guide # tooling

VS Code / WebStorm

TCSS 460 — Client/Server Programming

You have spent the last two years writing Java in IntelliJ IDEA (or perhaps Eclipse). Now you are switching to TypeScript and Node.js, and you need an editor that understands this new world. This guide walks you through setting up either **Visual Studio Code** or **WebStorm** for TypeScript development. Both are excellent choices — pick the one that feels right and get it configured before your first check-off.

---

1 Choosing an Editor

There is no single correct editor for this course. You need one that provides TypeScript type checking, integrated terminal access, and a way to test HTTP requests. Two editors stand out for TypeScript/Node.js work: VS Code and WebStorm.

1.1 Visual Studio Code

VS Code is a free, open-source editor from Microsoft. It is the most popular editor for TypeScript and JavaScript development by a wide margin. VS Code is lightweight out of the box — you install extensions to add the features you need. This means you control exactly what your editor does, but it also means you need to spend some time configuring it.

Attribute	Details
Cost	Free
TypeScript support	Built-in IntelliSense and type checking via the TypeScript language service
Extensions	Thousands available — you pick what you need
Terminal	Integrated terminal built in

Attribute	Details
Download	code.visualstudio.com

1.2 WebStorm

WebStorm is a full IDE from JetBrains — the same company that makes IntelliJ IDEA. If you used IntelliJ for TCSS 305, WebStorm will feel immediately familiar. The keyboard shortcuts, project navigation, refactoring tools, and debugger all work the same way. WebStorm includes TypeScript support, a built-in HTTP client, and test runners out of the box — no extensions required.

Attribute	Details
Cost	Free for non-commercial use (including coursework); also free with a JetBrains student license
TypeScript support	Built-in — uses the official TypeScript Language Service directly
Extensions	Fewer needed — most features are built in
Terminal	Integrated terminal built in
Download	jetbrains.com/webstorm

JetBrains Student License

As a UW student, you can get a free JetBrains Educational License that includes all JetBrains products (WebStorm, IntelliJ, DataGrip, and more). Apply at jetbrains.com/student using your `@uw.edu` email. The license renews annually while you are enrolled. Even without the student license, WebStorm is free for non-commercial use — coursework qualifies.

1.3 Which Should You Pick?

Here is honest guidance:

- **Pick WebStorm** if you liked IntelliJ in TCSS 305 and want everything to "just work" without configuring extensions. The transition from IntelliJ to WebStorm is minimal —

same keybindings, same project structure, same debugger feel.

- **Pick VS Code** if you want the editor that most professional TypeScript developers use, or if you prefer a lighter-weight tool that you customize yourself. The extension ecosystem is massive, and most online tutorials assume VS Code.
- **Either works for this course.** The course starter projects, check-offs, and group project do not depend on a specific editor.

⚠ Mixed Editors in Group Projects

When group members use different editors, a few friction points come up. Be aware of these before your group project starts:

- **Line endings.** VS Code defaults to LF (Unix-style) on macOS/Linux and CRLF (Windows-style) on Windows. WebStorm follows the OS default as well, but the two editors may auto-convert differently. If one teammate commits CRLF files and another commits LF, Git diffs become unreadable — every line shows as changed. **Fix:** Add a `.editorconfig` file to your project root (the course starter repos include one) and configure Git to normalize line endings: `git config --global core.autocrlf input` (macOS/Linux) or `git config --global core.autocrlf true` (Windows).
- **Formatting differences.** Prettier handles code formatting, and it produces identical output regardless of editor — but only if everyone has format-on-save enabled and is using the same Prettier config. If one teammate formats manually or has a different Prettier version, you will get noisy diffs full of whitespace changes. **Fix:** Run `npm install` so everyone uses the same Prettier version from `node_modules`, and make sure format-on-save is configured (Section 2.3 for VS Code, Section 3.3 for WebStorm).
- **Editor config files in Git.** VS Code stores workspace settings in `.vscode/`, WebStorm stores them in `.idea/`. These directories contain personal preferences (font size, color theme) that should not be shared. **Fix:** Both `.vscode/` and `.idea/` should be in `.gitignore`. The course starter repos already handle this, but if you create a project from scratch, do not forget to add them.
- **Path separators.** This is rare but can bite Windows users. If a teammate on Windows hardcodes a backslash path in a config file (e.g., `src\routes\index.ts`), it will break on macOS/Linux. Both editors handle this correctly in normal usage, but watch for it in manually written scripts or configuration.

