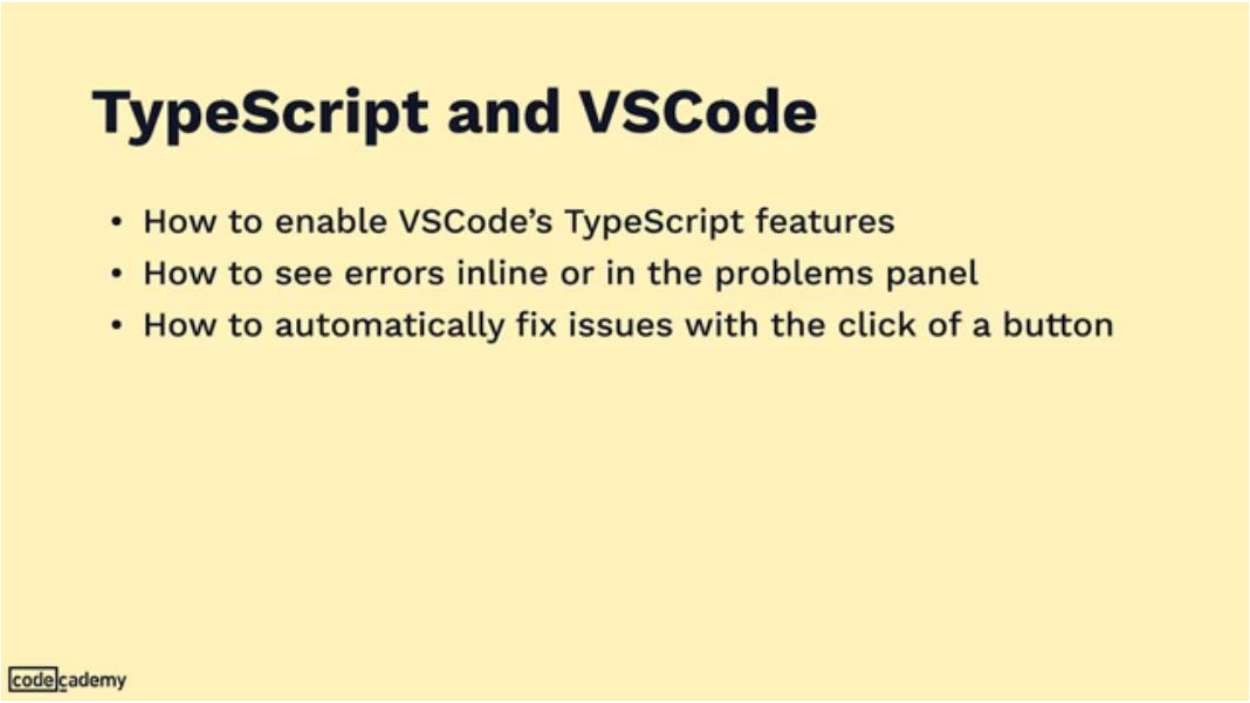
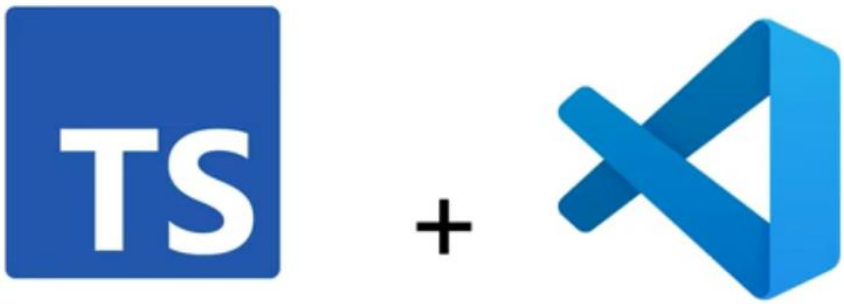
TypeScript Development in VSCode

VSCode has powerful tools built-in that can identify TypeScript type errors so that we can develop faster and with more confidence.

