Messages / s:
15

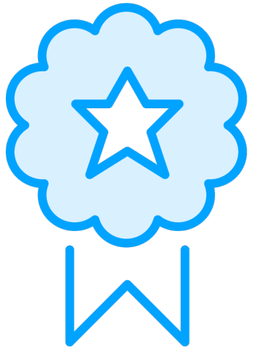
We are going through significant infrastructure changes to address and improve

Donate

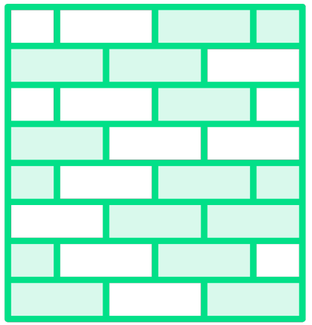
for better coverage

Data Types and Type Annotations in TypeScript

Importance of Data Types and Type Annotations



Precision and Correctness



Building Blocks of Error Free Integration

Boolean

Boolean.ts

```ts
let isAvailable: boolean = true;

let isHidden: boolean = false;

console.log(isAvailable);
console.log(isHidden);
```

Console

```
true
false
```

Number

Number.ts

```
let decimalNumber: number = 42;
let binaryNum: number = 0b1010;   // Binary literal
let octalNumber: number = 0o777;     // Octal literal
let hexdecNum: number = 0x1F; // Hexadecimal literal

console.log(decimalNumber);
console.log(binaryNumber);
console.log(octalNumber);
console.log(hexadecimalNumber);
```

Console

```
42
10
511
31
```

String

String.ts

```
let message: string = "Hello, TypeScript!";
console.log(message);
```

Console

```
Hello, TypeScript!
```

Template Strings

String.ts

```
let name: string = 'John';
let greeting: string = `Hello, ${name}!`;
console.log(greeting);
```

Console

```
Hello, John!
```

Array

Array.ts

```ts
let numbers: number[] = [1, 2, 3];
console.log(numbers);

let colors: Array<string> = ["red", "green", "blue"];
console.log(colors);
```

Console

```
[1, 2, 3]

["red", "green", "blue"]
```

Tuple

Tuple.ts

```ts
let person: [string, number] = ["Alice", 30];
console.log(person);
```

Console

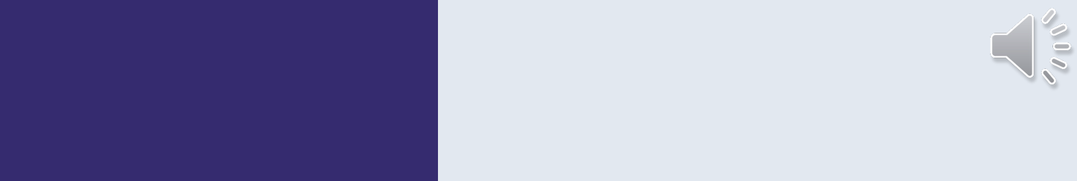
```
["Alice", 30]
```

Enum

Enum.ts

```ts
enum Day {
  Sunday,     // 0
  Monday,     // 1
  Tuesday,    // 2
  Wednesday,  // 3
  Thursday,   // 4
  Friday,     // 5
  Saturday    // 6
}

enum Color {
  Red = 1,
  Green = 2,
  Blue = 3
}
```

Console



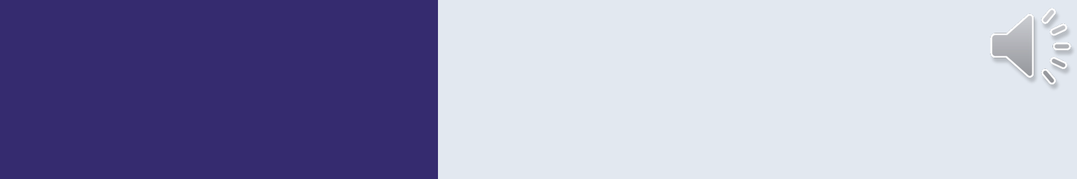
Enum

Enum.ts

```
enum Day {
  Sunday,     // 0
  Monday,     // 1
  Tuesday,    // 2
  Wednesday,  // 3
  Thursday,   // 4
  Friday,     // 5
  Saturday    // 6
}

let dayName: string = Day[2];
console.log(dayName);
```

Console

```
Tuesday
```



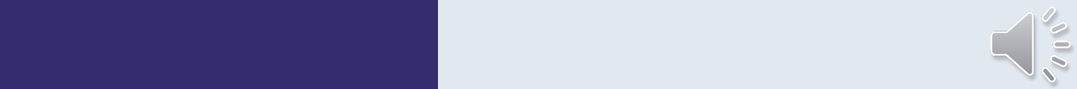
Any

Any.ts

```
let dynamicValue: any = "Hello, TypeScript!";
console.log(dynamicValue.length);
```

Console

```
17
```



Void

Void.ts

```ts
function greet(): void {
  console.log("Hello, TypeScript!");
}

const result: void = greet();
console.log(result);
```

Console

```
undefined
```

Null and Undefined

NullUndefined.ts

```
let data: string | null = null;
console.log(data);


let value: number;
console.log(value);
```

Console

```
null

Undefined
```

Never

Never.ts

```
function throwError(message: string): never {
throw new Error(message); }


function infiniteLoop(): never {
  while (true) {}
}


function checkNever(x: string | number): never {
  throw new Error('Unexpected value: ' + x);
}
```

Console

Unknown

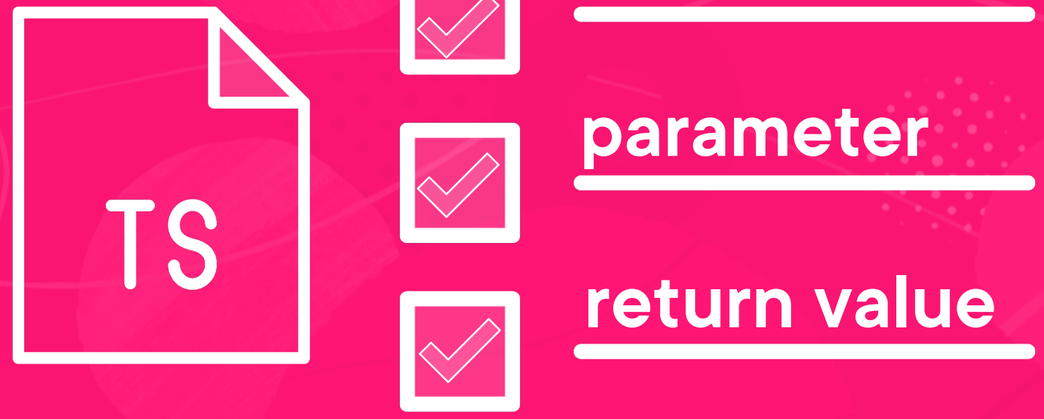
Unknown.ts

```ts
let userInput: unknown;
userInput = 5;          // Assign a number
console.log(userInput);
userInput = 'Hello';    // Assign a string
console.log(userInput);
userInput = true;       // Assign a boolean
console.log(userInput);
```

Console

```
5
Hello
true
```

What are Type Annotations?



Type Annotations

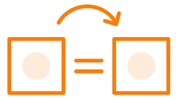
```
let age: number = 30;
let name: string = 'Alice';

function add(a: number, b: number): number {
  return a + b;
}
```

More Information

https://www.typescriptlang.org/docs/handbook/2/everyday-types.html#type-annotations-on-variables

Type Assertions in TypeScript

Specify Variable Type Explicitly.



Similar to Typecasting

Useful while Changing a Variable's Type

No impact on Runtime.

Type Assertions

Brackets.ts

```ts
let value: any = '42';
let numericValue: number = <number>value;

console.log(numericValue);

// Output: 42
```

AsKeyword.ts

```ts
let value: any = '42';
let numericValue: number = value as number;

console.log(numericValue);

// Output: 42
```

Functions, Arrow Functions, and Lambda Expressions

Importance of Functions

- Pivotal role in interacting with Web APIs.
- Modular Code: Readability & Maintainability.
- Code Reusability
- Ensure Data Integrity and Prevent Errors during API interactions.

Consequences of Inadequate Function Usage

- Disorganized and less maintainable code.
- Code duplication and reduced code reuse.
- Poor error handling and data validation.

