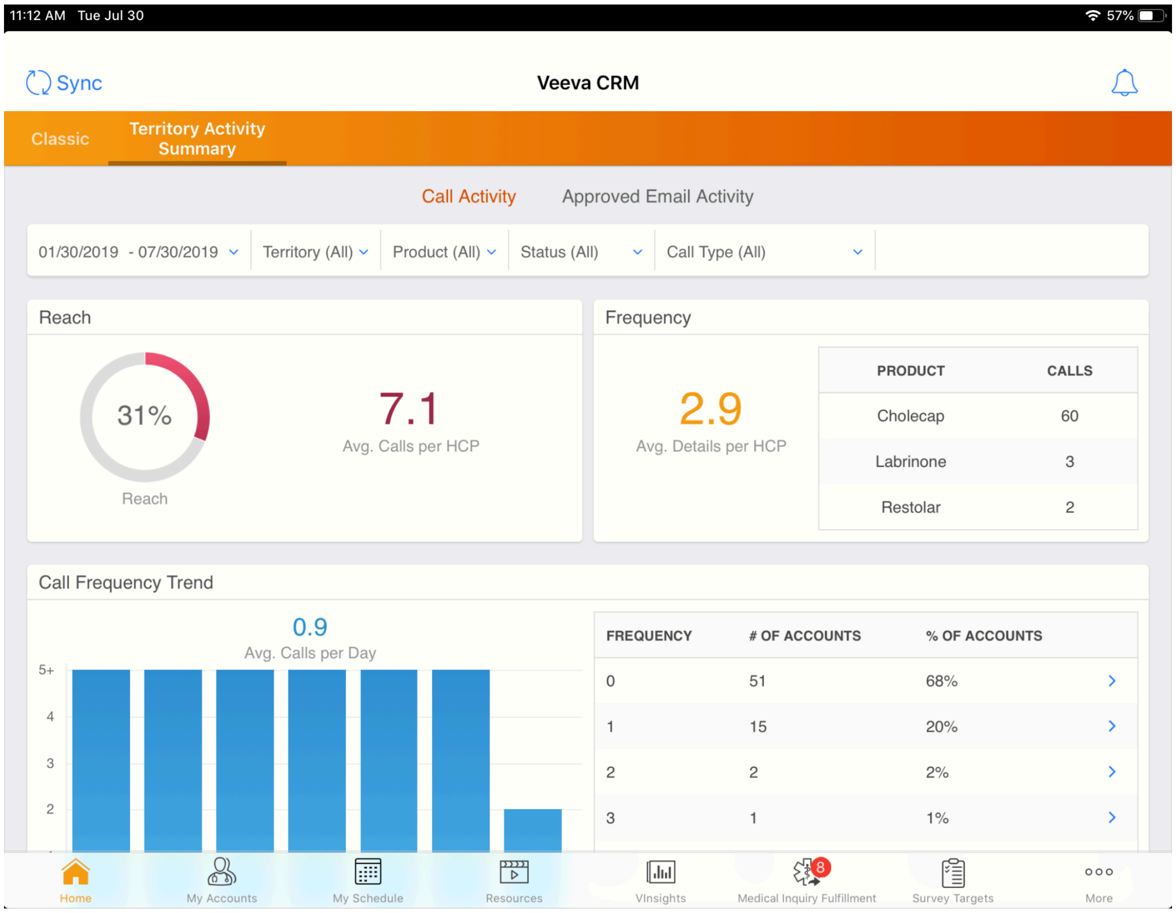
Building Custom Dashboards in Veeva CRM with MyInsights

By InuitionLabs • 3/31/2025 • 20 min read

veeva crm dashboard javascript enterprise-software



PRODUCT	CALLS
Cholecap	60
Labrinone	3
Restolar	2

FREQUENCY	# OF ACCOUNTS	% OF ACCOUNTS
0	51	68%
1	15	20%
2	2	2%
3	1	1%

Building Custom Dashboards in Veeva CRM with MyInsights

Overview: Veeva CRM MyInsights is a powerful data visualization framework within the Veeva CRM ecosystem, allowing developers to create custom, interactive dashboards for end users. This guide provides a deep dive into MyInsights for developers – from understanding what MyInsights is, to its architecture, data access APIs, and a step-by-step tutorial for building a custom dashboard with HTML, CSS, and JavaScript. We'll also cover best practices for dynamic data and how to embed these dashboards in Veeva CRM, along with real-world examples (rep activity dashboards, call summaries, KOL analytics, etc.) and code snippets. By the end, you should have a solid foundation to start developing **custom MyInsights dashboards** that surface the right data at the right time for your users.

What is Veeva MyInsights?

Veeva **MyInsights** is a feature of Veeva CRM that delivers real-time, contextual data visualizations to field teams directly within the CRM interface. In essence, it enables better decision-making by presenting tailored information (both current and historical) inside Veeva CRM, so users don't need to switch to external tools or aggregate data manually (MyInsights Overview). MyInsights dashboards can display virtually any data stored in (or connected to) Veeva CRM – for example, call frequencies, survey results, sales figures – in a user-friendly dashboard format (What is Veeva MyInsights? - Veeva Agency - twentyeightb) (Veeva upgrades its CRM software with MyInsights data visualization). This means **sales reps and managers get on-demand insights** (e.g. pre-call planning info, account trends, KOL profiles) within the same app they use for their daily activities, whether on an iPad or in the desktop browser.

From an ecosystem perspective, MyInsights is **fully integrated into Veeva CRM**. It leverages standard web technologies (HTML, CSS, JavaScript) and a Veeva-provided JavaScript library to query and display CRM data (MyInsights Overview). Veeva provides a library of out-of-the-box dashboard templates that cover common use cases, but customers can also collaborate with developers to build completely custom pages for specific roles or needs (MyInsights Overview). In other words, if the built-in reports don't meet your requirements, you can create tailored dashboards using MyInsights – effectively mini web applications embedded in CRM.