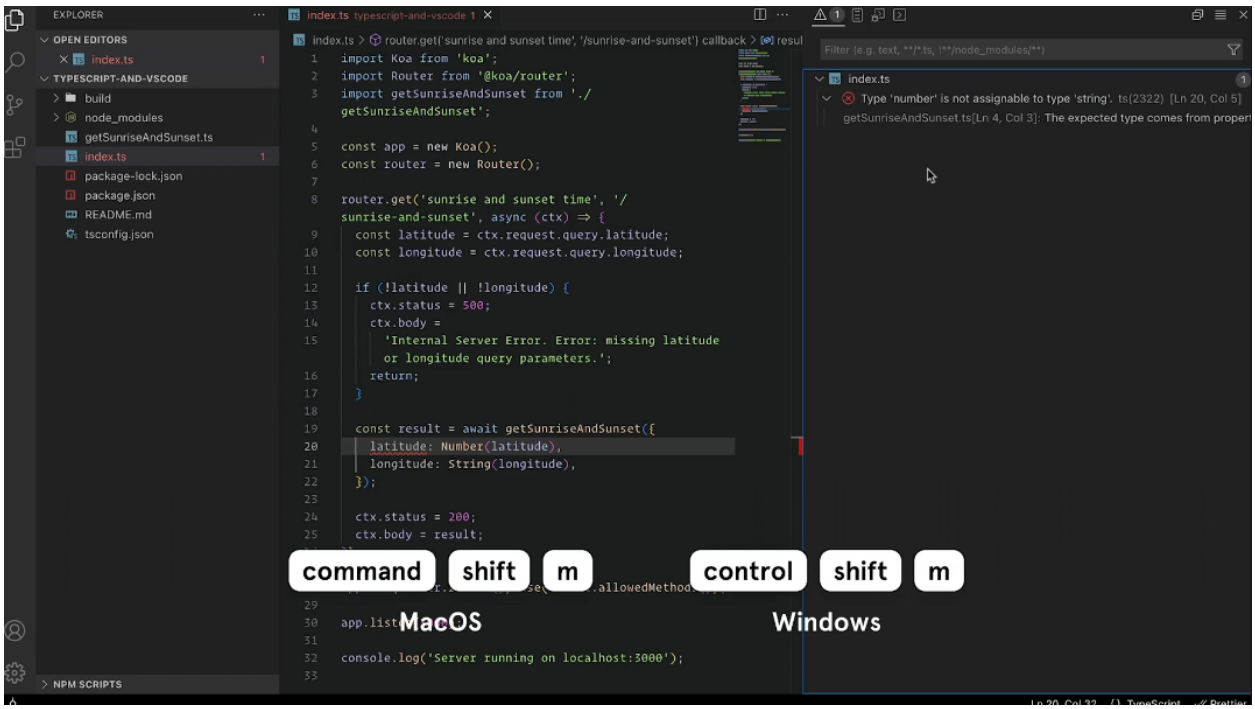
Introduction

TypeScript ensures our code is type checked. VSCode has powerful tools built-in that work really well with TypeScript. VSCode's tools help us auto-complete code, automatically fix problems, and expose type errors as we code nearly instantaneously. Let's get TypeScript and VSCode set up together and test out some of these features — you can follow along with the accompanying video or the provided text!