TypeScript Function Syntax

```
function functionName() {
  body
  }
```

TypeScript Function Syntax

```
function add(num1: number, num2: number): number {
  return a + b;
}
```

Example

```
function addNumbers(num1: number, num2: number): number {
  return num1 + num2;
}

// Test the function
const result = addNumbers(5,7);
console.log(`The sum of 5 and 7 is: ${result}`);
```

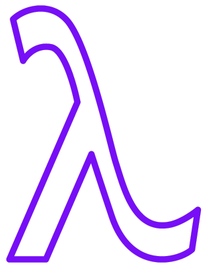
5+7 = 12

The sum of 5 and 7 is: 12

Understanding Arrow Functions in TypeScript

Shorthand Syntax for Defining Anonymous Functions

Fat Arrow Functions



Lambda Functions

Lexical Scoping of "this"

Arrow Function Syntax

```
(parameter1, parameter2, ..., parameterN) =>
 {
  expression;
 }
```

Arrow Function with Parameters

```
let addNumbers = (num1: number, num2: number): number => {
  return num1 + num2;
}
```

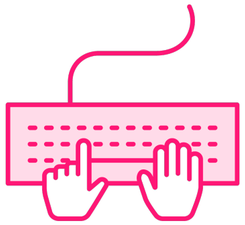
Arrow Function with Parameters

```
let sum = (x: number, y: number) => x + y;

console.log(sum(3, 4));
```

Arrow Function without Parameters

```
const greet = () => {
  return "Hello, World!";
};
```

Benefits of Arrow Functions

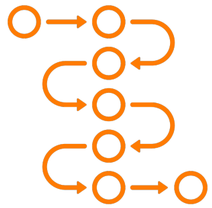


Conciseness

Lexical “this”

Ideal for Callbacks

Use Cases in Web API Consumption



Callback Functions

Event Listeners

Promise Chaining

Interfaces

Importance of Interfaces

Type Safety

Code Documentation

Code Reuse

Using Interfaces

- Define an Interface
- Implement the Interface

Defining an Interface

```
interface Person {
  firstName: string;
  lastName: string;
}
```

```
function greet(person: Person): string {
  return `Hello, ${person.firstName} ${person.lastName}!`;
}
```

```
const john: Person = {
  firstName: "John",
  lastName: "Doe",
};
```

```
console.log(greet(john));
```

Implementing an Interface

```
interface Shape {
  calculateArea(): number;
}
```

```
class Circle implements Shape {
  constructor(private radius: number) {}

  calculateArea(): number {
    return Math.PI * Math.pow(this.radius, 2);
  }
}
```

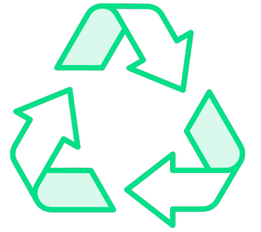
```
const circle = new Circle(5);
console.log(circle.calculateArea());
```

Modules and Namespaces for Code Organization

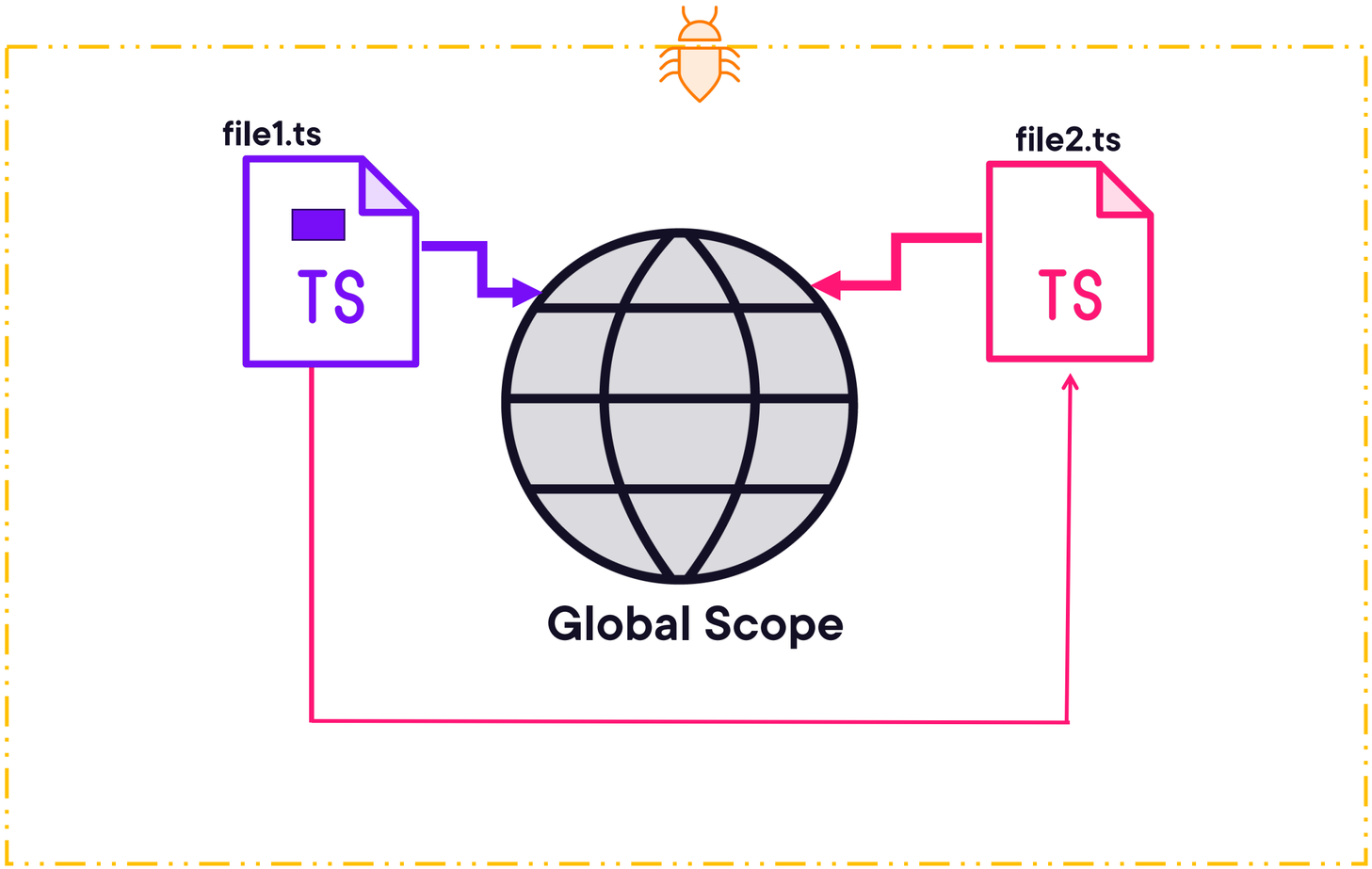
What are Modules?

Why Modules?

Encapsulation



Reusability


file1.ts
TS
file2.ts
TS
Global Scope

Referencing Modules in TypeScript

greet.ts

```
export var greeting : string = "Hello World!";
```

greetings.ts

```
import {greeting} from './greet';

console.log(greeting);
```

Namespaces

Interfaces | Class | Functions | Variables

Syntax

```
namespace <name>

{



}
```

Example

StringUtility.ts

```
namespace StringUtility
{
    function ToCapital(str: string): string {
        return str.toUpperCase();
    }

    function SubString(str: string, from: number, length: number = 0): string {
        return str.substr(from, length);
    }
}
```

Example

StringUtility.ts

```
namespace StringUtility {

    export function ToCapital(str: string): string {
        return str.toUpperCase();
    }

    export function SubString(str: string, from: number, length: number = 0): string {
        return str.substr(from, length);
    }
}
```

Namespace Vs. Modules

Namespace	VS	Module
• Logical Grouping and Avoiding Naming Collisions		• Code Encapsulation, Reusability, and Sharing
• Use export to expose functionality		• The export keyword controls visibility within
• Requires export for accessibility.		• Exports accessible outside module
• Include with triple-slash reference. e.g. *<reference path="path to namespace file"/>*		• Import using the import keyword
• Include with HTML <script> tag		• Import with module loader API
• Cannot declare Dependencies		• Can declare Dependencies

Practical Implementation

```
namespace FlightDataNamespace {
    export interface FlightData {
      states: StateData[];

    }

    export interface StateData {
      icao24: string;
      callsign: string;
      country: string;
      longitude: number;
      latitude: number;
      altitude: number;
      trueTrack: number;
    }

  }

export default FlightDataNamespace;
```

```
type HistoryElement = {
    latitude: number;
    longitude: number;
    timestamp: number;
  };
```

```
import { FlightData, StateData } from './flightData’;
```

```
import FlightDataNamespace from './flightData';
```

```
const initialFlightData: FlightDataNamespace.FlightData = { states: [] };
```

