# guide # tooling

# API Testing Tools

**TCSS 460 — Client/Server Programming**

Your Express API exposes routes for GET, POST, PUT, and DELETE — but a web browser can only send GET requests. To test anything beyond loading a page, you need a dedicated API testing tool. This guide covers three options you will use throughout the quarter: **Postman**, **Thunder Client** (VS Code extension), and **WebStorm's built-in HTTP Client**.

This guide is about **manual testing** — sending requests by hand to explore your API and verify that individual endpoints behave correctly during development. This is different from the automated test suite built into your project (Jest and Supertest), which runs programmatically, is checked into your repository, and is designed to catch regressions. Manual testing with these tools is informal and fast; automated tests are repeatable and verifiable by your whole team.

---

## 1 Why You Need an API Testing Tool

When you type a URL into your browser's address bar and hit Enter, the browser sends a **GET** request. That is the only HTTP method a browser address bar supports. There is no way to:

- Send a **POST** request with a JSON body (to create a resource)
- Send a **PUT** or **PATCH** request (to update a resource)
- Send a **DELETE** request (to remove a resource)
- Set custom **headers** like `Authorization: Bearer <token>`
- Inspect the **response headers** and **status code** in detail

For a course where you build and test a full REST API, you need a tool that can construct any HTTP request, send it, and show you exactly what came back.

### 1.1 Three Options

This course supports three tools. Pick the one that fits your editor and workflow — or use more than one:

| Tool | Where It Runs | Best For |
|---|---|---|
| **Postman** | Standalone desktop app | Full-featured GUI, team collaboration, API documentation |
| **Thunder Client** | VS Code extension | Lightweight testing without leaving your editor |
| **WebStorm HTTP Client** | Built into WebStorm/IntelliJ | Plain-text `.http` files that live in your Git repo |

All three do the same fundamental job — construct an HTTP request, send it to your server, and display the response. The differences are in workflow, collaboration features, and where your saved requests live.

**These Tools Are for Manual Testing**

All three tools include their own built-in test scripting frameworks — Postman's is the most full-featured, with a JavaScript-based runner and the `pm` assertion API; Thunder Client has a lightweight assertions tab; WebStorm's HTTP Client supports `client.test(...)` response handler scripts. In this course, we are not using those frameworks. We use these tools for **informal, manual testing**: construct a request, send it, read the response, confirm it looks right. The automated testing framework for your project (Jest and Supertest) is a separate concern covered elsewhere.

**Gen AI & Learning: Context Engineering with Saved Requests**

A well-organized collection of API test requests is more than a convenience — it is a form of **context engineering**. When you ask an AI coding agent to help debug an endpoint or add a new feature, it can read your saved requests to understand your API's structure, expected inputs, and response formats. The more organized your test requests are, the better context the agent has to work with. Think of your saved collections as documentation that both humans and AI agents can consume.

---

## 2 Installation and Setup

## 2.1 Postman

Postman is a standalone desktop application for API development and testing.

1. Go to https://www.postman.com/downloads/
2. Download the installer for your operating system (macOS, Windows, or Linux)
3. Run the installer
4. Create a free account when prompted (required for saving collections)

> **Postman Free Tier (2026)**
>
> Postman's free plan is limited to **one user** and is designed for individual use. For this course, the free tier is sufficient — you get unlimited collections, environments, collection runner executions, and mock servers. The free tier includes 50 AI credits per month and native Git integration. You do not need a paid plan for anything in TCSS 460.

## 2.2 Thunder Client (VS Code)

Thunder Client is a lightweight REST API client that runs as a VS Code extension.

1. Open VS Code
2. Click the Extensions icon in the left sidebar (or press `^ Ctrl` + `⇧ Shift` + `X`)
3. Search for **Thunder Client**
4. Click **Install**
5. A lightning bolt icon appears in the left sidebar — click it to open Thunder Client

> **Thunder Client Free Tier Limitations**
>
> As of 2025, Thunder Client's free version is restricted to **personal, non-commercial use only**. However, schools, colleges, and universities are explicitly permitted to use the free version. The free tier limits you to **3 collections** with **15 requests per collection**. For this course, that is enough for basic testing, but if you find yourself hitting limits, consider Postman or WebStorm's HTTP Client instead.

## 2.3 WebStorm HTTP Client

If you use WebStorm (or any JetBrains IDE), the HTTP Client is **built in** — there is nothing to install.