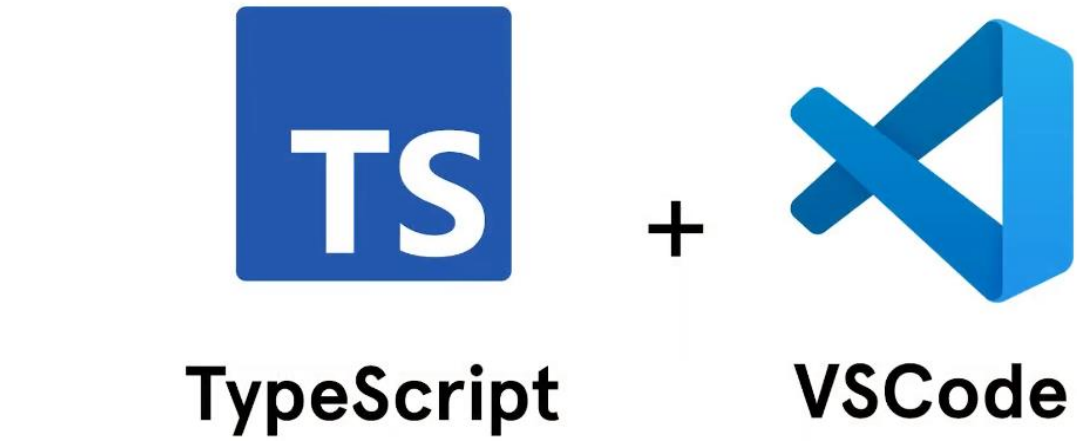


TypeScript + VSCode

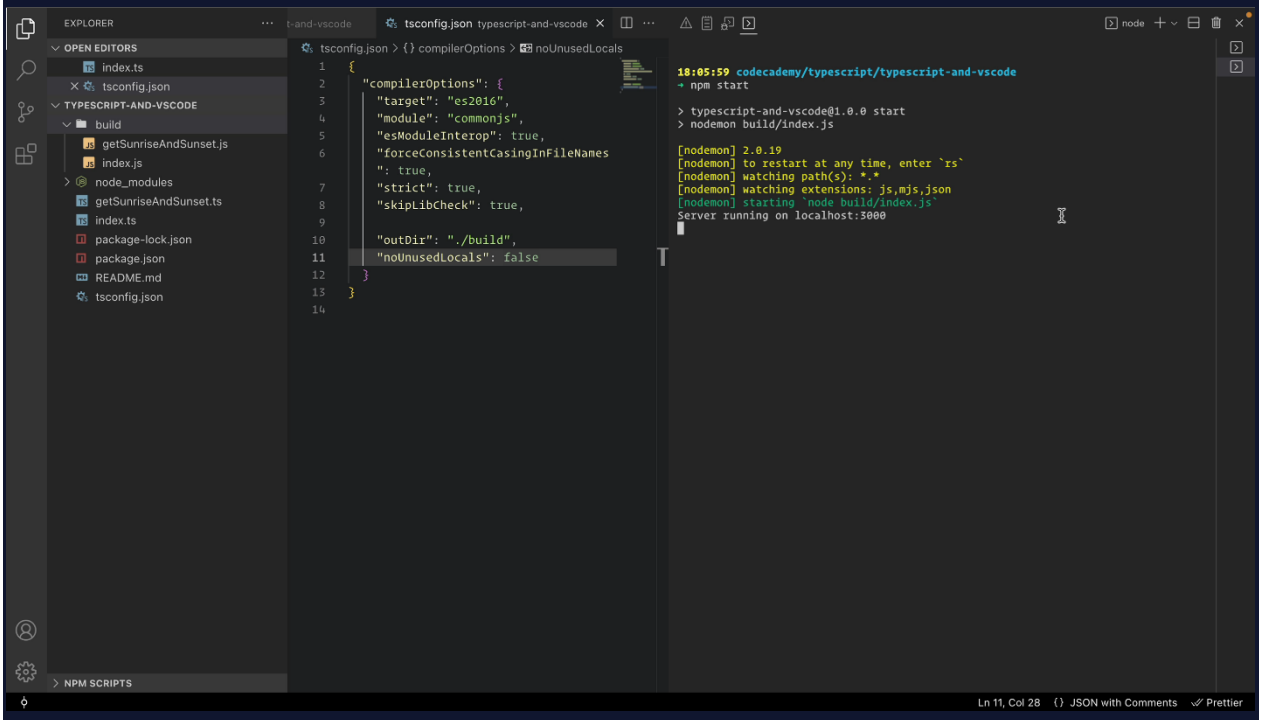


Summary

- We need to make sure that TypeScript Validation is enabled in Settings.
- We can hover errors or we can see them in the Problems Panel.
- As we make changes to our `tsconfig.json`, we can view all parts of our code that need attention.



Running the project



First, download our starting code and follow the instructions in the `README.md` file.

We're going to start out with this API written in TypeScript. This program takes a GET request and returns the sunrise and sunset time based on a provided longitude and latitude. Let's get the server started so that we can test it.

