

Pair-Programming Assignment 3 (PPA 3)

Creating Data with POST and Server-Side Validation

In PPA 3 you will extend the appointment scheduling system by allowing clients to create new appointment slots using the POST method. This version intentionally avoids request body streaming and avoids Promises and async await. You will send slot data using query parameters, so the server logic stays fully sequential inside the request handler.

Important Constraint for This Assignment

To keep all code sequential, you will not read a JSON request body in PPA 3. Instead, the client will send `startTime` and `endTime` in the URL query string. Later, after we cover asynchronous programming, you will switch to POST with a JSON body.

Connection to Today's Lesson (February 5)

This assignment applies REST resource design, HTTP methods and status codes, and client server responsibilities. The server validates input and protects system state. The client sends requests and updates the UI from responses.

Learning Objectives

- Build an interactive front end that submits a form, renders results, and displays server validation feedback without reloading
- Use the POST HTTP method correctly
- Validate input on the server using control flow and conditionals
- Return appropriate HTTP status codes and JSON responses

Learning Progression Across Assignments

- PPA 2 focused on reading data and displaying server responses
- PPA 3 focuses on creating data and reacting to server validation, with strong front-end interaction
- PPA 4 will focus on persisting data and maintaining state across server restarts

Reading Alignment

- Flanagan, JavaScript: The Definitive Guide
- Chapters 3–5 (types, control flow, equality)
- Chapter 6 (objects and arrays)

How This Fits into the Overall Project

By the end of this assignment, the system will support both reading and creating appointment slots. Future assignments will add persistence, updates, deletion, and more complex rules.

Assignment Tasks

- GET `/api/slots` returns an array of slot objects
- POST `/api/slots` accepts JSON input
- Parse JSON data from the request body
- Add the new slot to the in-memory data model
- Return appropriate HTTP status codes

- Valid POST returns 201 Created and the created slot object
- Invalid POST returns 400 Bad Request with a JSON error message

Starter Code (Server)

Create a file named `server.js`. Copy the starter code below. Then complete the TODO sections. Keep functions separated by two blank lines. Add comments where you make design decisions.

```
// server.js
// Appointment scheduling server (PPA 3)
// Uses Node.js http module only (no frameworks)
// Sequential POST handling: parameters are passed in the URL query string

const http = require("http");
const url = require("url");

// In memory data model (persistence added in PPA 4)
const slots = [];

function sendJson(res, statusCode, payload) {

  res.writeHead(statusCode, { "Content-Type": "application/json" });
  res.end(JSON.stringify(payload));

}

function nextId() {

  return slots.length + 1;

}

function validateSlotTimes(startTime, endTime) {

  if (typeof startTime !== "string" || startTime.trim().length === 0) {
    return { ok: false, message: "startTime is required" };
  }

  if (typeof endTime !== "string" || endTime.trim().length === 0) {
    return { ok: false, message: "endTime is required" };
  }

  // TODO (bonus): verify endTime is after startTime
  return { ok: true, message: "" };

}

function isDuplicate(startTime, endTime) {

  // TODO (bonus): return true if a slot with the same times already exists
  return false;

}
```

```

}

const server = http.createServer(function (req, res) {

  const parsedUrl = url.parse(req.url, true);
  const path      = parsedUrl.pathname;
  const query     = parsedUrl.query;

  if (req.method === "GET" && path === "/api/slots") {

    sendJson(res, 200, slots);
    return;

  }

  if (req.method === "POST" && path === "/api/slots") {

    const startTime = query.startTime;
    const endTime   = query.endTime;

    const result    = validateSlotTimes(startTime, endTime);

    if (!result.ok) {
      sendJson(res, 400, { error: result.message });
      return;
    }

    // TODO (bonus): prevent duplicates
    // if (isDuplicate(startTime, endTime)) {
    //   sendJson(res, 409, { error: "Duplicate slot" });
    //   return;
    // }

    const slot = {
      id       : nextId(),
      startTime : startTime,
      endTime   : endTime,
      status    : "available"
    };

    slots.push(slot);

    sendJson(res, 201, slot);
    return;

  }

  sendJson(res, 404, { error: "Not found" });

});

server.listen(3000, function () {

```

```
console.log("Server running at http://localhost:3000");
});
```

Interactive Front-End Guidance

Starting with this assignment, interactive front-end behavior is graded. Your page must react to server responses. Do not reload the page to see new slots. The server response is the source of truth.