1. In WebStorm, create a new file with the `.http` or `.rest` extension
2. WebStorm automatically recognizes it as an HTTP request file
3. Green "play" buttons appear next to each request — click to execute

> **Free for Students**
>
> WebStorm is free for students through the JetBrains Educational Pack. If you prefer an IDE over VS Code, this is an excellent option — and the built-in HTTP Client means one fewer tool to install.

---

## 3 Making Your First Request

Let us send a simple GET request to your running Express API. Make sure your development server is running first:

```
npm run dev
```

You should see output indicating the server is listening on port 3000.

### 3.1 Sending a GET Request

**In Postman:**

1. Click the **+** button to create a new request tab
2. Make sure the method dropdown says **GET**
3. Enter the URL: `http://localhost:3000/v1/hello`
4. Click **Send**

**In Thunder Client:**

1. Click the lightning bolt icon in the VS Code sidebar
2. Click **New Request**
3. Make sure the method is **GET**
4. Enter: `http://localhost:3000/v1/hello`
5. Click **Send**

**In WebStorm HTTP Client:**

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

```
### Hello endpoint
GET http://localhost:3000/v1/hello
```

Click the green play button next to the request.

## 3.2 Reading the Response

Regardless of which tool you use, the response display shows the same key information:

| Part | What It Tells You | Example |
|---|---|---|
| **Status code** | Whether the request succeeded | `200 OK`, `404 Not Found`, `500 Internal Server Error` |
| **Response headers** | Metadata about the response | `Content-Type: application/json`, `X-Powered-By: Express` |
| **Response body** | The actual data returned | `{"message": "Hello, World!"}` |
| **Response time** | How long the server took to respond | `12 ms` |

**Status Codes Are Your First Debugging Signal**

Before reading the response body, always check the status code. A `200` means success. A `4xx` means the client (you) sent something wrong — check your URL, headers, or body. A `5xx` means the server crashed — check your Express console for the error stack trace.

**Try It Yourself**

1. Start your development server with `npm run dev`
2. Open your preferred API testing tool
3. Send a GET request to `http://localhost:3000/v1/hello`
4. Verify you see a `200 OK` status code and a JSON response body
5. Now try a URL that does not exist, like `http://localhost:3000/v1/doesnotexist` — observe the `404` status code

# 4 Request Types

A REST API uses different HTTP methods for different operations. Your API testing tool lets you switch between them.

## 4.1 GET with Query Parameters

Query parameters go after a `?` in the URL, separated by `&`:

```
GET http://localhost:3000/v1/input/search?q=express&limit=5
```

All three tools also provide a **Params** section (or a key-value editor) where you can enter query parameters as name-value pairs instead of typing them into the URL manually. The tool assembles the full URL for you.

| Parameter | Value |
| --- | --- |
| `q` | `express` |
| `limit` | `5` |

> **Use the Params Editor**
>
> Typing query strings by hand is error-prone — you might forget to URL-encode special characters or miss an `&`. Use the params editor built into your tool. It handles encoding automatically.

## 4.2 POST with JSON Body

POST requests typically send data in the request body. To create a new user:

**In Postman / Thunder Client:**

1. Change the method to **POST**
2. Enter the URL: `http://localhost:3000/v1/input/users`
3. Click the **Body** tab
4. Select **JSON** (Postman) or **JSON** from the content type dropdown (Thunder Client)
5. Enter the body:

```
{
  "name": "Alice",
  "email": "alice@example.com"
}
```

Click **Send**. You should receive a `201 Created` response (or `200 OK`, depending on your API).

**In WebStorm HTTP Client:**

```
### Create a new user
POST http://localhost:3000/v1/input/users
Content-Type: application/json

{
  "name": "Alice",
  "email": "alice@example.com"
}
```

Notice the blank line between the headers and the body — this is required in `.http` file syntax. It mirrors the actual HTTP protocol, where a blank line separates headers from the body.

## 4.3 PUT and PATCH

PUT replaces an entire resource. PATCH updates specific fields. The request format is the same as POST — you include a JSON body:

```
### Update a user (full replacement)
PUT http://localhost:3000/v1/input/users/1
Content-Type: application/json

{
  "name": "Alice Updated",
  "email": "alice.updated@example.com"
}

### Partial update
PATCH http://localhost:3000/v1/input/users/1
Content-Type: application/json

{
  "email": "new.email@example.com"
}
```

In Postman and Thunder Client, the process is identical to POST — just change the method dropdown to PUT or PATCH.