TypeScript Fundamentals for API Consumption

Consuming Web APIs with Typescript 5

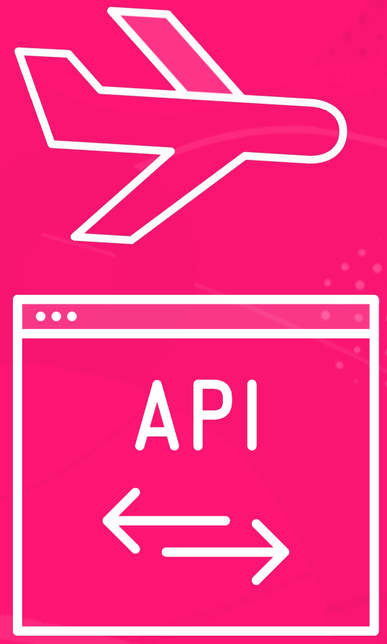
Defining Our Project

Allen O'Neill

CTO / Senior Engineer, Microsoft Regional Director & MVP

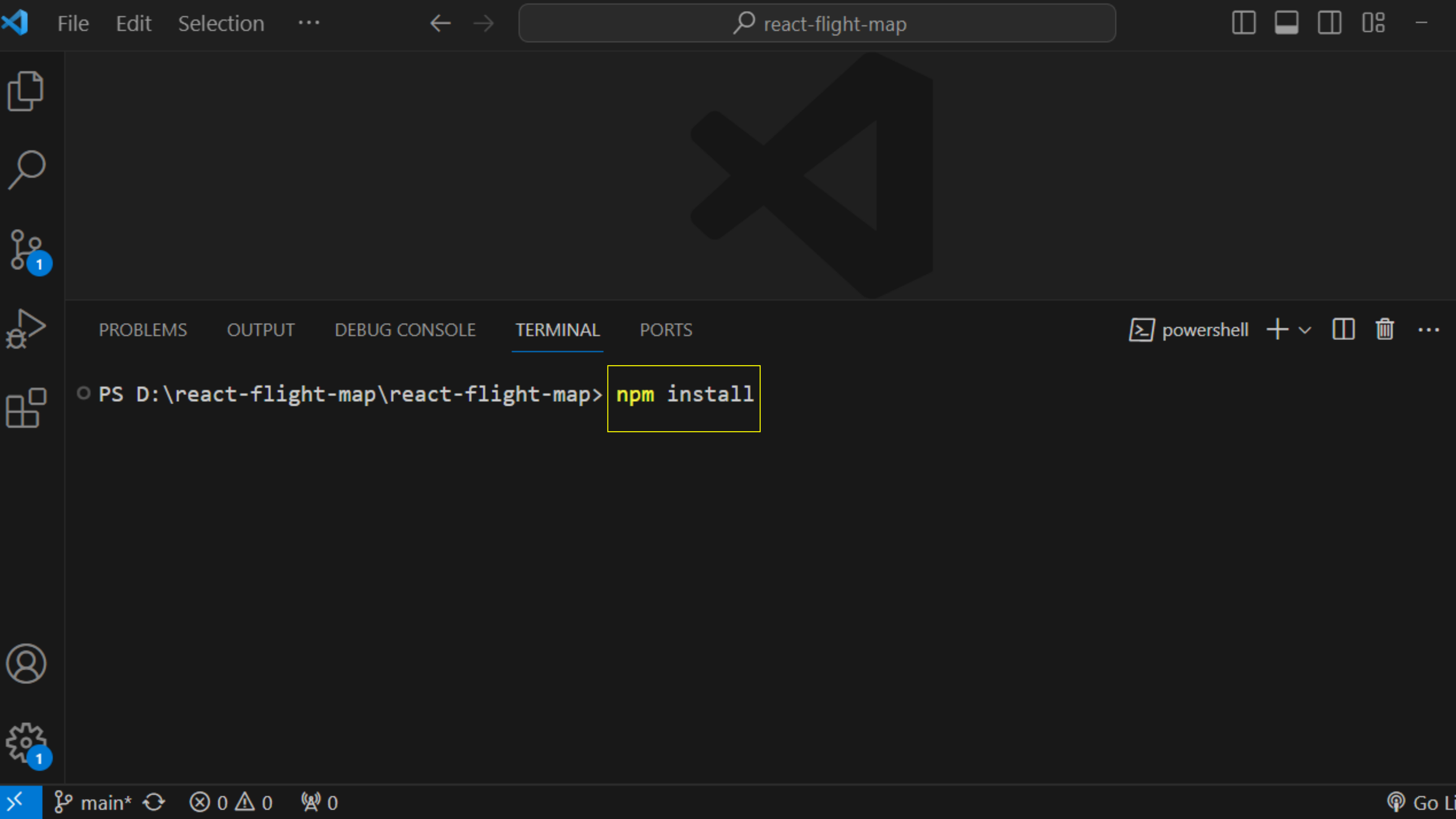
@DataBytesAI | www.datalabs.io

The Crucial Role of Web APIs in Our Project




React Flight App
localhost:3000
Geneva
Lyon
Milan
Venice
Slovenia
Croatia
Belgrade
Serbia
Bosnia and Herzegovina
Bordeaux
Monaco
San Marino
Marseille
Bilbao
Italy
Montenegro
Sofia
Vatican City
Skopje
Albania
Naples
Thessaloniki
Greece
Palermo
Athens
Catania
Annaba
Tunis
Constantine
Sousse
Malta
Gibraltar
Oujda
Fez
Casablanca
Tunisia
Zuara
Misrata
Benghazi
Ghardaia
Safi
Marrakesh
maptiler
© MapTiler © OpenStreetMap contributors

https://github.com/datalabs-io/pluralsight-webapi-typescript-v5-demo .

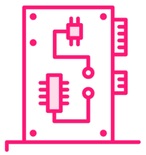

File
Edit
Selection
react-flight-map
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell
PS D:\react-flight-map\react-flight-map> npm install
main*
0
0
0