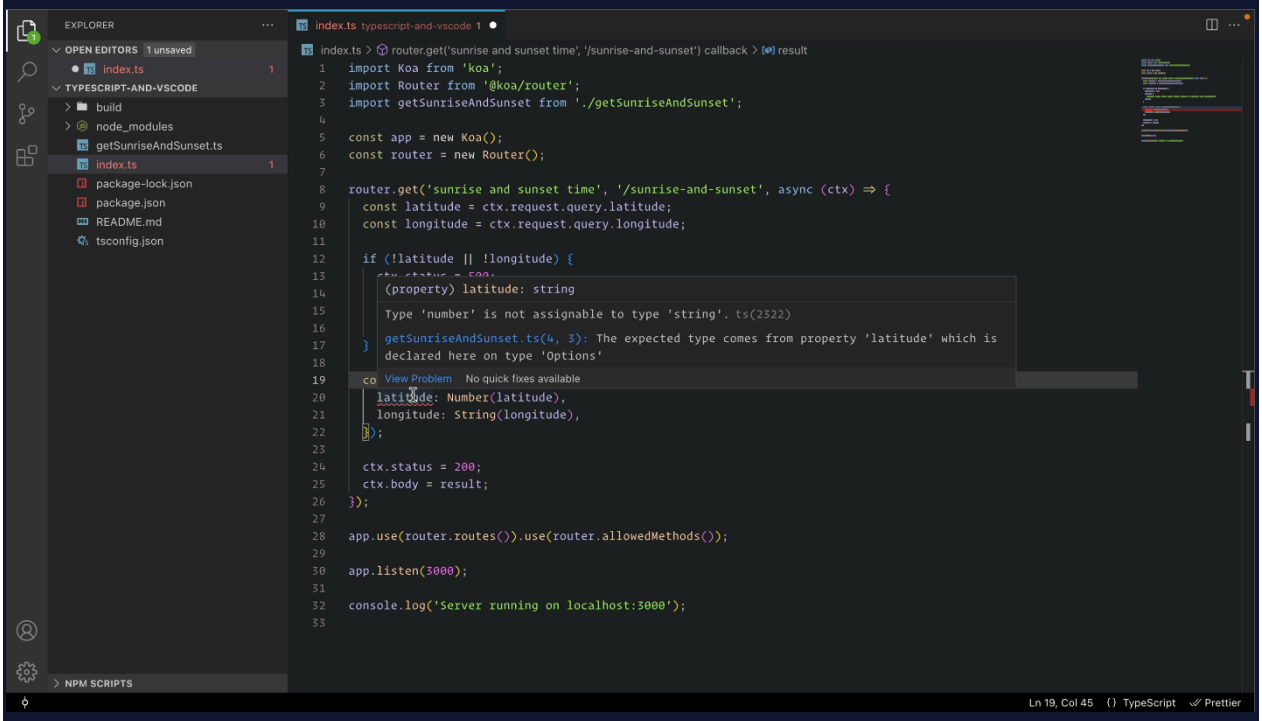
Open up an integrated terminal with View > Terminal. You can also do this with a keyboard shortcut: control and ` (backtick) on macOS or control and ` (backtick) on Windows.

In one terminal window, let's start tsc, which is TypeScript's CLI, in watch mode using the watch flag. We can do that by running `npm run tsc -- --watch`. The TypeScript configuration we're using is in the `tsconfig.json` file. At the top are all the default options provided by default. We added in the bottom two options: `outDir` and `noUnusedLocals`. The first is where tsc will output the built files — a directory named `./build`. The second one is an option that we'll talk about in a moment.

In another terminal, run `npm start -- --watch`, which will reload the server every time there's a change.

Let's give it a test. In a browser, we can navigate to http://localhost:3000 and hit the sunrise and sunset endpoint that also provides a latitude value and a longitude value. Now we'll send the request and we'll get back JSON for this location's sunrise and sunset times.