Notably, MyInsights content isn't limited to just CRM's core data. It can pull in data from **Veeva Nitro** (a cloud data warehouse for advanced analytics) and **Veeva Link** (external data sources like KOL databases) to enrich the dashboards (MyInsights Overview). This opens the door to more advanced insights – for example, combining real-time sales data with big data from Nitro or profile data from Veeva Link. (We'll touch on how to query Nitro/Link data later.) Veeva also offers **MyInsights Studio** – a codeless, WYSIWYG editor for building dashboards – but this article will focus on the **custom development approach** using code (MyInsights Overview).

To summarize, **MyInsights fits into the Veeva CRM ecosystem as the go-to solution for custom dashboards and visualizations**, delivering the right information to users when and where they need it (e.g. in an account's page, a call report, or a territory overview) without leaving CRM (MyInsights Overview) (Veeva upgrades its CRM software with MyInsights data visualization). Next, let's explore how MyInsights works under the hood and the components involved in building one.

MyInsights Architecture and Key Components

Building a MyInsights dashboard involves a few Veeva-specific architectural components that differ from a typical standalone web app. Understanding these will help you design and deploy your dashboard correctly:

- **HTML Report Records (Entry Points):** In Veeva CRM, each MyInsights dashboard is stored as an **HTML_Report_vod** record (often just called an "HTML Report"). This record acts as a container for your dashboard content and defines where the dashboard will appear in the CRM UI. Each HTML Report has a *record type* that corresponds to an **entry point** – e.g. Account, Call, Territory, etc. In other words, the record type determines in which context/users will see the dashboard (Creating MyInsights Content). For example, a dashboard with record type **Account_Reports_vod** will be available on the Account detail page (usually as a tab or button), whereas one with **Territory_Insights_vod** appears at a territory or home page level (Components of MyInsights) (Components of MyInsights). Veeva defines a set of these entry point record types (Account, Account Plan, Call, KOL Profile, Territory, etc.) and the MyInsights content is linked to them. Administrators create an HTML_Report_vod record, choose the appropriate record type (entry point) for it, and attach the content (the dashboard files) to that record (Creating MyInsights Content) (Creating MyInsights Content).

- **Content Zip File:** The actual dashboard content – your HTML, CSS, JS, and any assets – must be packaged as a single **zip file**. This zip must contain an **`index.html`** at its root (this is the entry file that Veeva CRM will load) (Creating MyInsights Content). You can include additional `.js` files, `.css` files, images, and libraries in the zip as needed. The key is that the zip's root should have `index.html` (do not nest everything inside an extra folder in the zip) (Creating MyInsights Content). The file size limit for attachments is 32 MB, which is usually plenty for dashboard assets. Once the HTML_Report record is saved, an admin (or you, if you have access) will upload this zip as an attachment (in Lightning, as a File) on that record (Creating MyInsights Content). The next time users sync their devices or refresh their CRM, the new content will be available. Essentially, the **zip deployment model** means your dashboard runs as a self-contained web page inside the CRM app.

- **Veeva JavaScript Library (Data Access Layer):** One of the most important components is Veeva's **JavaScript API library** (accessible via the global `com.veeva.clm` object). This library is what allows your custom HTML/JS page to talk to the **Veeva CRM data** on the device or in the cloud. MyInsights content uses the same JS library used for other Veeva content (like CLM presentations), providing functions to query data from the CRM database (Displaying Dynamic Content), fetch metadata, and even create or update records. For example, the library provides convenience functions like `getDataForCurrentObject` and `getDataForObject` to read field values from Veeva CRM (Displaying Dynamic Content), and `queryRecord` to run queries (similar to SOQL) and retrieve multiple records (Multichannel CRM). We will explore these APIs in detail in the next section. The **Veeva JS library abstracts away whether the user is online or offline** – if the data is available locally (on iPad) it will query the local database; if not, it may fetch from the server. As a developer, you just call the functions and handle the results asynchronously.

- **Offline and Online Architecture:** Veeva CRM is often used on iPads in the field, which means offline availability is crucial. MyInsights is designed with this in mind. When the HTML_Report record and its attached zip are downloaded via a sync, the content can run **offline** within the Veeva CRM iPad app. The data queries via the JS API will hit the local data store on the device (which is kept in sync with the server). On the other hand, if the user is on the desktop (Lightning Experience, online), the MyInsights content can be displayed via a Lightning Web Component container (provided by Veeva) that loads your HTML/JS content. In Lightning, the content might be served from Veeva's content delivery network (e.g. a `*.vod309.com` domain) and displayed in an iframe or similar (Viewing MyInsights Content in Lightning) – hence some extra setup (CSP trusted sites, etc.) is needed by an admin, but conceptually it works the same way. The main difference is **where the content is viewed**: in Veeva's mobile app or inside Salesforce Lightning. As a developer, you usually don't need to change your code for different platforms, but you do have to flag which platforms your content supports (via a field `Platform_vod__c` on the HTML_Report record, e.g.

"Large Mobile Devices", "Lightning", etc.) (Creating MyInsights Content). This ensures, for instance, that a desktop-only dashboard isn't synced to iPad, or vice versa.

- **Security and Access:** The MyInsights architecture respects Veeva CRM's security model. Users will only see data they have permission to see. If your dashboard queries a field or object the user doesn't have access to, the result will simply come back empty or without that field. Admins must grant appropriate object and field permissions for any data your dashboard needs. (For example, many out-of-the-box dashboards require read access to certain fields like product metrics, call details, etc., which need to be opened up to end users.) Keep this in mind during development – you may need to coordinate with your admin to ensure the relevant fields are visible to the target users.

In summary, **the architecture of a MyInsights dashboard** consists of an HTML_Report CRM record (entry point) which holds a packaged HTML/JS/CSS bundle. This bundle is rendered in the Veeva CRM app (or Lightning UI), and uses Veeva's JS APIs to pull data from the CRM data model in real-time. The next sections will cover how to work with that data model via the APIs and then walk through building an example dashboard.