https://github.com/datalabs-io/pluralsight-webapi-typescript-v5-demo-opensky

Interactive Flight-Map Application

Environment Variables

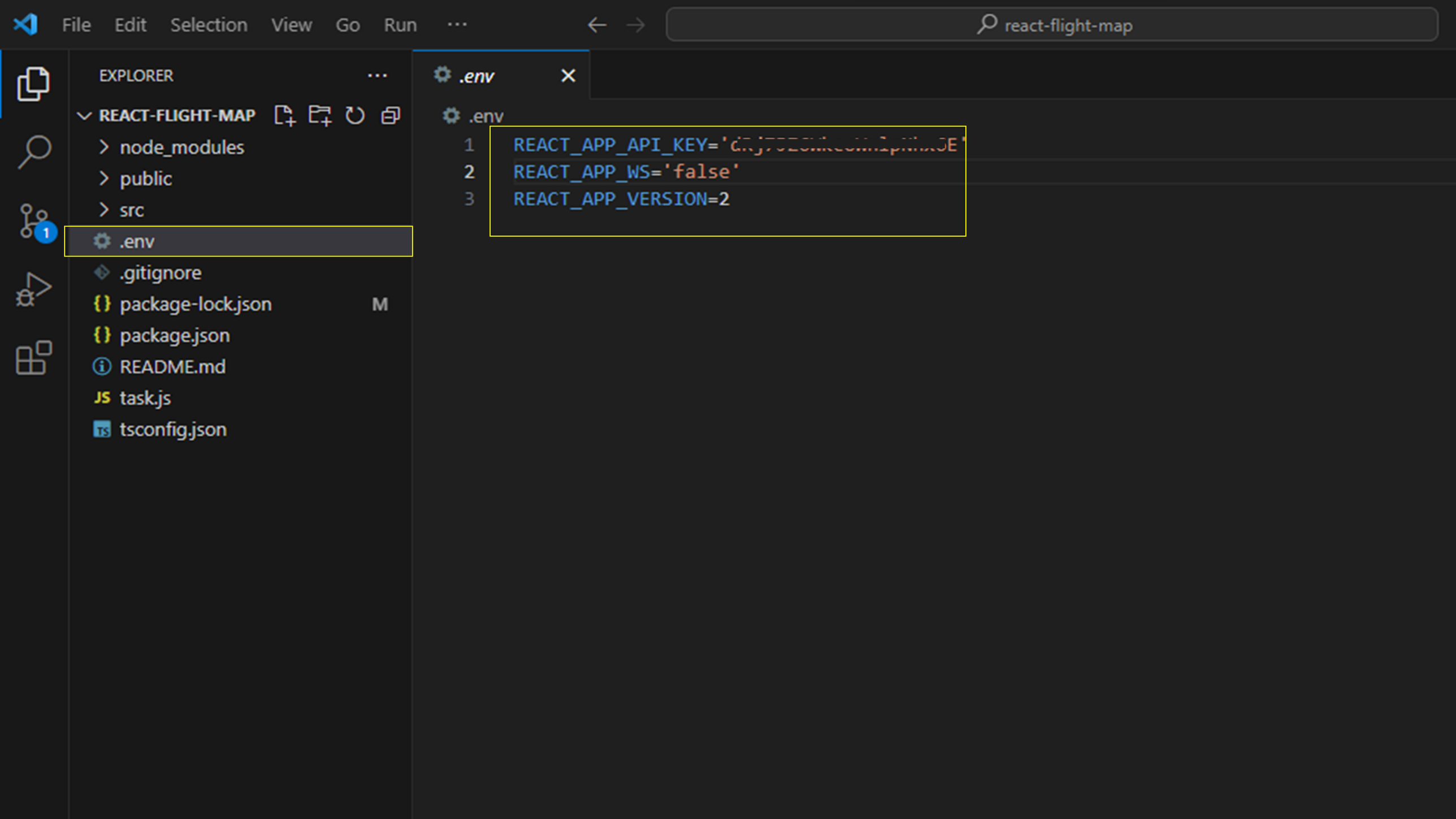
HTTP Requests

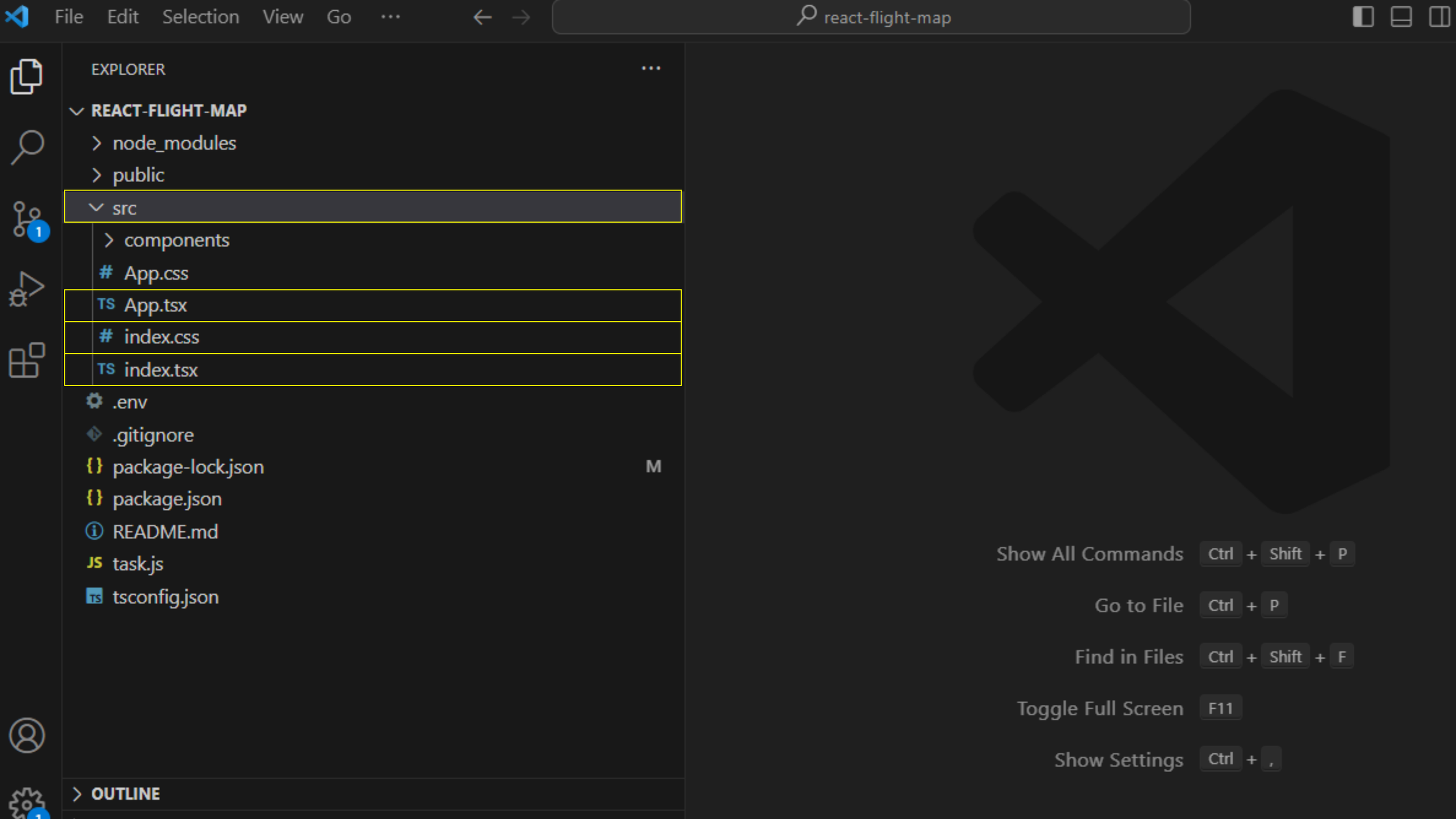


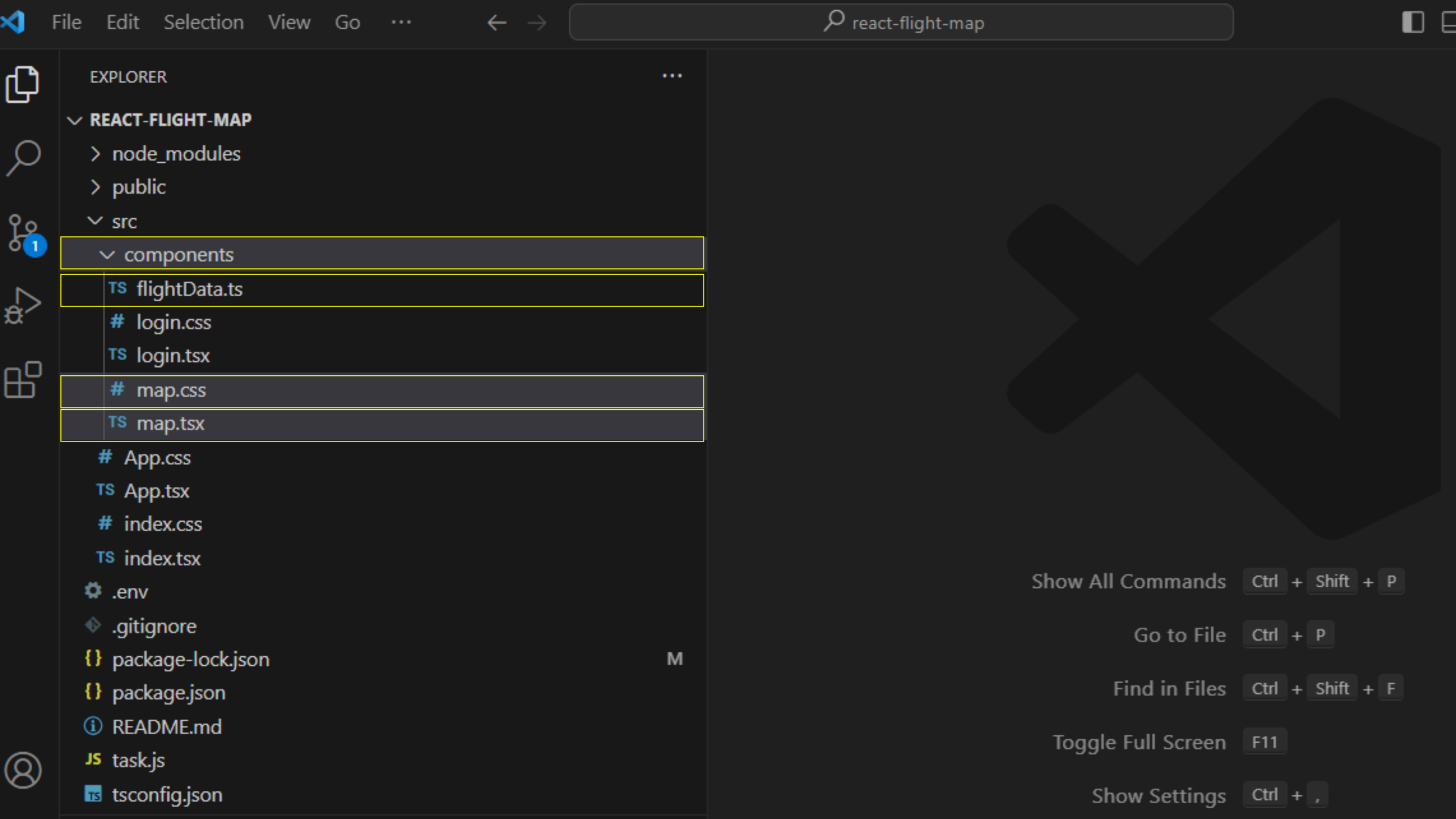
WebSocket

REACT_APP_API_KEY

REACT_APP_WS


File
Edit
Selection
View
Go
Run
react-flight-map
EXPLORER
REACT-FLIGHT-MAP
node_modules
public
src
.env
.gitignore
package-lock.json
M
package.json
README.md
task.js
tsconfig.json
.env
.env
1 REACT_APP_API_KEY='
2 REACT_APP_WS='false'
3 REACT_APP_VERSION=2


File
Edit
Selection
View
Go
react-flight-map
EXPLORER
REACT-FLIGHT-MAP
node_modules
public
src
components
App.css
App.tsx
index.css
index.tsx
.env
.gitignore
package-lock.json
M
package.json
README.md
task.js
tsconfig.json
OUTLINE
Show All Commands
Ctrl + Shift + P
Go to File
Ctrl + P
Find in Files
Ctrl + Shift + F
Toggle Full Screen
F11
Show Settings
Ctrl + ,

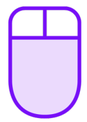

File
Edit
Selection
View
Go
react-flight-map
EXPLORER
REACT-FLIGHT-MAP
node_modules
public
src
components
flightData.ts
login.css
login.tsx
map.css
map.tsx
App.css
App.tsx
index.css
index.tsx
.env
.gitignore
package-lock.json
M
package.json
README.md
task.js
tsconfig.json
Show All Commands Ctrl + Shift + P
Go to File Ctrl + P
Find in Files Ctrl + Shift + F
Toggle Full Screen F11
Show Settings Ctrl + ,

