```
{
  "cognitive_node": {
    "id": "file:cognitive-modules()http-communication.pdf",
    "system": "API-GATEWAY",
    "module": "cognitive-modules",
    "role": "cognition-module",
    "manifesto": "This cognition-module declares a graph-centric, implementation-agnostic
specification for representing an HTTP communication surface as cognitive memories and
their interactions. Following cognition-module()creation-v2, it:\n\n1) treats every
PATH+CODE as a single atomic cognitive memory (node) with a stable id 'file:<path>';\n2)
constrains the module to a shared graph vocabulary (nodes, edges, constraints);\n3) defines
assembly, validation, and evolution rules so any HTTP surface can be captured consistently
and interoperably with other cognition-modules.",
    "graph_vocabulary": {
      "node_kinds": [
        { "kind": "controller", "semantics": "Declares HTTP entrypoints (routes)." },
        { "kind": "service", "semantics": "Executes side-effects (network, IO, computation) for
controllers." },
        { "kind": "dto", "semantics": "Shapes and/or validates inbound/outbound data (queries,
params, bodies)." },
        { "kind": "enum", "semantics": "Closed set of allowed values referenced by dto or logic."
},
        { "kind": "module", "semantics": "Composes nodes into an injectable boundary
(registration/scope)." },
        { "kind": "config", "semantics": "Holds environment-derived constants/URLs/keys." },
        { "kind": "bootstrap", "semantics": "Starts the runtime and applies global policies
(validation, pipes)." },
        { "kind": "test", "semantics": "Asserts wiring/behavioral contracts of any node(s)." },
        { "kind": "file", "semantics": "Fallback when role cannot be inferred without speculation."
}
      ],
      "edge_kinds": [
        { "type": "USES", "rule": "From a node that directly calls/constructs/types another node."
},
        { "type": "IMPORTS", "rule": "From a node that imports another symbol/file." },
        { "type": "REGISTERS", "rule": "From a module to the nodes it makes available
(controllers/services/etc.)." },
        { "type": "OWNS", "rule": "From a file to the primary symbol it declares (e.g.,
controller/service class)." },
        { "type": "BELONGS_TO", "rule": "From a file to its module namespace or feature
boundary." },
        { "type": "BOOTSTRAPS", "rule": "From bootstrap to the root module." }
      ]
    },
    "assembly_rules_v2": [
      "A1. One PATH+CODE → one node; id = 'file:<path>'.",
      "A2. Node kind is inferred only from direct, observable evidence (class decorators,
imports, exported symbols); otherwise use kind 'file'.",
      "A3. Create edges solely from evidence in code (imports, constructor injections, direct
method calls, annotations).",
      "A4. Do not invent routes, behaviors, or configs; record only what the code proves.",
      "A5. Tests link to the nodes they exercise but do not define production behavior."
    ],
    "validation_rules_v2": [
      "V1. No node without an id; no duplicate ids.",
      "V2. No edge without evidence; each edge must cite its source and target ids.",
      "V3. Module composition must be acyclic at the module level (imports/registration should
not create cycles).",
      "V4. If global validation/policies are in bootstrap, mark them as facts only if present in
code.",
      "V5. Prefer specific kinds (controller/service/dto/enum/module/config/bootstrap/test) over
'file' when safely inferable."
    ],
    "evolution_rules_v2": [
      "E1. Additions: new files → new nodes; connect edges from observable
imports/usages.",
      "E2. Modifications: update node fields minimally; adjust edges that changed due to code
edits.",
      "E3. Removals: delete node and dangling edges; preserve graph consistency.",
      "E4. Refactors: prefer stable ids (path-based) unless file is relocated; if relocated, update
id while preserving historic mapping separately if needed."
    ],
    "interoperability": [
      "I1. Produced graph must be consumable by other cognition-modules (e.g., api-gateway,
nestjs-current-understanding).",
      "I2. Shared vocabulary (node_kinds, edge_kinds) enables cross-module reasoning and
migration planning (e.g., REST → GraphQL).",
      "I3. HTTP-specific semantics live in node kinds, not in bespoke rules; the manifesto
remains framework-agnostic."
    ],
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["graph-manifesto", "normalized-nodes", "interaction-graph"],
      "exported_symbols": ["http-communication(graph)"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Normalization of PATH+CODE into nodes compatible with the shared vocabulary",
        "Deterministic graph construction from evidence-only relations"
      ],
      "error_model": [
       "If role inference is unsafe, node kind defaults to 'file'",
       "If an expected relation lacks evidence, omit the edge rather than speculate"
      ]
     },
     "contracts": {
      "invariants": [
       "Evidence-only modeling: no speculative nodes or edges",
       "Manifesto remains implementation-agnostic and reusable",
       "Graph produced is minimal, complete, and consistent with code"
      ],
      "security": [
       "Security/validation facts are recorded only when explicitly present (e.g., validation
pipes, guards)"
      ]
     },
     "why": {
      "associations": [
       "Provides a uniform, reusable blueprint to capture any HTTP surface as a graph",
       "Enables efficient communication and migration by keeping cognition-modules
structurally identical"
      ],
      "biases": [
       "Favors graph uniformity and evidence over narrative descriptions"
      ],
      "context": "An intermediate, graph-first step toward ascending comprehension levels and
cross-module interoperability."
     },
     "tests": {
      "spec_files": [],
      "key_cases": [
       "Single-file module yields one node with correct kind inference",
       "Interconnected files yield correct USES/IMPORTS/REGISTERS edges",
       "Graph remains acyclic at module level after incremental updates"
      ]
     },
     "links_hint": [
      "cognition-module()creation-v2.pdf",
      "cognitive-modules()api-gateway.pdf",
      "cognitive-modules()nestjs-current-understanding.pdf"
     ]
    },
    "graph_patch": {
     "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:cognitive-modules()http-communication.pdf", "kind": "file" },
      { "id": "cognition-module:http-communication", "kind": "cognition-module" }
     ],
```

  "edges": [
    { "type": "BELONGS_TO", "from": "file:cognitive-modules()http-communication.pdf", "to": "system:API-GATEWAY" },
    { "type": "OWNS", "from": "file:cognitive-modules()http-communication.pdf", "to": "cognition-module:http-communication" }
  ]
 }
}
2) NODES (Path, code, cognitive memory)
call-metrics-analytics\src\dashboard\calls-duration-summary\pipeline\index.ts

```
type DailyDataInput = {
  startDate: string;
  endDate: string;
  clinicTimezone: string;
};

// Total time

export function generatePipeline({
  startDate,
  endDate,
  clinicTimezone,
}: DailyDataInput) {
  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },
    {
      $project: {
        callOutcome: 1,
        callEndTime: 1,
        callTransferredTime: 1,
        callStartTime: 1,
        duration: {
          $cond: [
            { $eq: ["$callOutcome", "handled-by-agent"] },
            {
              $divide: [
                { $subtract: ["$callEndTime", "$callTransferredTime"]
},
                1000,
              ],
            },
            {
              $divide: [
                { $subtract: ["$callEndTime", "$callStartTime"] },
                1000,
              ],
            },
          ],
        },
      },
    },
    {
      $group: {
        _id: null,
        totalAiOperationTime: {
          $sum: {
            $cond: [
              {
                $or: [
                  { $eq: ["$callOutcome", "scheduled"] },
                  { $eq: ["$callOutcome", "rescheduled"] },
                  { $eq: ["$callOutcome", "cancellation"] },
                ],
              },
              "$duration",
              0,
            ],
          },
        },
        totalAiNonOperationTime: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "not-available"] },
"$duration", 0],
          },
        },
        totalHumanAgentTime: {
          $sum: {
            $cond: [
              { $eq: ["$callOutcome", "handled-by-agent"] },
              "$duration",
              0,
            ],
          },
        },
      },
    },
    {
      $project: {
        _id: 0,
        totalAiOperationTime: 1,
        totalAiNonOperationTime: 1,
        totalHumanAgentTime: 1,
      },
    },
  ];
}
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\pipeline\\index.ts",
    "system": "API-GATEWAY",
    "module": "http-communication",
    "role": "file",
    "manifesto": "Evidence-only memory of a pipeline builder that computes total durations by outcome buckets over a terminated-calls set. Exports `generatePipeline(input: DailyDataInput)` which returns a MongoDB aggregation pipeline. Locally defines `DailyDataInput` and uses no external imports.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [
        "type DailyDataInput = { startDate:string; endDate:string; clinicTimezone:string }"
      ],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "generatePipeline(input: DailyDataInput): any[]"
      ],
      "exported_symbols": [
```

      "generatePipeline"
    ],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": [],
    "error_model": [
      "Assumes `startDate` and `endDate` are parseable by `$toDate`; invalid values cause aggregation errors at execution time.",
      "Assumes documents contain `callStartTime`, `callEndTime`, `callTransferredTime`, `callStatus`, and `callOutcome`; missing fields affect computed durations.",
      "Duration division by 1000 expects millisecond timestamps; other units yield incorrect seconds."
    ]
  },
  "contracts": {
    "invariants": [
      "Stage order is fixed: $match (exclude 'ongoing-*' statuses and bound by [startDate,endDate]) → $project (fields + computed `duration`) → $group (accumulate totals per outcome class) → $project (remove _id and expose totals).",
      "Duration formula: if callOutcome == 'handled-by-agent' then (callEndTime - callTransferredTime)/1000 else (callEndTime - callStartTime)/1000.",
      "Totals produced: totalAiOperationTime (scheduled|rescheduled|cancellation), totalAiNonOperationTime (not-available), totalHumanAgentTime (handled-by-agent).",
      "Output document shape: { totalAiOperationTime, totalAiNonOperationTime, totalHumanAgentTime }."
    ],
    "security": []
  },
  "why": {
    "associations": [
      "Provides an aggregation for total time distribution across AI operations, AI non-operation, and human agent handling within a time window."
    ],
    "biases": [
      "Ignores ongoing call statuses explicitly; only terminated states contribute to totals.",
      "Timezone parameter is accepted but not used in the current pipeline stages."
    ],
    "context": "Utility node to assemble an analytics pipeline; integrated into the HTTP communication cognition module as a plain file node."
  },
  "tests": {
    "spec_files": [],
    "key_cases": [
      "Handled-by-agent uses transferred interval; others use full call interval.",
      "Ongoing statuses are excluded by match stage.",
      "Bounded by callStartTime within [startDate, endDate]."
    ]

```
    },
    "links_hint": []
  },
  "graph_patch": {
    "nodes": [
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\pipeline\\index.ts", "kind":
"file" },
      { "id": "type:DailyDataInput", "kind": "type" },
      { "id": "function:generatePipeline", "kind": "function" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\pipeline\\index.ts", "to":
"cognition-module:http-communication" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\pipeline\\index.ts", "to":
"type:DailyDataInput" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\pipeline\\index.ts", "to":
"function:generatePipeline" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\calls-duration-summary\types\index.ts

```
export type RawCallsSummaryDataType = {
  status: string;
  data: {
    aggregationResult: AggregationResult;
  };
};

type AggregationResult = CallsDurationSummaryType[];

export type CallsDurationSummaryType = {
  totalAiOperationTime: number;
  totalAiNonOperationTime: number;
  totalHumanAgentTime: number;
};
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts",
    "system": "call-metrics-analytics",
    "module": "calls-duration-summary",
    "role": "types",
    "purpose": "Defines TypeScript types for representing summarized call duration data
across AI and human agents.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["RawCallsSummaryDataType", "CallsDurationSummaryType"],
      "exported_symbols": ["RawCallsSummaryDataType", "CallsDurationSummaryType"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "AggregationResult is always an array of CallsDurationSummaryType.",
        "RawCallsSummaryDataType.data.aggregationResult must conform to
AggregationResult."
      ],
      "security": []
    },
    "why": {
      "associations": ["Provides typed structures for analytics results in call duration summary
dashboards."],
      "biases": [],
      "context": "Type-only file ensuring strong typing for dashboard data rendering and
backend responses."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": []
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:calls-duration-summary", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts", "kind":
"file" },
      { "id": "type:RawCallsSummaryDataType", "kind": "type" },
```

    { "id": "type:CallsDurationSummaryType", "kind": "type" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:calls-duration-summary" },
    { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts", "to": "module-node:calls-duration-summary" },
    { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts", "to": "type:RawCallsSummaryDataType" },
    { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts", "to": "type:CallsDurationSummaryType" },
    { "type": "USES_TYPE", "from": "type:RawCallsSummaryDataType", "to": "type:CallsDurationSummaryType" }
  ]
 }
}

call-metrics-analytics\src\dashboard\calls-duration-summary\utils\process-received-data.ts

```
import {
  type RawCallsSummaryDataType,
  type CallsDurationSummaryType,
} from "../types";

type SummaryTotalsWithUnit = CallsDurationSummaryType & {
  unit: string;
};

export function processReceivedData(
  rawData: RawCallsSummaryDataType
): SummaryTotalsWithUnit {
  const aggregation = rawData.data.aggregationResult[0];

  if (!aggregation) {
    return {
      totalAiOperationTime: 0,
      totalAiNonOperationTime: 0,
      totalHumanAgentTime: 0,
      unit: "seconds",
    };
  }
  return {
    totalAiOperationTime: aggregation.totalAiOperationTime,
    totalAiNonOperationTime: aggregation.totalAiNonOperationTime,
    totalHumanAgentTime: aggregation.totalHumanAgentTime,
    unit: "seconds",
  };
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\utils\\process-received-data.ts",
    "system": "call-metrics-analytics",
    "module": "calls-duration-summary",
    "role": "util",
    "purpose": "Transforms raw aggregated call summary data into a standardized object with totals and a fixed unit.",
    "inputs": {
      "di_injections": [],
      "imports": ["../types"],
      "schema_types": ["RawCallsSummaryDataType", "CallsDurationSummaryType"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["processReceivedData"],
      "exported_symbols": ["processReceivedData"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "If aggregationResult is empty, returns default totals with zeros and unit 'seconds'."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns an object containing AI operation, AI non-operation, and human agent totals.",
        "The result always includes a unit field with the value 'seconds'."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Utility function to normalize backend aggregation responses for UI consumption."
      ],

      "biases": ["Assumes time measurement unit is always 'seconds'."],
      "context": "Part of the calls-duration-summary dashboard pipeline, ensuring consistent structure for display or further processing."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Handles missing aggregationResult by returning default values.",
        "Correctly maps totals from the first element of aggregationResult when present."
      ]
    },
    "links_hint":
["file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:calls-duration-summary", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\utils\\process-received-data.ts", "kind": "file" },
      { "id": "util:processReceivedData", "kind": "util" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:calls-duration-summary" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\utils\\process-received-data.ts", "to": "module-node:calls-duration-summary" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\utils\\process-received-data.ts", "to": "util:processReceivedData" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\utils\\process-received-data.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts" },
      { "type": "USES_TYPE", "from": "util:processReceivedData", "to":
"type:RawCallsSummaryDataType" },
      { "type": "USES_TYPE", "from": "util:processReceivedData", "to":
"type:CallsDurationSummaryType" }
    ]
  }
}

call-metrics-analytics\src\dashboard\calls-duration-summary\index.ts

```
import express from "express";
import { stringifyPipeline } from "../utils/stringifyPipeline";
import { generatePipeline } from "./pipeline";
import { processReceivedData } from "./utils/process-received-data";
import { type RawCallsSummaryDataType } from "./types";

import dotenv from "dotenv";
dotenv.config();

const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

const router = express.Router();

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const pipelineDailyData = generatePipeline({
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
      clinicTimezone: "America/Los_Angeles",
    });

    const pipeline = stringifyPipeline(pipelineDailyData);

    const url =
`${callMetricsRecordsURL}/call-records/aggregate?${pipeline}`;
    const response = await fetch(url);

    const json: RawCallsSummaryDataType = await response.json();

    if (!response.ok) {
      console.error("Error", response.status);
    }

    const processedData = processReceivedData(json);

    res.status(200).json(processedData);
  } catch (err) {
    console.log(err);
  }
});

export default router;
```
{

```
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts",
    "system": "call-metrics-analytics",
    "module": "calls-duration-summary",
    "role": "controller",
    "purpose": "Defines an Express router endpoint that builds a MongoDB aggregation
pipeline, forwards the request to the call-metrics backend, processes the response, and
returns summarized call duration data.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "dotenv",
        "../utils/stringifyPipeline",
        "./pipeline",
        "./utils/process-received-data",
        "./types"
      ],
      "schema_types": ["RawCallsSummaryDataType"],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["GET /"],
      "exported_symbols": ["router"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads environment variable CALL_METRICS_RECORDS_URL",
        "Makes HTTP request to call-metrics backend",
        "Logs errors to console"
      ],
      "error_model": [
        "Logs error status if backend response is not ok",
        "Catches and logs runtime errors"
      ]
    },
    "contracts": {
      "invariants": [
        "startDate and endDate query parameters must be provided in request",
        "Response always returns processed data in standardized format"
      ],
      "security": ["Depends on dotenv for configuration management"]
    },
    "why": {
      "associations": [
        "Acts as the entrypoint for call duration summary analytics in the dashboard",
        "Bridges frontend queries with backend aggregation results"
    ],
    "biases": ["Hardcodes clinicTimezone as America/Los_Angeles"],
    "context": "Provides API endpoint for dashboard summaries by delegating pipeline
generation, stringification, and response processing."
  },
  "tests": {
    "spec_files": [],
    "key_cases": [
      "Handles successful fetch and returns processed data",
      "Handles failed fetch and logs error",
      "Handles empty aggregation result gracefully through processReceivedData"
    ]
  },
  "links_hint": [
    "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\pipeline\\index.ts",

"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\utils\\process-received-d
ata.ts",
    "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts"
  ]
 },
 "graph_patch": {
  "nodes": [
    { "id": "system:call-metrics-analytics", "kind": "system" },
    { "id": "module-node:calls-duration-summary", "kind": "module-node" },
    { "id": "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts",
"kind": "file" },
    { "id": "controller:calls-duration-summary-router", "kind": "controller" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:calls-duration-summary" },
    { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts", "to":
"module-node:calls-duration-summary" },
    { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts", "to":
"controller:calls-duration-summary-router" },
    { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\pipeline\\index.ts" },
    { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\utils\\process-received-d
ata.ts" },
    { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\types\\index.ts" },
```