The bottom line: mixed editors work fine as long as the team agrees on Prettier, `.editorconfig`, and `.gitignore`. These are project-level settings, not editor-level — they travel with the repo and keep everyone consistent regardless of which editor they use.

Important

Whichever editor you choose, make sure you complete the setup in this guide before the first check-off assignment. You need a working TypeScript environment with linting, formatting, and a way to send HTTP requests.

---

2 VS Code Setup for TypeScript

If you chose VS Code, follow this section. If you chose WebStorm, skip to Section 3.

2.1 Installing VS Code

Download VS Code from code.visualstudio.com and run the installer for your operating system. On macOS, drag the app to your Applications folder. On Windows, the installer handles everything.

After installation, open VS Code and verify it launches. You should see the Welcome tab.

Command Line Launcher (macOS)

On macOS, open the Command Palette (`Cmd + Shift + P`), type "Shell Command: Install 'code' command in PATH", and press Enter. This lets you open projects from the terminal with `code .` — you will use this constantly.

2.2 Essential Extensions

VS Code's power comes from extensions. Open the Extensions panel (`Ctrl + Shift + X` on Windows/Linux, `Cmd + Shift + X` on macOS) and install the following:

ESLint — Linting for TypeScript and JavaScript

ESLint analyzes your code for potential errors and style violations as you type. Red and yellow squiggles appear under problems in real time. In this course, every starter project ships with an ESLint configuration, so you just need the extension installed.

- Search: `ESLint` by Microsoft
- Extension ID: `dbaeumer.vscode-eslint`

Prettier — Code Formatter — Automatic code formatting

Prettier reformats your code to a consistent style every time you save. No more arguing about indentation, semicolons, or line length — Prettier decides for you. This is the same tool used by most professional TypeScript projects.

- Search: `Prettier - Code formatter` by Prettier
- Extension ID: `esbenp.prettier-vscode`

Thunder Client — API testing inside VS Code

Thunder Client is a lightweight REST client built into VS Code. It replaces the need for a separate tool like Postman for testing your API endpoints. You can send GET, POST, PUT, and DELETE requests, inspect response bodies and headers, and organize requests into collections — all without leaving your editor.

- Search: `Thunder Client` by Ranga Vadhineni
- Extension ID: `rangav.vscode-thunder-client`

Error Lens — Inline error display

Error Lens displays error and warning messages directly on the line where they occur, instead of requiring you to hover over a squiggly underline. This small change dramatically speeds up your feedback loop when writing TypeScript.

- Search: `Error Lens` by Alexander
- Extension ID: `usernamehw.errorlens`

Pretty TypeScript Errors — Readable error messages

TypeScript error messages can be dense and hard to parse, especially when they involve complex generic types. This extension reformats those errors into a more readable layout. Paired with Error Lens, it makes TypeScript errors far less intimidating.

- Search: `Pretty TypeScript Errors` by yoavbls
- Extension ID: `yoavbls.pretty-ts-errors`

Install from the Command Line

You can install all five extensions at once from your terminal:

```
code --install-extension dbaeumer.vscode-eslint
code --install-extension esbenp.prettier-vscode
code --install-extension rangav.vscode-thunder-client
code --install-extension usernamehw.errorlens
code --install-extension yoavbls.pretty-ts-errors
```

2.3 Configuring Format on Save

The single most important setting in VS Code for this course is **format on save**. When enabled, Prettier automatically reformats your file every time you press `Ctrl + S` (or `Cmd + S` on macOS). This keeps your code consistent without any extra effort.

Open VS Code settings (`Ctrl + ,` or `Cmd + ,`) and search for the following settings, or edit your `settings.json` directly (`Ctrl + Shift + P` then "Preferences: Open User Settings (JSON)"):

```
{
  "editor.defaultFormatter": "esbenp.prettier-vscode",
  "editor.formatOnSave": true,
  "editor.codeActionsOnSave": {
    "source.fixAll.eslint": "explicit"
  }
}
```

This configuration does three things:

1. `editor.defaultFormatter` — Tells VS Code to use Prettier (not the built-in formatter) for all files.
2. `editor.formatOnSave` — Runs Prettier every time you save.
3. `editor.codeActionsOnSave` — Runs ESLint's auto-fix on save, which corrects linting issues that have automatic fixes (like removing unused imports).