The Map Component

Delivers the Interactive Map

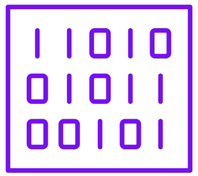
MapLibre GL Library

Functions for customizing marker styling and rendering flight data

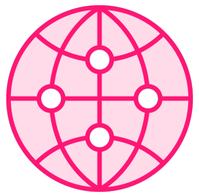


Manages user interactions, allows left and right clicks

WebSocket Integration



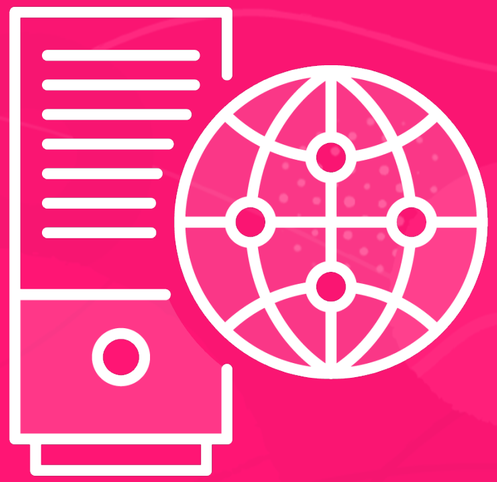
Display real-time flight data



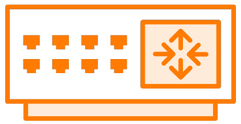
Establish WebSocket Connection

Creation of a WebSocket Server

Creating the Server



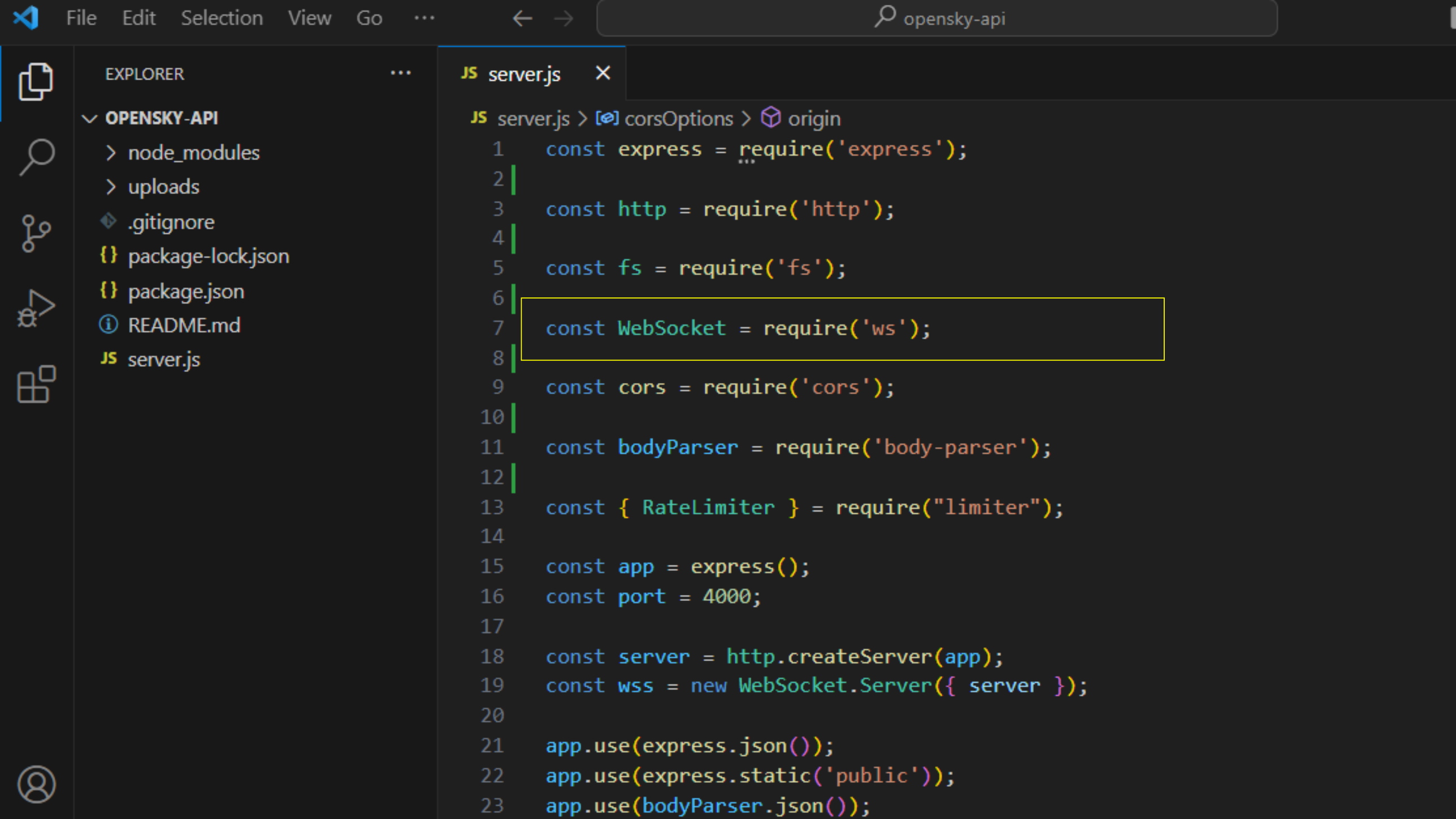
Server Setup



The server is set up to listen on port 4000

CORS is enabled to allow cross-origin requests

A WebSocket server is created for real-time data communication


File Edit Selection View Go ···
opensky-api
EXPLORER
OPENSKY-API
node_modules
uploads
.gitignore
package-lock.json
package.json
README.md
server.js
server.js
server.js > corsOptions > origin
1 const express = require('express');
2
3 const http = require('http');
4
5 const fs = require('fs');
6
7 const WebSocket = require('ws');
8
9 const cors = require('cors');
10
11 const bodyParser = require('body-parser');
12
13 const { RateLimiter } = require("limiter");
14
15 const app = express();
16 const port = 4000;
17
18 const server = http.createServer(app);
19 const wss = new WebSocket.Server({ server });
20
21 app.use(express.json());
22 app.use(express.static('public'));
23 app.use(bodyParser.json());

Restful APIs

/opensky-local

/history

File Edit Selection View Go ··· opensky-api

EXPLORER

OPENSKY-API

- node_modules
- uploads
- .gitignore
- package-lock.json
- package.json
- README.md
- server.js M

OUTLINE

TIMELINE

server.js M

server.js > ...

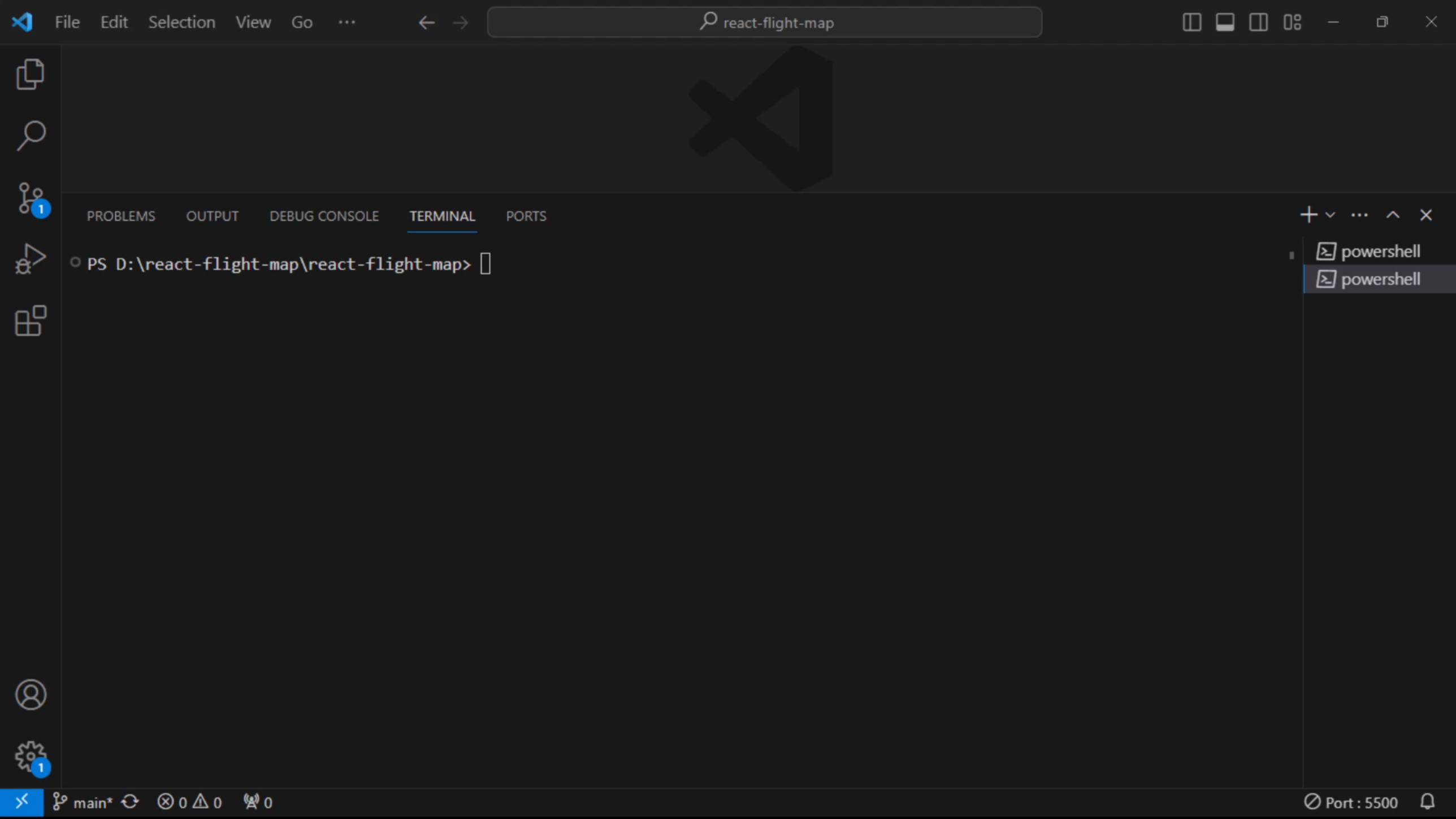
```
1  const express = require('express');
2
3  const http = require('http');
4
5  const fs = require('fs');
6
7  const WebSocket = require('ws');
8
9  const cors = require('cors');
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS D:\opensky-api\opensky-api>
```

powersh

powersh


File
Edit
Selection
View
Go
react-flight-map
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
PS D:\react-flight-map\react-flight-map>
powershell
powershell
main*
0 0
0
Port : 5500

