assignment # check-off # express # proxy

Check-Off 2 — Proxy API

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

Due Date

Sunday, April 19, 2026, 11:59 PM

Description

In this check-off, you will explore the proxy routes added to Backend-1 during Week 3 lectures, then build your own proxy routes that call a third-party API. You will create one pass-through proxy that forwards a response as-is and one transformed proxy that reshapes verbose API data into a clean, purpose-built response.

This assignment reinforces the proxy pattern — a server-side intermediary that hides API keys from clients, simplifies complex third-party responses, and gives you full control over your API's contract.

Learning Objectives

By completing this check-off, you will:

- Explain the proxy pattern and why it is used in client/server architecture
- Make server-side HTTP requests to a third-party API using `fetch`
- Create a pass-through proxy route that forwards a third-party response
- Create a transformed proxy route that reshapes data into a new schema
- Apply validation middleware to proxy routes
- Handle errors from third-party API calls (network failures, non-OK responses)
- Document proxy routes using OpenAPI annotations
- Write integration tests for proxy routes

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** — analyzing third-party API responses and designing transformed response schemas

---

Before You Begin

Ensure you have completed:

- **Check-Off 1** — You already have Backend-1 cloned and working
- **An OpenWeatherMap API key** — Sign up at openweathermap.org if you have not already (the free tier is sufficient)
- **An API testing tool installed** — Postman, Thunder Client (VS Code extension), or similar

Guides for This Check-Off

These guides cover the concepts and patterns you will use:

- The Proxy Pattern — Why proxy, pass-through vs. transformed, hiding API keys
- Routing & Middleware — Middleware chain, Express Router, organizing routes in files
- Error Handling & Validation — Input validation, error responses, status codes
- Async Concepts — Promises, async/await, error handling
- Async in TypeScript — Using async/await with fetch and Express

---

Project Setup

Requirement 1: Pull Latest and Set Up

TCSS 460 Backend 1

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

You already have Backend-1 cloned from Check-Off 1. The Week 3 lectures added proxy routes to the repository — pull the latest changes:

```
cd TCSS-460-Backend-1
git pull
npm install
```

Create a `.env` file in the project root with your OpenWeatherMap API key:

```
WEATHER_API_KEY=your_api_key_here
```

Start the development server:

```
npm run dev
```

Verify the Proxy Routes Work

Make these two requests with your API tool:

- `GET http://localhost:3000/proxy/weather?city=Seattle` — pass-through proxy (returns raw OWM data)
- `GET http://localhost:3000/proxy/summary?city=Seattle` — transformed proxy (returns a simplified response)

Both should return `200` with JSON. If you get a `500` or the server crashes, check that your `.env` file is in the project root and the key is valid.

Requirements

Requirement 2: Explore Existing Proxy Routes

Before building your own, explore what the lecture demo already provides. Use your API tool to make requests to all existing proxy routes:

Pass-through routes (forward raw OWM responses):

- `GET /proxy/weather?city=Seattle`

- `GET /proxy/weather/Seattle`
- `GET /proxy/forecast?city=Seattle`

Transformed routes (reshape OWM data):

- `GET /proxy/summary?city=Seattle`
- `GET /proxy/summary/Seattle`

Compare the responses:

- Look at `/proxy/weather?city=Seattle` and `/proxy/summary?city=Seattle` side by side. The first is OWM's raw response — verbose, deeply nested, full of fields your client might not need. The second is a purpose-built response shaped by your server.

Explore the code:

- Open `src/controllers/proxy/` — find the controller that handles these routes
- Open `src/routes/proxy/` — find the router that maps paths to handlers
- Find the `requireEnvVar` middleware — how does it protect the API key?
- Find the `requireCity` middleware — how does it validate input?

You will demo this exploration to your checker and explain the difference between pass-through and transformed proxy responses.

Requirement 3: Build a Pass-Through Proxy Route

> **Reference**
>
> See The Proxy Pattern guide for the full pattern, including pass-through examples and error handling.

Create a new proxy route that calls a **different** OpenWeatherMap endpoint — one that the demo does not already cover. Good options include:

- Air Pollution API — air quality data for a location
- Geocoding API — convert city names to coordinates
- Any other OWM endpoint not already used by the demo

Your route must:

1. **Live in its own controller and route file** — follow the project's existing file organization pattern
2. **Be mounted under** `/proxy` — your route path should be `/proxy/yourpath`
3. **Use the existing** `requireEnvVar('WEATHER_API_KEY')` **middleware** — do not create a duplicate
4. **Include validation middleware** — validate required input (reuse existing middleware like `requireCity` or create new middleware appropriate for your chosen endpoint)
5. **Forward the OWM response as-is** — this is a pass-through proxy, not a transformation
6. **Handle errors** — check `result.ok` before parsing, and wrap the `fetch` call in try/catch for network failures

> ⚠ **Do Not Duplicate Existing Routes**
>
> The demo already covers `/proxy/weather`, `/proxy/forecast`, and `/proxy/summary`. Choose an endpoint that does something different.

