assignment # check-off # express

Check-Off 1 — Running Web API + API Testing

School of Engineering and Technology, University of Washington Tacoma
TCSS 460 — Client/Server Programming, Spring 2026

Due Date

Sunday, April 12, 2026, 11:59 PM

Description

In this check-off, you will clone an existing Express + TypeScript web API, run it locally, explore its routes and documentation, and then extend it with your own routes, tests, and API documentation. You will demonstrate your work to a peer who will verify each step.

This is your first hands-on experience with a real Express project. By the end, you will have a working mental model of how routes, controllers, middleware, and tests fit together.

Learning Objectives

By completing this check-off, you will:

- Clone and run a Node.js/Express project from a Git repository
- Use Postman, Thunder Client, or another visual API tool to make HTTP requests and inspect responses
- Navigate an Express project's file structure (routes, controllers, middleware)
- Create new route and controller files following an established project pattern
- Write integration tests for your new routes using Jest and Supertest
- Document your routes using OpenAPI/Swagger annotations
- Demonstrate your understanding to a peer

Course Learning Objectives

This check-off contributes to the following course learning objectives:

- **LO 1:** Design and implement RESTful web APIs using a server-side framework
- **LO 6:** Transfer object-oriented programming skills to a new language and ecosystem

It also supports these course outcomes:

- **Inquiry and Critical Thinking** — navigating an unfamiliar project structure and reasoning about route/middleware interactions

---

Before You Begin

Ensure you have completed:

- **Node.js installed** (v22+) — see the Node.js Setup guide
- **VS Code or WebStorm configured** — see the VS Code / WebStorm guide
- **An API testing tool installed** — Postman, Thunder Client (VS Code extension), or similar
- **Git installed** and you can clone repositories

Guides for This Check-Off

These guides cover the concepts and patterns you will use:

- Intro to Express — Routes, HTTP methods, params, query strings, request body
- Routing & Middleware — Middleware chain, Express Router, organizing routes in files
- Error Handling & Validation — Input validation, error responses, status codes
- API Testing — Jest + Supertest, writing route tests, mocking

---

Project Setup

Requirement 1: Clone and Run the Lecture Demo

TCSS 460 Backend 1

github.com/UWT-SET-TCSS460-LECTURE-MATERIALS/TCSS-460-Backend-1

Clone the repository:

```
git clone https://github.com/UWT-SET-TCSS460-LECTURE-MATERIALS/TCSS-460-Backend-1.git
cd TCSS-460-Backend-1
```

Install dependencies:

```
npm install
```

Start the development server:

```
npm run dev
```

You should see output like:

```
Server running at http://localhost:3000
API docs at http://localhost:3000/api-docs
```

> **Verify It Works**
>
> Open your browser and visit `http://localhost:3000/v1/hello`. You should see:
>
> ```
> { "message": "Hello, you sent a GET request" }
> ```

Requirements

Requirement 2: Explore with an API Tool

Open Postman, Thunder Client, or your preferred API tool and make requests to the existing API. You will demo these to your checker.

Minimum requests to prepare:

1. `GET /v1/hello` — verify the basic route works
2. `POST /v1/hello` — notice the `201` status code
3. `GET /v1/input/search?q=express&limit=5` — query string parameters
4. `GET /v1/input/users/42` — route parameter
5. `POST /v1/input/users` with JSON body `{"name": "Your Name", "email": "you@example.com"}` — request body
6. `GET /v2/input/users/abc` — see the validation middleware return a `400` error

> **Save These in a Collection**
>
> Create a collection called "TCSS 460 — Check-Off 1" and save each request. This makes your demo smoother and gives you a reference for later.

Requirement 3: Review the API Documentation

Visit `http://localhost:3000/api-docs` in your browser. This is the Swagger UI — an interactive documentation page generated from the project's `openapi.yaml` file.

Explore:

- Browse the available endpoints grouped by tag
- Use the "Try it out" button to send a request directly from the docs
- Notice how each endpoint documents its parameters, request body, and response format

You will add documentation for your new routes in Requirement 6.

Requirement 4: Run the Existing Tests

Stop the dev server (Ctrl+C) and run the test suite:

```
npm test
```

You should see all tests pass:

```
Test Suites: 5 passed, 5 total
Tests:       42 passed, 42 total
```

> **42 Tests to Start**
>
> The repo ships with **42 passing tests** across 5 test suites. Your checker will verify that your final test count is **higher than 42** — meaning you added new tests.

Requirement 5: Create Your Own Routes

Now it is your turn. Create a **new route file and controller file** that follow the project's existing pattern. Your new routes should be mounted under `/v1`.