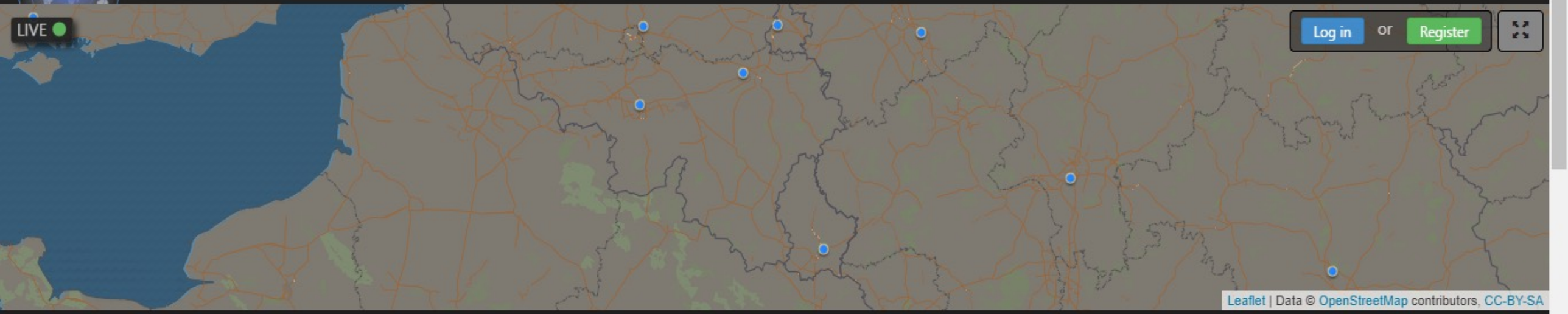
The OpenSky Network - Free AD

opensky-network.org

Network Community Data About Contribute My OpenSky

LIVE

Log in or Register



Open Air Traffic Data for Research.

×1000 MODE-S messages collected so far:
26,209,958,543

tracked a/c MODE-S:
64

MODE-S Messages / s:
53,939

# of seen aircraft:
506,048

FLARM messages collected so far:
6,776,346,006

tracked a/c FLARM:
0

FLARM Messages / s:
15

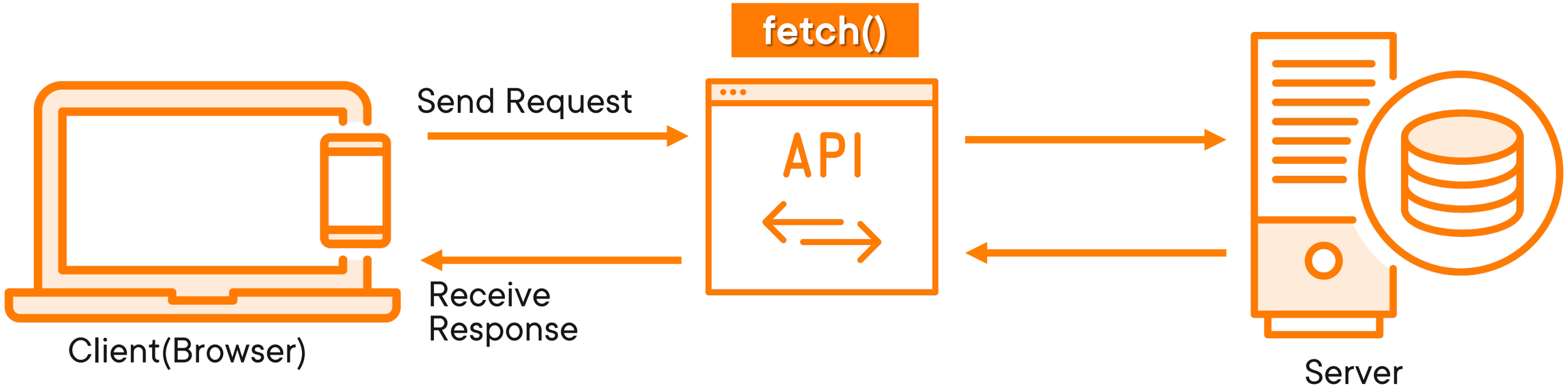
We are going through significant infrastructure changes to address and improve

Donate

for better coverage

Fetch API

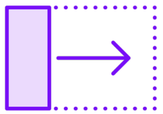
Fetch API


fetch()
Send Request
API
Receive
Response
Client(Browser)
Server

Key Features and Importance of Fetch API

Simplicity and Promises

Better Handling of Responses

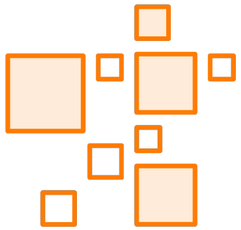


Flexibility

Cross-Origin Requests

Key Features and Importance of Fetch API

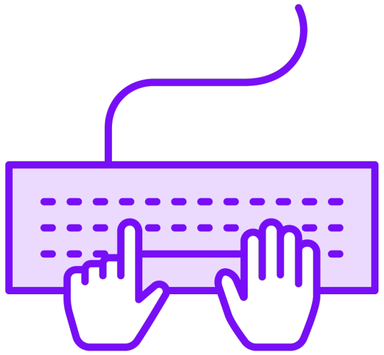
Streaming and Progress Tracking



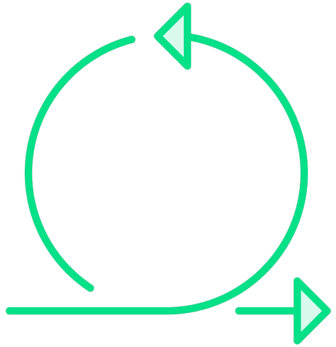
Native Integration

Modern Web Development

What is New in Fetch API in TypeScript 5?



Improved Type Safety



Support for Async/Await

Support for Promises

Fetch API Alternatives

XMLHttpRequest

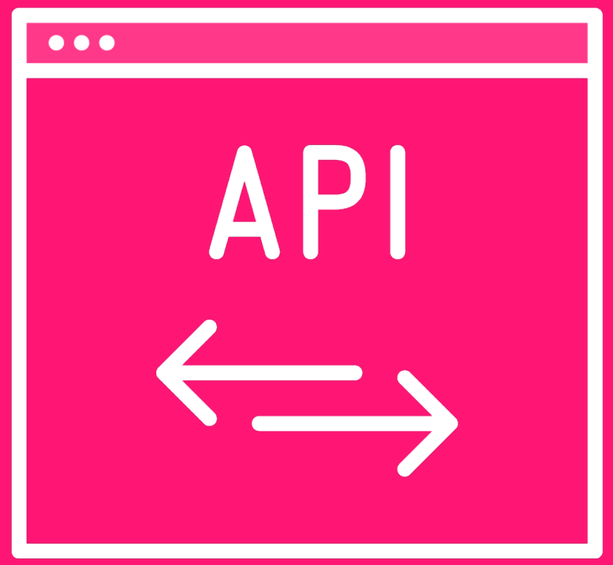
Axios

SuperAgent

Got

Making GET and POST Requests with Fetch

Introduction to REST API Concept and HTTP Verbs



What is a REST API?