Working with Veeva CRM Data Models and APIs

One of the biggest challenges (and opportunities) in custom MyInsights development is **querying and handling Veeva CRM data**. Veeva CRM is built on the Salesforce platform, so the data model will include standard Salesforce objects (Account, Contact, Opportunity, etc.) as well as Veeva-specific objects and fields. Veeva's convention is often to suffix custom objects/fields with "_vod__c" (vod = Veeva object/data). For example, call records in Veeva CRM use the **Call2_vod__c** object (which extends the Activity concept for CRM calls), and that object has fields like Call_Date_vod__c, Call_Type_vod__c, etc. An Account's profile might have additional fields like Specialty_1_vod__c. As a developer, you'll want to familiarize yourself with the specific objects and fields relevant to your dashboard (your business analysts or admin can provide the data model details). Once you know what to query, you will use the **Veeva JS API** to retrieve that data.

Veeva CRM JavaScript API Overview: The `com.veeva.clm` library exposes a set of functions to read (and write) CRM data. These calls are all asynchronous – you provide a callback function to handle the result when it returns. Here are some of the most commonly used functions for MyInsights:

- **`getDataForCurrentObject(object, field, callback)`** – This retrieves a field value for the "current" record of a given object type (Multichannel CRM) (Multichannel CRM). The "current object" depends on context. For MyInsights, if your dashboard is opened from an Account, then `getDataForCurrentObject("Account", "Name", callback)` would fetch the Name of that account. If it's opened in a Call report context, `"Call"` could fetch the current call's fields, etc. Veeva restricts the object names here to certain keywords: e.g. `"Account"`, `"Call"`, `"User"`, `"TSF"` (Territory Sales Force), etc. The result passed to the callback will contain an object with the value. For example: `result.Account.Name` might equal "Dr. Alice Smith" (if success is true).

- **`getDataForObject(object, recordId, field, callback)`** – This retrieves a field value from a specific record (by ID) of a given object (Multichannel CRM) (Multichannel CRM). You'll use this if you already have a record ID and want another field from that record. For example, if you have an Account's Id, you could get its Phone or Address via this function. It's like a one-record SOQL query (SELECT field FROM Object WHERE Id = '...'). The result will be similar to above, e.g. `result.Account.Phone` would have the value.

- **`queryRecord(object, fields, where, sort, limit, callback)`** – This is the most powerful function for dashboards, as it lets you query multiple records with filters (akin to a SOQL query). You specify the object API name, an array of fields (or a comma-separated string of field API names) to retrieve, an optional WHERE clause (in Salesforce SOQL syntax without the leading "WHERE"), optional sort order, and limit (Multichannel CRM) (Multichannel CRM). The library then returns an array of records matching that query. For example, you could query all Call2_vod__c where AccountId = X and Status = "Completed" to get all past calls for an account. The result in the callback might have a structure like `result.Call2_vod__c` as an array of objects, each with the fields you requested. (Also included is a `result.count` typically, indicating how many records were returned.) The query syntax supports basic operators and AND/OR logic (Multichannel CRM), much like a normal SOQL, but be mindful of not making the query too complex – stick to standard filters and joins (you generally can't join objects in one call except through the defined relationship functions Veeva provides). If you need related data (e.g. related child records), you may need to perform multiple queries in sequence.

- **Other Useful Functions:** The API includes functions like `getFieldLabel(object, fields, callback)` to get the display labels of fields (useful if you want to show field names dynamically in the user's language) (Multichannel CRM) (Multichannel CRM). It also provides the ability to modify data (`createRecord` , `updateRecord` , etc.), though those are less commonly used in dashboards focused on visualization. There are specialized convenience methods as well (for example, methods to get list of products or surveys associated to an account, used in CLM context). For the most part, **MyInsights dashboards stick to read-only data queries**, but it's good to know the library can do more if needed.

Callback Pattern: Each API call takes a `callback(result)` function. The `result` object will contain `result.success = true/false` . If true, it will also contain a property named after the object you queried, with the data. If false, it may contain an `error` message. Always check `result.success` in your callback. You may also want to handle the case where `success:true` but the data array is empty (no records found).

Example – Querying CRM Data: Suppose we want to retrieve the last 5 completed Calls for the current Account and display them. We can do something like:

```
// Step 1: Get the current Account’s Id
com.veeva.clm.getDataForCurrentObject("Account", "Id", fun
    if (res.success) {
        let accountId = res.Account.Id;
        // Step 2: Query Call records for that Account
        let fields = ["Call_Date_vod__c", "Call_Type_vod__
        let whereClause = "Status_vod__c = 'Completed' AND
        let sortClause = ["Call_Date_vod__c, DESC"];
        let limitCount = "5";
        com.veeva.clm.queryRecord("Call2_vod__c", fields,
            if (result.success) {
                // result.Call2_vod__c is an array of up t
                displayCalls(result.Call2_vod__c);
            } else {
                console.error("Query failed: " + result.er
            }
        });
    } else {
        console.error("Failed to get Account ID");
    }
});
```

In this snippet, we first retrieve the Account Id using `getDataForCurrentObject`. Then we construct a query for Call2_vod__c records associated with that Account, filtering to only completed calls, sorting by date (descending), and limiting to 5 results. The resulting array is passed to a custom function `displayCalls` which would handle rendering the data (more on rendering later). This illustrates the general approach: **chain your data calls logically**. You can nest callbacks as shown, or use modern techniques (like wrapping these calls in Promises or using async/await with a promise wrapper) to avoid “callback hell.” Since the Veeva API is asynchronous, your code must be structured to work with the data only after it arrives (inside the callback).

Working with the Data Model: When writing queries, use the **API names** of objects and fields. For standard objects (Account, Contact, etc.), those are just “Account”, “Contact” etc. For Veeva custom objects, it could be something like “Call2_vod__c”, “Medical_Event_vod__c” etc. Field API names often end with “__c” for custom fields or have “_vod__c” if provided by Veeva. You can consult the Veeva CRM data dictionary or your org’s schema for exact names. For example, the account field that links a Call to an Account might be `Account_vod__c` on Call2_vod__c. Using the wrong API name will simply return nothing. Also, note that Veeva’s API sometimes expects you to use specific context objects for `getDataForCurrentObject` (like “Account” or “Call” as strings, not the full API name with __c). The documentation indicates which keywords are allowed (Multichannel CRM).