Formatting Not Working on Save

If saving does not reformat your code, check these common issues:

- **Prettier is not the default formatter.** Right-click in any file, select "Format Document With...", and choose Prettier. VS Code may ask you to set a default — select Prettier.
- **The project has no Prettier config.** Prettier looks for a `.prettierrc` or `prettier` key in `package.json`. Our starter projects include this, but if you created a project from scratch, add a `.prettierrc` file:

```
{
  "semi": true,
  "singleQuote": true,
  "trailingComma": "all",
  "printWidth": 100
}
```

- **Extension conflict.** If you have another formatter extension installed (like Beautify), it may conflict. Disable other formatters and keep only Prettier.

2.4 IntelliSense and Type Checking

VS Code includes built-in TypeScript support through the TypeScript language service. This gives you:

- **Auto-completion** — As you type, VS Code suggests properties, methods, and types based on your code's type information. Press `Tab` or `Enter` to accept a suggestion.
- **Hover information** — Hover over any variable, function, or type to see its inferred type and documentation.
- **Go to Definition** — `Ctrl + Click` (or `Cmd + Click` on macOS) on a function or variable name to jump to where it is defined.
- **Error highlighting** — Red squiggles appear for type errors before you even run your code.

This all works automatically when you open a TypeScript project with a `tsconfig.json` file. Every starter project in this course includes one.

Quick Fix

When you see a red or yellow squiggle, place your cursor on it and press `Ctrl + .` (or `Cmd + .` on macOS). VS Code will suggest automatic fixes — adding missing imports, correcting type annotations, or wrapping code in try/catch blocks.

2.5 Using Thunder Client

You will test your API endpoints frequently in this course. Thunder Client lets you do this without leaving VS Code.

To send a request:

1. Click the Thunder Client icon in the Activity Bar (left sidebar — it looks like a lightning bolt).
2. Click **New Request**.
3. Choose the HTTP method (GET, POST, PUT, DELETE) from the dropdown.
4. Enter the URL, for example: `http://localhost:3000/hello`.
5. For POST/PUT requests, click the **Body** tab, select **JSON**, and enter your request body.
6. Click **Send**.

The response appears in the right panel with the status code, headers, and body. You can save requests into collections for reuse.

> **Collections for Check-Offs**
>
> Create a Thunder Client collection for each check-off assignment. Save all the requests you need to test. This saves you from retyping URLs every time you restart your server, and you can share collections with your group members by exporting them.

---

3 WebStorm Setup for TypeScript

If you chose WebStorm, follow this section. If you already set up VS Code in Section 2, skip to Section 4.

3.1 Installing WebStorm

Download WebStorm from jetbrains.com/webstorm and run the installer. On first launch, WebStorm will ask you to activate a license. Choose one of these options:

- **Non-commercial use** — Free. Select "Non-commercial use" during activation. Coursework qualifies.

- **Student license** — Free. If you already have a JetBrains Educational License from IntelliJ, log in with the same JetBrains account. Otherwise, apply at jetbrains.com/student.

After activation, WebStorm is ready to go. Open your project folder with **File > Open** and select the project's root directory (the folder that contains `package.json`).

3.2 Built-in TypeScript Support

Unlike VS Code, WebStorm requires no extensions for TypeScript. Everything is built in:

- **Type checking and error highlighting** — WebStorm uses the official TypeScript Language Service directly. As of WebStorm 2026.1, this is the service-powered TypeScript engine, which offloads analysis to the TypeScript compiler for faster navigation, refactoring, and error detection in large projects.
- **Auto-completion** — Type-aware suggestions as you type, including auto-imports.
- **Refactoring** — Rename, extract method, extract variable, change signature — the same refactoring tools you used in IntelliJ for Java. Press `Shift + F6` to rename, `Ctrl + Alt + M` (or `Cmd + Option + M` on macOS) to extract a method.
- **Navigation** — `Ctrl + Click` to go to definition. `Ctrl + Shift + N` to find any file by name. `Ctrl + N` to find any symbol.

If you are coming from IntelliJ, notice that these are the exact same shortcuts. This is intentional — all JetBrains IDEs share the same keymap.

> **TypeScript Version**
>
> WebStorm uses the TypeScript compiler bundled with your project (from `node_modules/typescript`). This means it always matches the version your project uses — no version mismatch between your editor and your build. You can verify this in **Settings > Languages & Frameworks > TypeScript**.

3.3 Configuring ESLint and Prettier in WebStorm

WebStorm detects ESLint and Prettier configurations automatically from your project's `package.json` and config files. The course starter projects include both, so setup is minimal.

ESLint: Go to **Settings > Languages & Frameworks > JavaScript > Code Quality Tools > ESLint** and select **Automatic ESLint configuration**. WebStorm will find the ESLint config in your project and highlight violations in the editor.