- Representational State Transfer
- REST API Principles:
  - Client-server architecture
  - Stateless design
  - Uniform interface
  - Standard HTTP methods

HTTP Verbs

GET

POST

PUT

DELETE

HTTP Verbs

HEAD

CONNECT

OPTIONS

TRACE

HTTP Verbs (Non-Standard)

PATCH

MERGE

PROPFIND

PROPPATCH

MKCOL

COPY

MOVE

LOCK

UNLOCK

GET Vs. POST

Aspect	GET	VS	POST
Purpose	Retrieve data from a resource		Create or modify a resource
Data in Requests	Appends data to URL (query params)		Sends data in the request body
Caching	Responses are cacheable		Responses are typically not cacheable
Idempotence	Idempotent		Non-idempotent
Bookmarks/History	Suitable for bookmarks and revisits		May not be suitable for bookmarks or revisits
Security	Less secure for sensitive data, as data is visible in URL		More secure for sensitive data, as data is in the request body
Request Size	Limited by URL length and browser restrictions		Typically allows larger payloads due to request body usage

```
fetch('https://jsonplaceholder.typicode.com/users/2')
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json(); // Parse JSON response
  })
  .then(data => {
    // Use the retrieved data
    console.log('Response Data:', data);
  })
  .catch(error => {
    console.error('GET Error:', error);
  });
```

```
// Define data to send in the POST request
const postData = {
  name: 'John Doe',
  email: 'john@example.com',
};
// Perform a POST request using Fetch
fetch('https://jsonplaceholder.typicode.com/users', {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
  },
  body: JSON.stringify(postData), // Convert data to JSON
})
  .then(response => {
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }
    return response.json(); // Parse JSON response
  })
  .then(data => {
    // Handle the response from the server
    console.log('POST Response Data:', data);
  })
  .catch(error => {
    console.error('POST Error:', error);
  });
```

Handling JSON Responses and Data Parsing

Simple JSON Object

```
const data = {
  name: "John Doe",
  age: 30,
  active: true,
};
```

JSON Arrays

```
const numbers = [1, 2, 3, 4, 5];
```

Nested JSON Objects

```
const person = {
  name: "John Doe",
  address: {
    street: "123 Main St",
    city: "Exampleville",
  },
};
```

Array of Nested JSON Objects

```
const people = [
  {
    name: "Alice",
    age: 25,
  },
  {
    name: "Bob",
    age: 32,
  },
];
```

Complex JSON Structures

```
const complexData = {
  users: [
    {
      name: "User 1",
      posts: [
        {
          title: "Post 1",
          comments: ["Comment 1", "Comment 2"],
        },
        // More posts...
      ],
    },
    // More users...
  ],
};
```

JSON API Responses

```
fetch("https://api.example.com/data")
  .then((response) => response.json())
  .then((data) => {
    // Handle the JSON data
  });
```

Parsing Errors

```
try {
  const parsedData = JSON.parse("invalid json");
} catch (error) {
  // Handle the parsing error
}
```

External Data Sources

```
import fs from 'fs’;

const jsonDataFromFile = fs.readFileSync('data.json', 'utf-8’);

const parsedFileData = JSON.parse(jsonDataFromFile);
```

Custom Data Transformation

```
const rawData = '{ "value": 42 }';

const parsedData = JSON.parse(rawData, (key, value) => {

  if (key === "value") return value * 2; // Custom transformation

  return value;
});
```

Typed JSON Parsing

```
interface User {
  id: number;
  username: string;
}

const rawData = '{ "id": 1, "username": "user123" }';
const parsedData = JSON.parse(rawData) as User;
```

Asynchronous Programming in TypeScript

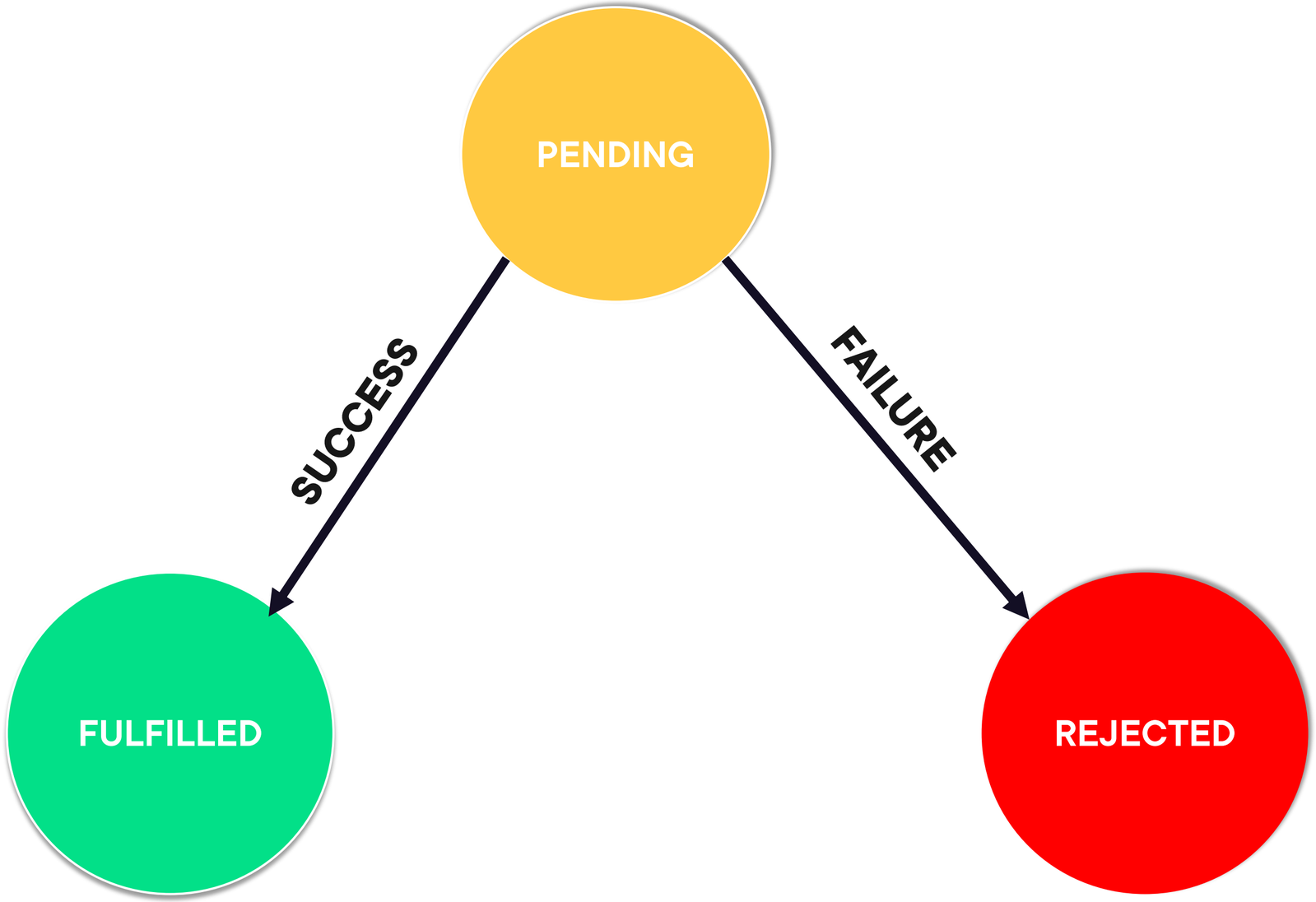
What & Why Promises?

What are Promises?

- Promises resolve future results or errors.
- Useful for non-blocking, asynchronous code execution.

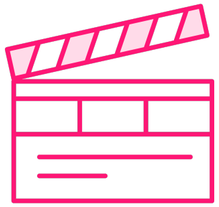
Why are Promises Useful?

- Promises enhance code structure.
- Simplify error handling and control.


PENDING
SUCCESS
FAILURE
FULFILLED
REJECTED

How to Use Promises?

Create a Promise



Specify what should happen when the Promise is resolved successfully

Handle errors if the Promise is rejected

```
function fetchData() {
  return new Promise((resolve, reject) => {
    // Use the fetch() function to make an HTTP GET request
    fetch('https://jsonplaceholder.typicode.com/posts/1')
      .then(response => {
        // Check if the response status is OK (status code 200)
        if (!response.ok) {
          throw new Error('Network response was not ok');
        }
        // Parse the response body as JSON
        return response.json();
      })
      .then(data => {
        // Resolve the Promise with the JSON data
        resolve(data);
      })
      .catch(error => {
        // Reject the Promise with an error message
        reject(error);
      });
  });
}

// Use the fetchData() function
fetchData()
  .then(result => {
    console.log('Success:', result);
  })
  .catch(error => {
    console.error('Error:', error);
  });
```

Async/Await

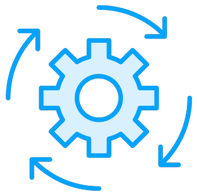
Why async/await?

Improved Readability

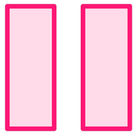
Reduced Errors

Improved Performance

How to Use Promises?



Define a Function as async

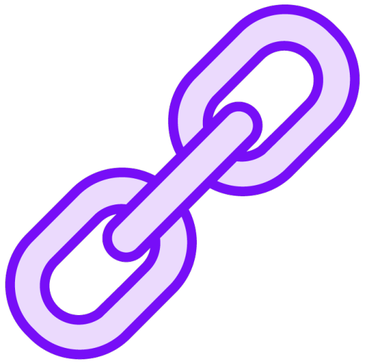


Use await until Promise is resolved, then continue

Handle errors using try-catch blocks

```javascript
async function fetchData() {
  try {
    // Use the await keyword before the fetch() function to pause execution until the Pr
    const response = await fetch('https://jsonplaceholder.typicode.com/posts/1');

    // Check if the response status is OK (status code 200)
    if (!response.ok) {
      throw new Error('Network response was not ok');
    }

    // Use await to parse the response body as JSON
    const data = await response.json();

    // Log the retrieved data
    console.log('Success:', data);
  } catch (error) {
    // Handle errors gracefully
    console.error('Error:', error);
  }
}
// Call the asynchronous function
fetchData();
```

Chaining Promises



Chaining Promises:

- Chain Promises with .then() for sequential async operations.
- Sequential, one-after-another execution in chained Promises.

Why Chain Promises?