{ "type": "CONFIG", "from": "controller:calls-duration-summary-router", "to": "config:CALL_METRICS_RECORDS_URL" }
    ]
  }
}

call-metrics-analytics\src\dashboard\carousel\external-services\fetchAgentCallTimeData.ts

```
import { generateAgentCallTimePipeline } from "../pipeline";
import { type DateRange } from "./../types";

const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

type CallTimeData = {
  totalCallTime: number;
  fullName: string;
  extension: string;
  imageUrl: string;
};

export type Aggregation = {
  status: string;
  data: {
    aggregationResult: CallTimeData[];
  };
};

export async function fetchAgentCallTimeData({
  startDate,
  endDate,
}: DateRange) {
  const callRecordsService =
`${callMetricsRecordsURL}/call-records/aggregate?`;

  const response = await fetch(
    `${callRecordsService}${generateAgentCallTimePipeline({
      startDate,
      endDate,
    })}`
  );

  if (!response.ok) {
    console.error("Failed to fetch agent data:", response.status);
    throw new Error("Call records service responded with error");
  }

  const json: Aggregation = await response.json();

  return json.data.aggregationResult;
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeData.ts",
    "system": "call-metrics-analytics",
    "module": "carousel",
    "role": "service",
    "purpose": "Fetches aggregated agent call time data from the call-metrics backend using a generated pipeline and returns structured results.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "../pipeline",
        "./../types"
      ],
      "schema_types": ["DateRange", "Aggregation"],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["fetchAgentCallTimeData"],
      "exported_symbols": ["fetchAgentCallTimeData", "Aggregation"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["HTTP request to call-records service", "Logs error to console"],
      "error_model": [
        "Throws an error if the backend responds with a non-OK status.",
        "Fails if network request cannot complete."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns an array of CallTimeData when successful.",
        "Relies on generateAgentCallTimePipeline to produce valid query string."
      ],
      "security": ["Depends on environment variable CALL_METRICS_RECORDS_URL for target service URL."]
    },
    "why": {

```
      "associations": ["Bridges UI carousel with backend call time analytics for agents."],
      "biases": ["Assumes backend always responds with expected Aggregation shape."],
      "context": "Used by dashboard carousel to show call time statistics per agent."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Successful fetch returns parsed aggregation result.",
        "Non-OK response triggers error handling.",
        "Handles empty aggregationResult gracefully."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\carousel\\types\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:carousel", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeD
ata.ts", "kind": "file" },
      { "id": "service:fetchAgentCallTimeData", "kind": "service" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:carousel" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeD
ata.ts", "to": "module-node:carousel" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeD
ata.ts", "to": "service:fetchAgentCallTimeData" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeD
ata.ts", "to": "file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeD
ata.ts", "to": "file:call-metrics-analytics\\src\\dashboard\\carousel\\types\\index.ts" },
      { "type": "CONFIG", "from": "service:fetchAgentCallTimeData", "to":
"config:CALL_METRICS_RECORDS_URL" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\carousel\pipeline\index.ts

```
import { stringifyPipeline } from "../../utils/stringifyPipeline";
import { type DateRange } from "../types";

export function generateAgentCallTimePipeline({
  startDate,
  endDate,
}: DateRange) {
  const page = 1;
  const pageSize = 4;
  const skipCount = (page - 1) * pageSize;

  return stringifyPipeline([
    {
      $match: {
        $expr: {
          $and: [
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },
    {
      $match: {
        $or: [
          {
            callOutcome: { $in: ["scheduled", "rescheduled",
"cancellation"] },
          },
          {
            $and: [
              { callOutcome: "handled-by-agent" },
              { "agent.fullName": { $ne: null } },
            ],
          },
        ],
      },
    },
    {
      $project: {
        callOutcome: 1,
        callEndTime: 1,
        callTransferredTime: 1,
        callStartTime: 1,
        originalAgentFullName: "$agent.fullName",
        agentLabel: {
          $cond: [
            {
              $in: [
                "$callOutcome",
                ["scheduled", "rescheduled", "cancellation"],
              ],
            },
            "AI agent",
            "$agent.fullName",
          ],
        },
        duration: {
          $cond: [
            { $eq: ["$callOutcome", "handled-by-agent"] },
            {
              $divide: [
                {
                  $subtract: [
                    { $toDate: "$callEndTime" },
                    { $toDate: "$callTransferredTime" },
                  ],
                },
                1000,
              ],
            },
            {
              $divide: [
                {
                  $subtract: [
                    { $toDate: "$callEndTime" },
                    { $toDate: "$callStartTime" },
                  ],
                },
                1000,
              ],
            },
          ],
        },
      },
    },
    {
      $group: {
        _id: "$agentLabel",
        totalCallTime: { $sum: "$duration" },
      },
    },
    {
      $project: {
        _id: 0,
        fullName: "$_id",
        totalCallTime: 1,
        extension: { $literal: "(461)-476-8795" },
        imageUrl: { $literal: "diane-russell.png" },
      },
    },
    { $sort: { totalCallTime: -1 } },
    { $skip: skipCount },
    { $limit: pageSize },

    // Fallback step to inject default AI agent if result is empty
    {
      $facet: {
        paginatedData: [],
      },
    },
    {
      $project: {
        paginatedData: {
          $cond: [
            { $eq: [{ $size: "$paginatedData" }, 0] },
            [
              {
                fullName: "AI agent",
                totalCallTime: 0,
                extension: "(461)-476-8795",
                imageUrl: "diane-russell.png",
              },
            ],
            "$paginatedData",
          ],
        },
      },
    },
    { $unwind: "$paginatedData" },
    { $replaceRoot: { newRoot: "$paginatedData" } },
  ]);
}
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts",
    "system": "call-metrics-analytics",
    "module": "carousel",
    "role": "pipeline",
    "purpose": "Generates a MongoDB aggregation pipeline string to calculate total call time
per agent or AI within a given date range, returning a paginated and normalized dataset.",
    "inputs": {
      "di_injections": [],
      "imports": ["../../utils/stringifyPipeline", "../types"],
      "schema_types": ["DateRange"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generateAgentCallTimePipeline"],
      "exported_symbols": ["generateAgentCallTimePipeline"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "If startDate or endDate are invalid, MongoDB $toDate may throw runtime errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Pipeline filters calls within startDate and endDate.",
        "Includes both AI operations (scheduled, rescheduled, cancellation) and
handled-by-agent calls with valid agent names.",
        "Calculates call duration differently for AI vs human-handled calls.",
        "Always returns at least one record, injecting a fallback AI agent if results are empty."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Supports carousel visualization by providing consistent agent call time data.",
        "Handles pagination, sorting, and fallback cases to ensure UI always has displayable
data."
      ],
      "biases": ["Hardcodes extension and imageUrl fields for agents."],
```

      "context": "Part of dashboard carousel analytics for displaying call time rankings of
agents versus AI."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid date range returns aggregated results grouped by agent label.",
        "Handled-by-agent calls use transferred time for duration.",
        "AI calls use start-to-end time for duration.",
        "Empty results trigger fallback AI agent insertion."
      ]
    },
    "links_hint":
["file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTime
Data.ts"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:carousel", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts", "kind": "file"
},
      { "id": "pipeline:generateAgentCallTimePipeline", "kind": "pipeline" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:carousel" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts", "to":
"module-node:carousel" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts", "to":
"pipeline:generateAgentCallTimePipeline" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\utils\\stringifyPipeline.ts" },
      { "type": "USES_TYPE", "from": "pipeline:generateAgentCallTimePipeline", "to":
"type:DateRange" }
    ]
  }
}

call-metrics-analytics\src\dashboard\carousel\types\index.ts

```
export type DateRange = {
  startDate: string;
  endDate: string;
};
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\carousel\\types\\index.ts",
    "system": "call-metrics-analytics",
    "module": "carousel",
    "role": "types",
    "purpose": "Defines a DateRange type representing the start and end dates used in carousel-related queries and pipelines.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["DateRange"],
      "exported_symbols": ["DateRange"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "DateRange must include both startDate and endDate as strings."
      ],
      "security": []
    },
    "why": {
      "associations": ["Used by pipelines and services to enforce consistent date range structure."],
      "biases": [],
      "context": "Shared type alias to guarantee consistency across carousel module utilities and services."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\carousel\\pipeline\\index.ts",

"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeData.ts"
    ]
  },
  "graph_patch": {

```
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:carousel", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\carousel\\types\\index.ts", "kind": "file" },
      { "id": "type:DateRange", "kind": "type" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:carousel" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\types\\index.ts", "to":
"module-node:carousel" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\types\\index.ts", "to": "type:DateRange"
}
    ]
  }
}
```

call-metrics-analytics\src\dashboard\carousel\index.ts

```
import express from "express";
import { fetchAgentCallTimeData } from
"./external-services/fetchAgentCallTimeData";
import dotenv from "dotenv";

dotenv.config();

const router = express.Router();

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const dateRange = {
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
    };

    const agentsCallTimeData = await fetchAgentCallTimeData(dateRange);

    res.status(200).json(agentsCallTimeData);
  } catch (err) {
    console.log(err);
  }
});

export default router;
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\carousel\\index.ts",
    "system": "call-metrics-analytics",
    "module": "carousel",
    "role": "controller",
    "purpose": "Defines an Express router endpoint that fetches aggregated agent call time data within a given date range and returns it as JSON.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "dotenv",
        "./external-services/fetchAgentCallTimeData"
      ],
      "schema_types": [],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["GET /"],
      "exported_symbols": ["router"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads environment variable CALL_METRICS_RECORDS_URL",
        "Performs HTTP request via fetchAgentCallTimeData",
        "Logs errors to console"
      ],
      "error_model": [
        "Errors in fetchAgentCallTimeData propagate and are caught, logged, but not rethrown."
      ]
    },
    "contracts": {
      "invariants": [
        "Requires startDate and endDate query parameters.",
        "Response always returns an array of agent call time data when successful."
      ],
      "security": ["Depends on dotenv for configuration management."]
    },
    "why": {
      "associations": [
        "Acts as an API entrypoint for carousel dashboard data.",
        "Delegates actual data fetching and transformation to fetchAgentCallTimeData service."
      ],
```

      "biases": [],
      "context": "Enables frontend carousel to request and display agent call time summaries."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Handles valid date ranges and returns agent call time data.",
        "Logs and swallows errors instead of crashing the server."
      ]
    },
    "links_hint": [

"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeData.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:carousel", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\carousel\\index.ts", "kind": "file" },
      { "id": "controller:carousel-router", "kind": "controller" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:carousel" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\index.ts", "to": "module-node:carousel"
},
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\carousel\\index.ts",
"to": "controller:carousel-router" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\carousel\\external-services\\fetchAgentCallTimeData.ts" },
      { "type": "CONFIG", "from": "controller:carousel-router", "to":
"config:CALL_METRICS_RECORDS_URL" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\ai-operation-breakdown\pipeline\index.ts

```
type DailyDataInput = {
  startDate: string;
  endDate: string;
  page: number;
  pageSize: number;
  granularity: string;
  clinicTimezone: string;
};

export function generatePipeline({
  startDate,
  endDate,
  page,
  pageSize,
  granularity,
  clinicTimezone,
}: DailyDataInput) {
  let dateFormat;

  if (granularity === "daily") {
    dateFormat = "%Y-%m-%d";
  } else if (granularity === "monthly") {
    dateFormat = "%Y-%m";
  } else if (granularity === "yearly") {
    dateFormat = "%Y";
  }

  const skipCount = (page - 1) * pageSize;

  return [
    { $project: { callStartTime: 1, callOutcome: 1, _id: 0 } },

    {
      $match: {
        $expr: {
          $and: [
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },
    {
      $group: {
        _id: {
          $dateToString: {
            format: dateFormat,
            date: "$callStartTime",
            timezone: clinicTimezone,
          },
        },
        scheduled: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "scheduled"] }, 1, 0],
          },
        },
        rescheduled: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "rescheduled"] }, 1, 0],
          },
        },
        cancellation: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "cancellation"] }, 1, 0],
          },
        },
        humanAgent: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "handled-by-agent"] }, 1,
0],
          },
        },
        nonOperation: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "not-available"] }, 1, 0],
          },
        },
      },
    },
    {
      $sort: { _id: 1 },
    },
    {
      $facet: {
        total: [{ $count: "count" }],
        paginatedData: [{ $skip: skipCount }, { $limit: pageSize }],
      },
    },
  ];
}
```

{

```
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\pipeline\\index.ts
",
    "system": "call-metrics-analytics",
    "module": "ai-operation-breakdown",
    "role": "pipeline",
    "purpose": "Generates a MongoDB aggregation pipeline to group call outcomes by date
and classify them into scheduled, rescheduled, cancellation, human agent, and
non-operation categories, supporting pagination and granularity.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["DailyDataInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generatePipeline"],
      "exported_symbols": ["generatePipeline"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "Invalid granularity value results in undefined dateFormat, causing runtime error.",
        "Invalid startDate or endDate strings may cause $toDate errors in MongoDB."
      ]
    },
    "contracts": {
      "invariants": [
        "Groups calls by formatted callStartTime using the provided granularity and timezone.",
        "Counts calls by outcome: scheduled, rescheduled, cancellation, handled-by-agent,
not-available.",
        "Returns results sorted by date ascending.",
        "Supports pagination using skip and limit."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Supports AI operation breakdown charts by providing categorized call outcome data.",
        "Pagination and total count allow efficient frontend rendering."
      ],
      "biases": ["Granularity limited to daily, monthly, or yearly."],
      "context": "Used for charting and analytics to monitor AI vs human agent operations over
time."
    },
    "tests": {
```

      "spec_files": [],
      "key_cases": [
        "Daily granularity produces YYYY-MM-DD formatted keys.",
        "Monthly granularity produces YYYY-MM formatted keys.",
        "Yearly granularity produces YYYY formatted keys.",
        "Pipeline returns counts for each call outcome type.",
        "Pagination correctly slices the results."
      ]
    },
    "links_hint": []
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:ai-operation-breakdown", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\pipeline\\index.ts
", "kind": "file" },
      { "id": "pipeline:generatePipeline(ai-operation-breakdown)", "kind": "pipeline" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:ai-operation-breakdown" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\pipeline\\index.ts
", "to": "module-node:ai-operation-breakdown" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\pipeline\\index.ts
", "to": "pipeline:generatePipeline(ai-operation-breakdown)" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\ai-operation-breakdown\utils\process-received-data.ts

```
import { formatDate } from "../../../utils/format-date";
export type DataResponse = {
  status: string;
  data: {
    aggregationResult: AggregationResult[];
  };
};

type AggregationResult = {
  total: TotalCount[];
  paginatedData: Aggregation[];
```

```
};

type TotalCount = {
  count: number;
};

type Aggregation = {
  _id: string;
  humanAgent: number;
  nonOperation: number;
  scheduled: number;
  rescheduled: number;
  cancellation: number;
};

export function processReceivedData(rawData: DataResponse): {
  xAxis: string[];
  callsHandledByHuman: number[];
  callsHandledByAIWithoutOperation: number[];
  callsScheduled: number[];
  callsRescheduled: number[];
  callsCancelled: number[];
  year: number;
  totalItems: number;
} {
  const result = rawData.data.aggregationResult[0];
  const totalItems = result.total[0]?.count || 0;
  const paginatedData = result.paginatedData;

  const xAxis = paginatedData.map((entry) => formatDate(entry._id));
  const callsHandledByHuman = paginatedData.map((entry) =>
entry.humanAgent);
  const callsHandledByAIWithoutOperation = paginatedData.map(
    (entry) => entry.nonOperation
  );
  const callsScheduled = paginatedData.map((entry) => entry.scheduled);
  const callsRescheduled = paginatedData.map((entry) =>
entry.rescheduled);
  const callsCancelled = paginatedData.map((entry) =>
entry.cancellation);

  return {
    xAxis,
    callsHandledByHuman,
    callsHandledByAIWithoutOperation,
    callsScheduled,
    callsRescheduled,
    callsCancelled,
    year: 0,
    totalItems,
  };
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-received-data.ts",
    "system": "call-metrics-analytics",
    "module": "ai-operation-breakdown",
    "role": "util",
    "purpose": "Transforms raw aggregation results from the AI operation breakdown pipeline into chart-ready data structures with arrays for x-axis labels and categorized call counts.",
    "inputs": {
      "di_injections": [],
      "imports": ["../../../utils/format-date"],
      "schema_types": ["DataResponse", "AggregationResult", "Aggregation", "TotalCount"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["processReceivedData"],
      "exported_symbols": ["processReceivedData", "DataResponse"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "If rawData.data.aggregationResult is empty, accessing result.total or result.paginatedData may cause runtime errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns arrays for xAxis and each call category, even if empty.",
        "Total count is derived from the first element in result.total or defaults to 0.",
        "Year is currently hardcoded to 0."
      ],
      "security": []
    },
    "why": {

```
      "associations": [
        "Acts as a normalization layer between backend aggregation results and frontend chart rendering."
      ],
      "biases": ["Hardcodes year as 0 instead of deriving from data."],
      "context": "Utility specifically designed to prepare AI operation breakdown data for visualization."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid aggregation result produces mapped arrays.",
        "Empty total array results in totalItems = 0.",
        "Empty paginatedData results in empty arrays for all categories."
      ]
    },
    "links_hint": [
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\pipeline\\index.ts",
      "file:call-metrics-analytics\\src\\utils\\format-date.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:ai-operation-breakdown", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-received-data.ts", "kind": "file" },
      { "id": "util:processReceivedData(ai-operation-breakdown)", "kind": "util" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:ai-operation-breakdown" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-received-data.ts", "to": "module-node:ai-operation-breakdown" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-received-data.ts", "to": "util:processReceivedData(ai-operation-breakdown)" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-received-data.ts", "to": "file:call-metrics-analytics\\src\\utils\\format-date.ts" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\ai-operation-breakdown\index.ts

```
import express from "express";
import { stringifyPipeline } from "../../utils/stringifyPipeline";

import { generatePipeline } from "./pipeline";
import {
  type DataResponse,
  processReceivedData,
} from "./utils/process-received-data";
import dotenv from "dotenv";
dotenv.config();
const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

const router = express.Router();