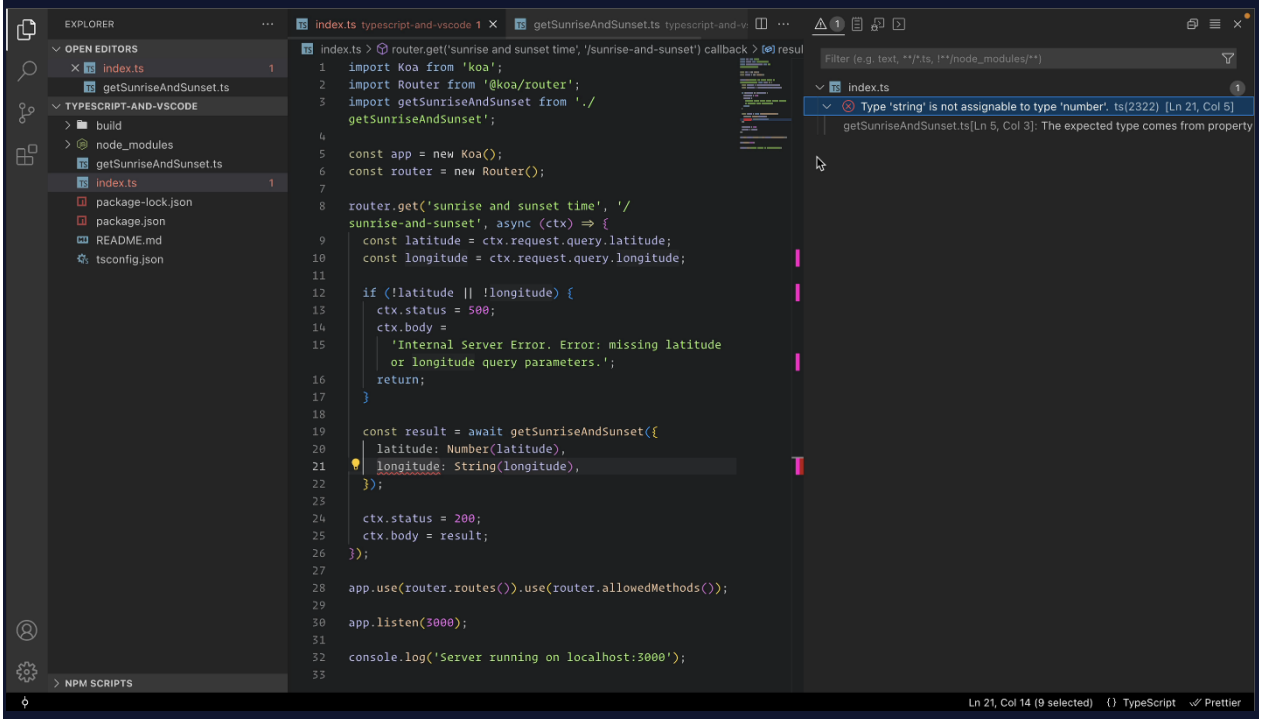
Viewing TypeScript errors



Now that we know it works, let's make some changes to the project. Here is the project opened in VSCode. We highly recommend VSCode as a code editor. VSCode has one setting we need to make sure is toggled on: TypeScript Validation. To see that setting, go to Code > Preferences > Settings, then type in "TypeScript validation". It should be enabled by default.

In the project, let's make a change to the latitude we are passing to the `getSunriseAndSunset()` function. Instead of casting the latitude as a `String`, cast it as a `Number`. Immediately, we'll see it highlighted in red. We can hover over the error and when clicking on the blue name, there is a link to the type definition that the argument is erroring against.