- Clear and organized way to structure and execute a sequence of asynchronous tasks.
- Handling dependencies between asynchronous operations.

```
const fetchUserData = () => {
  return fetch('https://jsonplaceholder.typicode.com/users/1')
    .then((response) => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then((userData) => {
      console.log('User Data:', userData);
    })
    .catch((error) => {
      console.error('Error:', error);
    });
};

fetchUserData();
```

Practical Implementation

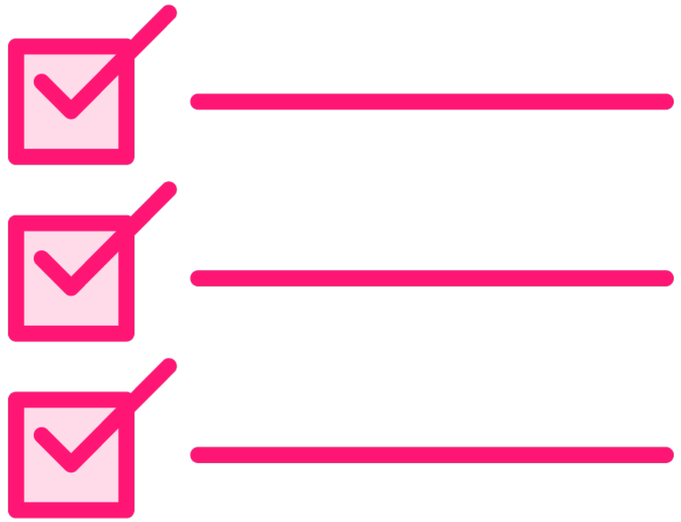
```
const postDataToServer = async () => {
  const dataToSend = { key: 'value' };

  try {
    const response = await fetch(`http://localhost:4000/opensky-
local?fileId=data${counterId}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json', // Set the appropriate content type
      },
      body: JSON.stringify(dataToSend),
    });

    if (!response.ok) {
      throw new Error(`Request failed with status ${response.status}`);
    }

    const textResponse = await response.text();
    setFlightData(textResponse);

  } catch (error) {
    console.error('Error:', error);
  }
};
```

```
const postJSONDataToServer = async () => {
  const dataToSend = { key: 'value' };
  try {
    const response = await fetch(`http://localhost:4000/opensky-
local?fileId=data${counterId}`, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json', // Set the appropriate content type
      },
      body: JSON.stringify(dataToSend),
    });
    if (!response.ok) {
      throw new Error(`Request failed with status ${response.status}`);
    }
    const jsonData = await response.json();
    setFlightData(jsonData);
  } catch (error) {
    console.error('Error:', error);
  }
};
```

```
const fetchDataFromOpenSkyLocal = async () => {
  if (requestStatus === 'ok') {
    try {
      const response = await
fetch(`http://localhost:4000/v${version}/opensky-
local?fileId=data${counterId}`);
      if (response.ok) {
        const data = await response.json();
        setFlightData(data.data);
        setRequestStatus('ok');
      } else {
        console.error('Error fetching data from the
mock OpenSky API:', response.status,
response.statusText);
        setRequestStatus('error');
      }
    } catch (error) {
      console.error('Error fetching data from the mock
OpenSky API:', error);
    }
  }
};
```

Type Checking and Type Inference

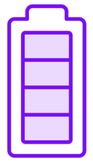
What is Type Checking?



Why is Type Checking Important?

Prevent Errors

Improve Readability

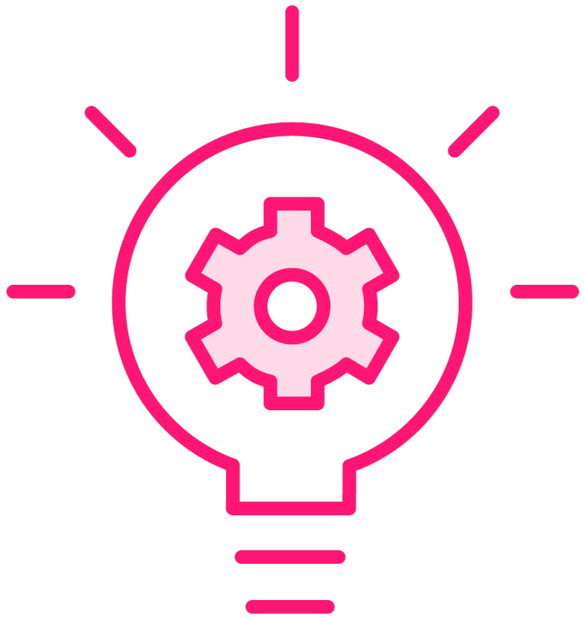


Improve Maintainability

Improve Code Quality

```typescript
interface UserProfile {
  id: number;
  name: string;
  email: string;
}

async function fetchUserProfile(userId: number): Promise<UserProfile> {
  const response = await fetch(`https://jsonplaceholder.typicode.com/users/${userId}`);
  if (!response.ok) {
    throw new Error('Failed to fetch user profile');
  }
  const userProfile = await response.json();

  // TypeScript enforces type checking for 'userProfile'
  console.log('User Profile:', userProfile);

  return userProfile;
}

// Usage
const userId = 1; // Replace with an actual user ID
fetchUserProfile(userId)
  .then((userProfile) => {
    // Accessing 'userProfile' with confidence
    console.log('User ID:', userProfile.id);
    console.log('Name:', userProfile.name);
    console.log('Email:', userProfile.email);
  })
  .catch((error) => {
    console.error('Error:', error);
  });
```

What Is Type Inference?



Importance of Type Inference

Save developers time and effort

Improve the readability of code

Reduce errors

```typescript
interface Post {
    userId: number;
    id: number;
    title: string;
    body: string;
  }

  async function fetchPost(postId: number): Promise<Post> {
    const response = await fetch(`https://jsonplaceholder.typicode.com/posts/${postId}`);
    if (!response.ok) {
      throw new Error('Failed to fetch post');
    }
    const post = await response.json();

    // TypeScript automatically infers the structure of 'post'
    console.log('Post Data:', post);

    return post;
  }

  // Usage
  const postId = 1; // Replace with an actual post ID
  fetchPost(postId)
    .then((post) => {
      // Accessing 'post' with confidence
      console.log('User ID:', post.userId);
      console.log('Title:', post.title);
      console.log('Body:', post.body);
    })
    .catch((error) => {
      console.error('Error:', error);
    });
```

Defining Custom Types for API Responses

What are Custom Types?

Importance of Custom Types

Data Precision

Code Clarity

Type Safety

Improved Documentation

Better Error Messages

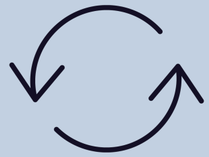
Setting Up Custom Types in TypeScript

Using Type Aliases and Intersection Types

```
interface
type User = {
type Product = {
  id: number;
  name: string;
  username: string;
  price: number;
};id: 1,
};
  username: 'example_user',
};
type Admin = {
  // TypeScript infers the custom type for
//id: number;
'user'.
  role: 'admin';
};

type UserOrAdmin = User | Admin;
```

Type Casting and Type Assertions

Type Casting and Type Assertions



Type Casting

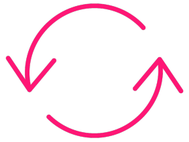
`as` `Number()` `String()`

JavaScript-style Casting Functions

Type Assertion

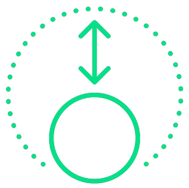
`<Type>` `as`

Importance of Type Casting and Type Assertion



Tell TypeScript a value's type

Handling Dynamic Data



Data Transformation

```typescript
// Define a custom type for post data
interface Post {
    userId: number;
    id: number;
    title: string;
    body: string;
  }

  // Fetch post data from JSONPlaceholder API
  fetch('https://jsonplaceholder.typicode.com/posts/1')
    .then((response) => {
      if (!response.ok) {
        throw new Error('Network response was not ok');
      }
      return response.json();
    })
    .then((postData) => {
      // Type assertion to specify the expected structure
      const post = postData as Post;

      // Type casting to ensure 'userId' is treated as a number
      const userId = Number(post.userId);

      console.log('Post Data:', post);
      console.log('User ID as Number:', userId);
    })
    .catch((error) => {
      console.error('Error:', error);
    });
```

Practical Implementation

```
interface TextApiResponse {
  text: string;
}
```

```
interface JsonApiResponse {
  data: FlightDataNamespace.FlightData;
}
```

```
// Fetch and Handle Text Response with Type Assertion
const fetchDataFromServer = async () => {
  try {
      // ... (fetch request)
      const textData = await response.text();

      // Type assertion to treat textData as TextApiResponse
      const response: TextApiResponse = {
          text: textData,
      };

      // Now we can work with response as a TextApiResponse
      setFlightData(response);

  } catch (error) {
      console.error('Error:', error);
  }
};
```

```
// Fetch and handle JSON response
const fetchJSONDataFromServer = async () => {
  try {
    const response = await fetch(`http://localhost:4000/opensky-local?fileId=data${counterId}`, {
      method: 'GET',
    });

    if (!response.ok) {
      throw new Error(`Request failed with status ${response.status}`);
    }

    const jsonData = await response.json();

    // Type assertion to treat jsonData as JsonApiResponse
    const response: JsonApiResponse = {
      data: jsonData,
    };

    // Now we can work with response as a JsonApiResponse
    setFlightData(response);
  } catch (error) {
    console.error('Error:', error);
  }
};
```