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const pipelineDailyData = generatePipeline({
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
      page: Number(req.query.page),
      pageSize: Number(req.query.pageSize),
      granularity: req.query.granularity as string,
      clinicTimezone: req.query.clinicTimezone as string,
    });

    const pipeline = stringifyPipeline(pipelineDailyData);

    const url =
`${callMetricsRecordsURL}/call-records/aggregate?${pipeline}`;
    const response = await fetch(url);

    if (!response.ok) {
      console.error("Error", response.status);
    }

    const json: DataResponse = await response.json();

    const processedData = processReceivedData(json);

    const output = {
      xAxis: processedData.xAxis,
      callsHandledByHuman: processedData.callsHandledByHuman,
      callsHandledByAIWithoutOperation:
        processedData.callsHandledByAIWithoutOperation,
      callsScheduled: processedData.callsScheduled,
      callsRescheduled: processedData.callsRescheduled,
      callsCancelled: processedData.callsCancelled,
      year: processedData.year,
      paginationInformation: {
        pageNumber: Number(req.query.page),
        pageSize: Number(req.query.pageSize),
        totalItems: processedData.totalItems,
      },
    };

    res.status(200).json(output);
    // res.status(200).json(json);
  } catch (err) {
    console.log(err);
  }
});

export default router;
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts",
    "system": "call-metrics-analytics",
    "module": "ai-operation-breakdown",
    "role": "controller",
    "purpose": "Defines an Express router endpoint that generates an aggregation pipeline, queries the call-metrics backend, processes results, and returns AI operation breakdown data with pagination information.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "dotenv",
        "../../utils/stringifyPipeline",
        "./pipeline",
        "./utils/process-received-data"
      ],
      "schema_types": ["DataResponse"],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["GET /"],
      "exported_symbols": ["router"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads environment variable CALL_METRICS_RECORDS_URL",
        "Performs HTTP request to call-records backend",
        "Logs errors to console"
      ],
      "error_model": [
        "If fetch returns non-OK, logs error and still attempts to process JSON.",
        "If query parameters are missing or invalid, may cause runtime errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Accepts query parameters: startDate, endDate, page, pageSize, granularity,
clinicTimezone.",
        "Always returns structured JSON with xAxis, categorized calls arrays, year, and
paginationInformation."
      ],
      "security": ["Depends on dotenv for configuration management."]
    },
    "why": {
      "associations": [
        "Acts as API entrypoint for AI operation breakdown charts.",
        "Bridges backend pipeline results with frontend charting needs."
      ],
      "biases": ["Returns year from processedData even if hardcoded."],
      "context": "Supports dashboard analytics by exposing an endpoint for visualizing AI vs
human agent call outcomes over time."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid request returns structured breakdown with pagination.",
        "Non-OK backend response logs error.",
        "Empty aggregation result handled gracefully."
      ]
    },
    "links_hint": [

"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\pipeline\\index.ts
",

"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-rec
eived-data.ts"
```

    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:ai-operation-breakdown", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts", "kind":
"file" },
      { "id": "controller:ai-operation-breakdown-router", "kind": "controller" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:ai-operation-breakdown" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts", "to":
"module-node:ai-operation-breakdown" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts", "to":
"controller:ai-operation-breakdown-router" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\pipeline\\index.ts
" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-rec
eived-data.ts" },
      { "type": "CONFIG", "from": "controller:ai-operation-breakdown-router", "to":
"config:CALL_METRICS_RECORDS_URL" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\call-frequency-outcome\pipeline\index.ts

```
type DateRange = {
  startDate: string;
  endDate: string;
};

export function generateCallFrequencyOutcomePipeline({
  startDate,
  endDate,
}: DateRange) {
  return [
    {
      $match: {
        $expr: {
          $and: [
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },
    {
      $project: {
        callerPhoneNumber: 1,
        callOutcome: 1,
        outcomeType: {
          $switch: {
            branches: [
              {
                case: {
                  $in: [
                    "$callOutcome",
                    ["scheduled", "rescheduled", "cancellation"],
                  ],
                },
                then: "operation",
              },
              {
                case: { $eq: ["$callOutcome", "handled-by-agent"] },
                then: "handledByAgent",
              },
              {
                case: { $eq: ["$callOutcome", "not-available"] },
                then: "nonOperation",
              },
            ],
            default: null,
          },
        },
      },
    },
    {
      $match: {
        outcomeType: { $in: ["operation", "handledByAgent",
"nonOperation"] },
      },
    },
    {
      $group: {
        _id: {
          callerPhoneNumber: "$callerPhoneNumber",
          outcomeType: "$outcomeType",
        },
        count: { $sum: 1 },
      },
    },
    {
      $group: {
        _id: "$_id.callerPhoneNumber",
        totalCalls: { $sum: "$count" },
        breakdown: {
          $push: {
            k: "$_id.outcomeType",
            v: "$count",
          },
        },
      },
    },
    {
      $project: {
        callCountKey: {
          $cond: [
            { $gt: ["$totalCalls", 10] },
            "10+",
            { $toString: "$totalCalls" },
          ],
        },
        breakdown: {
          $arrayToObject: "$breakdown",
        },
      },
    },
    {
      $group: {
        _id: "$callCountKey",
        operation: { $sum: { $ifNull: ["$breakdown.operation", 0] } },
        handledByAgent: { $sum: { $ifNull:
["$breakdown.handledByAgent", 0] } },
        nonOperation: { $sum: { $ifNull: ["$breakdown.nonOperation", 0]
} },
      },
    },
    {
      $project: {
        _id: 0,
        calls: "$_id",
        operation: 1,
        handledByAgent: 1,
        nonOperation: 1,
      },
    },
    { $sort: { calls: 1 } },
  ];
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\pipeline\\index.ts"
,
    "system": "call-metrics-analytics",
    "module": "call-frequency-outcome",
    "role": "pipeline",
    "purpose": "Generates a MongoDB aggregation pipeline that groups calls by caller phone number, classifies outcomes into operation, handledByAgent, or nonOperation, and then produces a frequency distribution of calls per phone number bucket.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["DateRange"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generateCallFrequencyOutcomePipeline"],
      "exported_symbols": ["generateCallFrequencyOutcomePipeline"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "Invalid startDate or endDate strings may cause $toDate errors in MongoDB."
      ]
    },
    "contracts": {

```
      "invariants": [
        "Filters calls between startDate and endDate.",
        "Classifies callOutcome into three categories: operation, handledByAgent,
nonOperation.",
        "Groups calls by callerPhoneNumber to calculate total calls and outcome breakdown.",
        "Buckets phone numbers by total call count, collapsing >10 calls into '10+'.",
        "Aggregates totals per callCountKey, producing overall category counts."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Supports charts analyzing frequency of calls per phone number and their outcome
types.",
        "Provides insights into distribution of caller behavior (one-time vs frequent callers)."
      ],
      "biases": ["Buckets all callers with more than 10 calls into '10+'. This simplifies
distribution but loses detail for heavy callers."],
      "context": "Used in dashboard analytics to understand frequency and categorization of
caller outcomes."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Caller with only one operation call should appear in bucket '1' with operation count 1.",
        "Caller with multiple mixed outcomes aggregated correctly by outcomeType.",
        "Caller with more than 10 calls bucketed under '10+'.",
        "Aggregation result sorted by calls ascending."
      ]
    },
    "links_hint": []
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:call-frequency-outcome", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\pipeline\\index.ts"
, "kind": "file" },
      { "id": "pipeline:generateCallFrequencyOutcomePipeline", "kind": "pipeline" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:call-frequency-outcome" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\pipeline\\index.ts"
, "to": "module-node:call-frequency-outcome" },
```

      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\pipeline\\index.ts"
, "to": "pipeline:generateCallFrequencyOutcomePipeline" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\call-frequency-outcome\index.ts

```
import express from "express";
import { stringifyPipeline } from "../../utils/stringifyPipeline";
import { generateCallFrequencyOutcomePipeline } from "./pipeline";

import dotenv from "dotenv";
dotenv.config();

const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

const router = express.Router();

export type CallFrequencyOutcomeDataType = {
  calls: string; // "1", "2", ..., "10+"
  operation: number;
  handledByAgent: number;
  nonOperation: number;
};

export type CallFrequencyOutcomeResponse = {
  status: string;
  data: {
    aggregationResult: CallFrequencyOutcomeDataType[];
  };
};

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const pipelineDailyData = generateCallFrequencyOutcomePipeline({
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
    });

    const pipeline = stringifyPipeline(pipelineDailyData);
    const url =
`${callMetricsRecordsURL}/call-records/aggregate?${pipeline}`;
    const response = await fetch(url);

    const json: CallFrequencyOutcomeResponse = await response.json();

    if (!response.ok) {
      console.error("Error", response.status);
    }

    res.status(200).json(processRawData(json));
  } catch (err) {
    console.log(err);
  }
});

export default router;

function processRawData(
  response: CallFrequencyOutcomeResponse
): CallFrequencyOutcomeDataType[] {
  const dbResults = response.data.aggregationResult;

  const defaultBuckets: CallFrequencyOutcomeDataType[] = [
    { calls: "1", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "2", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "3", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "4", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "5", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "6", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "7", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "8", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "9", operation: 0, handledByAgent: 0, nonOperation: 0 },
    { calls: "10+", operation: 0, handledByAgent: 0, nonOperation: 0 },
  ];

  const frequencyMap = new Map<string, CallFrequencyOutcomeDataType>();
  dbResults.forEach((entry) => {
    frequencyMap.set(entry.calls, entry);
  });

  return defaultBuckets.map((bucket) => {
    return frequencyMap.get(bucket.calls) ?? bucket;
  });
}
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\index.ts",
    "system": "call-metrics-analytics",
    "module": "call-frequency-outcome",
    "role": "controller",
    "purpose": "Defines an Express router endpoint that executes the call frequency outcome pipeline, queries the backend, and returns normalized frequency distribution data for calls by outcome type.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "dotenv",
        "../../utils/stringifyPipeline",
        "./pipeline"
      ],
      "schema_types": [
        "CallFrequencyOutcomeDataType",
        "CallFrequencyOutcomeResponse"
      ],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["GET /"],
      "exported_symbols": ["router"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads environment variable CALL_METRICS_RECORDS_URL",
        "Performs HTTP request to call-records backend",
        "Logs errors to console"
      ],
      "error_model": [
        "If fetch fails or response is invalid, logs error and may return incomplete results."
      ]
    },
    "contracts": {
      "invariants": [
        "Response always returns data for buckets '1' through '9' and '10+' even if backend does not return them.",
        "processRawData ensures default buckets are filled with zeros when missing."
      ],
      "security": ["Depends on dotenv for environment configuration."]
    },
    "why": {
      "associations": [
        "Provides normalized chart-ready data for call frequency outcome analytics.",
        "Bridges backend pipeline results with frontend chart requirements by filling missing
buckets."
      ],
      "biases": ["Hardcodes buckets up to 9 and collapses all higher counts into '10+'."],
      "context": "Enables dashboard visualization of call distribution by number of calls per
phone number and outcome category."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid backend response populates buckets with actual data.",
        "Missing buckets are backfilled with zeros.",
        "Error in fetch is logged but server continues execution."
      ]
    },
    "links_hint": [

"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\pipeline\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:call-frequency-outcome", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\index.ts", "kind":
"file" },
      { "id": "controller:call-frequency-outcome-router", "kind": "controller" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:call-frequency-outcome" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\index.ts", "to":
"module-node:call-frequency-outcome" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\index.ts", "to":
"controller:call-frequency-outcome-router" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\pipeline\\index.ts"
}
    ]
```

```
 }
}
```

call-metrics-analytics\src\dashboard\charts\call-volume\model\generate-call-volume-histogram.ts

```
import { UnifiedModel } from "../../../../models";

type CallVolumeRawDataType = {
  _id: string;
  frequency: number;
}[];

export default async function generateCallVolumeHistogram(
  startDate: number,
  endDate: number
) {
  const aggregation = await UnifiedModel.aggregate([
    {
      // Find records between the date range
      $match: {
        callInitializationTime: {
          $gte: new Date(startDate),
          $lte: new Date(endDate),
        },
      },
    },
    {
      // Create a histogram: Count how many documents started at a
given hour.
      // The result is a maximum of 24 documents with the shape { _id:
number, frequency: number }
      $group: {
        _id: {
          $toInt: {
            $dateToString: {
              format: "%H",
              date: "$callInitializationTime",
            },
          },
        },
        frequency: { $sum: 1 },
      },
    },
    {
      $sort: {
        _id: 1,
      },
    },
  ]);

  return aggregation.map((document) => ({
    hour: document._id,
    frequency: document.frequency,
  }));
}
```

```
{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\model\\generate-call-volume-histogram.ts",
    "system": "call-metrics-analytics",
    "module": "call-volume",
    "role": "model",
    "purpose": "Generates a histogram of call volumes by hour within a given date range using MongoDB aggregation on UnifiedModel.",
    "inputs": {
      "di_injections": [],
      "imports": ["../../../../models"],
      "schema_types": ["CallVolumeRawDataType"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generateCallVolumeHistogram"],
      "exported_symbols": ["generateCallVolumeHistogram"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Performs database aggregation on UnifiedModel."],
      "error_model": [
        "If startDate or endDate are invalid, aggregation may return empty or throw errors.",
        "If UnifiedModel is not connected, aggregation will fail."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns an array of objects with hour and frequency keys.",
        "Groups callInitializationTime by hour (0–23).",
        "Result is sorted by hour ascending."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Provides backend data for call volume charts at hourly granularity."
      ],
      "biases": ["Assumes callInitializationTime is always present and valid date."],
      "context": "Used in analytics dashboards to display hourly call volume histograms."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid range returns counts grouped by hour.",
        "Empty dataset returns empty array.",
        "Data spanning multiple days aggregates counts per hour across days."
      ]
    },
    "links_hint": [
      "model:UnifiedModel"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:call-volume", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\model\\generate-call-volume-
histogram.ts", "kind": "file" },
      { "id": "model:generateCallVolumeHistogram", "kind": "model" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:call-volume"
},
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\model\\generate-call-volume-
histogram.ts", "to": "module-node:call-volume" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\model\\generate-call-volume-
histogram.ts", "to": "model:generateCallVolumeHistogram" },
      { "type": "USES", "from": "model:generateCallVolumeHistogram", "to":
"model:UnifiedModel" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\call-volume\pipeline\index.ts

```
type DailyDataInput = {
  startDate: string;
  endDate: string;
  clinicTimezone: string;
};

export function generatePipeline({
  startDate,
  endDate,
  clinicTimezone,
}: DailyDataInput) {
  return [
    { $project: { callStartTime: 1, callOutcome: 1, _id: 0 } },
    {
      $match: {
        $expr: {
          $and: [
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },
    {
      $group: {
        _id: {
          $dateToString: {
            format: "%H",
            date: "$callStartTime",
            timezone: clinicTimezone,
          },
        },
        humanAgent: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "handled-by-agent"] }, 1,
0],
          },
        },
        aiNonOperation: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "not-available"] }, 1, 0],
          },
        },
        scheduled: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "scheduled"] }, 1, 0],
          },
        },
        rescheduled: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "rescheduled"] }, 1, 0],
          },
        },
        cancellation: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "cancellation"] }, 1, 0],
          },
        },
      },
    },
    {
      $sort: { _id: 1 },
    },
  ];
}
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\pipeline\\index.ts",
    "system": "call-metrics-analytics",
    "module": "call-volume",
    "role": "pipeline",
    "purpose": "Generates a MongoDB aggregation pipeline that groups calls by hour of day and counts outcomes categorized as human agent, AI non-operation, scheduled, rescheduled, and cancellation.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["DailyDataInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generatePipeline"],
      "exported_symbols": ["generatePipeline"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "Invalid startDate or endDate strings may cause $toDate errors.",
      "If clinicTimezone is invalid, $dateToString may throw runtime error."
     ]
   },
   "contracts": {
     "invariants": [
      "Pipeline projects only callStartTime and callOutcome.",
      "Filters calls between provided startDate and endDate.",
      "Groups calls by hour of callStartTime using clinicTimezone.",
      "Counts outcomes into humanAgent, aiNonOperation, scheduled, rescheduled, and
cancellation categories.",
      "Results are sorted by hour ascending."
     ],
     "security": []
   },
   "why": {
     "associations": [
      "Supports call volume charts by providing per-hour counts of categorized call
outcomes."
     ],
     "biases": ["Granularity limited to hourly buckets."],
     "context": "Used for dashboard analytics to visualize distribution of calls across hours of
the day and outcome types."
   },
   "tests": {
     "spec_files": [],
     "key_cases": [
      "Hourly grouping produces 0–23 keys with counts per category.",
      "Empty dataset returns empty aggregation result.",
      "Timezone parameter shifts grouping to clinic’s local time."
     ]
   },
   "links_hint": []
  },
  "graph_patch": {
   "nodes": [
     { "id": "system:call-metrics-analytics", "kind": "system" },
     { "id": "module-node:call-volume", "kind": "module-node" },
     { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\pipeline\\index.ts",
"kind": "file" },
     { "id": "pipeline:generatePipeline(call-volume)", "kind": "pipeline" }
   ],
   "edges": [
     { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:call-volume"
},
     { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\pipeline\\index.ts", "to":
"module-node:call-volume" },
```

      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\pipeline\\index.ts", "to":
"pipeline:generatePipeline(call-volume)" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\call-volume\utils\parse-query.ts

```
export default function parseQuery(query: {
  endDate: string;
  startDate: string;
}) {
  const { startDate, endDate } = query;
  return [new Date(startDate).getTime(), new Date(endDate).getTime()];
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\utils\\parse-query.ts",
    "system": "call-metrics-analytics",
    "module": "call-volume",
    "role": "util",
    "purpose": "Parses a query object containing startDate and endDate strings into numeric timestamps representing milliseconds since epoch.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["parseQuery"],
      "exported_symbols": ["parseQuery"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "If startDate or endDate are invalid date strings, Date parsing will result in NaN timestamps."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns a two-element array [startDateMs, endDateMs].",
        "Input query must contain startDate and endDate as strings."
      ],

```
      "security": []
    },
    "why": {
      "associations": [
        "Utility function to normalize request query parameters into a format suitable for
database operations."
      ],
      "biases": ["Assumes all date inputs are valid ISO strings or parseable formats."],
      "context": "Used in call volume analytics to prepare query ranges for MongoDB
aggregations."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid ISO date strings return correct timestamps.",
        "Invalid date strings return NaN values."
      ]
    },
    "links_hint": []
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:call-volume", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\utils\\parse-query.ts", "kind":
"file" },
      { "id": "util:parseQuery(call-volume)", "kind": "util" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:call-volume"
},
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\utils\\parse-query.ts", "to":
"module-node:call-volume" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\utils\\parse-query.ts", "to":
"util:parseQuery(call-volume)" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\call-volume\index.ts

```
import express from "express";
import { stringifyPipeline } from "../../utils/stringifyPipeline";
import { generatePipeline } from "./pipeline";
import dotenv from "dotenv";
dotenv.config();

const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

const router = express.Router();

export type DataResponse = {
  status: string;
  data: {
    aggregationResult: CallVolumeRawDataType[];
  };
};

type CallVolumeRawDataType = {
  _id: string;
  humanAgent: number;
  aiNonOperation: number;
  scheduled: number;
  rescheduled: number;
  cancellation: number;
};