Advanced Data – Nitro and External Sources: If your org has Veeva Nitro (a data lake) or uses Veeva Link, MyInsights can leverage those as well. Veeva has methods like `queryVDSRecord` (VDS = Veeva Data Services, Nitro) to fetch data from Nitro directly into the dashboard (Viewing MyInsights Content in Lightning). For example, a Nitro dataset of prescription data could be queried to show trends for a product. Similarly, data from Veeva Link (such as KOL profiles) can be surfaced via Link’s APIs within MyInsights. These are advanced use-cases and typically require additional setup (e.g. ensuring the user is online, proper API keys, etc.). For most custom dashboards focusing on CRM data, you will primarily use the core CRM data via the JS library as shown above. Just be aware that MyInsights isn’t limited to the CRM database; with the appropriate connectors, it can be a window into broader data as well (Link) (Link).

Now that we’ve covered how to get data, let’s walk through building a custom dashboard step by step – putting everything together (HTML/CSS/JS, data queries, and deployment).

Step-by-Step Guide: Building a Custom MyInsights Dashboard

In this section, we'll outline a step-by-step approach to create a custom MyInsights dashboard. This will be a generic guide that you can adapt to your specific use case. We assume you have some idea of what you want to build (e.g. a dashboard showing certain metrics) and you have access to a Veeva CRM sandbox or org to upload and test your content.

Step 1: Define the Dashboard's Purpose and Entry Point
Before coding, clarify **what information the dashboard will show and where it will live in CRM**. Is this a dashboard for a specific Account (e.g. sales and interaction summary for one account)? Or is it a territory-level overview for a sales rep (aggregated data across many accounts)? Or perhaps a Key Opinion Leader profile page? The answer determines the **entry point (record type)** you'll use and how users will access it. For example:

- If it's account-specific, you might choose the *Account_Reports_vod* entry point so that the dashboard can be opened from the Account detail screen.
- If it's meant to be a rep's home dashboard, you could use *Territory_Insights_vod* (or Territory_Insights_Default_vod if you want it as a default landing screen).
- For something that appears on a Call report (during call entry), use *Call2_vod*. Knowing the entry point guides your design because it tells you what **context data** is readily available. (E.g., on an Account page, the "current object" is Account; on a call report, the current object is Call.) Also consider what *questions* the dashboard will help answer for the user – this will drive what data you need to fetch. Sketch out the charts or metrics you want. For our example, let's say we want an "Account Summary Dashboard" that a rep can view on an account page to see recent activity (calls, key messages, etc.) and maybe some sales numbers for that account.

Step 2: Set Up Your Project – HTML, CSS, JS Files

Create a working directory on your computer for the dashboard. At minimum, you will need an `index.html` file. This will be the main file loaded by MyInsights. You can also create separate CSS and JavaScript files if you prefer, or just embed styles and script in the HTML. For organization, let's create:

- `index.html` – the HTML structure of the dashboard.
- `style.css` – custom styles (optional, you can also use a CSS framework if it's not too large and you include it in the zip).
- `main.js` – your JavaScript code for data retrieval and rendering. Make sure to reference `style.css` and `main.js` in your HTML (as local files). For example, in the head of your HTML: `<link rel="stylesheet" href="style.css" />` and at the bottom of body: `<script src="main.js"></script>`. **Do not include any hard-coded absolute URLs to these files**; since they will be in the zip together, just referencing by filename works. If you plan to use external libraries like Chart.js or D3.js for charts, you have two options: either include their files in your zip (and reference them via a script tag), or use a CDN link. However, note that if your dashboard is to work offline, a CDN link (which requires internet) may not be available – so it's often safer to include any library scripts locally in the zip. Keep an eye on file size though; include only what you need.

Now, set up a basic HTML skeleton in `index.html`. For example:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8" />
  <title>Account Summary Dashboard</title>
  <meta name="viewport" content="width=device-width, initi
  <link rel="stylesheet" href="style.css" />
</head>
<body>
```

```
  <div class="dashboard-container">
    <h2 id="accountName">Account: <!-- Account Name will g
    <div id="metrics">
      <!-- dynamic content (charts, tables) will be inject
    </div>
  </div>
  <script src="main.js"></script>
</body>
</html>
```

This is a simple starting point. We include the doctype and basic meta (important: Veeva recommends including the `<!DOCTYPE html>` for standards mode (Viewing MyInsights Content in Lightning)). The body has a container, a header for the account name, and a placeholder div for metrics. In `style.css`, you might add styles for `.dashboard-container` and other elements (fonts, colors, etc.) to make it look nice and perhaps responsive. Keep CSS simple and avoid using any external fonts unless they're critical (again, offline considerations).

Step 3: Initialize and Test Basic Page Structure
At this stage, it's good to ensure your page loads and the basic elements appear. You can actually test the static HTML in a normal web browser *to some extent*. Open `index.html` in a browser; you should see your header and empty metrics section. Obviously, it won't show real data yet, but you can at least verify your HTML and CSS are working (and fix any typos in paths). Note that any Veeva-specific JS calls (like `com.veeva.clm.getDataForCurrentObject`) **will not work in a regular browser** (they only work in the Veeva CRM context). So for now, don't call them until you're in CRM.

However, you can stub out some script functions in `main.js` to test layout. For instance, you can write a dummy function to populate #metrics with some placeholder text or chart. This way, you can iterate on layout/design without needing to constantly deploy to CRM. Once you're happy with the layout, you'll integrate the actual data calls.

Step 4: Retrieve Data using Veeva APIs

Now it’s time to pull in the real data. In `main.js`, you will use the Veeva JS library functions discussed earlier to get data. Remember, these will only execute properly when running inside Veeva CRM (e.g. on the iPad app or Lightning container) where the `com.veeva.clm` object is available. A common pattern is to trigger your data load when the page is ready. Since this is a simple page, you can call a function at the bottom of your script file or use `window.onload`. For example:

```
// main.js
function loadDashboardData() {
    // 1. Get current Account Name (for header)
    com.veeva.clm.getDataForCurrentObject("Account", "Name
        if(res.success) {
            document.getElementById("accountName").innerTe
        }
    });
    // 2. Get recent Calls for this account
    com.veeva.clm.getDataForCurrentObject("Account", "Id",
        if(res.success) {
            let acctId = res.Account.Id;
            let fields = ["Call_Date_vod__c", "Call_Type_v
            let where = "Status_vod__c = 'Completed' AND A
            com.veeva.clm.queryRecord("Call2_vod__c", fiel
                if(result.success) {
                    let calls = result.Call2_vod__c;
                    renderCallsTable(calls);
                }
            });
        }
    });
}
```

