PPA 5 Objects and Monthly Calendar Interface

Purpose

In PPA 5 you will improve the appointment scheduling interface by rendering appointments in a monthly calendar view. You will fetch data from the server using `GET` and refresh the interface after creating new appointments using `POST`. `DELETE` will be introduced in PPA 6.

Scope rules for this assignment

- Appointments must fit within a single day. No multi day appointments.
- Use ISO strings like 2026-03-01T09:00 for inputs and server data.
- Client networking must remain callback based. No async and no await.
- Focus on objects, arrays, DOM creation, and CSS classes.

Learning objectives

- Use objects and arrays to represent appointments and calendar cells
- Render a monthly calendar using DOM operations
- Explain the `GET` and `POST` update loop in a client server app
- Use CSS classes to control presentation without changing HTML structure
- Write readable code with indentation and comments

Application flow

Your page follows this flow. This flow is the key idea for today.

1. Page loads and the browser builds the DOM from HTML
2. provider.js runs after the DOM exists because the script tag is at the end of the body
3. `refreshCalendar` sends `GET /api/slots`
4. When `GET` returns, `renderCalendar` rebuilds the month view
5. User creates a new slot and clicks Create slot
6. `sendCreateSlot` sends `POST /api/slots`
7. When `POST` returns `201`, `refreshCalendar` runs again so the UI matches server data

Beginner support understanding functions

What a function is:

A function is a reusable block of code. You define it once and call it when needed. Functions help you break one large problem into smaller steps.

```
// Define a function once
function sayHello() {
  console.log("Hello");
}


// Call the function to run it
sayHello();
```

Parameters and return values

Parameters are inputs. Return values are outputs. Return sends a value back to the caller.

```
// add takes two inputs a and b and returns the sum
function add(a, b) {
  return a + b;
}


const total = add(3, 5);
console.log(total);
```

Beginner support CSS and classes

What CSS does:

CSS controls presentation. HTML provides structure. JavaScript can change classes on elements to change how they look without changing the underlying data.

How class changes affect presentation

A class selector begins with a dot. If an element has class slotItem, then rules for .slotItem apply. If JavaScript adds a second class like available, rules for .available also apply.

```
<!-- Same HTML element, different classes -->
<div class="slotItem available">09:00 to 09:30</div>
<div class="slotItem booked">09:00 to 09:30</div>
```

In JavaScript you can change the class to change the presentation.

```
const el = document.GETElementById("message");

el.className = "ok";        // one class
el.className = "error";     // different class, different look
```

You can also add or remove classes without replacing all classes.

```
const cell = document.querySelector(".dayCell");

cell.classList.add("today");
cell.classList.remove("today");
```

Files you will create or edit

- public/provider.html
- public/style.css
- public/provider.js
- server.js must serve provider.html, provider.js, and style.css

Starter file provider.html

Use these ids exactly so your JavaScript can find elements with document.getElementById.

```
<!-- public/provider.html -->
<!doctype html>
<html>
  <head>
    <meta charset="utf-8">
    <title>Provider Calendar</title>
    <link rel="stylesheet" href="/style.css">
  </head>
  <body>

    <h1 id="monthTitle"></h1>

    <div id="message"></div>

    <section id="createSlotPanel">
      <h2>Create a slot</h2>

      <label>Start time ISO
        <input id="startTimeInput" placeholder="2026-03-01T09:00">
      </label>

      <label>End time ISO
        <input id="endTimeInput" placeholder="2026-03-01T09:30">
      </label>

      <button id="createSlotButton">Create slot</button>
    </section>

    <section>
      <h2>Month view</h2>
      <div id="calendarGrid"></div>
    </section>

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

  </body>
</html>
```

Why the script tag is at the end of the body

The browser reads HTML from top to bottom. If your JavaScript runs before the DOM elements exist, document.getElementById returns null. Placing the script tag near the end of body ensures the elements exist before your code tries to use them.

```
<!-- public/provider.html -->
<body>

  <!-- HTML elements appear first so the browser builds them into the DOM -
->
```

```
  <div id="calendarGrid"></div>

  <!-- Script runs after the DOM exists -->
  <script src="/provider.js"></script>

</body>
```

Starter file style.css

This CSS creates a clean month grid. You can add small improvements as bonus work.

```
/* public/style.css */

#calendarGrid {
  display: grid;
  grid-template-columns: repeat(7, 1fr);
  gap: 8px;
}

.dayCell {
  border: 1px solid #ccc;
  padding: 8px;
  min-height: 120px;
}

.dayNumber {
  font-weight: bold;
  margin-bottom: 6px;
}

.slotItem {
  border: 1px solid #ddd;
  padding: 4px;
  margin-bottom: 4px;
  display: flex;
  justify-content: space-between;
  gap: 6px;
}

/* Message styles controlled by className in JavaScript */
.ok {
  border: 1px solid #2e7d32;
  padding: 8px;
}

.error {
  border: 1px solid #c62828;
  padding: 8px;
}

/* Bonus example: highlight today */
.today {
  outline: 3px solid #1976d2;
}
```