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const pipelineDailyData = generatePipeline({
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
      clinicTimezone: req.query.clinicTimezone as string,
    });

    const pipeline = stringifyPipeline(pipelineDailyData);

    const url =
`${callMetricsRecordsURL}/call-records/aggregate?${pipeline}`;
    const response = await fetch(url);

    const json: DataResponse = await response.json();

    if (!response.ok) {
      console.error("Error", response.status);
    }
    const processedData = generateCallVolumeHistogram(json);

    res.status(200).json(processedData);
  } catch (err) {
    console.log(err);
  }
});

function generateCallVolumeHistogram(rawData: DataResponse) {
  const dbResults = rawData.data.aggregationResult;

  // Build a map for fast lookup by hour
  const frequencyMap = new Map<string, CallVolumeRawDataType>(
    dbResults.map((entry) => [entry._id, entry])
  );

  // Generate 24-hour range and map frequencies
  const histogram = Array.from({ length: 24 }, (_, h) => {
    const hour = h.toString().padStart(2, "0");
    const data = frequencyMap.get(hour);

    const humanAgent = data?.humanAgent ?? 0;
    const aiNonOperation = data?.aiNonOperation ?? 0;
    const scheduled = data?.scheduled ?? 0;
    const rescheduled = data?.rescheduled ?? 0;
    const cancellation = data?.cancellation ?? 0;

    return {
      hour,
      humanAgent,
      aiNonOperation,
      scheduled,
      rescheduled,
      cancellation,
      total:
        humanAgent + aiNonOperation + scheduled + rescheduled +
cancellation,
    };
  });

  return histogram;
}
```

```
export default router;
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\index.ts",
    "system": "call-metrics-analytics",
    "module": "call-volume",
    "role": "controller",
    "purpose": "Defines an Express router endpoint that generates a pipeline, queries the call-metrics backend, and returns a 24-hour histogram of call outcomes grouped by hour.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "dotenv",
        "../../utils/stringifyPipeline",
        "./pipeline"
      ],
      "schema_types": ["DataResponse", "CallVolumeRawDataType"],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["GET /"],
      "exported_symbols": ["router"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads environment variable CALL_METRICS_RECORDS_URL",
        "Performs HTTP request to call-records backend",
        "Logs errors to console"
      ],
      "error_model": [
        "If fetch returns non-OK, logs error but continues processing response.",
        "If query parameters are invalid or missing, may produce empty or invalid results."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns a histogram with 24 entries, one for each hour of the day.",
        "Each entry includes counts for humanAgent, aiNonOperation, scheduled, rescheduled, cancellation, and total."
      ],
      "security": ["Depends on dotenv for configuration management."]
    },
    "why": {

      "associations": [
        "Bridges backend call volume pipeline with frontend visualization by ensuring all 24 hours are represented.",
        "Normalizes incomplete backend results by filling missing hours with zeros."
      ],
      "biases": ["Histogram assumes fixed 24-hour cycle regardless of actual data span."],
      "context": "Provides chart-ready call volume data for dashboards."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid request returns histogram with correct totals.",
        "Missing hours default to zero counts.",
        "Non-OK backend response still produces a histogram if data is present."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\pipeline\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:call-volume", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\index.ts", "kind": "file" },
      { "id": "controller:call-volume-router", "kind": "controller" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:call-volume" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\index.ts", "to": "module-node:call-volume" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\index.ts", "to": "controller:call-volume-router" },
      { "type": "IMPORTS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\index.ts", "to": "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\pipeline\\index.ts" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\handling-overview\pipeline\index.ts

```
type DailyDataInput = {
  startDate: string;
  endDate: string;
  page: number;
  pageSize: number;
  granularity: string;
  clinicTimezone: string;
};

export function generatePipeline({
  startDate,
  endDate,
  page,
  pageSize,
  granularity,
  clinicTimezone,
}: DailyDataInput) {
  let dateFormat;

  if (granularity === "daily") {
    dateFormat = "%Y-%m-%d";
  } else if (granularity === "monthly") {
    dateFormat = "%Y-%m";
  } else if (granularity === "yearly") {
    dateFormat = "%Y";
  }

  console.log("START: ", startDate);
  console.log("END: ", endDate);
  console.log("TIMEZONE: ", clinicTimezone);
  console.log("--------------------------------");

  const skipCount = (page - 1) * pageSize;

  return [
    { $project: { callStartTime: 1, callOutcome: 1, _id: 0 } },

    {
      $match: {
        $expr: {
          $and: [
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },
    {
      $group: {
        _id: {
          $dateToString: {
            format: dateFormat,

            date: "$callStartTime",
            timezone: clinicTimezone,
          },
        },
        humanAgent: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "handled-by-agent"] }, 1,
0],
          },
        },
        nonOperation: {
          $sum: {
            $cond: [{ $eq: ["$callOutcome", "not-available"] }, 1, 0],
          },
        },
        operation: {
          $sum: {
            $cond: [
              {
                $or: [
                  { $eq: ["$callOutcome", "scheduled"] },
                  { $eq: ["$callOutcome", "rescheduled"] },
                  { $eq: ["$callOutcome", "cancellation"] },
                ],
              },
              1,
              0,
            ],
          },
        },
      },
    },
    {
      $sort: { _id: 1 },
    },
    {
      $facet: {
        total: [{ $count: "count" }],
        paginatedData: [{ $skip: skipCount }, { $limit: pageSize }],
      },
    },
  ];
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\pipeline\\index.ts",
    "system": "call-metrics-analytics",
    "module": "handling-overview",
    "role": "pipeline",
    "purpose": "Generates a MongoDB aggregation pipeline that groups calls by time granularity and classifies outcomes into humanAgent, nonOperation, and operation categories, supporting pagination and sorting.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["DailyDataInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generatePipeline"],
      "exported_symbols": ["generatePipeline"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Logs startDate, endDate, and clinicTimezone to console."],
      "error_model": [
        "Invalid granularity leads to undefined dateFormat, causing $dateToString errors.",
        "Invalid startDate or endDate strings may throw MongoDB $toDate errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Groups calls by date according to granularity (daily, monthly, yearly) and clinicTimezone.",
        "Counts calls into three categories: humanAgent, nonOperation, and operation.",
        "Supports pagination via skip and limit stages.",
        "Returns both total count and paginated data via $facet."
      ],
      "security": []
    },

    "why": {
      "associations": [
        "Supports handling overview charts by showing distribution of outcomes over time.",
        "Pagination makes results scalable for large datasets."
      ],
      "biases": ["Groups multiple AI outcomes into a single operation category."],
      "context": "Enables dashboard analytics to visualize AI vs human handling proportions over time."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Daily granularity groups by YYYY-MM-DD.",
        "Monthly granularity groups by YYYY-MM.",
        "Yearly granularity groups by YYYY.",
        "Pagination slices results as expected.",
        "Empty dataset still returns facet structure."
      ]
    },
    "links_hint": []
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:handling-overview", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\pipeline\\index.ts", "kind": "file" },
      { "id": "pipeline:generatePipeline(handling-overview)", "kind": "pipeline" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:handling-overview" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\pipeline\\index.ts", "to": "module-node:handling-overview" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\pipeline\\index.ts", "to": "pipeline:generatePipeline(handling-overview)" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\handling-overview\utils\process-received-data.ts

```
import { formatDate } from "../../../utils/format-date";
export type DataResponse = {
  status: string;
  data: {
    aggregationResult: AggregationResult[];
  };
};

type AggregationResult = {
  total: TotalCount[];
  paginatedData: Aggregation[];
};

type TotalCount = {
  count: number;
};

type Aggregation = {
  _id: string;
  humanAgent: number;
  nonOperation: number;
  operation: number;
};

export function processReceivedData(rawData: DataResponse): {
  xAxis: string[];
  callsHandledByHuman: number[];
  callsHandledByAI: number[];
  callsHandledByAIWithoutOperation: number[];
  year: number;
  totalItems: number;
} {
  const result = rawData.data.aggregationResult[0];
  const totalItems = result.total[0]?.count || 0;
  const paginatedData = result.paginatedData;

  const xAxis = paginatedData.map((entry) => {
    return formatDate(entry._id);
  });
  const callsHandledByHuman = paginatedData.map((entry) =>
entry.humanAgent);
  const callsHandledByAI = paginatedData.map((entry) =>
entry.operation);

  const callsHandledByAIWithoutOperation = paginatedData.map(
    (entry) => entry.nonOperation
  );

  return {
    xAxis: xAxis,
    callsHandledByHuman: callsHandledByHuman,
    callsHandledByAI: callsHandledByAI,
    callsHandledByAIWithoutOperation: callsHandledByAIWithoutOperation,
    year: 0,
    totalItems: totalItems,
  };
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\utils\\process-received-data.ts",
    "system": "call-metrics-analytics",
    "module": "handling-overview",
    "role": "util",
    "purpose": "Transforms raw aggregation results from the handling-overview pipeline into chart-ready structures with arrays for time axis and categorized call counts.",
    "inputs": {
      "di_injections": [],
      "imports": ["../../../utils/format-date"],
      "schema_types": ["DataResponse", "AggregationResult", "Aggregation", "TotalCount"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["processReceivedData"],
      "exported_symbols": ["processReceivedData", "DataResponse"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "If aggregationResult is empty, accessing result.total or result.paginatedData may cause runtime errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns arrays for xAxis and each call category, even if empty.",
        "Total count derived from first element of result.total or defaults to 0.",
        "Year is currently hardcoded to 0."

```
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Acts as a normalization layer between backend aggregation results and frontend
handling overview chart rendering."
      ],
      "biases": ["Hardcodes year as 0 rather than deriving from data."],
      "context": "Utility tailored for handling-overview charts to structure call outcome data."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid aggregation result produces populated arrays.",
        "Empty total array results in totalItems = 0.",
        "Empty paginatedData results in empty arrays for all categories."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\pipeline\\index.ts",
      "file:call-metrics-analytics\\src\\utils\\format-date.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:handling-overview", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\utils\\process-received
-data.ts", "kind": "file" },
      { "id": "util:processReceivedData(handling-overview)", "kind": "util" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:handling-overview" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\utils\\process-received
-data.ts", "to": "module-node:handling-overview" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\utils\\process-received
-data.ts", "to": "util:processReceivedData(handling-overview)" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\utils\\process-received
-data.ts", "to": "file:call-metrics-analytics\\src\\utils\\format-date.ts" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\median-call-duration\pipeline

```
import express from "express";
import { stringifyPipeline } from "../../utils/stringifyPipeline";

import { generatePipeline } from "./pipeline";
import {
  type DataResponse,
  processReceivedData,
} from "./utils/process-received-data";
import dotenv from "dotenv";
dotenv.config();
const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

const router = express.Router();

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const pipelineDailyData = generatePipeline({
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
      page: Number(req.query.page),
      pageSize: Number(req.query.pageSize),
      granularity: req.query.granularity as string,
      clinicTimezone: req.query.clinicTimezone as string,
    });

    const pipeline = stringifyPipeline(pipelineDailyData);

    const url =
`${callMetricsRecordsURL}/call-records/aggregate?${pipeline}`;
    const response = await fetch(url);

    if (!response.ok) {
      console.error("Error", response.status);
    }

    const json: DataResponse = await response.json();

    const processedData = processReceivedData(json);

    const output = {
      xAxis: processedData.xAxis,
      callsHandledByHuman: processedData.callsHandledByHuman,
      callsHandledByAI: processedData.callsHandledByAI,
      callsHandledByAIWithoutOperation:
        processedData.callsHandledByAIWithoutOperation,
      year: processedData.year,
      paginationInformation: {
        pageNumber: Number(req.query.page),
        pageSize: Number(req.query.pageSize),
        totalItems: processedData.totalItems,
      },
    };

    res.status(200).json(output);
    // res.status(200).json(json);
  } catch (err) {
    console.log(err);
  }
});

export default router;
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline",
    "system": "call-metrics-analytics",
    "module": "median-call-duration",
    "role": "controller",
    "purpose": "Defines an Express router endpoint that generates an aggregation pipeline, queries the call-metrics backend, processes results, and returns median call duration data with pagination information.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "dotenv",
        "../../utils/stringifyPipeline",
        "./pipeline",
        "./utils/process-received-data"
      ],
      "schema_types": ["DataResponse"],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["GET /"],

```
      "exported_symbols": ["router"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads environment variable CALL_METRICS_RECORDS_URL",
        "Performs HTTP request to call-records backend",
        "Logs errors to console"
      ],
      "error_model": [
        "If fetch returns non-OK, logs error and still attempts to process JSON.",
        "If query parameters are missing or invalid, may cause runtime errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Accepts query parameters: startDate, endDate, page, pageSize, granularity,
clinicTimezone.",
        "Always returns structured JSON with xAxis, categorized calls arrays, year, and
paginationInformation."
      ],
      "security": ["Depends on dotenv for configuration management."]
    },
    "why": {
      "associations": [
        "Acts as API entrypoint for median call duration charts.",
        "Bridges backend pipeline results with frontend charting needs."
      ],
      "biases": ["Returns year from processedData even if hardcoded."],
      "context": "Supports dashboard analytics by exposing an endpoint for visualizing median
call durations across AI and human agents."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid request returns structured breakdown with pagination.",
        "Non-OK backend response logs error.",
        "Empty aggregation result handled gracefully."
      ]
    },
    "links_hint": [

"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts",

"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\utils\\process-receiv
ed-data.ts"
    ]
  },
```

  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:median-call-duration", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline", "kind": "file" },
      { "id": "controller:median-call-duration-router", "kind": "controller" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:median-call-duration" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline", "to": "module-node:median-call-duration" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline", "to": "controller:median-call-duration-router" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\median-call-duration\pipeline\index.ts

```
type DailyDataInput = {
  startDate: string;
  endDate: string;
  page: number;
  pageSize: number;
  granularity: string;
  clinicTimezone: string;
};
export function generatePipeline({
  startDate,
  endDate,
  page,
  pageSize,
  granularity,
  clinicTimezone,
}: DailyDataInput) {
  let dateFormat;

  if (granularity === "daily") {
    dateFormat = "%Y-%m-%d";
  } else if (granularity === "monthly") {
    dateFormat = "%Y-%m";
  } else if (granularity === "yearly") {
    dateFormat = "%Y";
  }

  const localOffsetTimeInHours = new Date().getTimezoneOffset() / 60;

  const skipCount = (page - 1) * pageSize;

  return [
    {
      $match: {
        $expr: {
          $and: [
            { $ne: ["$callStatus", "ongoing-bot"] },
            { $ne: ["$callStatus", "ongoing-agent"] },
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },
    {
      $project: {
        callStartTime: 1,
        callOutcome: 1,
        callEndTime: 1,
        callTransferredTime: 1,
        callStatus: 1,
        duration: {
          $cond: [
            { $eq: ["$callOutcome", "handled-by-agent"] },
            {
              $divide: [
                { $subtract: ["$callEndTime", "$callTransferredTime"]
},
                1000,
              ],
            },
            {
              $divide: [
                { $subtract: ["$callEndTime", "$callStartTime"] },
                1000,
              ],
            },
          ],
        },
        _id: 0,
      },
    },
    {
      $group: {
        _id: {
          $dateToString: {
            format: dateFormat,
            date: "$callStartTime",
            timezone: clinicTimezone,
          },
        },

        aiOperationDurations: {
          $push: {
            $cond: [
              {
                $or: [
                  { $eq: ["$callOutcome", "scheduled"] },
                  { $eq: ["$callOutcome", "rescheduled"] },
                  { $eq: ["$callOutcome", "cancellation"] },
                ],
              },
              "$duration",
              "$$REMOVE",
            ],
          },
        },

        aiNonOperationDurations: {
          $push: {
            $cond: [
              { $eq: ["$callOutcome", "not-available"] },
              "$duration",
              "$$REMOVE",
            ],
          },
        },

        humanAgentDurations: {
          $push: {
            $cond: [
              { $eq: ["$callOutcome", "handled-by-agent"] },
              "$duration",
              "$$REMOVE",
            ],
          },
        },
      },
    },
    {
      $sort: { _id: 1 },
    },
    {
      $facet: {
        total: [{ $count: "count" }],
        paginatedData: [{ $skip: skipCount }, { $limit: pageSize }],
      },
    },
  ];
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts",
    "system": "call-metrics-analytics",
    "module": "median-call-duration",
    "role": "pipeline",
    "purpose": "Generates a MongoDB aggregation pipeline that groups calls by date granularity and collects call durations into arrays for AI operations, AI non-operations, and human agent calls to enable median calculation.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["DailyDataInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generatePipeline"],
      "exported_symbols": ["generatePipeline"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [

```
        "Invalid granularity leads to undefined dateFormat, causing $dateToString errors.",
        "Invalid startDate or endDate strings may throw MongoDB $toDate errors.",
        "If call times are missing, duration may compute as null or incorrect."
      ]
    },
    "contracts": {
      "invariants": [
        "Filters calls within the date range excluding ongoing statuses.",
        "Durations for handled-by-agent calls are computed as callEndTime -
callTransferredTime.",
        "Durations for AI-related calls are computed as callEndTime - callStartTime.",
        "Groups by date string based on provided granularity and clinic timezone.",
        "Pushes durations into arrays categorized as aiOperationDurations,
aiNonOperationDurations, and humanAgentDurations.",
        "Supports pagination via skip and limit in a $facet."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Prepares raw data for median calculation of call durations by category and time
bucket.",
        "Used to analyze efficiency and time distribution of AI vs human handling."
      ],
      "biases": ["Assumes callOutcome values are limited to scheduled, rescheduled,
cancellation, not-available, handled-by-agent."],
      "context": "Backend analytics pipeline feeding dashboard median call duration charts."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Daily granularity produces per-day arrays of durations.",
        "Monthly granularity produces per-month arrays.",
        "Yearly granularity produces per-year arrays.",
        "Pipeline excludes ongoing-bot and ongoing-agent statuses.",
        "Empty dataset results in empty arrays for each category."
      ]
    },
    "links_hint": [