## 4.4 DELETE

DELETE requests typically have no body — you identify the resource to delete in the URL:

```
### Delete a user
DELETE http://localhost:3000/v1/input/users/1
```

In Postman or Thunder Client, select **DELETE** from the method dropdown and enter the URL. No body is needed.

> **Try It Yourself**
>
> Test all four operations against your running API:
>
> 1. **GET** `http://localhost:3000/v1/input/users` — list all users
> 2. **POST** `http://localhost:3000/v1/input/users` with a JSON body — create a new user
> 3. **PUT** `http://localhost:3000/v1/input/users/1` with a JSON body — update the user
> 4. **DELETE** `http://localhost:3000/v1/input/users/1` — delete the user
> 5. **GET** `http://localhost:3000/v1/input/users/1` again — confirm deletion (expect `404`)

---

## 5 Setting Headers

HTTP headers carry metadata about the request. Some are set automatically by your tool; others you need to add manually.

### 5.1 Content-Type

The `Content-Type` header tells the server what format the request body is in.

| Content-Type | Meaning |
|---|---|
| `application/json` | JSON data (most common for APIs) |
| `application/x-www-form-urlencoded` | Form data (HTML forms) |
| `multipart/form-data` | File uploads |

When you select "JSON" as the body type in Postman or Thunder Client, the tool **automatically** sets `Content-Type: application/json` for you. In WebStorm's HTTP Client,

you write the header explicitly in the `.http` file.

## 5.2 Authorization

Once your API requires authentication (starting with the auth-squared check-off), you will send a **Bearer token** in the `Authorization` header:

```
### Access a protected endpoint
GET http://localhost:3000/v1/protected/resource
Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...
```

In Postman and Thunder Client, there is a dedicated **Auth** tab where you can select "Bearer Token" and paste your token. The tool adds the `Authorization` header for you.

> **Common Header Mistakes**
>
> **"415 Unsupported Media Type"** — You forgot the `Content-Type: application/json` header (or your tool did not set it automatically). Verify it is present in the request headers.
>
> **"401 Unauthorized"** — Your `Authorization` header is missing, the token is expired, or there is a typo. Copy the token carefully — Bearer tokens are long and easy to truncate. Make sure there is exactly one space between `Bearer` and the token value.
>
> **"400 Bad Request" with valid-looking JSON** — Check that `Content-Type` is `application/json`, not `text/plain`. Some tools default to `text/plain` if you type in the body without selecting the JSON option.

## 5.3 Custom Headers

You can add any custom header your API expects. In all three tools, there is a **Headers** section where you add key-value pairs:

| Header | Value | Purpose |
|---|---|---|
| `Content-Type` | `application/json` | Request body format |
| `Authorization` | `Bearer <token>` | Authentication |
| `Accept` | `application/json` | Preferred response format |
| `X-Request-ID` | `abc-123` | Request tracing (custom) |

## 6 Saving and Organizing Requests

Testing an API is not a one-time activity — you will come back to the same endpoints repeatedly throughout the quarter. Saving and organizing your requests saves time and reduces errors.

### 6.1 Postman Collections

In Postman, requests are organized into **Collections** — folders that group related requests together.

1. Click **Collections** in the left sidebar
2. Click **+** to create a new collection (e.g., "TCSS 460 — Sprint 1")
3. Right-click the collection and select **Add Request**
4. Name the request descriptively (e.g., "GET all users")
5. Configure and save the request

You can export a collection as a JSON file and share it with teammates. They import it into their Postman and have all the same requests ready to go.

> **Naming Convention for Collections**
>
> A clear naming structure helps everyone on your team. Consider organizing by sprint or feature area:
>
> - `Sprint 1 – User Registration`
> - `Sprint 2 – Movie Proxy`
> - `Auth – Login and Token`

### 6.2 Thunder Client Collections

Thunder Client has a similar collections feature inside VS Code:

1. Click the **Collections** tab in the Thunder Client sidebar
2. Click the folder icon to create a new collection
3. Add requests to the collection

Thunder Client stores collection data locally (or in your workspace if you enable workspace-level storage). The free tier limits you to 3 collections with 15 requests each.

## 6.3 WebStorm `.http` Files

WebStorm takes a fundamentally different approach — your requests are **plain text files** with the `.http` extension. This has a significant advantage: `.http` files are just text, so they can be **committed to Git** alongside your source code.

```
### === User Endpoints ===

### List all users
GET http://localhost:3000/v1/input/users

### Create a user
POST http://localhost:3000/v1/input/users
Content-Type: application/json

{
  "name": "Alice",
  "email": "alice@example.com"
}

### Get user by ID
GET http://localhost:3000/v1/input/users/1

### Delete a user
DELETE http://localhost:3000/v1/input/users/1
```

Each `###` starts a new request. You can have as many requests as you want in a single file, or split them across multiple files (e.g., `users.http`, `auth.http`, `movies.http`).