Starter file provider.js

This version is intentionally straightforward. It uses nested loops so beginners can follow the logic. Next week you can refactor for elegance.

```
// public/provider.js
// Provider calendar UI for PPA 5
// GET and POST only


let currentMonth = 3;   // 1 to 12
let currentYear  = 2026;


// Run once when the page loads
refreshCalendar();


// Show a user facing message
function showMessage(text, kind) {

  const el       = document.getElementById("message");
  el.textContent = text;
  el.className   = kind;

}


// GET all slots then re render the month view
function refreshCalendar() {

  const xhr = new XMLHttpRequest();
  xhr.open("GET", "/api/slots");

  xhr.onload = function () {

    if (xhr.status === 200) {

      const rawSlots = JSON.parse(xhr.responseText);
      renderCalendar(rawSlots);

    } else {

      showMessage("GET failed " + String(xhr.status), "error");

    }

  };

  xhr.send();
```

```
}


// Render the month grid, then insert slot items into each day cell
function renderCalendar(rawSlots) {

  setMonthTitle(currentMonth, currentYear);

  const grid = document.getElementById("calendarGrid");
  grid.innerHTML = "";

  const firstDay     = new Date(currentYear, currentMonth - 1, 1);
  const startWeekday = firstDay.getDay();  // 0 Sunday to 6 Saturday
  const daysInMonth  = new Date(currentYear, currentMonth, 0).getDate();

  for (let i = 0; i < 42; i += 1) {

    const dayNumber = i - startWeekday + 1;

    const cell     = document.createElement("div");
    cell.className = "dayCell";

    if (dayNumber >= 1 && dayNumber <= daysInMonth) {

      // Day label at the top of the cell
      const label = document.createElement("div");
      label.className   = "dayNumber";
      label.textContent = String(dayNumber);
      cell.appendChild(label);

      // Insert all matching slots for this day
      for (let j = 0; j < rawSlots.length; j += 1) {

        const slot = rawSlots[j];

        // Extract yyyy-mm-dd and compare the day number
        const datePart = slot.startTime.split("T")[0];
        const slotDay  = Number(datePart.split("-")[2]);

        if (slotDay === dayNumber) {

          const item = document.createElement("div");
          item.className = "slotItem";

          // Display just the clock times to keep it readable
          const startClock = slot.startTime.split("T")[1];
          const endClock   = slot.endTime.split("T")[1];

          const text = document.createElement("span");
          text.textContent = startClock + " to " + endClock;

          item.appendChild(text);
          cell.appendChild(item);
```

```
        }

      }

    } else {

      // Cells outside the current month remain empty
      cell.className += " empty";

    }

    grid.appendChild(cell);

  }

}


// Send POST then refresh the calendar on success
function sendCreateSlot(startTime, endTime) {

  const xhr = new XMLHttpRequest();

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

  xhr.open("POST", path);

  xhr.onload = function () {

    if (xhr.status === 201) {

      showMessage("Slot created", "ok");
      refreshCalendar();

    } else {

      const data = JSON.parse(xhr.responseText || "{}");
      showMessage(data.error || "Create failed", "error");

    }

  };

  xhr.send();

}


// Update the month title header
function setMonthTitle(month, year) {
```

```
  const names = [
    "January","February","March","April","May","June",
    "July","August","September","October","November","December"
  ];

  document.getElementById("monthTitle").textContent =
    names[month - 1] + " " + String(year);

}


// Button click creates a slot
document.getElementById("createSlotButton").addEventListener("click",
function () {

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

  sendCreateSlot(startTime, endTime);

});
```

Student tasks

- Confirm `provider.html` loads and `provider.js` runs without console errors
- Render the month grid with correct day numbers
- Place each appointment into the correct day cell
- Send `POST` to create a new slot and refresh the calendar on success
- Add at least one visual improvement using CSS classes, for example style ok and error messages

Rubric

1 point Partial work

- Month grid renders with day numbers
- `GET` triggers a render at least once

2 points Full completion

- Appointments display on correct days after `GET`
- POST creates a slot and calendar refresh shows it
- Code is properly indented and includes meaningful comments

3 points Bonus

- Display a friendly message when inputs are missing
- Highlight today using a CSS class
- Show a small count of slots per day
- Add status styling using CSS classes such as available and booked