"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\utils\\process-receiv
ed-data.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:median-call-duration", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts",
"kind": "file" },
      { "id": "pipeline:generatePipeline(median-call-duration)", "kind": "pipeline" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:median-call-duration" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts",
"to": "module-node:median-call-duration" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts",
"to": "pipeline:generatePipeline(median-call-duration)" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\median-call-duration\index.ts

```
import express from "express";
import { stringifyPipeline } from "../../utils/stringifyPipeline";
import { generatePipeline } from "./pipeline";

import { formatDate } from "../../utils/format-date";

import dotenv from "dotenv";
dotenv.config();

const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

const router = express.Router();

type DataResponse = {
  status: string;
  data: {
    aggregationResult: AggregationResult[];
  };
};

type AggregationResult = {
  total: TotalCount[];
  paginatedData: Aggregation[];
};

type TotalCount = {
  count: number;
};

type Aggregation = {
  _id: string;
  aiOperationDurations: number[];
  aiNonOperationDurations: number[];
  humanAgentDurations: number[];
};

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const page = Number(req.query.page);
    const pageSize = Number(req.query.pageSize);

    const pipelineDailyData = generatePipeline({
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
      page,
      pageSize,
      granularity: req.query.granularity as string,
      clinicTimezone: req.query.clinicTimezone as string,
    });
    const pipeline = stringifyPipeline(pipelineDailyData);

    const url =
`${callMetricsRecordsURL}/call-records/aggregate?${pipeline}`;
    const response = await fetch(url);

    if (!response.ok) {
      console.error("Error", response.status);
    }

    const json: DataResponse = await response.json();

    const paginationInformation = {
      pageNumber: page,
      pageSize,
      totalItems: 0,
    };

    const medians = processReceivedData(json, paginationInformation);
    res.status(200).json(medians);
  } catch (err) {
    console.log(err);
  }
});

export default router;

function processReceivedData(
  rawData: DataResponse,
  paginationInformation: {
    pageNumber: number;
    pageSize: number;
    totalItems: number;
  }
): {
  xAxis: string[];
  aiOperation: number[];
  aiNonOperation: number[];
  humanAgent: number[];
  year: number;
  totalItems: number;
  paginationInformation: {
    pageNumber: number;
    pageSize: number;
    totalItems: number;
  };
} {
  const result = rawData.data.aggregationResult[0];
  const totalItems = result.total[0]?.count || 0;
  const paginatedData = result.paginatedData;

  function median(array: number[]): number {
    if (!array.length) return 0;
    const sorted = [...array].sort((a, b) => a - b);
    const mid = Math.floor(sorted.length / 2);
    return sorted.length % 2 === 0
      ? (sorted[mid - 1] + sorted[mid]) / 2
      : sorted[mid];
  }

  const xAxis = paginatedData.map((entry) => formatDate(entry._id));
  const aiOperationMedian = paginatedData.map(
    (entry) => Math.round(median(entry.aiOperationDurations) * 100) /
100
  );

  const aiNonOperationMedian = paginatedData.map(
    (entry) => Math.round(median(entry.aiNonOperationDurations) * 100)
/ 100
  );

  const humanAgentMedian = paginatedData.map(
    (entry) => Math.round(median(entry.humanAgentDurations) * 100) /
100
  );

  return {
    xAxis: xAxis,
    aiOperation: aiOperationMedian,
    aiNonOperation: aiNonOperationMedian,
    humanAgent: humanAgentMedian,
    year: 0,
    totalItems: totalItems,
    paginationInformation: {
      pageNumber: paginationInformation.pageNumber,
      pageSize: paginationInformation.pageSize,
      totalItems: totalItems,
    },
  };
}
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\index.ts",
    "system": "call-metrics-analytics",
    "module": "median-call-duration",
    "role": "controller",
    "purpose": "Defines an Express router endpoint that generates an aggregation pipeline, queries the backend, calculates medians of call durations by category, and returns chart-ready data with pagination information.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
```

```json
        "dotenv",
        "../../utils/stringifyPipeline",
        "./pipeline",
        "../../utils/format-date"
      ],
      "schema_types": [
        "DataResponse",
        "AggregationResult",
        "Aggregation",
        "TotalCount"
      ],
      "config_keys": ["CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["GET /"],
      "exported_symbols": ["router"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads environment variable CALL_METRICS_RECORDS_URL",
        "Performs HTTP request to backend",
        "Logs errors to console"
      ],
      "error_model": [
        "If fetch response is non-OK, logs error but still attempts to parse JSON.",
        "If aggregationResult is empty, accessing result.total or result.paginatedData may throw errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Accepts query parameters: startDate, endDate, page, pageSize, granularity, clinicTimezone.",
        "Always returns arrays for xAxis and median values for aiOperation, aiNonOperation, and humanAgent.",
        "PaginationInformation mirrors request and includes totalItems."
      ],
      "security": ["Depends on dotenv for configuration management."]
    },
    "why": {
      "associations": [
        "Acts as the API entrypoint for median call duration charts.",
        "Provides normalized median values per category, ready for visualization."
      ],
      "biases": ["Year is hardcoded as 0."],
      "context": "Supports dashboard analytics by exposing median duration comparisons across AI operations, AI non-operations, and human agent handled calls."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid data produces medians rounded to two decimals.",
        "Empty arrays return 0 for median.",
        "Pagination info is returned as part of response."
      ]
    },
    "links_hint": [

"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:median-call-duration", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\index.ts",
"kind": "file" },
      { "id": "controller:median-call-duration-router", "kind": "controller" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:median-call-duration" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\index.ts", "to":
"module-node:median-call-duration" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\index.ts", "to":
"controller:median-call-duration-router" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\repeated-callers\pipeline\index.ts

```
type DailyDataInput = {
  startDate: string;
  endDate: string;
  page: number;
  pageSize: number;
  granularity: string;
};
```

```
export function generatePipeline({
  startDate,
  endDate,
  page,
  pageSize,
  granularity,
}: DailyDataInput) {
  let dateFormat;

  if (granularity === "daily") {
    dateFormat = "%Y-%m-%d";
  } else if (granularity === "monthly") {
    dateFormat = "%Y-%m";
  } else if (granularity === "yearly") {
    dateFormat = "%Y";
  }

  const skipCount = (page - 1) * pageSize;
  const localOffsetTimeInHours = new Date().getTimezoneOffset() / 60;

  return [
    // Step 1: Filter calls within the date range
    {
      $match: {
        $expr: {
          $and: [
            { $gte: ["$callStartTime", { $toDate: startDate }] },
            { $lte: ["$callStartTime", { $toDate: endDate }] },
          ],
        },
      },
    },

    // Step 2: Group by time interval and phone number
    {
      $group: {
        _id: {
          timeGroup: {
            $dateToString: {
              format: dateFormat,
              date: {
                $subtract: [
                  "$callStartTime",
                  1000 * 60 * 60 * localOffsetTimeInHours,
                ],
              },
            },
          },
          phoneNumber: "$callerPhoneNumber",
        },
        count: { $sum: 1 },
      },
    },

    // Step 3: Group by time interval only and classify callers
    {
      $group: {
        _id: "$_id.timeGroup",
        uniqueCallers: {
          $sum: {
            $cond: [{ $eq: ["$count", 1] }, 1, 0], // count = 1 →
unique
          },
        },
        repeatedCallers: {
          $sum: {
            $cond: [{ $gt: ["$count", 1] }, 1, 0], // count > 1 → 1
repeated caller
          },
        },
      },
    },

    // Step 4: Sort results by time interval
    {
      $sort: { _id: 1 },
    },

    // Step 5: Apply pagination and return total count
    {
      $facet: {
        total: [{ $count: "count" }],
        paginatedData: [{ $skip: skipCount }, { $limit: pageSize }],
      },
    },
  ];
```

```
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\pipeline\\index.ts",
    "system": "call-metrics-analytics",
    "module": "repeated-callers",
    "role": "pipeline",
    "purpose": "Generates a MongoDB aggregation pipeline that groups calls by time interval and caller phone number to classify and count unique vs repeated callers, supporting pagination.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["DailyDataInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["generatePipeline"],
      "exported_symbols": ["generatePipeline"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "Invalid startDate or endDate may cause $toDate errors.",
        "Undefined granularity causes $dateToString errors.",
        "Timezone adjustment assumes system offset, which may not align with clinic timezone."
      ]
    },
    "contracts": {
      "invariants": [
        "Filters calls between given startDate and endDate.",
        "First groups by date and phone number to count calls per caller per interval.",
        "Then classifies callers as unique (count=1) or repeated (count>1).",
        "Aggregates totals per interval, sorted chronologically.",
        "Supports pagination via $facet with total count and paginatedData."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Enables analysis of caller behavior by distinguishing one-time vs repeated callers per interval.",
        "Provides backend support for repeated callers charting."

```
      ],
      "biases": [
        "Uses system timezone offset instead of explicitly provided clinic timezone."
      ],
      "context": "Backend analytics to support visualization of unique vs repeated caller
trends."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Daily granularity produces YYYY-MM-DD buckets.",
        "Monthly granularity produces YYYY-MM buckets.",
        "Yearly granularity produces YYYY buckets.",
        "Counts unique callers correctly when count=1.",
        "Counts repeated callers correctly when count>1."
      ]
    },
    "links_hint": []
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:repeated-callers", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\pipeline\\index.ts",
"kind": "file" },
      { "id": "pipeline:generatePipeline(repeated-callers)", "kind": "pipeline" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:repeated-callers" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\pipeline\\index.ts", "to":
"module-node:repeated-callers" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\pipeline\\index.ts", "to":
"pipeline:generatePipeline(repeated-callers)" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\repeated-callers\index.ts

```
import express from "express";
import { stringifyPipeline } from "../../utils/stringifyPipeline";
import { generatePipeline } from "./pipeline";
import { formatDate } from "../../utils/format-date";
import dotenv from "dotenv";
dotenv.config();
const callMetricsRecordsURL = process.env.CALL_METRICS_RECORDS_URL;

const router = express.Router();

router.get("/", async (req: express.Request, res: express.Response) =>
{
  try {
    const page = Number(req.query.page);
    const pageSize = Number(req.query.pageSize);

    const pipelineDailyData = generatePipeline({
      startDate: req.query.startDate as string,
      endDate: req.query.endDate as string,
      page,
      pageSize,
      granularity: req.query.granularity as string,
    });

    const pipeline = stringifyPipeline(pipelineDailyData);

    const url =
`${callMetricsRecordsURL}/call-records/aggregate?${pipeline}`;
    const response = await fetch(url);

    if (!response.ok) {
      console.error("Error", response.status);
    }

    const json = await response.json();

    const paginationInformation = {
      pageNumber: page,
      pageSize,
      totalItems: 0,
    };

    const output = processReceivedData(json, paginationInformation);

    res.status(200).json(output);
  } catch (err) {
    console.log(err);
  }
});

export default router;
type DataResponse = {
  status: string;
  data: {
    aggregationResult: AggregationResult[];
  };
};

type AggregationResult = {
  total: TotalCount[];
  paginatedData: Aggregation[];
};

type TotalCount = {
  count: number;
};

type Aggregation = {
  _id: string;
  uniqueCallers: number;
  repeatedCallers: number;
};

function processReceivedData(
  rawData: DataResponse,
  paginationInformation: {
    pageNumber: number;
    pageSize: number;
    totalItems: number;
  }
) {
  const result = rawData.data.aggregationResult[0];

  const output = result.paginatedData.reduce(
    (accumulator, current) => {
      accumulator.xAxis.push(formatDate(current._id));
      accumulator.uniqueCallers.push(current.uniqueCallers);
      accumulator.repeatedCallers.push(current.repeatedCallers);
      return accumulator;
    },
    {
      xAxis: [],
      uniqueCallers: [],
      repeatedCallers: [],
      paginationInformation: {
        pageNumber: 0,
        pageSize: 0,
        totalItems: 0,
      },
    } as {
      xAxis: string[];
      uniqueCallers: number[];
      repeatedCallers: number[];
      paginationInformation: {
        pageNumber: number;
        pageSize: number;
        totalItems: number;
      };
    }
  );

  let total;

  try {
    total = result.total[0].count;
  } catch {
    total = 0;
  }

  output.paginationInformation.pageNumber =
paginationInformation.pageNumber;
  output.paginationInformation.pageSize =
paginationInformation.pageSize;
  output.paginationInformation.totalItems = total;

  return output;
}
```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts",
    "system": "call-metrics-analytics",
    "module": "repeated-callers",

```
  "role": "controller",
  "purpose": "Defines an Express router endpoint that generates a repeated callers pipeline,
queries the backend, processes the results, and returns chart-ready data with pagination
information.",
  "inputs": {
   "di_injections": [],
   "imports": [
    "express",
    "dotenv",
    "../../utils/stringifyPipeline",
    "./pipeline",
    "../../utils/format-date"
   ],
   "schema_types": [
    "DataResponse",
    "AggregationResult",
    "Aggregation",
    "TotalCount"
   ],
   "config_keys": ["CALL_METRICS_RECORDS_URL"]
  },
  "outputs": {
   "api_surface": ["GET /"],
   "exported_symbols": ["router"],
   "events_pubsub": []
  },
  "behavior": {
   "side_effects": [
    "Reads CALL_METRICS_RECORDS_URL from environment variables",
    "Performs HTTP request to backend",
    "Logs errors to console"
   ],
   "error_model": [
    "If fetch fails or backend returns non-OK, logs error but still attempts to process JSON.",
    "If aggregationResult is missing or malformed, processReceivedData may default
totalItems to 0."
   ]
  },
  "contracts": {
   "invariants": [
    "Always returns arrays for xAxis, uniqueCallers, and repeatedCallers even if empty.",
    "Pagination information is included and derived from request plus backend total count."
   ],
   "security": ["Depends on dotenv for environment variable management."]
  },
  "why": {
   "associations": [
    "Provides chart-ready analytics on caller frequency (unique vs repeated).",
```

    "Bridges raw backend aggregation with structured frontend data."
   ],
   "biases": ["Assumes backend always provides aggregationResult[0] structure."],
   "context": "Supports dashboard visualization of unique vs repeated callers over time."
  },
  "tests": {
   "spec_files": [],
   "key_cases": [
    "Valid dataset produces arrays of xAxis, uniqueCallers, repeatedCallers.",
    "Empty dataset still returns arrays and pagination info with totalItems=0.",
    "Pagination info matches request values with totalItems from backend."
   ]
  },
  "links_hint": [
   "file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\pipeline\\index.ts",
   "file:call-metrics-analytics\\src\\utils\\format-date.ts"
  ]
 },
 "graph_patch": {
  "nodes": [
   { "id": "system:call-metrics-analytics", "kind": "system" },
   { "id": "module-node:repeated-callers", "kind": "module-node" },
   { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts",
"kind": "file" },
   { "id": "controller:repeated-callers-router", "kind": "controller" }
  ],
  "edges": [
   { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:repeated-callers" },
   { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts", "to":
"module-node:repeated-callers" },
   { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts", "to":
"controller:repeated-callers-router" },
   { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\pipeline\\index.ts" }
  ]
 }
}

call-metrics-analytics\src\dashboard\charts\types\index.ts

```
export interface ProcessInput {
  startDate: string;
  endDate: string;
  page: number;
  pageSize: number;
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\types\\index.ts",
    "system": "call-metrics-analytics",
    "module": "charts",
    "role": "types",
    "purpose": "Defines a ProcessInput interface to standardize input parameters for chart-related data processing functions.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["ProcessInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["ProcessInput"],
      "exported_symbols": ["ProcessInput"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "Improperly typed inputs (non-string dates or non-numeric pagination values) could cause runtime issues in consuming functions."
      ]
    },
    "contracts": {
      "invariants": [
        "startDate and endDate must be provided as strings.",
        "page and pageSize must be numbers."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Ensures consistent structure of input parameters across multiple chart modules.",
        "Facilitates type safety in TypeScript when handling pagination and date ranges."
      ],
      "biases": [],
      "context": "Shared type definitions used by various chart pipelines and controllers."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [

        "Valid ProcessInput object with correct types is accepted.",
        "Invalid types cause compile-time TypeScript errors."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\pipeline\\index.ts",

"file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\pipeline\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\pipeline\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\types\\index.ts", "kind": "file" },
      { "id": "type:ProcessInput", "kind": "type" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\types\\index.ts", "to":
"module-node:charts" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\types\\index.ts", "to": "type:ProcessInput" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\validators\common\types.ts

```
export interface QueryType {
  startDate: string;
  endDate: string;
  page: number;
  pageSize: number;
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\common\\types.ts",
    "system": "call-metrics-analytics",
    "module": "charts-validators",
    "role": "types",
    "purpose": "Defines a QueryType interface to standardize query parameter structures
(date range and pagination) for chart validators.",
    "inputs": {
      "di_injections": [],

```
      "imports": [],
      "schema_types": ["QueryType"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["QueryType"],
      "exported_symbols": ["QueryType"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": [
        "Improper typing (invalid dates or non-numeric pagination values) may cause runtime
issues in validation or pipeline construction."
      ]
    },
    "contracts": {
      "invariants": [
        "startDate and endDate must be strings representing valid date values.",
        "page and pageSize must be numbers."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Ensures uniformity of query parameters across validator logic for chart APIs.",
        "Provides type safety in TypeScript for validating request parameters before pipeline
execution."
      ],
      "biases": [],
      "context": "Part of validation layer to ensure request integrity in chart-related endpoints."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid QueryType object compiles correctly.",
        "TypeScript enforces correct types, preventing invalid query shapes at compile time."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\types\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts-validators", "kind": "module-node" },
```

    { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\common\\types.ts", "kind": "file" },
    { "id": "type:QueryType", "kind": "type" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:charts-validators" },
    { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\common\\types.ts", "to": "module-node:charts-validators" },
    { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\common\\types.ts", "to": "type:QueryType" }
  ]
 }
}

call-metrics-analytics\src\dashboard\charts\validators\input-validation\index.ts

```
import { ProcessInput } from "../../types";
import { Request } from "express";
import { validateDateFormat } from "./validate_date_format";
import { validateNumericValue } from "./validate_numeric_format";

/**
 * Parses and validates the incoming query parameters for fetching
daily, monthly, or yearly data.
 *
 * @param query - The query object extracted from the Express request.
 * @returns - A strongly typed and validated ProcessInput object.
 * @throws - An error if any parameter is missing or invalid.
 */
export function queryValidation(query: Request["query"]): ProcessInput
{
  try {
    const startDate = query.startDate as string;
    const endDate = query.endDate as string;
    const page = query.page as string;
    const pageSize = query.pageSize as string;

    return {
      startDate: validateDateFormat(startDate),
      endDate: validateDateFormat(endDate),
      page: validateNumericValue(page),
      pageSize: validateNumericValue(pageSize),
    };
  } catch (error) {
    throw new Error(`Invalid query parameters: ${(error as
Error).message}`);
  }
}
```

{
  "cognitive_node": {
    "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts",
    "system": "call-metrics-analytics",
    "module": "charts-validators",
    "role": "validator",
    "purpose": "Provides queryValidation function to parse and validate incoming Express request query parameters, ensuring they match expected formats and returning a strongly typed ProcessInput object.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "../../types",
        "express",
        "./validate_date_format",
        "./validate_numeric_format"
      ],
      "schema_types": ["ProcessInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["queryValidation"],
      "exported_symbols": ["queryValidation"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Throws error if validation fails."],
      "error_model": [
        "Throws error if startDate or endDate is missing or invalid format.",
        "Throws error if page or pageSize are not numeric values.",
        "Error messages are wrapped with 'Invalid query parameters:' prefix."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns a valid ProcessInput object if no error is thrown.",
        "Ensures startDate and endDate are valid date strings.",
        "Ensures page and pageSize are valid numeric values."
      ],
      "security": [