Front End Requirements

- A form where the provider enters startTime and endTime
- On submit, send a POST request that includes startTime and endTime
- If response status is 201, render the returned slot into the page immediately
- If response status is 400 or 409, show the error message returned by the server
- Show slots in a readable UI (table or cards)

Suggested UI Behaviors (Choose at least two)

- Clear the form on success
- Keep keyboard focus in the startTime field after success
- Show a green success banner and a red error banner
- Disable the submit button while a request is in progress (simple UI state)

Starter Code: provider.html

Create provider.html in the same folder. This starter includes a table for slots and a message banner.

```
<!-- provider.html -->
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Provider Slot Manager</title>
    <style>
      body { font-family: Arial, sans-serif; margin: 24px; }
      label { display: block; margin-top: 12px; }
      input { width: 340px; padding: 6px; }
      button { margin-top: 12px; padding: 8px 12px; }
      .error { color: #b00020; }
      .ok { color: #0b6b0b; }
      table { border-collapse: collapse; margin-top: 16px; width: 720px; }
      th, td { border: 1px solid #ccc; padding: 8px; text-align: left; }
      th { background: #f2f2f2; }
    </style>
  </head>

  <body>
    <h1>Provider Slot Manager</h1>

    <p id="message"></p>
```

```

<form id="slotForm">
  <label>Start time (example: 2026-03-01T09:00)</label>
  <input id="startTime" type="text" placeholder="YYYY-MM-DDTHH:MM">

  <label>End time (example: 2026-03-01T09:30)</label>
  <input id="endTime" type="text" placeholder="YYYY-MM-DDTHH:MM">

  <button id="createBtn" type="submit">Create slot</button>
</form>

<h2>Current slots</h2>

<table>
  <thead>
    <tr>
      <th>Start time</th>
      <th>End time</th>
      <th>Status</th>
    </tr>
  </thead>
  <tbody id="slotTableBody"></tbody>
</table>

<script src="provider.js"></script>
</body>
</html>

```

Starter Code: provider.js

Create provider.js. This starter uses XMLHttpRequest (no Promises, no async/await). Your main work is to implement onload so the UI updates based on the HTTP status code.

```

// provider.js
// Interactive UI logic for PPA 3
// Uses XMLHttpRequest (no Promises, no async/await)

function setMessage(text, kind) {

  const p = document.getElementById("message");
  p.textContent = text;

  if (kind === "error") {
    p.className = "error";
  } else {
    p.className = "ok";
  }
}

function addSlotRow(slot) {

  const tbody = document.getElementById("slotTableBody");

```

```

const tr = document.createElement("tr");
const td1 = document.createElement("td");
const td2 = document.createElement("td");
const td3 = document.createElement("td");

td1.textContent = slot.startTime;
td2.textContent = slot.endTime;
td3.textContent = slot.status;

tr.appendChild(td1);
tr.appendChild(td2);
tr.appendChild(td3);

tbody.appendChild(tr);
}

function parseJsonSafely(text) {
  try {
    return { ok: true, value: JSON.parse(text) };
  } catch (err) {
    return { ok: false, value: null };
  }
}

// POST /api/slots?startTime=...&endTime=...
function submitNewSlot(startTime, endTime) {

  const xhr = new XMLHttpRequest();

  const requestUrl =
    "/api/slots?startTime=" + encodeURIComponent(startTime) +
    "&endTime=" + encodeURIComponent(endTime);

  xhr.open("POST", requestUrl);

  xhr.onload = function () {

    // TODO: parse JSON response safely
    // TODO: if status 201, addSlotRow(slot) and show a success message
    // TODO: if status 400 or 409, show the server error message
    // TODO: otherwise, show a generic error message

  };

  xhr.send();
}

```

```
document.getElementById("slotForm").addEventListener("submit", function
(event) {

    event.preventDefault();

    const startTime = document.getElementById("startTime").value;
    const endTime   = document.getElementById("endTime").value;

    submitNewSlot(startTime, endTime);

});
```

Student Decisions (You Must Make Choices)

You must decide what makes an appointment slot valid. At minimum, consider whether required fields are present and whether their values make sense.

- Choose how to format startTime and endTime and explain your choice in comments
- Decide what a good success message looks like and when it should disappear
- Decide what the UI should do after success (clear inputs, focus, disable button, etc.)
- Choose a readable slot presentation (table is provided, but you may redesign as cards)

Bonus (Optional)

- Prevent duplicate time slots
- Automatically assign unique IDs
- Return 400 status codes for invalid input with helpful messages

Grading Rubric

This assignment is graded on a 3-point scale.

1 point:	POST endpoint exists and returns a response; UI integration is incomplete or unclear
2 points:	Server creates slots with validation and correct status codes; client renders created slots and shows server messages
3 points:	Bonus rules implemented (duplicates or time ordering) and polished UI behavior; documented in README

Submission Checklist

- Server starts without errors
- GET /api/slots still works
- POST /api/slots creates new slots and returns 201
- Invalid input returns 400 with JSON error message
- UI updates without page reload and shows success or error messages
- Code is committed to GitHub