Requirement 4: Build a Transformed 5-Day Forecast Route

> 📖 **Reference**
>
> See the "Transformed Proxy" section of The Proxy Pattern guide for examples of reshaping API responses.

Create a transformed proxy route that calls OWM's 5-day / 3-hour forecast endpoint and reshapes the response into a **daily summary**.

The problem: OWM's forecast endpoint returns up to 40 entries — one for every 3-hour interval over 5 days. That is a lot of data, and most clients do not need 3-hour granularity. Your job is to aggregate this into something useful.

The goal: Return a response that provides a **5-day daily summary**. For each day, include at minimum:

- The date
- The daily high temperature
- The daily low temperature
- A general weather condition (e.g., "Clouds", "Rain", "Clear")

You design the response shape. There is no single correct schema. Decide what fields to include, how to name them, and how to structure the JSON. The only requirement is that it is YOUR schema — not raw OWM JSON passed through.

Your route must:

1. Call OWM's 5-day forecast endpoint server-side
2. Transform the response — aggregate the 3-hour entries into daily summaries
3. Return your custom-shaped JSON response
4. Include proper error handling (non-OK responses, network failures)

> **Aggregation Strategy**
>
> Think about how to determine the "daily high" from multiple 3-hour entries. You will need to group entries by date and then find the max/min temperatures within each group. The weather condition could be the most frequently occurring condition for the day, or the condition at midday — your choice.

Requirement 5: Document Your Routes

Add OpenAPI documentation for **both** 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)
- Response schema — especially for the transformed route, document YOUR response shape
- At least one example response for each route

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

> **Reference**
>
> See the OpenAPI Documentation guide for the YAML syntax and how to define schemas, parameters, and responses.

Requirement 6: Write Tests

Create tests for your new routes. Your tests must cover:

1. **Happy path for the pass-through route** — valid input returns `200` with data
2. **Happy path for the transformed route** — valid input returns `200` with your custom schema
3. **At least one error case** — missing or invalid input returns an appropriate error (e.g., `400`)

Follow the testing patterns from the existing test files in the `tests/` directory.

> **Reference**
>
> See the API Testing guide for Jest + Supertest patterns.

```
npm test
```

Your total test count should be **higher** than the starting count.

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 existing proxy routes respond (test `/proxy/weather?city=Seattle`)	☐
2	Student explains the difference between a pass-through and a transformed proxy response	☐

#	Verify	✓
3	Student shows the `requireEnvVar` middleware and explains how it protects the API key	☐
4	New pass-through proxy route exists under `/proxy` and calls a different OWM endpoint	☐
5	Student demos the pass-through route with valid input — returns `200` with raw OWM data	☐
6	New transformed forecast route exists and returns a daily summary (not raw 3-hour data)	☐
7	Student demos the transformed route and explains how they aggregated 3-hour entries into daily summaries	☐
8	Student demos an error case (missing/invalid input) and the route returns an appropriate error response	☐
9	Both new routes appear in the API docs at `/api-docs` with parameters and response schemas	☐
10	`npm test` passes and includes new tests for both routes	☐
11	New controller and route files follow the project's existing file organization pattern	☐
12	Student can explain the overall request flow: client request, server fetch to OWM, response back to client	☐

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 2: Proxy API

Guide Reference

Guide	What It Covers
The Proxy Pattern	Pass-through vs. transformed proxies, hiding API keys, error handling
Routing & Middleware	Router pattern, file organization, middleware chains
Error Handling & Validation	Validation, error responses, status codes
API Testing	Jest + Supertest patterns
OpenAPI Documentation	Documenting routes with OpenAPI/YAML
Async Concepts	Promises, async/await fundamentals
Async in TypeScript	Using async/await with fetch and Express
Intro to Express	Routes, HTTP methods, params, query, body

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

You are welcome to use AI coding assistants to help you write proxy routes, transformation logic, tests, and OpenAPI documentation. However, your checker will ask you to explain the proxy pattern, your transformation strategy, and your error handling. If you cannot explain why a proxy hides the API key, how you aggregated forecast data, or what happens when the third-party API is down, 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.