```
        "Prevents injection of invalid or malicious query parameters into pipeline construction."
      ]
    },
    "why": {
      "associations": [
        "Acts as input validation layer for chart endpoints.",
        "Protects downstream pipelines from invalid or malformed query parameters."
      ],
      "biases": ["Validation strictly enforces formats, rejecting anything unexpected."],
      "context": "Ensures safe and predictable inputs for analytics dashboards."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid query returns correct ProcessInput object.",
        "Invalid date format throws error.",
        "Non-numeric page or pageSize throws error.",
        "Missing query parameters throw error."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\types\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\common\\types.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts-validators", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts",
"kind": "file" },
      { "id": "validator:queryValidation", "kind": "validator" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:charts-validators" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts", "to":
"module-node:charts-validators" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts", "to":
"validator:queryValidation" },
      { "type": "IMPORTS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts", "to":
"file:call-metrics-analytics\\src\\dashboard\\charts\\types\\index.ts" }
    ]
  }
```

}

call-metrics-analytics\src\dashboard\charts\validators\input-validation\validate_date_format.ts

```
/**
 * Validates that a given string matches the ISO 8601 date format.
 * Example of expected format: 2022-09-01T00:00:00.000Z
 *
 * @param value - The date string to validate.
 * @throws - Error if the value does not match the expected ISO format.
 */
export function validateDateFormat(value: string): string {
  const isoRegex = /^\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}\.\d{3}Z$/;

  if (!isoRegex.test(value)) {
    throw new Error(
      ` must be in ISO 8601 format (e.g., 2022-09-01T00:00:00.000Z)`
    );
  }

  return value;
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_date_format.ts",
    "system": "call-metrics-analytics",
    "module": "charts-validators",
    "role": "validator",
    "purpose": "Provides validateDateFormat function to ensure date strings conform to the ISO 8601 format (YYYY-MM-DDTHH:mm:ss.sssZ).",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["validateDateFormat"],
      "exported_symbols": ["validateDateFormat"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Throws error if value does not match expected regex."],

```
      "error_model": [
        "Throws error with message 'must be in ISO 8601 format (e.g., ...)' if input does not
match regex."
      ]
    },
    "contracts": {
      "invariants": [
        "Accepts only ISO 8601 formatted strings with full precision and UTC (Z) indicator.",
        "Returns the input string unchanged if valid."
      ],
      "security": [
        "Prevents non-standard or malformed date strings from being used in database
queries."
      ]
    },
    "why": {
      "associations": [
        "Ensures consistency of date formats across all chart endpoints.",
        "Protects downstream MongoDB queries from invalid date inputs."
      ],
      "biases": ["Strict validation may reject valid but differently formatted ISO strings (e.g.,
without milliseconds)."],
      "context": "Used in input validation layer for chart analytics endpoints."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid ISO string '2022-09-01T00:00:00.000Z' passes validation.",
        "String missing milliseconds or Z suffix fails validation.",
        "Completely invalid date string fails validation."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts-validators", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_date_
format.ts", "kind": "file" },
      { "id": "validator:validateDateFormat", "kind": "validator" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:charts-validators" },
```

    { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_date_format.ts", "to": "module-node:charts-validators" },
    { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_date_format.ts", "to": "validator:validateDateFormat" }
  ]
 }
}

call-metrics-analytics\src\dashboard\charts\validators\input-validation\validate_numeric_format.ts

```
/**
 * Validates that a given value is a numeric integer (used for page and
pageSize).
 *
 * @param value - The value to validate.
 * @param fieldName - The name of the field (for error messaging).
 * @throws - Error if the value is not a valid integer.
 */
export function validateNumericValue(value: string): number {
  const numericRegex = /^\d+$/;

  if (!numericRegex.test(String(value))) {
    throw new Error(`Must be a numeric value.`);
  }
  return parseInt(value, 10);
}
```

{
 "cognitive_node": {
  "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_numeric_format.ts",
  "system": "call-metrics-analytics",
  "module": "charts-validators",
  "role": "validator",
  "purpose": "Provides validateNumericValue function to ensure inputs like page and pageSize are numeric integers.",
  "inputs": {
   "di_injections": [],
   "imports": [],
   "schema_types": [],
   "config_keys": []
  },
  "outputs": {

```
      "api_surface": ["validateNumericValue"],
      "exported_symbols": ["validateNumericValue"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Throws error if value is not a valid integer."],
      "error_model": [
        "Throws error with message 'Must be a numeric value.' if validation fails."
      ]
    },
    "contracts": {
      "invariants": [
        "Accepts only string values consisting entirely of digits.",
        "Returns an integer parsed in base 10 if valid."
      ],
      "security": [
        "Prevents injection of non-numeric values into pagination logic."
      ]
    },
    "why": {
      "associations": [
        "Ensures safe and predictable pagination parameters in chart endpoints.",
        "Prevents runtime errors in downstream MongoDB pipeline pagination stages."
      ],
      "biases": ["Strict regex rejects negative numbers, decimals, or formatted numbers."],
      "context": "Used in queryValidation to validate page and pageSize query parameters."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid integer string like '5' returns 5.",
        "Non-numeric string like 'abc' throws error.",
        "Decimal string like '3.5' throws error.",
        "Negative number string like '-2' throws error."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts-validators", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_nume
ric_format.ts", "kind": "file" },
      { "id": "validator:validateNumericValue", "kind": "validator" }
```

    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:charts-validators" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_numeric_format.ts", "to": "module-node:charts-validators" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\validate_numeric_format.ts", "to": "validator:validateNumericValue" }
    ]
  }
}

call-metrics-analytics\src\dashboard\charts\validators\index.ts

```
export { validateAndAdjustDailyRange } from "./validation_daily";
export { validateAndAdjustMonthlyRange } from "./validation_monthly";
export { validateAndAdjustYearlyRange } from "./validation_yearly";
export { queryValidation } from "./input-validation";
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\index.ts",
    "system": "call-metrics-analytics",
    "module": "charts-validators",
    "role": "barrel",
    "purpose": "Acts as an index barrel file re-exporting validation functions for daily, monthly, yearly ranges, and query parameter validation.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "./validation_daily",
        "./validation_monthly",
        "./validation_yearly",
        "./input-validation"
      ],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "validateAndAdjustDailyRange",
        "validateAndAdjustMonthlyRange",
        "validateAndAdjustYearlyRange",
        "queryValidation"
      ],
      "exported_symbols": [

```
        "validateAndAdjustDailyRange",
        "validateAndAdjustMonthlyRange",
        "validateAndAdjustYearlyRange",
        "queryValidation"
      ],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Always re-exports all validator functions for simplified imports across modules."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Centralizes exports for validation logic.",
        "Simplifies import paths for consumers in chart modules."
      ],
      "biases": [],
      "context": "Entry point for validator utilities in dashboard charts."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Importing from this index provides access to all validation functions.",
        "Functions maintain type safety and correct linkage to underlying files."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\input-validation\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts-validators", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\index.ts", "kind": "file"
},
      { "id": "barrel:validators-index", "kind": "barrel" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:charts-validators" },
```

    { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\index.ts", "to": "module-node:charts-validators" },
    { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\index.ts", "to": "barrel:validators-index" }
   ]
  }
}

call-metrics-analytics\src\dashboard\charts\validators\validation_daily.ts

```
/**
 * Validates and adjusts the date range for daily call data.
 *
 * FUNCTION:
 *     MAKE ALL THE NECESSARY VALIDATIONS TO THE INPUT: SECURITY,
LOGIC, TYPES, ETC.
 * CURRENT:
 *     The range is logically ordered (i.e., `startDate` is before
`endDate`).
 *
 * Unlike previous versions, this version no longer restricts the range
to 30 days,
 * allowing users to request daily data over extended periods.
 *
 * This change is important to support wider historical analysis,
 * and to prevent issues where the original date range was silently
overwritten
 * (which previously led to data from the earliest year being
returned).
 *
 * @param startDate - The requested start date in ISO format.
 * @param endDate - The requested end date in ISO format.
 * @returns A tuple `[validatedStartDate, validatedEndDate]` in ISO
string format.
 */
export function validateAndAdjustDailyRange(
  startDate: string,
  endDate: string
): [string, string] {
  let start = new Date(startDate);
  let end = new Date(endDate);
  // Ensure `startDate` is always before `endDate` by swapping if
necessary
  if (start > end) {
    [start, end] = [end, start];
  }

  return [start.toISOString(), end.toISOString()];
}
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_daily.ts",
    "system": "call-metrics-analytics",
    "module": "charts-validators",
    "role": "validator",
    "purpose": "Provides validateAndAdjustDailyRange function to validate and adjust daily
query date ranges, ensuring logical ordering without limiting the number of days.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["validateAndAdjustDailyRange"],
      "exported_symbols": ["validateAndAdjustDailyRange"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Swaps start and end dates if provided out of order."],
      "error_model": [
        "Invalid date strings may result in 'Invalid Date' objects, which when converted
toISOString() throw errors."
      ]
    },
    "contracts": {
      "invariants": [
        "Ensures returned dates are always logically ordered: start <= end.",
        "Returns both start and end dates in ISO 8601 string format."
      ],
      "security": [
        "Protects against misordered date inputs that could otherwise yield inconsistent query
behavior."
      ]
    },
    "why": {
      "associations": [
```

"Critical for ensuring accurate daily aggregations in pipelines.",
"Supports historical analysis over arbitrary ranges without restricting duration."
],
"biases": [
"Assumes JavaScript Date parsing works consistently across all ISO inputs.",
"Does not sanitize invalid ISO inputs beyond reordering."
],
"context": "Used in input validation for daily chart analytics to prevent logical errors in date ranges."
},
"tests": {
"spec_files": [],
"key_cases": [
"Valid ordered range returns unchanged ISO strings.",
"Swapped input dates return corrected order.",
"Invalid date string throws error during toISOString()."
]
},
"links_hint": [
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\index.ts"
]
},
"graph_patch": {
"nodes": [
{ "id": "system:call-metrics-analytics", "kind": "system" },
{ "id": "module-node:charts-validators", "kind": "module-node" },
{ "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_daily.ts", "kind": "file" },
{ "id": "validator:validateAndAdjustDailyRange", "kind": "validator" }
],
"edges": [
{ "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:charts-validators" },
{ "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_daily.ts", "to": "module-node:charts-validators" },
{ "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_daily.ts", "to": "validator:validateAndAdjustDailyRange" }
]
}
}

call-metrics-analytics\src\dashboard\charts\validators\validation_monthly.ts

```
/**
 * 📌 Validates and adjusts the date range for monthly call data.
 * Ensures:
```

```
 *  - Proper date format and logical order.
 *  - If the range exceeds 12 months, it adjusts to the first 12 months
from the start date.
 *  - If input is missing or invalid, it defaults to a valid 12-month
range.
 *
 * @param startDate - The requested start date in ISO format.
 * @param endDate - The requested end date in ISO format.
 * @returns A tuple `[validatedStartDate, validatedEndDate]` after
validation and adjustment.
 */
export function validateAndAdjustMonthlyRange(
  startDate: string,
  endDate: string
): [string, string] {
  // Handle missing dates by defaulting to a 12-month range ending
today
  if (!startDate || !endDate) {
    const now = new Date();
    const oneYearBefore = new Date(now);
    oneYearBefore.setMonth(oneYearBefore.getMonth() - 12);
    return [oneYearBefore.toISOString(), now.toISOString()];
  }

  let start = new Date(startDate);
  let end = new Date(endDate);

  //  Check if the input dates are invalid; default to a 12-month range
  if (isNaN(start.getTime()) || isNaN(end.getTime())) {
    const now = new Date();
    const oneYearBefore = new Date(now);
    oneYearBefore.setMonth(oneYearBefore.getMonth() - 12);
    return [oneYearBefore.toISOString(), now.toISOString()];
  }

  //  Ensure `startDate` is before `endDate`, swapping if necessary
  if (start > end) {
    [start, end] = [end, start];
  }

  //  Calculate the total number of months in the range
  const totalMonths =
    (end.getUTCFullYear() - start.getUTCFullYear()) * 12 +
    (end.getUTCMonth() - start.getUTCMonth());

  //  If the range exceeds 12 months, adjust it to 12 months from the
start date
  if (totalMonths > 12) {
    end = new Date(start);
    end.setMonth(start.getMonth() + 12);
  }

  return [start.toISOString(), end.toISOString()];
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_monthly.ts",
    "system": "call-metrics-analytics",
    "module": "charts-validators",
    "role": "validator",
    "purpose": "Provides validateAndAdjustMonthlyRange to validate and constrain monthly query ranges, ensuring a maximum of 12 months and logical ordering.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["validateAndAdjustMonthlyRange"],
      "exported_symbols": ["validateAndAdjustMonthlyRange"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Swaps start and end dates if reversed.", "Defaults to last 12 months if inputs are missing or invalid."],
      "error_model": [
        "Invalid ISO strings are caught and replaced with default 12-month range."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns a valid ISO range of start and end dates.",
        "Ensures start <= end.",
        "Caps maximum range to 12 months."
      ],
      "security": [
        "Protects against excessively long ranges that could overload the system."

```
      ]
    },
    "why": {
      "associations": [
        "Used in chart modules to validate monthly queries.",
        "Ensures consistent and efficient queries for monthly aggregations."
      ],
      "biases": ["Limits ranges strictly to 12 months, disallowing longer spans."],
      "context": "Supports analytics dashboards where monthly summaries are queried."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Missing dates defaults to last 12 months.",
        "Invalid date strings defaults to last 12 months.",
        "Swapped inputs corrected into proper order.",
        "Ranges longer than 12 months truncated to 12 months."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts-validators", "kind": "module-node" },
      { "id":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_monthly.ts", "kind":
"file" },
      { "id": "validator:validateAndAdjustMonthlyRange", "kind": "validator" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:charts-validators" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_monthly.ts", "to":
"module-node:charts-validators" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_monthly.ts", "to":
"validator:validateAndAdjustMonthlyRange" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\validators\validation_yearly.ts

```
/**
 * Validates and adjusts the date range for yearly call data.
 * Ensures:
 *  - Proper date format and logical order.
 *  - If the range exceeds 5 years, it adjusts to the last 5 years from
the end date.
 *  - If input is missing or invalid, it defaults to a valid 5-year
range.
 *
 * @param startDate - The requested start date in ISO format.
 * @param endDate - The requested end date in ISO format.
 * @returns A tuple `[validatedStartDate, validatedEndDate]` after
validation and adjustment.
 */
export function validateAndAdjustYearlyRange(
  startDate: string,
  endDate: string
): [string, string] {
  // Handle missing dates by defaulting to a 5-year range ending today
  if (!startDate || !endDate) {
    const now = new Date();
    const fiveYearsBefore = new Date(now);
    fiveYearsBefore.setFullYear(fiveYearsBefore.getFullYear() - 5);
    return [fiveYearsBefore.toISOString(), now.toISOString()];
  }

  let start = new Date(startDate);
  let end = new Date(endDate);

  // Check if the input dates are invalid; default to a 5-year range
  if (isNaN(start.getTime()) || isNaN(end.getTime())) {
    const now = new Date();
    const fiveYearsBefore = new Date(now);
    fiveYearsBefore.setFullYear(fiveYearsBefore.getFullYear() - 5);
    return [fiveYearsBefore.toISOString(), now.toISOString()];
  }

  // Ensure `startDate` is before `endDate`, swapping if necessary
  if (start > end) {
    [start, end] = [end, start];
  }

  // Calculate the total number of years in the range
  const totalYears = end.getUTCFullYear() - start.getUTCFullYear();
  // If the range exceeds 5 years, adjust it to the last 5 years from
the end date
  if (totalYears > 5) {
    start.setFullYear(end.getUTCFullYear() - 5);
  }

  return [start.toISOString(), end.toISOString()];
}
```

```
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_yearly.ts",
    "system": "call-metrics-analytics",
    "module": "charts-validators",
    "role": "validator",
    "purpose": "Provides validateAndAdjustYearlyRange function to validate and constrain
yearly query ranges, ensuring logical ordering and a maximum span of 5 years.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["validateAndAdjustYearlyRange"],
      "exported_symbols": ["validateAndAdjustYearlyRange"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Swaps start and end dates if reversed.", "Defaults to last 5 years if
inputs are missing or invalid."],
      "error_model": [
        "Invalid ISO strings are caught and replaced with a default 5-year range."
      ]
    },
    "contracts": {
      "invariants": [
        "Always returns a valid ISO range with start <= end.",
        "Caps the maximum range to 5 years.",
        "Returns values in ISO 8601 string format."
      ],
      "security": [
        "Protects system from queries spanning excessive year ranges that could cause
performance issues."
      ]
    },
    "why": {
      "associations": [
        "Ensures yearly chart data queries are valid and performant.",
        "Prevents user errors such as reversed or overly long ranges."
      ],
      "biases": ["Strictly limits range to 5 years, disallowing broader historical queries."],
      "context": "Used in analytics dashboards for yearly aggregated call data."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Missing dates default to last 5 years.",
        "Invalid dates default to last 5 years.",
        "Swapped dates corrected into proper order.",
        "Ranges longer than 5 years truncated to last 5 years from end date."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts-validators", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_yearly.ts",
"kind": "file" },
      { "id": "validator:validateAndAdjustYearlyRange", "kind": "validator" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:charts-validators" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_yearly.ts", "to":
"module-node:charts-validators" },
      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\validators\\validation_yearly.ts", "to":
"validator:validateAndAdjustYearlyRange" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\charts\index.ts

```
export { default as handlingOverview } from "./handling-overview";
export { default as medianCallDuration } from "./median-call-duration";
export { default as repeatedCallers } from "./repeated-callers";
export { default as callVolume } from "./call-volume";
export { default as callFrequencyOutcome } from
"./call-frequency-outcome";
export { default as aiOperationBreakdown } from
"./ai-operation-breakdown";
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\index.ts",
    "system": "call-metrics-analytics",
    "module": "charts",
    "role": "barrel",
    "purpose": "Acts as an index barrel file re-exporting all chart modules for simplified imports across the system.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "./handling-overview",
        "./median-call-duration",
        "./repeated-callers",
        "./call-volume",
        "./call-frequency-outcome",
        "./ai-operation-breakdown"
      ],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "handlingOverview",
        "medianCallDuration",
        "repeatedCallers",
        "callVolume",
        "callFrequencyOutcome",
        "aiOperationBreakdown"
      ],
      "exported_symbols": [
        "handlingOverview",
        "medianCallDuration",
        "repeatedCallers",
        "callVolume",
        "callFrequencyOutcome",
        "aiOperationBreakdown"
      ],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],