> **Why `.http` Files Are Great for Group Projects**
>
> When your team commits `.http` files to your shared repository, every teammate gets a complete set of working test requests when they pull. No exporting, importing, or account linking required. When someone adds a new endpoint, they add the test request to the `.http` file in the same pull request. The test requests evolve with the code.

## 6.4 Why Saving Matters

Typing out the same request URL, headers, and body from memory every time you want to test an endpoint is slow and error-prone. Saved requests give you:

- **One-click retesting** — after a code change, re-run the request instantly

- **Shared test suites** — teammates can reproduce bugs using your exact requests
- **Living documentation** — a collection of requests describes your API's capabilities
- **Regression testing** — quickly verify that existing endpoints still work after changes

> **Gen AI & Learning: Generating Collections from Documentation**
>
> If your API has a Swagger/OpenAPI specification (covered in the Swagger/OpenAPI guide), you can ask an AI agent to generate a complete collection of test requests from it. Give the agent your OpenAPI spec and ask: "Generate a Postman collection (or `.http` file) that tests every endpoint with realistic sample data." This can save significant time, especially when your API has dozens of endpoints. The agent can also generate edge-case requests — empty bodies, missing required fields, invalid IDs — that you might not think to test manually.

---

# 7 Environment Variables

During development, your API runs at `http://localhost:3000`. When deployed, it runs at a public URL like `https://your-api.onrender.com`. Hardcoding the URL in every request means changing every request when you switch environments. Environment variables solve this.

## 7.1 The Concept

Instead of hardcoding URLs:

```
GET http://localhost:3000/v1/input/users
```

You use a variable:

```
GET {{base_url}}/v1/input/users
```

Then you define `base_url` differently for each environment:

| Environment | `base_url` |
| --- | --- |
| Local Development | `http://localhost:3000` |
| Deployed (Production) | `https://your-api.onrender.com` |

Switch environments, and every request automatically points to the right server.

## 7.2 Postman Environments

1. Click the **Environments** tab in the left sidebar
2. Click **+** to create a new environment (e.g., "Local")
3. Add a variable: `base_url` = `http://localhost:3000`
4. Create another environment (e.g., "Production") with `base_url` = `https://your-api.onrender.com`
5. Select the active environment from the dropdown in the top-right corner

In your requests, use `{{base_url}}`:

```
GET {{base_url}}/v1/input/users
```

Postman replaces `{{base_url}}` with the value from whichever environment is active.

## 7.3 Thunder Client Environments

Thunder Client handles environments similarly:

1. Click the **Env** tab in the Thunder Client sidebar
2. Create a new environment and add variables
3. Reference them in requests with `{{variable_name}}` syntax

## 7.4 WebStorm HTTP Client Environments

WebStorm uses a JSON file called `http-client.env.json` in your project root:

```
{
  "local": {
    "base_url": "http://localhost:3000"
  },
  "production": {
    "base_url": "https://your-api.onrender.com"
  }
}
```

In your `.http` files, use the same `{{base_url}}` syntax:

```
### List all users
GET {{base_url}}/v1/input/users
```

When you run a request, WebStorm prompts you to select an environment. Since `http-client.env.json` is a plain text file, it can be committed to Git — the whole team shares the same environment definitions.

> **Do Not Commit Secrets**
>
> Environment variables often include auth tokens and API keys. WebStorm supports a separate `http-client.private.env.json` file for sensitive values — add this file to `.gitignore` so secrets never reach your repository. Postman and Thunder Client store environment data in their own application storage, but be careful when exporting collections that include environment values.

## 7.5 Common Variables

These are useful variables to define in any API testing environment:

| Variable | Example Value | Purpose |
|---|---|---|
| `base_url` | `http://localhost:3000` | Server base URL |
| `token` | `eyJhbGciOi...` | Reusable auth token |
| `user_id` | `42` | Test user ID for CRUD operations |
| `api_version` | `v1` | API version prefix |

> **Gen AI & Learning: Debugging Failing Requests**
>
> When a request that used to work suddenly returns an error, paste both the request and the error response into your AI coding agent and ask: "This request was returning 200 OK yesterday but now gives a 500 Internal Server Error. Here is the request and response — what changed?" The agent can often spot issues like missing headers, changed field names, or expired tokens faster than manually comparing old and new behavior. If your API changed recently, share the relevant code diff too — the agent can correlate the code change with the test failure.