```
// Kick off data load when page is ready:
loadDashboardData();
```

In this snippet, `loadDashboardData()` does two things: first, it gets the Account’s Name and injects it into the page (`#accountName` element). Second, it gets the Account’s Id and uses it to query the last 5 completed calls (as in the earlier example). We then call a `renderCallsTable` function with the results. You would define `renderCallsTable(calls)` to process that array of call records and display them. For instance, you could build an HTML table listing call date, type, etc., or perhaps compute some summary (like count of CLM presentations shown). For brevity, imagine `renderCallsTable` creates a small HTML table:

```
function renderCallsTable(calls) {
    if (!calls || calls.length === 0) {
        document.getElementById("metrics").innerHTML = "<p
        return;
    }
    let html = "<h3>Recent Calls</h3><table><tr><th>Date</
    for (let c of calls) {
        let date = c.Call_Date_vod__c ? new Date(c.Call_Da
        let type = c.Call_Type_vod__c || "";
        let channel = c.CLM_vod__c ? "CLM" : "Standard"; /
        html += `<tr><td>${date}</td><td>${type}</td><td>$
    }
    html += "</table>";
    document.getElementById("metrics").innerHTML = html;
}
```

This would output a simple table under the “Recent Calls” heading. You can of course make the rendering more elaborate – perhaps show a chart (e.g., number of calls per month), or list key message usage, etc. The principle is the same: **fetch data via the API, then dynamically update the DOM with that data**.

While developing, it's helpful to use `console.log` or `alert` to debug data shapes. On an iPad, you won't have a console readily visible, but if you're using Lightning Experience to test, you can open the browser dev tools console to see logs. Alternatively, you can temporarily add a `<textarea>` to your HTML and dump JSON into it for debugging.

Step 5: Implement Dynamic Visualizations (Charts, etc.)

Now that basic data retrieval is working, you can enhance the dashboard with more visuals. MyInsights pages often include charts (bar graphs, pie charts) or KPIs. You can manually construct charts using HTML/CSS (for example, a simple bar chart using div widths), but using a library like **Chart.js** can save time. If you choose to use Chart.js, download the library and include `chart.min.js` in your zip, then add a `<canvas>` element in your HTML where the chart should render. For example, to show a pie chart of call types in the last year, you might gather counts of each type from your calls data, then use Chart.js:

```
// Assuming Chart.js is loaded and a <canvas id="callChart
function renderCallTypeChart(calls) {
    let typeCounts = {};
    calls.forEach(c => {
        let type = c.Call_Type_vod__c || "Other";
        typeCounts[type] = (typeCounts[type] || 0) + 1;
    });
    // Prepare data for chart
    const labels = Object.keys(typeCounts);
    const data = Object.values(typeCounts);
    const ctx = document.getElementById('callChart').getCo
    new Chart(ctx, {
        type: 'pie',
        data: {
            labels: labels,
            datasets: [{ data: data, backgroundColor: ['#3
        },
        options: { title: { display: true, text: 'Call Typ
    });
}
```

This would draw a pie chart of call types with some default colors. The key is to ensure the Chart.js script is available (include it before your main.js in the HTML). The specifics of charting are beyond our scope, but many MyInsights developers leverage such libraries for slick visuals. Keep in mind file size and complexity – don't overload the page with massive libraries or too many heavy charts, as it may affect performance on an iPad.

Step 6: Test in Veeva CRM
Now that your dashboard code is ready, it's crucial to test it inside a Veeva CRM environment. The process is typically:

1. **Zip up your content** – select the files (index.html, main.js, style.css, and any others) and zip *the files* (not the containing folder). You should get a zip file where index.html is at the root inside the zip. (As noted earlier, only one zip attachment is supported per HTML Report record (Creating MyInsights Content), and compressing the folder itself rather than just the files can lead to an extra directory layer that will break the content – so compress the files directly).
2. **Upload to an HTML_Report_vod record** – in your Veeva CRM sandbox or developer org, go to the **HTML Reports** tab (or ask your admin to). Create a new record with the appropriate record type (entry point) you decided in Step 1 (Creating MyInsights Content). Give it a name (e.g. "Account Summary Dashboard"). Choose the Platform (e.g. Large Mobile Devices if for iPad, or Lightning if you also want it online) (Creating MyInsights Content). Save, then attach the zip file to this record (via Notes & Attachments or Files, depending on setup) (Creating MyInsights Content).
3. **Sync or Publish** – If you're testing on an iPad CRM app, perform a sync so the new content is downloaded. If testing in Lightning, ensure the content is marked for Lightning (Platform_vod = Lightning) and possibly use the *Publish* button (for Lightning content, some orgs require hitting "Publish" on

the HTML_Report record to make it live (Enabling MyInsights Custom Account Visualizations)).

4. **View the Dashboard** – Navigate to the entry point to see your dashboard. For example, if it's account-level, go to an account's page. In Veeva CRM on iPad, a tab or button (often labeled "Insights" or the name of the report) should be visible (in the account's Sunrise bar or details area) – tapping it will open your MyInsights page. In Lightning, if configured, the dashboard might appear as a component on the account page or as a tab. (Lightning embedding requires an admin to add the *MyInsights LWC* to the Lightning record page layout and select your report from a dropdown (Viewing MyInsights Content in Lightning) (Viewing MyInsights Content in Lightning), but if you have system admin access, you can do this via the Lightning App Builder). Once you open the page, observe if all data loads correctly.

During testing, you might find issues like blank sections or errors. Common causes include:

- Mistyped object or field API names in queries (leading to no data returned).
- Forgetting to grant the user read access to certain fields/objects – if data isn't coming through, check Field Level Security and Object permissions for the user profile on those fields.
- JavaScript errors – if something is wrong in your JS logic, parts of the script might not run. Using `console.log` and having the browser console open (in Lightning) helps. On iPad, you might use the **Console (beta)** in Veeva CRM (there's a feature to view console logs on the iPad app if enabled by support).
- Case sensitivity or missing the `DOCTYPE` causing quirks mode (always include `<!DOCTYPE html>` to avoid HTML/CSS issues (Viewing MyInsights Content in Lightning)).

Iterate by fixing issues, re-zipping, re-uploading (replace the attachment with your new zip), and re-testing. It can be a bit of a cycle, but it ensures your dashboard works in the real environment.

Step 7: Deploy to Users
Once you are satisfied with the dashboard in your dev/test environment, deploying to end users involves promoting the HTML_Report record and content to production (if you built in a sandbox) and making sure profiles have access. Your Veeva CRM admin will likely handle this part, but in summary: they would recreate or deploy the HTML_Report_vod record (via metadata deployment or manually), upload the zip, and then expose it to users (e.g. adding the button/tab for it, or in Lightning adding the component to the page). End users might need to sync their devices to get the latest content. One great aspect of MyInsights is that **if you update the dashboard code**, you can simply replace the zip attachment on the HTML Report record, and users will get the new version on next sync (or near real-time if online). No app update required – it's very agile.

That's it! You have built and deployed a custom MyInsights dashboard. Next, we'll discuss some best practices to keep your dashboards efficient and user-friendly.

Best Practices for Dynamic MyInsights Dashboards

When developing MyInsights content, keep these best practices in mind to ensure performance and maintainability:

- **Optimize Data Retrieval:** Query only the data you need for the dashboard. Avoid unbounded queries that could return hundreds of records unless absolutely necessary. Use filters (`WHERE` clauses) to narrow down results (e.g., last 1 year of data, or specific status flags). Also use the `limit` parameter in `queryRecord` if you only need top N records (Multichannel CRM). This prevents slowing down the dashboard or exhausting device memory with too much data. If you need to show large data sets (e.g., thousands of records or very heavy analytics), consider moving that logic to

Veeva Nitro or an external system that pre-aggregates data, and then fetch only summary results into MyInsights.

- **Use Asynchronous Calls Wisely:** The JS API is asynchronous; if you need to do multiple queries (and you often will), chain them logically as shown, or use modern JS patterns to avoid deeply nested callbacks. Always handle the callback results – check for `result.success` and handle errors (maybe display an error message or at least fail silently). Do not assume data is there without checking, or your page might try to read undefined properties and throw errors.

- **Leverage the Context:** Use `getDataForCurrentObject` for quick access to context like the current record's info (Account name, etc.) (Displaying Dynamic Content). It's efficient and saves you from doing a full query when you just need one field from the current record. Also, consider using the built-in **relationship functions** if available. (For example, Veeva might have special APIs like `getChildObjects` or specific methods to retrieve related records in one go – check the developer documentation for any that apply to your scenario.)

- **Keep the User Interface Responsive:** Avoid doing heavy computations in the JavaScript on the main thread that freeze the UI. Because data calls are async, you usually won't freeze the UI during fetch, but if you process a lot of data, consider splitting into chunks or giving feedback. It's good UX to provide a **loading indicator** when the dashboard is fetching data. For example, show a spinner or a "Loading data..." message in your #metrics div before the data arrives, then remove it. This way the user knows something is happening. If the dashboard is complex and data is slow (maybe Nitro queries can take a second or two), this feedback is important.

- **Use the Latest JS Library:** Veeva occasionally updates the MyInsights JavaScript library to improve performance and fix issues. Ensure your org is using the **MyInsights v2.0 library** (which has better performance especially when multiple dashboards are used). Using outdated libraries can degrade query performance and even lead to incorrect data in certain scenarios (Viewing MyInsights Content in Lightning). In practice, this means if your org upgraded CRM, make sure to also update any "veeva library static resource" if that was needed. (Often, Veeva provides a file to include for older content. Check Veeva release notes for MyInsights v2 library.)