```
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Always re-exports all chart router modules as default exports."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Centralizes chart module exports for easier integration into higher-level modules like
app routing."
      ],
      "biases": [],
      "context": "Acts as entry point for all dashboard chart routes."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Importing from this index provides access to all chart router modules.",
        "Ensures consistent module exposure."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\charts\\index.ts", "kind": "file" },
      { "id": "barrel:charts-index", "kind": "barrel" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\charts\\index.ts", "to": "module-node:charts" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\charts\\index.ts",
"to": "barrel:charts-index" }
    ]
  }
}
```

call-metrics-analytics\src\dashboard\utils\format-date.ts

```
export function formatDate(date: string): string {
  const dateDivided = date.split("-");
  const elementCounter = dateDivided.length;

  const evaluatedDate = new Date(date);

  const yyyy = evaluatedDate.getUTCFullYear();
  const mm = String(evaluatedDate.getUTCMonth() + 1).padStart(2, "0");
  const dd = String(evaluatedDate.getUTCDate()).padStart(2, "0");

  switch (elementCounter) {
    case 1:
      return `${yyyy}`;
    case 2:
      return `${mm}/${yyyy}`;
    case 3:
      return `${mm}/${dd}/${yyyy}`;
    default:
      throw new Error("No such case");
  }
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\utils\\format-date.ts",
    "system": "call-metrics-analytics",
    "module": "utils",
    "role": "util",
    "purpose": "Provides formatDate function to format input date strings into human-readable strings based on granularity (yearly, monthly, daily).",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["formatDate"],
      "exported_symbols": ["formatDate"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Throws error if input date does not match expected split granularity."],

```
      "error_model": [
        "Invalid date strings may produce 'Invalid Date' and yield NaN components.",
        "Unexpected split length throws 'No such case' error."
      ]
    },
    "contracts": {
      "invariants": [
        "Input date must be in string format with components separated by '-' (YYYY,
YYYY-MM, or YYYY-MM-DD).",
        "Outputs formatted string depending on granularity: YYYY, MM/YYYY, or
MM/DD/YYYY."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Used by chart data processing utilities to produce consistent x-axis labels.",
        "Ensures date strings are presented in a user-friendly format depending on aggregation
level."
      ],
      "biases": ["Assumes only three possible cases (yearly, monthly, daily)."],
      "context": "Supports analytics dashboards by converting ISO dates to simplified chart
labels."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Input '2025' returns '2025'.",
        "Input '2025-07' returns '07/2025'.",
        "Input '2025-07-15' returns '07/15/2025'.",
        "Invalid or unexpected input throws error."
      ]
    },
    "links_hint": [

"file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\utils\\process-received
-data.ts",

"file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\utils\\process-rec
eived-data.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:utils", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\utils\\format-date.ts", "kind": "file" },
```

    { "id": "util:formatDate", "kind": "util" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:utils" },
    { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\utils\\format-date.ts", "to": "module-node:utils" },
    { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\utils\\format-date.ts", "to": "util:formatDate" }
  ]
 }
}

call-metrics-analytics\src\dashboard\utils\stringifyPipeline.ts

```
export function stringifyPipeline(pipeline: unknown) {
  const searchParams = new URLSearchParams();
  searchParams.append("pipeline", JSON.stringify(pipeline));

  return searchParams.toString();
}
```

{
 "cognitive_node": {
  "id": "file:call-metrics-analytics\\src\\dashboard\\utils\\stringifyPipeline.ts",
  "system": "call-metrics-analytics",
  "module": "utils",
  "role": "util",
  "purpose": "Provides stringifyPipeline function to serialize a MongoDB aggregation
pipeline into a query string format for use in HTTP requests.",
  "inputs": {
   "di_injections": [],
   "imports": [],
   "schema_types": [],
   "config_keys": []
  },
  "outputs": {
   "api_surface": ["stringifyPipeline"],
   "exported_symbols": ["stringifyPipeline"],
   "events_pubsub": []
  },
  "behavior": {
   "side_effects": [],
   "error_model": [
    "If pipeline contains circular references, JSON.stringify will throw an error.",
    "Large or deeply nested pipelines may result in overly long query strings."
   ]
  },
  "contracts": {

```
      "invariants": [
        "Always returns a valid URL-encoded query string with a 'pipeline' key.",
        "Pipeline must be serializable with JSON.stringify."
      ],
      "security": [
        "Encodes pipeline as a safe query string parameter, preventing raw injection into
URLs."
      ]
    },
    "why": {
      "associations": [
        "Used by multiple chart controllers to send pipelines to backend aggregation
endpoints.",
        "Provides consistent way to transmit pipeline definitions over HTTP GET requests."
      ],
      "biases": ["Assumes pipelines will fit into URL length limits."],
      "context": "Supports dashboard routes by converting pipeline objects to transportable
strings."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Valid pipeline array serialized to query string.",
        "Empty pipeline serialized correctly.",
        "Invalid (non-serializable) pipeline throws error."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\charts\\ai-operation-breakdown\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\handling-overview\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\median-call-duration\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\repeated-callers\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\call-frequency-outcome\\index.ts",
      "file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:utils", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\utils\\stringifyPipeline.ts", "kind": "file" },
      { "id": "util:stringifyPipeline", "kind": "util" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:utils" },
      { "type": "BELONGS_TO", "from":
"file:call-metrics-analytics\\src\\dashboard\\utils\\stringifyPipeline.ts", "to": "module-node:utils"
},
```

      { "type": "OWNS", "from":
"file:call-metrics-analytics\\src\\dashboard\\utils\\stringifyPipeline.ts", "to":
"util:stringifyPipeline" }
    ]
  }
}

call-metrics-analytics\src\dashboard\index.ts

```
export {
  handlingOverview,
  medianCallDuration,
  repeatedCallers,
  callVolume,
  callFrequencyOutcome,
  aiOperationBreakdown,
} from "./charts";

export { default as callsDurationSummary } from
"./calls-duration-summary";
export { default as carousel } from "./carousel";
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\dashboard\\index.ts",
    "system": "call-metrics-analytics",
    "module": "dashboard",
    "role": "barrel",
    "purpose": "Acts as an index barrel file re-exporting all dashboard modules, including chart
modules, calls duration summary, and carousel.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "./charts",
        "./calls-duration-summary",
        "./carousel"
      ],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "handlingOverview",
        "medianCallDuration",
        "repeatedCallers",
        "callVolume",
        "callFrequencyOutcome",

```
      "aiOperationBreakdown",
      "callsDurationSummary",
      "carousel"
    ],
    "exported_symbols": [
      "handlingOverview",
      "medianCallDuration",
      "repeatedCallers",
      "callVolume",
      "callFrequencyOutcome",
      "aiOperationBreakdown",
      "callsDurationSummary",
      "carousel"
    ],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": [],
    "error_model": []
  },
  "contracts": {
    "invariants": [
      "Always re-exports chart modules and dashboard feature modules as part of the unified
dashboard API surface."
    ],
    "security": []
  },
  "why": {
    "associations": [
      "Provides centralized access to all dashboard features for integration into higher-level
application modules."
    ],
    "biases": [],
    "context": "Acts as the main entry point for the dashboard package."
  },
  "tests": {
    "spec_files": [],
    "key_cases": [
      "Importing from this index provides access to all chart and dashboard modules.",
      "Ensures consistent module exposure for app routing."
    ]
  },
  "links_hint": [
    "file:call-metrics-analytics\\src\\dashboard\\charts\\index.ts",
    "file:call-metrics-analytics\\src\\dashboard\\calls-duration-summary\\index.ts",
    "file:call-metrics-analytics\\src\\dashboard\\carousel\\index.ts"
  ]
 },
```

  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:dashboard", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\dashboard\\index.ts", "kind": "file" },
      { "id": "barrel:dashboard-index", "kind": "barrel" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:dashboard"
},
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\dashboard\\index.ts",
"to": "module-node:dashboard" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\dashboard\\index.ts", "to":
"barrel:dashboard-index" }
    ]
  }
}

call-metrics-analytics\src\models\index.ts

```
import mongoose, { Model } from "mongoose";
import { Schema } from "mongoose";

interface RawCallRecord {
  callId: string;
  phoneNumber: string;
  callInitializationTime: Date;
  callFinalizationTime: Date;
  callReason: "scheduling" | "rescheduling" | "cancelation";
  completedByAI: boolean;
}

// Define the schema
export const callSchema: Schema = new Schema({
  callId: {
    type: String,
    required: true,
    unique: true,
    immutable: true,
  },
  phoneNumber: {
    type: String,
    required: true,
    immutable: true,
  },
  callInitializationTime: {
    type: Date,
    required: true,
    immutable: true,
  },
  callFinalizationTime: {
    type: Date,
    required: true,
    immutable: true,
  },
  callReason: {
    type: String,
    enum: ["scheduling"],
    required: true,
    immutable: true,
  },
  completedByAI: {
    type: Boolean,
    required: true,
    immutable: true,
  },
});

type SourceType = "ringCentral" | "generated";

/**
 * Dynamically returns the correct model based on the source type.
 */
export function getUnifiedModel(sourceType: SourceType):
Model<RawCallRecord> {
  if (sourceType === "ringCentral") {
    return (
      mongoose.models.CallFromRingCentral ||
      mongoose.model<RawCallRecord>(
        "CallFromRingCentral",
        callSchema,
        "callsFromRingCentral"
      )
    );
  }

  return (
    mongoose.models.CallInformation ||
    mongoose.model<RawCallRecord>(
      "CallInformation",
      callSchema,
      "callsInformation"
    )
  );
}

export const UnifiedModel = getUnifiedModel("generated");
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\models\\index.ts",
    "system": "call-metrics-analytics",
    "module": "models",
    "role": "model-definition",
    "purpose": "Defines the Mongoose schema and dynamic model retrieval for call records, supporting multiple data sources (ringCentral or generated).",
    "inputs": {
      "di_injections": [],
      "imports": ["mongoose", "Schema"],
      "schema_types": ["RawCallRecord"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["callSchema", "getUnifiedModel", "UnifiedModel"],
      "exported_symbols": ["callSchema", "getUnifiedModel", "UnifiedModel"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Registers Mongoose models dynamically at runtime."],
      "error_model": [
        "Mongoose will throw if schema definitions conflict with existing registered models."
      ]
    },
    "contracts": {
      "invariants": [
        "callId, phoneNumber, callInitializationTime, callFinalizationTime, callReason, and completedByAI must always exist in each record.",
        "callId and phoneNumber are immutable and unique per record."
      ],
      "security": [
        "Schema enforces immutability and uniqueness to prevent tampering with key record identifiers."
      ]
    },
    "why": {
      "associations": [

"Central entry point for Mongoose models in the system.",
"Ensures a unified data interface across different data origins (ringCentral and generated)."
],
"biases": [
"Currently restricts callReason enum in schema to 'scheduling' only, despite broader type definition."
],
"context": "Used by analytics and dashboard pipelines to query call record data consistently."
},
"tests": {
"spec_files": [],
"key_cases": [
"Correct model is returned when sourceType is 'ringCentral'.",
"Correct model is returned when sourceType is 'generated'.",
"Schema enforces required and immutable fields."
]
},
"links_hint": [
"file:call-metrics-analytics\\src\\dashboard\\charts\\call-volume\\model\\generate-call-volume-histogram.ts"
]
},
"graph_patch": {
"nodes": [
{ "id": "system:call-metrics-analytics", "kind": "system" },
{ "id": "module-node:models", "kind": "module-node" },
{ "id": "file:call-metrics-analytics\\src\\models\\index.ts", "kind": "file" },
{ "id": "schema:callSchema", "kind": "schema" },
{ "id": "function:getUnifiedModel", "kind": "function" },
{ "id": "model:UnifiedModel", "kind": "model" }
],
"edges": [
{ "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:models" },
{ "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\models\\index.ts", "to": "module-node:models" },
{ "type": "OWNS", "from": "file:call-metrics-analytics\\src\\models\\index.ts", "to": "schema:callSchema" },
{ "type": "OWNS", "from": "file:call-metrics-analytics\\src\\models\\index.ts", "to": "function:getUnifiedModel" },
{ "type": "OWNS", "from": "file:call-metrics-analytics\\src\\models\\index.ts", "to": "model:UnifiedModel" }
]
}
}

call-metrics-analytics\src\routes\index.ts

```
import express from "express";
import {
  handlingOverview,
  medianCallDuration,
  repeatedCallers,
  callVolume,
  callsDurationSummary,
  carousel,
  callFrequencyOutcome,
  aiOperationBreakdown,
} from "./../dashboard";

const router = express.Router();

router.use("/agreggated/handling-overview", handlingOverview);
router.use("/agreggated/median-duration", medianCallDuration);
router.use("/agreggated/call-volume", callVolume);
router.use("/agreggated/repeated-callers", repeatedCallers);
router.use("/agreggated/call-frequency-outcome", callFrequencyOutcome);
router.use("/agreggated/ai-operation-breakdown", aiOperationBreakdown);
//
router.use("/calls-duration-summary/handler-totals",
callsDurationSummary);

router.use("/call-time-data", carousel);

export default router;
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\routes\\index.ts",
    "system": "call-metrics-analytics",
    "module": "routes",
    "role": "router",
    "purpose": "Defines the main Express router entry point, wiring up all dashboard and analytics submodules into HTTP routes.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "./../dashboard"
      ],
      "schema_types": [],
      "config_keys": []

```
  },
  "outputs": {
    "api_surface": [
      "/agreggated/handling-overview",
      "/agreggated/median-duration",
      "/agreggated/call-volume",
      "/agreggated/repeated-callers",
      "/agreggated/call-frequency-outcome",
      "/agreggated/ai-operation-breakdown",
      "/calls-duration-summary/handler-totals",
      "/call-time-data"
    ],
    "exported_symbols": ["router"],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": ["Mounts multiple dashboard feature routers under aggregated paths."],
    "error_model": [
      "Relies on subrouters for error handling. If a subrouter throws, Express default error
handling applies."
    ]
  },
  "contracts": {
    "invariants": [
      "Each mounted route delegates to a corresponding dashboard module.",
      "Path naming conventions use '/agreggated/*' for analytics and
'/calls-duration-summary/*' for summaries."
    ],
    "security": [
      "Does not perform input validation directly; relies on submodules and validators."
    ]
  },
  "why": {
    "associations": [
      "Acts as a single entry point for all API routes in call-metrics-analytics system.",
      "Connects dashboard analytics pipelines with external clients via Express."
    ],
    "biases": [
      "Uses misspelling 'agreggated' instead of 'aggregated' in all route paths, which could
propagate to client usage."
    ],
    "context": "Provides REST endpoints that map to analytics and dashboard
computations."
  },
  "tests": {
    "spec_files": [],
    "key_cases": [
      "GET /agreggated/handling-overview routes to handlingOverview module.",
        "GET /calls-duration-summary/handler-totals routes to callsDurationSummary module.",
        "GET /call-time-data routes to carousel module."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\dashboard\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:routes", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\routes\\index.ts", "kind": "file" },
      { "id": "router:main-router", "kind": "router" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:routes" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\routes\\index.ts", "to":
"module-node:routes" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\routes\\index.ts", "to":
"router:main-router" }
    ]
  }
}
```

call-metrics-analytics\src\server.ts

```
import express, { Application } from "express";
import mongoose from "mongoose";
import bodyParser from "body-parser";
import dotenv from "dotenv";
import routes from "./routes";

dotenv.config();

const mongoURL = process.env.MONGO_URL;

// Initialize Express
const app: Application = express();
const PORT = process.env.PORT;

// Middleware
app.use(bodyParser.json()); // Only body-parser is needed

// MongoDB Connection
const mongoURI = mongoURL || "";
```

```
mongoose.connect(mongoURI, {} as mongoose.ConnectOptions);

// MongoDB Connection Success/Error Handlers
const db = mongoose.connection;
db.on("error", console.error.bind(console, "MongoDB connection
error:"));
db.once("open", () => {
  console.log("Connected to MongoDB");
});

app.use("/api/call-records", routes);

// Start the server
app.listen(PORT, (): void => {
  console.log(`Server is running on port: ${PORT}`);
});
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\src\\server.ts",
    "system": "call-metrics-analytics",
    "module": "server",
    "role": "entrypoint",
    "purpose": "Bootstraps the Express application, connects to MongoDB, mounts all routes, and starts the HTTP server.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "express",
        "mongoose",
        "body-parser",
        "dotenv",
        "./routes"
      ],
      "schema_types": [],
      "config_keys": ["MONGO_URL", "PORT"]
    },
    "outputs": {
      "api_surface": ["/api/call-records/*"],
      "exported_symbols": [],
      "events_pubsub": ["MongoDB connection events", "Express HTTP server"]
    },
    "behavior": {
      "side_effects": [
        "Establishes MongoDB connection on startup.",
        "Starts listening for HTTP requests on configured PORT."

```
      ],
      "error_model": [
        "Logs MongoDB connection errors to console.",
        "Throws runtime errors if PORT or MONGO_URL environment variables are missing or
invalid."
      ]
    },
    "contracts": {
      "invariants": [
        "MongoDB must be available and reachable for system functionality.",
        "Express must be configured with JSON body parsing before routes are mounted."
      ],
      "security": [
        "Relies on environment variables for sensitive data like database connection strings."
      ]
    },
    "why": {
      "associations": [
        "Entry point of the system tying together models, routes, and utilities.",
        "Ensures persistent database connection before serving analytics routes."
      ],
      "biases": [
        "Assumes a single database connection for all models.",
        "Assumes body-parser JSON middleware is sufficient for all request types."
      ],
      "context": "Provides runtime environment for analytics dashboard and API endpoints."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Successful MongoDB connection prints 'Connected to MongoDB'.",
        "Server starts and logs the configured port.",
        "Routes under /api/call-records/* return expected responses."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\routes\\index.ts",
      "file:call-metrics-analytics\\src\\models\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:server", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\src\\server.ts", "kind": "file" },
      { "id": "entrypoint:express-server", "kind": "entrypoint" }
    ],
    "edges": [
```

      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:server" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\src\\server.ts", "to":
"module-node:server" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\src\\server.ts", "to":
"entrypoint:express-server" }
    ]
  }
}

call-metrics-analytics\.env

```
MONGO_URL=mongodb://localhost:27017/data_analytics
PORT=5000
CALL_METRICS_RECORDS_URL = http://localhost:3002
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\.env",
    "system": "call-metrics-analytics",
    "module": "config",
    "role": "environment",
    "purpose": "Defines environment variables for MongoDB connection, server port, and
external call metrics records service URL.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": ["MONGO_URL", "PORT", "CALL_METRICS_RECORDS_URL"]
    },
    "outputs": {
      "api_surface": ["MONGO_URL", "PORT", "CALL_METRICS_RECORDS_URL"],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Loaded by dotenv to configure runtime environment."],
      "error_model": [
        "Missing or misconfigured variables may cause server startup failure or inability to
connect to MongoDB/external services."
      ]
    },
    "contracts": {
      "invariants": [
        "MONGO_URL must be a valid MongoDB connection string.",
        "PORT must be a valid integer for Express to bind on.",
        "CALL_METRICS_RECORDS_URL must be a valid URL to an external service."
      ],
      "security": [

```
        "Sensitive values like MongoDB connection strings must not be committed to public
repositories."
      ]
    },
    "why": {
      "associations": [
        "Used in server.ts for MongoDB connection and server port binding.",
        "Used in dashboard routes for communicating with external call records service."
      ],
      "biases": ["Configured for local development with hardcoded localhost addresses."],
      "context": "Centralizes environment configuration for analytics system."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Server starts correctly with MONGO_URL pointing to local DB.",
        "Routes can access external service via CALL_METRICS_RECORDS_URL.",
        "Changing PORT reflects in server listening log."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\server.ts",
      "file:call-metrics-analytics\\src\\routes\\index.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:config", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\.env", "kind": "file" },
      { "id": "config:MONGO_URL", "kind": "config" },
      { "id": "config:PORT", "kind": "config" },
      { "id": "config:CALL_METRICS_RECORDS_URL", "kind": "config" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to": "module-node:config" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\.env", "to":
"module-node:config" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\.env", "to": "config:MONGO_URL" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\.env", "to": "config:PORT" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\.env", "to":
"config:CALL_METRICS_RECORDS_URL" }
    ]
  }
}
```

call-metrics-analytics\Dockerfile

```
FROM node:23-slim AS builder
```

```dockerfile
WORKDIR /app

COPY package*.json ./

RUN npm install

COPY . .