---

# 8 Which Should You Use?

All three tools get the job done. Here is how they compare:

| Feature | Postman | Thunder Client | WebStorm HTTP Client |
|---|---|---|---|
| **Runs in** | Standalone app | VS Code extension | Built into WebStorm |
| **Cost** | Free (1 user) | Free (personal/edu) | Free (student license) |
| **Setup effort** | Download + account | One-click install | Nothing (built in) |
| **Request saving** | Collections (cloud/local) | Collections (local) | `.http` files (plain text) |
| **Git-friendly** | Export JSON manually | Workspace storage option | `.http` files commit directly |
| **Environment variables** | GUI-based, per-environment | GUI-based, per-environment | `http-client.env.json` file |
| **Team sharing** | Export/import collections | Manual file sharing | Commit `.http` files to repo |
| **GUI polish** | Highly polished | Clean, lightweight | Code editor (text-based) |
| **Works offline** | Yes | Yes | Yes |

## 8.1 Recommendations

**If you use VS Code:** Start with **Postman**. It has the most polished interface, the best documentation, and the largest community. Thunder Client is convenient for quick one-off tests without leaving your editor, but its free tier limitations make it less practical as your primary tool.

**If you use WebStorm:** Use the **built-in HTTP Client**. It requires no extra installation, and `.http` files integrate naturally into your Git workflow. Your test requests travel with your code.

**You can use more than one.** Many developers use Postman for complex testing workflows and their editor's built-in tools for quick checks. There is no requirement to pick a single tool.

### For Group Projects

Consider having your team agree on one approach for shared test requests. If you commit `.http` files (WebStorm format), anyone can read them regardless of their editor — the syntax is plain text. Postman collections require everyone to have Postman installed and to import the exported JSON.

### Gen AI & Learning: Generating `.http` Files from Route Definitions

Once your Express API has several route files, you can ask an AI agent to scan your route definitions and generate a complete `.http` file. Point the agent at your `src/routes/` directory and ask: "Generate an `.http` file with sample requests for every route, including realistic request bodies for POST and PUT endpoints." The agent reads your route handlers, identifies the expected parameters and body shapes, and produces a ready-to-use test file. This is especially useful at the start of each sprint when your team adds new endpoints.

---

## 9 Summary

| Concept | Key Point |
| --- | --- |
| Why API testing tools? | Browsers can only send GET requests — you need tools for POST, PUT, DELETE, and custom headers |
| Postman | Standalone desktop app with polished GUI, free for individual use |
| Thunder Client | Lightweight VS Code extension, free for personal/educational use |
| WebStorm HTTP Client | Built into WebStorm, uses `.http` files that commit to Git |
| Response reading | Always check the status code first, then headers, then body |
| Request body | POST/PUT/PATCH send JSON in the body with `Content-Type: application/json` |

| Concept | Key Point |
| --- | --- |
| Headers | `Content-Type` and `Authorization` are the two you will use most |
| Collections | Save and organize requests for one-click retesting and team sharing |
| `.http` files | Plain-text request files — Git-friendly alternative to GUI collections |
| Environment variables | Use `{{base_url}}` to switch between localhost and deployed URLs |
| Team workflow | Commit shared test requests so every teammate has the same test suite |

---

## 10 References

**Official Documentation:**

- Postman Learning Center — Getting started guides, collection management, and environment variables
- Postman Downloads — Desktop app for macOS, Windows, and Linux
- Thunder Client — Official site with feature overview and pricing
- Thunder Client — VS Code Marketplace — Install directly from VS Code
- WebStorm HTTP Client Documentation — Complete reference for `.http` file syntax and features
- WebStorm HTTP Request Syntax — Detailed `.http` file format specification
- JetBrains Educational Pack — Free WebStorm license for students

**HTTP Fundamentals:**

- MDN — HTTP Request Methods — Official reference for GET, POST, PUT, PATCH, DELETE
- MDN — HTTP Response Status Codes — What each status code means

---

## 11 Further Reading

**External Resources**

- Postman Blog — Getting Started with Collections — How to organize and share API test suites
- WebStorm Blog — HTTP Client Tips and Tricks — Advanced `.http` file techniques including response handling and scripting
- REST API Tutorial — Background on RESTful API design and HTTP methods
- HTTP Cats — A lighthearted way to remember HTTP status codes (seriously, bookmark this)

---

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