- **Design for Multiple Platforms:** If your dashboard will be used on iPads and in desktop Lightning, test in both. The rendering might differ due to screen size or Salesforce styling on Lightning. Ensure your CSS is robust (use relative sizes, flexible layouts). Veeva CRM on iPad uses a Safari webview, so generally if it works on a modern WebKit browser it will work there. In Lightning, your content might be embedded inside a container with limited height – you might need to make the container scrollable or adjust the LWC height setting (admins can configure the MyInsights LWC height or you can make your content height dynamic). If something should only be used on one platform, use the Platform_vod field and/or separate HTML_Report records to target different platforms.

- **Avoid Platform-Specific Code:** Related to the above, try not to rely on any browser-specific or platform-specific hacks. The same content should run on both iOS and desktop if needed. In fact, Veeva notes that "MyInsights content using platform-specific syntax is not supported" (Creating MyInsights Content). Stick to standard HTML5/JS/CSS that works cross-platform. If you have to detect platform (perhaps to adjust styling), use feature detection or the user agent as a last resort.

- **Include Images/Assets Appropriately:** If your dashboard needs logos or images, include them in the zip and reference them by relative path. There's no guarantee of internet connectivity, so don't hotlink images from an external site. Also, optimize images for size since large images can bloat the zip.

- **Security and Compliance:** (While we won't focus on validation/compliance here, one best practice note:) If your dashboard displays sensitive data, ensure it's pulling from the right fields and that those fields aren't free-text fields containing inappropriate content. Also, MyInsights pages can't easily be restricted by record beyond what the record type entry provides and the user's data visibility. If certain data should only be shown to certain roles, you may need to create separate HTML_Report content with profile-based visibility. Work with your admin to set up appropriate visibility rules (there are profile and group fields on the HTML_Report object to control who sees it).

- **Performance Testing:** Try the dashboard with real data volumes and a typical network scenario. If reps are offline, test it offline. If online, test with a slow connection to see if any part is sluggish. The goal is to have the dashboard load within a few seconds at most and be interactive. If you find a query is slow, consider if you can reduce the data or move logic to a background process (maybe precompute something and store results in a field that MyInsights just reads).

By following these best practices, you'll create dashboards that are efficient, maintainable, and provide real value to the end users without hiccups.

Embedding and Accessing MyInsights Dashboards in Veeva CRM

We've touched on this in bits earlier, but it's worth clarifying how users actually access these custom dashboards once deployed, and what "embedding" entails:

- **Mobile (iPad, Windows Tablet)** – In the Veeva CRM mobile app, MyInsights content is usually accessed via a tab or button in the UI. For example, on an iPad, when viewing an Account, there may be a tab in the account's horizontal menu (Sunrise bar) for "Account Reports" or whatever label the admin gave to the HTML Report (Creating MyInsights Content). Tapping that will load the MyInsights page (your HTML content) in a full-screen view within the app. Some entry points behave slightly differently: e.g., a Call report MyInsights (Call2_vod) might show as an embedded section within the call editing screen rather than a separate page. For instance, Veeva

CRM supports adding a MyInsights **widget on the Call Report page** itself, showing custom content while the user is filling out the call (in Edit mode) (Displaying MyInsights Content on the Call Report) (Displaying MyInsights Content on the Call Report). In such cases, the admin adds a special field (zvod_MyInsights_Widget_vod) to the page layout which serves as a placeholder for the MyInsights content (Displaying MyInsights Content on the Call Report). The takeaway is that on mobile, MyInsights either appears as a tab/page or as an embedded section, depending on configuration. As a developer, you don't have to change anything for one or the other – it's purely admin configuration how it's presented.

- **Online (Salesforce Lightning)** – In Lightning Experience (desktop browser), MyInsights dashboards can be embedded into Lightning pages via a provided Lightning Web Component. Veeva delivers a component (named something like "MyInsights" or specifically *myInsights V2 LWC*) which admins can drag onto a Lightning record page (Account, Contact, etc.) (Viewing MyInsights Content in Lightning). Once added, the admin can either configure it to show a specific HTML Report or leave it generic to show all available MyInsights for that context (Viewing MyInsights Content in Lightning). For example, on an Account Lightning page, the admin might add the component and fix it to the "Account Sales Dashboard" report. Then whenever a user views an Account, that component loads your content for that account. Alternatively, if left generic, it might show a menu of all MyInsights pages for that account record type. The user can interact with the dashboard just as they would on iPad (the content is the same). The Lightning container ensures the content is only visible to those with permission and appropriate record context. Under the hood, if your content is marked for Lightning (Platform_vod field includes Lightning), it won't sync to devices but will be available to the LWC which fetches it from the cloud. Some Salesforce security settings (Content Security Policy, CORS) might need adjustment by the admin (as indicated by Veeva's documentation) to allow the content to load (Viewing MyInsights Content in Lightning), but that's a one-time org setup. In short, **embedding in Lightning** means your dashboard can appear alongside standard Salesforce components on a user's screen (Viewing MyInsights Content in Lightning) – for example, a territory dashboard on the Home page, or a profile visualization on an Account page – making it seamlessly part of the CRM workflow.

- **Multiple Dashboards and Navigation:** If a user has multiple MyInsights available in one place, how do they choose? On mobile, if multiple HTML Reports share the same entry point (say two different Account-level dashboards), the UI might list both as separate tabs or menu items. In Lightning, as mentioned, the MyInsights LWC can list all available reports if not configured to a single one (Viewing MyInsights Content in Lightning). It's generally a good idea to not overload any single entry point with too many different dashboards – it could confuse users. Instead, consolidate or use separate entry points if logically distinct. Also note, you can hyperlink between MyInsights pages or out to other CRM pages using URL schemes or deep links (for advanced use – e.g., clicking a data point could navigate the user to a related record or different MyInsights page). Veeva even supports a concept of deep linking into CRM from MyInsights (Deep Linking in MyInsights - Veeva CRM Help), which can enable some nifty interactions (like jumping to a record detail, or pre-populating a new record form from the dashboard context).

- **Refreshing Data:** MyInsights content doesn't auto-refresh in real-time; it shows data as of the user's last sync (for offline) or last page load (for online). If data changes (like a new call is logged) and the user wants to see it, they would typically exit and re-enter the page or perform a manual sync (offline). You could provide a "Refresh" button in your dashboard that re-runs the queries (just call your load function again) – this can be helpful especially for online use where new data might have come in. Just be cautious to not spam queries (don't refresh endlessly in a loop).

Embedding is largely about how the user sees the content. As a developer, **your focus is to ensure the dashboard behaves well in the embedded context** (e.g., sizing and not breaking the page layout). Test the final user experience: if in Lightning, does your dashboard fit in the allocated space? If it's too tall, maybe instruct admin to give it a full tab or increase height. If on iPad, does it handle rotations and different screen sizes (if supporting Windows tablet which might be different aspect ratio)? Using flexible CSS (like width:100% on tables, etc.) can help.

To wrap up this section: once deployed, MyInsights dashboards become a natural part of the Veeva CRM interface. Whether it's a rep checking an Account dashboard before a call, or a manager reviewing a territory summary on the web, the goal is that the custom dashboard feels integrated. Veeva's platform handles the heavy lifting of delivering and displaying your content in the right place (Creating MyInsights Content) (Viewing MyInsights Content in Lightning).