Prettier: Go to **Settings > Languages & Frameworks > JavaScript > Prettier** and check **On save**. Select the Prettier package from your project's `node_modules`. WebStorm will format your code with Prettier every time you save — the same behavior as VS Code with the Prettier extension.

> **ESLint Not Detecting Issues**
>
> If ESLint is not highlighting problems in WebStorm:
>
> - Make sure you have run `npm install` in the project directory so `node_modules` contains the ESLint package.
> - Check that the ESLint setting is set to "Automatic" (not "Manual" or "Disabled").
> - Restart WebStorm after changing ESLint settings — sometimes the language service needs a fresh start.

3.4 Run Configurations

In IntelliJ, you ran Java programs with a Run Configuration (the green play button). WebStorm works the same way for Node.js.

Running an npm script:

1. Open `package.json` in the editor.
2. You will see green play icons next to each script in the `"scripts"` section.
3. Click the play icon next to `"dev"` (or whatever script starts your server).
4. WebStorm creates a Run Configuration automatically and runs it in the built-in terminal.

You can also create run configurations manually via **Run > Edit Configurations**. For this course, the npm script approach is usually sufficient.

> **Run/Debug from the Gutter**
>
> WebStorm shows small green arrows (gutter icons) next to `app.listen()` calls and test functions. Click one to run or debug that specific entry point. This is the same behavior as IntelliJ's gutter icons for Java `main` methods.

3.5 The Built-in HTTP Client

WebStorm includes a built-in HTTP client that serves the same purpose as Thunder Client in VS Code or standalone tools like Postman. Instead of a GUI, it uses `.http` files — plain text

files where you write your requests.

Create a new file called `requests.http` in your project and add:

```
### Get hello endpoint
GET http://localhost:3000/hello

### Create a new item
POST http://localhost:3000/items
Content-Type: application/json

{
  "name": "Test Item",
  "description": "A test item"
}
```

Click the green play icon next to any request to send it. The response appears in a panel below with the status code, headers, and body.

Advantages of `.http` files over GUI-based tools:

Feature	Benefit
Plain text	You can commit them to Git alongside your code
Shareable	Team members can use the same request files
Environment variables	Use `{{host}}` variables for different environments (local, staging, production)
Response scripting	Write assertions in JavaScript to validate responses automatically

Share Request Files with Your Group

Commit your `.http` files to your group project repository. When a teammate pulls the latest code, they immediately have all the API test requests ready to run — no need to recreate them manually in Postman.

---

4 Common Editor Tasks

Regardless of which editor you chose, you will do these tasks constantly throughout the course. This section covers the workflow for both editors.

4.1 Running the Dev Server from the Terminal

Both VS Code and WebStorm include an integrated terminal. You will use it to start your development server, run scripts, and install packages.

Opening the terminal:

Editor	Shortcut
VS Code	`Ctrl + ` ` (backtick) or **View > Terminal**
WebStorm	`Alt + F12` or **View > Tool Windows > Terminal**

Once the terminal is open, the commands are identical in both editors:

```
# Install project dependencies
npm install

# Start the development server (with auto-reload)
npm run dev

# Compile TypeScript (check for errors without running)
npx tsc --noEmit
```

The dev server runs in the terminal and watches for file changes. When you save a file, the server automatically restarts. You will see output like:

```
Server running on http://localhost:3000
```

Leave this terminal running while you develop. Open a second terminal tab (`Ctrl + Shift + ``` in VS Code, or click the `+` icon in WebStorm's terminal) if you need to run other commands.

Port Already in Use

If you see `Error: listen EADDRINUSE :::3000`, another process is already using port 3000. This usually means you have a previous server instance still running. Find and stop it:

```
# macOS / Linux
lsof -ti:3000 | xargs kill

# Windows (PowerShell)
Get-Process -Id (Get-NetTCPConnection -LocalPort
3000).OwningProcess | Stop-Process
```

Alternatively, change the port in your project configuration (usually in `.env` or directly in the `app.listen()` call).

4.2 Debugging with Breakpoints

Debugging with breakpoints is more effective than scattering `console.log()` statements through your code. Both editors support setting breakpoints and stepping through TypeScript code line by line.

Setting a breakpoint:

- **VS Code:** Click in the gutter (the narrow area to the left of line numbers) to place a red dot on any line.
- **WebStorm:** Click in the gutter to the left of line numbers — same as IntelliJ.