What to build:

Create a new feature with **at least 3 routes** that accept different types of input. For example, you might build a "greeting" feature, a "math" feature, or a "quote" feature — the theme is up to you. The routes must demonstrate:

1. **At least one route with a route parameter** (e.g., `/v1/yourfeature/:id`)
2. **At least one route with a query string** (e.g., `/v1/yourfeature/search?q=...`)
3. **At least one POST route with a JSON body**

Files to create:

Following the project's pattern, you need:

1. A **controller file** in `src/controllers/` — your route handler functions
2. A **route file** in `src/routes/v1/` — maps paths to your handlers
3. **Mount your router** in `src/routes/v1/index.ts`

> **How to Structure Routes**
>
> See the Routing & Middleware guide, Section 7 — "Organizing Routes in Files" for the exact pattern.

Example structure (if you chose a "greeting" theme):

```
src/
├── controllers/
│   └── greeting.ts        ← Your handler functions
├── routes/
│   └── v1/
│       ├── index.ts        ← Mount your router here
│       └── greeting.ts     ← Your route definitions
```

> **Don't Modify Existing Routes**
>
> Add your files alongside the existing ones. Do not edit `hello.ts`, `input.ts`, or any other existing file (except `routes/v1/index.ts` where you mount your new router).

Requirement 6: Document Your Routes

Add OpenAPI documentation for your new routes in the `openapi.yaml` file. Your documentation should include:

- The path and HTTP method for each route
- A description of what each route does
- Parameter definitions (path params, query params, or request body)
- At least one example response

After updating the file, restart the dev server and verify your routes appear in the Swagger UI at `http://localhost:3000/api-docs`.

Requirement 7: Write Tests for Your Routes

Create a test file for your new routes in the `tests/v1/` directory. Your tests must cover:

1. **Happy path** — each route responds correctly with valid input
2. **At least one sad path** — a route returns an appropriate error (e.g., `400`) for invalid input

Follow the testing patterns from the existing test files.

> **How to Write Route Tests**
>
> See the API Testing guide for the full walkthrough of Jest + Supertest patterns.

Example test file location:

```
tests/
└── v1/
    └── greeting.test.ts    ← Your tests
```

Run your tests to verify they pass:

```
npm test
```

Your total test count should now be **higher than 42**.

Peer Check-Off

Find a classmate to verify your work.

> **For the Checker**
>
> The checker has 10 points to award. There is no rubric — use your judgment. Did they do the work? Can they explain it? Award points based on completeness and understanding — not perfection.

Check-Off Checklist

#	Verify	✓
1	Server starts with `npm run dev` and responds at `http://localhost:3000/v1/hello`	☐
2	Student demos at least 4 requests to existing routes using an API tool (GET, POST, params, validation error)	☐
3	Student shows Swagger docs at `/api-docs`	☐
4	`npm test` passes with 42 tests (the starting count)	☐
5	New controller file exists in `src/controllers/`	☐
6	New route file exists in `src/routes/v1/` and is mounted in `v1/index.ts`	☐
7	New routes include: a route param, a query string, and a POST with body	☐
8	Student demos new routes in their API tool with valid input	☐
9	Student demos at least one new route with invalid input showing an error response	☐
10	New routes appear in Swagger docs at `/api-docs`	☐
11	`npm test` passes with **more than 42 tests** (new tests added)	☐

How to Submit

1. Complete the check-off requirements above.
2. Find a classmate to be your checker.
3. Demo your work — walk them through each item on the checklist.
4. The checker has 10 points to award. There is no rubric — use your judgment. Did they do the work? Can they explain it? Award points based on completeness and understanding — not perfection.
5. Both of you fill in the shared spreadsheet linked from the Canvas assignment: the student enters their name in Column A, the checker enters their name in Column B, and the checker enters the score in Column C.

Canvas — Check-Off 1: Running Web API + API Testing

Guide Reference

Guide	What It Covers
Intro to Express	Routes, HTTP methods, params, query, body, responses
Routing & Middleware	Router pattern, file organization, middleware
Error Handling & Validation	Validation, error responses, status codes
API Testing	Jest + Supertest patterns
Node.js Setup	Installing Node.js
VS Code / WebStorm	Editor setup

Gen AI & Learning: Using AI for This Check-Off

You are welcome to use AI coding assistants to help you write routes, tests, and OpenAPI documentation. However, your checker will ask you to explain what your code does. If you cannot explain a route handler, a test assertion, or why a status code is appropriate, the check-off is not complete. Use AI as a tool, but make sure you understand every line.

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