Now, let's look at some **examples and use cases** of MyInsights dashboards to spark inspiration and see how these pieces come together in real scenarios.

Examples and Use Cases of Custom MyInsights Dashboards

(Displaying MyInsights Content on the Call Report) *Example: MyInsights dashboard embedded in a Call Report, showing key metrics for each attendee (HCP) to guide the conversation during a group call.* MyInsights pages can take many forms depending on business needs. They might show a sales rep all the recent interactions and key messages for an account (as in the call report example above, where each attendee's last call, key messages, and insights are listed), or provide high-level performance metrics on a rep's home screen. Here are some common use cases:

- **Territory/Rep Activity Dashboard:** A territory-level dashboard that gives a rep an overview of their recent activity and sales performance across all accounts in their territory. For example, Veeva's out-of-the-box *Territory Insights* page shows operational and historical data like call activity summaries and Approved Email usage, helping the rep gauge reach and frequency of their engagements (Using the Veeva-Provided Custom MyInsights Territory Insights Page). Custom territory dashboards might include charts for total calls this quarter vs target, top products detailed, or a leaderboard of top accounts by sales. The entry point for this is typically *Territory_Insights_vod* (accessible from the Home page or a global Insights tab). This empowers reps and managers to do territory planning and tracking without leaving CRM.

- **Call Interaction Summary (Account 360° View):** An account-centric dashboard that summarizes all key interactions and data for a given account. This could be used for pre-call planning – before a meeting, the rep opens this dashboard to see the last call date and notes, which key messages were shown, what samples were given, recent order volumes, etc., for that account. Veeva provides an example *Interaction Summary* page which displays call interactions, key message usage, and sample drops for an account (Using the Veeva-Provided Custom MyInsights Interaction Summary Page). Your custom version might also pull in data like open service requests or pending medical inquiries for a full 360° view. This usually uses the *Account_Reports_vod* entry point (on the Account detail). By having all relevant info on one screen, the rep can quickly tailor the upcoming discussion with the HCP. (In the screenshot above, you can see a variant used in a Call Report context for group calls – showing insights for each attendee, which is an extension of the same idea to multiple accounts at once.)
- **KOL Profile Analytics:** For medical liaisons or sales reps who engage with Key Opinion Leaders (KOLs), a KOL profile dashboard provides deep insight into that stakeholder. This might include the KOL's background (e.g. specialty, institution, credentials), their engagement history (events attended, scientific inquiries made, publications if linked, etc.), and influence metrics. The MyInsights *KOL Profile* page, for example, displays information about a stakeholder's career stats, recent interactions (emails, inquiries), and medical event activity (Using the Veeva-Provided Custom MyInsights KOL Profile Page). This can be enriched with data from Veeva Link (which might provide things like publications or social media activity of the KOL). Such a page is usually tied to *KOL_Profile_vod* entry point and would appear when viewing an account that is designated as a KOL. It helps

the user approach interactions with KOLs more strategically, armed with all relevant info.

- **Product Sales and Ordering Dashboard:** Another common use case is a product-centric dashboard for an account or territory. For example, an Account Sales dashboard might show sales trends for key products in that account over time, current year-to-date sales vs last year, and maybe an order history. If your org uses Veeva Orders, you could query order data to populate this. This kind of dashboard could be used by a rep before a sales call to review buying patterns. It might also be used by a manager to spot which products are lagging. Depending on scope, this might be an Account-level report (focused on that account's numbers) or a territory-level (aggregated). Chart.js comes in handy here to plot trends (line charts for sales over months, etc.). Make sure to aggregate data either via queries (if the dataset is small) or precompute if large (maybe Nitro if thousands of rows).

- **Coaching or Field Insight Dashboard:** Some organizations create MyInsights for managers to review rep performance or for reps to do self-coaching. For example, a manager could have a dashboard showing call quality metrics for each rep, or a rep could see their call execution vs peers. These are more custom but highlight that MyInsights isn't just for HCP-facing data; it can be rep-facing as well (since the *User* object is accessible, you can get the current rep's info and build a personalized dashboard for them).

Each of these examples uses the same building blocks we discussed: querying the relevant objects (Calls, Accounts, Orders, etc.) and presenting the data in an intuitive format (tables, charts, KPIs). Veeva has provided many of these as templates which can be downloaded and studied (Using Veeva-Provided Custom MyInsights Pages). If you're new, it might be worthwhile to obtain some of the Veeva-provided MyInsights pages (like those mentioned: Account Plan, Inventory Monitoring, etc.) and see how they implemented certain features. They serve as technical examples and can be further customized (Using Veeva-Provided Custom MyInsights Pages).

Finally, always design with the end-user in mind: what decisions or actions should this dashboard inform? Keeping the dashboard focused and not cluttered will ensure it's used and loved by the field.

Resources and Further Reading

For more detailed information and official documentation on Veeva MyInsights and related development, check out the following resources:

- **Veeva CRM Help – MyInsights Overview:** Official help documentation providing an overview of MyInsights, its capabilities, and admin tasks (MyInsights Overview) (MyInsights Overview). (Navigate the Veeva CRM Help site for subtopics on MyInsights configuration, best practices, etc.)

- **Veeva CRM Developer Guide (JavaScript Library Reference):** The developer documentation for the Veeva JavaScript Library (available on Veeva's developer website) details all functions (e.g. `queryRecord`, `getDataForObject`, etc.) and their usage (Multichannel CRM) (Displaying Dynamic Content). This is extremely useful when you need to find what functions are available and expected parameters. *(Requires login to Veeva developer community in some cases.)*

- **MyInsights Studio Guide:** If you are interested in the no-code approach or using custom components within MyInsights, the MyInsights Studio documentation on Veeva CRM Help describes how to use the Studio UI and even how to create **Custom Display Elements (CDEs)** (which are like web components for reuse in Studio) (GitHub - veeva/CRM-MyInsights-CDE-Examples) (GitHub - veeva/CRM-MyInsights-CDE-Examples). Advanced developers might create CDEs with code that can be handed off to content creators who use Studio.

- **Veeva Nitro and Link Integration:** To learn about pulling Nitro data into MyInsights, see the **Nitro Help** documentation and the *Integrating Veeva Link with CRM MyInsights* topic (Using Veeva-Provided Custom MyInsights

Pages) (Link). These explain the data models and query functions (`queryVDSRecord`) when working with external datasets.

- **Veeva CRM Release Notes:** MyInsights is continually improved. Reading through recent release notes can highlight new features or changes (for example, when MyInsights v2 library was introduced, or performance enhancements). This can be found on Veeva's support site or PDF release documentation.
- **Community and Blogs:** Veeva's online community (Veeva Connect) has forums for MyInsights where developers and users discuss questions. Additionally, some agencies and partners (like 28b.co.uk, etc.) have blog posts about MyInsights use cases and tips (e.g. *What is Veeva MyInsights?* (What is Veeva MyInsights? - Veeva Agency - twentyeightb) (What is Veeva MyInsights? - Veeva Agency - twentyeightb)). These can provide inspiration and practical insights from real implementations.

By leveraging these resources and the guidance in this article, you should be well-equipped to build and refine custom MyInsights dashboards. **Happy coding**, and may your Veeva CRM users gain valuable insights from the dashboards you create!