Problems panel

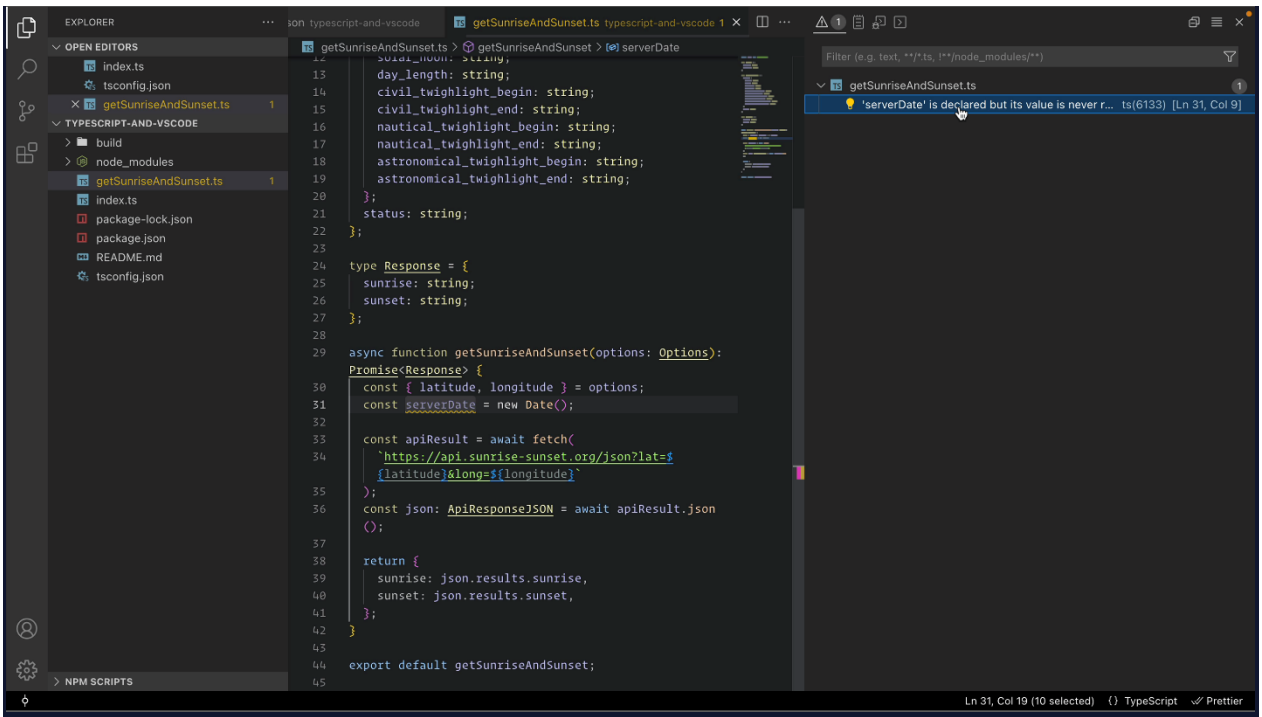


While hovering to discover issues works, when there are many issues, it's easier to view a complete list of issues all at once. To see a list, go to View > Problems. You can also use command + shift + m on macOS or control + shift + mon Windows to open the problems panel.

Inside the problems panel, we can click on the error, which will take us to the type definition.

Since we made that change, let's now make the longitude a number type. We immediately see a new issue appear in the problems panel. Clicking on it takes us to the call site and we can update the type cast there to match the type we just changed.

tsconfig.json changes



Now that we've seen some errors with code changes, let's make a change to our TypeScript configuration instead. In `tsconfig.json`, set `noUnusedLocals` to `true`, which means that if there are any defined variables that are unused, TypeScript will show a warning.

After a few seconds, we see that there is a `serverDate` variable that is declared but never used. We could click on the warning and see the line directly. Alternatively, if we hover over the warning, we'll see a light bulb appear, which can fix problems like these for us.

When making changes to a codebase's `tsconfig.json` or applying refactors that impact multiple files, the problem panel can be really helpful for exposing all the issues present in the codebase.

We encourage you to experiment with this API server and with the `tsconfig.json` file. The code for this project is linked in the video's description, and it has instructions to get it running in its readme. Also, you'll find the full `tsconfig.json` reference and all of its options at https://www.typescriptlang.org/tsconfig.

Wrap up

In this article, we covered how to use VSCode's TypeScript tools. To recap:

- We need to make sure that TypeScript Validation is enabled in Settings.
- We can hover over errors or we can see them in the Problems Panel.
- As we make changes to our `tsconfig.json`, we can view all parts of our code that need attention.

As you code, we hope these features will improve your experience developing TypeScript programs. Intelligent suggestions based on types can make us feel more confident about our code, and hopefully, make it easier to identify problem spots.