Starting a debug session:

In VS Code:

1. Open the Run and Debug panel (`Ctrl + Shift + D` or `Cmd + Shift + D`).
2. If you do not have a launch configuration yet, click "create a launch.json file" and select **Node.js**.
3. Add this configuration to `.vscode/launch.json` for debugging an Express app:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "launch",
      "name": "Debug Server",
      "runtimeExecutable": "npx",
      "runtimeArgs": ["ts-node", "src/index.ts"],
```

```
      "restart": true,
      "console": "integratedTerminal"
    }
  ]
}
```

1. Press `F5` to start debugging.

In WebStorm:

1. Create a Node.js Run/Debug Configuration (or use the npm script approach from Section 3.4).
2. Click the **Debug** button (the bug icon) instead of the Run button.
3. WebStorm launches the server with the debugger attached.

Once the debugger is attached and your server hits a breakpoint, both editors show:

- **Variables panel** — Current values of all variables in scope.
- **Call stack** — The chain of function calls that led to this line.
- **Step controls** — Step Over (next line), Step Into (enter function), Step Out (finish current function), Continue (resume execution).

> **Debugging Express Route Handlers**
>
> Place a breakpoint on the first line inside a route handler function. Then send a request to that route (using Thunder Client, the WebStorm HTTP client, or your browser). The debugger will pause execution at your breakpoint, letting you inspect `req.params`, `req.body`, `req.query`, and any other data.

4.3 Git Integration

Both editors include built-in Git support. You do not need a separate Git GUI.

VS Code Git features:

- The Source Control panel (`Ctrl + Shift + G`) shows changed files, lets you stage/unstage changes, write commit messages, and push/pull.
- Inline diff view shows what changed in each file.
- The built-in terminal works for any Git operation the GUI does not cover.

WebStorm Git features:

- The Git tool window (`Alt + 9`) shows the commit log, branches, and diffs — identical to IntelliJ's Git integration.

- The Commit dialog (`Ctrl + K` or `Cmd + K`) lets you select files, write a message, and commit.
- Interactive rebase, cherry-pick, and stash are available from the Git menu.

For this course, you will primarily use these Git operations:

Operation	Command	When
`git clone`	Clone the starter repo	Start of each assignment
`git add` + `git commit`	Save your work	After meaningful progress
`git push`	Push to GitHub	Before each deadline, or when collaborating
`git pull`	Pull teammates' changes	Group project sprints
`git branch` + `git checkout`	Create/switch branches	Group project feature work

Commit Early, Commit Often

Do not wait until your code is "done" to commit. Commit after each small, working change. A commit message like "add GET route for /users" is far more useful than "finished check-off" with 500 lines of changes. Both editors make committing fast — use the keyboard shortcut and make it a habit.

---

5 Summary

Topic	Key Point
VS Code	Free, lightweight, extension-driven. Install ESLint, Prettier, Thunder Client, Error Lens, and Pretty TypeScript Errors.
WebStorm	JetBrains IDE, familiar from IntelliJ. TypeScript, ESLint, Prettier, HTTP client, and debugger are all built in.

Topic	Key Point
Free access	VS Code is free. WebStorm is free for non-commercial use and free with a JetBrains student license.
Format on save	Configure Prettier to format on save in both editors. This is non-negotiable for consistent code.
API testing	Use Thunder Client (VS Code) or `.http` files (WebStorm) to test your API endpoints without leaving your editor.
Debugging	Use breakpoints, not `console.log()`. Both editors support stepping through TypeScript code line by line.
Git	Both editors have built-in Git support. Commit early and often.
Bottom line	Either editor works. Pick one, set it up completely, and start building.

---

6 References

Official Documentation:

- Visual Studio Code — TypeScript — VS Code's official TypeScript documentation
- WebStorm — TypeScript Support — JetBrains' TypeScript documentation for WebStorm
- WebStorm — HTTP Client — Built-in HTTP client documentation
- ESLint — Getting Started — ESLint official setup guide
- Prettier — Install — Prettier official installation guide
- Thunder Client — Thunder Client official site and documentation

Licensing and Access:

- JetBrains Student License — Free access to all JetBrains IDEs for students
- WebStorm Non-Commercial Use — JetBrains announcement on free non-commercial licensing

---

7 Further Reading

External Resources

- WebStorm 2026.1 Release Notes — Latest WebStorm features including service-powered TypeScript engine
- VS Code Tips and Tricks — Productivity shortcuts and hidden features
- WebStorm vs VS Code 2026 — Detailed comparison of both editors for JavaScript/TypeScript development
- Thunder Client Documentation — Collections, environments, and scripting in Thunder Client

---

This guide is part of TCSS 460 — Client/Server Programming, School of Engineering and Technology, University of Washington Tacoma.