RUN npm run build


FROM node:23-slim AS runtime

WORKDIR /app

COPY --from=builder /app/dist ./dist

COPY --from=builder /app/package*.json ./
# Install only production dependencies (excluding devDependencies)
RUN npm install --only=production --no-audit --prefer-offline
CMD ["node", "dist/server.js"]
```

```json
{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\Dockerfile",
    "system": "call-metrics-analytics",
    "module": "infrastructure",
    "role": "containerization",
    "purpose": "Defines a multi-stage Docker build for the call-metrics-analytics service, separating build and runtime environments for optimized image size.",
    "inputs": {
      "di_injections": [],
      "imports": ["node:23-slim"],
      "schema_types": [],
      "config_keys": ["package.json", "package-lock.json", "dist/server.js"]
    },
    "outputs": {
      "api_surface": ["Docker image for call-metrics-analytics service"],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Build stage installs all dependencies and compiles TypeScript to dist.",
        "Runtime stage installs only production dependencies."
      ],
      "error_model": [
        "Build fails if package.json or build script is invalid.",
        "Runtime container fails if dist/server.js is missing after build."
      ]
    },
    "contracts": {
      "invariants": [
        "Dist artifacts must exist before runtime stage.",
        "Only production dependencies are installed in runtime stage."
      ],
      "security": [
        "Uses --no-audit and --prefer-offline to reduce dependency audit overhead in runtime
stage."
      ]
    },
    "why": {
      "associations": [
        "Supports containerized deployment of analytics API.",
        "Ensures smaller runtime image by excluding build tools and devDependencies."
      ],
      "biases": ["Assumes Node.js v23 is stable and available in deployment environment."],
      "context": "Optimized for deployment pipelines that separate build and runtime
concerns."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Docker build succeeds and produces working dist/server.js runtime.",
        "Running container starts server on configured PORT."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\server.ts",
      "file:call-metrics-analytics\\package.json"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:infrastructure", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\Dockerfile", "kind": "file" },
      { "id": "container:Dockerfile", "kind": "container" }
    ],
    "edges": [
```

    { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:infrastructure" },
    { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\Dockerfile", "to":
"module-node:infrastructure" },
    { "type": "OWNS", "from": "file:call-metrics-analytics\\Dockerfile", "to":
"container:Dockerfile" }
   ]
  }
}

call-metrics-analytics\package.json

```
{
  "name": "backend",
  "version": "1.3.0",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node dist/server.js",
    "dev": "ts-node-dev src/server.ts",
    "build": "tsc"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "description": "",
  "dependencies": {
    "body-parser": "^1.20.3",
    "dotenv": "^16.4.7",
    "express": "^4.21.2",
    "mongoose": "^8.9.5"
  },
  "devDependencies": {
    "@types/body-parser": "^1.19.5",
    "@types/cors": "^2.8.17",
    "@types/express": "^5.0.0",
    "@types/node": "^22.10.7",
    "ts-node-dev": "^2.0.0",
    "typescript": "^5.7.3"
  }
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\package.json",

```
  "system": "call-metrics-analytics",
  "module": "infrastructure",
  "role": "package-config",
  "purpose": "Defines project metadata, npm scripts, dependencies, and development
tooling for the call-metrics-analytics backend service.",
  "inputs": {
    "di_injections": [],
    "imports": [],
    "schema_types": [],
    "config_keys": []
  },
  "outputs": {
    "api_surface": ["npm run dev", "npm run build", "npm start", "npm test"],
    "exported_symbols": [],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": ["Controls how project is built, tested, and run."],
    "error_model": [
      "`npm test` is currently a placeholder that always exits with error.",
      "Missing build artifacts in dist may cause npm start to fail."
    ]
  },
  "contracts": {
    "invariants": [
      "start script runs compiled JavaScript at dist/server.js.",
      "dev script runs TypeScript directly via ts-node-dev.",
      "build script compiles all TypeScript files via tsc."
    ],
    "security": []
  },
  "why": {
    "associations": [
      "Provides entry points for developer workflow.",
      "Defines dependencies required for runtime (Express, Mongoose, dotenv) and
devDependencies for TypeScript ecosystem."
    ],
    "biases": [
      "No testing framework configured, tests always fail.",
      "Project named 'backend' without detailed description."
    ],
    "context": "Enables Node.js + TypeScript backend development and container builds."
  },
  "tests": {
    "spec_files": [],
    "key_cases": [
      "npm run dev starts hot-reload server.",
      "npm run build generates dist folder.",
        "npm start successfully runs server from dist.",
        "npm test fails as expected until a real test framework is added."
      ]
    },
    "links_hint": [
      "file:call-metrics-analytics\\src\\server.ts",
      "file:call-metrics-analytics\\Dockerfile"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:infrastructure", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\package.json", "kind": "file" },
      { "id": "config:package-scripts", "kind": "config" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:infrastructure" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\package.json", "to":
"module-node:infrastructure" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\package.json", "to":
"config:package-scripts" }
    ]
  }
}
```

call-metrics-analytics\tsconfig.json

```
{
  "compilerOptions": {
    /* Visit https://aka.ms/tsconfig to read more about this file */
    /* Projects */
    // "incremental": true,                              /* Save
.tsbuildinfo files to allow for incremental compilation of projects. */
    // "composite": true,                                /* Enable
constraints that allow a TypeScript project to be used with project
references. */
    // "tsBuildInfoFile": "./.tsbuildinfo",              /* Specify the
path to .tsbuildinfo incremental compilation file. */
    // "disableSourceOfProjectReferenceRedirect": true,  /* Disable
preferring source files instead of declaration files when referencing
composite projects. */
    // "disableSolutionSearching": true,                 /* Opt a
project out of multi-project reference checking when editing. */
    // "disableReferencedProjectLoad": true,             /* Reduce the
number of projects loaded automatically by TypeScript. */
```

```
    /* Language and Environment */
    "target": "es2016" /* Set the JavaScript language version for
emitted JavaScript and include compatible library declarations. */,
    // "lib": [],                                        /* Specify a
set of bundled library declaration files that describe the target
runtime environment. */
    // "jsx": "preserve",                                /* Specify
what JSX code is generated. */
    // "experimentalDecorators": true,                   /* Enable
experimental support for legacy experimental decorators. */
    // "emitDecoratorMetadata": true,                    /* Emit
design-type metadata for decorated declarations in source files. */
    // "jsxFactory": "",                                 /* Specify the
JSX factory function used when targeting React JSX emit, e.g.
'React.createElement' or 'h'. */
    // "jsxFragmentFactory": "",                         /* Specify the
JSX Fragment reference used for fragments when targeting React JSX emit
e.g. 'React.Fragment' or 'Fragment'. */
    // "jsxImportSource": "",                            /* Specify
module specifier used to import the JSX factory functions when using
'jsx: react-jsx*'. */
    // "reactNamespace": "",                             /* Specify the
object invoked for 'createElement'. This only applies when targeting
'react' JSX emit. */
    // "noLib": true,                                    /* Disable
including any library files, including the default lib.d.ts. */
    // "useDefineForClassFields": true,                  /* Emit
ECMAScript-standard-compliant class fields. */
    // "moduleDetection": "auto",                        /* Control
what method is used to detect module-format JS files. */

    /* Modules */
    "module": "commonjs" /* Specify what module code is generated. */,
    "rootDir": "./src" /* Specify the root folder within your source
files. */,
    // "moduleResolution": "node10",                     /* Specify how
TypeScript looks up a file from a given module specifier. */
    // "baseUrl": "./",                                  /* Specify the
base directory to resolve non-relative module names. */
    // "paths": {},                                      /* Specify a
set of entries that re-map imports to additional lookup locations. */
```

```
    // "rootDirs": [],                                   /* Allow
multiple folders to be treated as one when resolving modules. */
    // "typeRoots": [],                                  /* Specify
multiple folders that act like './node_modules/@types'. */
    // "types": [],                                      /* Specify
type package names to be included without being referenced in a source
file. */
    // "allowUmdGlobalAccess": true,                     /* Allow
accessing UMD globals from modules. */
    // "moduleSuffixes": [],                             /* List of
file name suffixes to search when resolving a module. */
    // "allowImportingTsExtensions": true,               /* Allow
imports to include TypeScript file extensions. Requires
'--moduleResolution bundler' and either '--noEmit' or
'--emitDeclarationOnly' to be set. */
    // "resolvePackageJsonExports": true,                /* Use the
package.json 'exports' field when resolving package imports. */
    // "resolvePackageJsonImports": true,                /* Use the
package.json 'imports' field when resolving imports. */
    // "customConditions": [],                           /* Conditions
to set in addition to the resolver-specific defaults when resolving
imports. */
    // "resolveJsonModule": true,                        /* Enable
importing .json files. */
    // "allowArbitraryExtensions": true,                 /* Enable
importing files with any extension, provided a declaration file is
present. */
    // "noResolve": true,                                /* Disallow
'import's, 'require's or '<reference>'s from expanding the number of
files TypeScript should add to a project. */

    /* JavaScript Support */
    // "allowJs": true,                                  /* Allow
JavaScript files to be a part of your program. Use the 'checkJS' option
to get errors from these files. */
    // "checkJs": true,                                  /* Enable
error reporting in type-checked JavaScript files. */
    // "maxNodeModuleJsDepth": 1,                        /* Specify the
maximum folder depth used for checking JavaScript files from
'node_modules'. Only applicable with 'allowJs'. */

    /* Emit */
    // "declaration": true,                              /* Generate
.d.ts files from TypeScript and JavaScript files in your project. */
    // "declarationMap": true,                           /* Create
sourcemaps for d.ts files. */
    // "emitDeclarationOnly": true,                      /* Only output
d.ts files and not JavaScript files. */
    // "sourceMap": true,                                /* Create
source map files for emitted JavaScript files. */
    // "inlineSourceMap": true,                          /* Include
sourcemap files inside the emitted JavaScript. */
    // "outFile": "./",                                  /* Specify a
file that bundles all outputs into one JavaScript file. If
'declaration' is true, also designates a file that bundles all .d.ts
output. */
    "outDir": "./dist" /* Specify an output folder for all emitted
files. */,
    // "removeComments": true,                           /* Disable
emitting comments. */
    // "noEmit": true,                                   /* Disable
emitting files from a compilation. */
    // "importHelpers": true,                            /* Allow
importing helper functions from tslib once per project, instead of
including them per-file. */
    // "importsNotUsedAsValues": "remove",               /* Specify
emit/checking behavior for imports that are only used for types. */
    // "downlevelIteration": true,                       /* Emit more
compliant, but verbose and less performant JavaScript for iteration. */
    // "sourceRoot": "",                                 /* Specify the
root path for debuggers to find the reference source code. */
    // "mapRoot": "",                                    /* Specify the
location where debugger should locate map files instead of generated
locations. */
    // "inlineSources": true,                            /* Include
source code in the sourcemaps inside the emitted JavaScript. */
    // "emitBOM": true,                                  /* Emit a
UTF-8 Byte Order Mark (BOM) in the beginning of output files. */
    // "newLine": "crlf",                                /* Set the
newline character for emitting files. */
    // "stripInternal": true,                            /* Disable
emitting declarations that have '@internal' in their JSDoc comments. */
    // "noEmitHelpers": true,                            /* Disable
generating custom helper functions like '__extends' in compiled output.
*/
```

```
    // "noEmitOnError": true,                            /* Disable
emitting files if any type checking errors are reported. */
    // "preserveConstEnums": true,                       /* Disable
erasing 'const enum' declarations in generated code. */
    // "declarationDir": "./",                           /* Specify the
output directory for generated declaration files. */
    // "preserveValueImports": true,                     /* Preserve
unused imported values in the JavaScript output that would otherwise be
removed. */

    /* Interop Constraints */
    // "isolatedModules": true,                          /* Ensure that
each file can be safely transpiled without relying on other imports. */
    // "verbatimModuleSyntax": true,                     /* Do not
transform or elide any imports or exports not marked as type-only,
ensuring they are written in the output file's format based on the
'module' setting. */
    // "allowSyntheticDefaultImports": true,             /* Allow
'import x from y' when a module doesn't have a default export. */
    "esModuleInterop": true /* Emit additional JavaScript to ease
support for importing CommonJS modules. This enables
'allowSyntheticDefaultImports' for type compatibility. */,
    // "preserveSymlinks": true,                         /* Disable
resolving symlinks to their realpath. This correlates to the same flag
in node. */
    "forceConsistentCasingInFileNames": true /* Ensure that casing is
correct in imports. */,

    /* Type Checking */
    "strict": true /* Enable all strict type-checking options. */,
    // "noImplicitAny": true,                            /* Enable
error reporting for expressions and declarations with an implied 'any'
type. */
    // "strictNullChecks": true,                         /* When type
checking, take into account 'null' and 'undefined'. */
    // "strictFunctionTypes": true,                      /* When
assigning functions, check to ensure parameters and the return values
are subtype-compatible. */
    // "strictBindCallApply": true,                      /* Check that
the arguments for 'bind', 'call', and 'apply' methods match the
original function. */
    // "strictPropertyInitialization": true,             /* Check for
class properties that are declared but not set in the constructor. */
```

```
    // "noImplicitThis": true,                            /* Enable
error reporting when 'this' is given the type 'any'. */
    // "useUnknownInCatchVariables": true,                /* Default
catch clause variables as 'unknown' instead of 'any'. */
    // "alwaysStrict": true,                              /* Ensure 'use
strict' is always emitted. */
    // "noUnusedLocals": true,                            /* Enable
error reporting when local variables aren't read. */
    // "noUnusedParameters": true,                        /* Raise an
error when a function parameter isn't read. */
    // "exactOptionalPropertyTypes": true,                /* Interpret
optional property types as written, rather than adding 'undefined'. */
    // "noImplicitReturns": true,                         /* Enable
error reporting for codepaths that do not explicitly return in a
function. */
    // "noFallthroughCasesInSwitch": true,                /* Enable
error reporting for fallthrough cases in switch statements. */
    // "noUncheckedIndexedAccess": true,                  /* Add
'undefined' to a type when accessed using an index. */
    // "noImplicitOverride": true,                        /* Ensure
overriding members in derived classes are marked with an override
modifier. */
    // "noPropertyAccessFromIndexSignature": true,        /* Enforces
using indexed accessors for keys declared using an indexed type. */
    // "allowUnusedLabels": true,                         /* Disable
error reporting for unused labels. */
    // "allowUnreachableCode": true,                      /* Disable
error reporting for unreachable code. */

    /* Completeness */
    // "skipDefaultLibCheck": true,                       /* Skip type
checking .d.ts files that are included with TypeScript. */
    "skipLibCheck": true /* Skip type checking all .d.ts files. */
  }
}
```

{
  "cognitive_node": {
    "id": "file:call-metrics-analytics\\tsconfig.json",
    "system": "call-metrics-analytics",
    "module": "infrastructure",
    "role": "compiler-config",
    "purpose": "Configures the TypeScript compiler for the call-metrics-analytics backend, defining compilation targets, module system, paths, and strictness settings.",

```
  "inputs": {
    "di_injections": [],
    "imports": [],
    "schema_types": [],
    "config_keys": []
  },
  "outputs": {
    "api_surface": ["./dist/* compiled JS artifacts"],
    "exported_symbols": [],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": ["Controls how TypeScript source is compiled into JavaScript."],
    "error_model": [
      "Misconfiguration may result in compilation errors or missing output files.",
      "Strict mode may reject unsafe or loosely typed code."
    ]
  },
  "contracts": {
    "invariants": [
      "Compiles TypeScript in ./src to JavaScript in ./dist.",
      "Targets ECMAScript 2016 using CommonJS modules.",
      "Applies strict type-checking rules and enforces case sensitivity in imports."
    ],
    "security": []
  },
  "why": {
    "associations": [
      "Ensures consistent build outputs for Dockerfile and npm scripts.",
      "Balances modern TypeScript strictness with compatibility for Node.js runtime."
    ],
    "biases": [
      "Disables full lib checking with skipLibCheck for faster builds at the cost of missing
some type errors."
    ],
    "context": "Forms the foundation for developer and CI/CD compilation of the backend
service."
  },
  "tests": {
    "spec_files": [],
    "key_cases": [
      "tsc compiles without error and emits to dist.",
      "Strict mode errors appear for type violations.",
      "skipLibCheck prevents type errors in @types packages from breaking builds."
    ]
  },
  "links_hint": [
    "file:call-metrics-analytics\\package.json",
      "file:call-metrics-analytics\\Dockerfile"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:call-metrics-analytics", "kind": "system" },
      { "id": "module-node:infrastructure", "kind": "module-node" },
      { "id": "file:call-metrics-analytics\\tsconfig.json", "kind": "file" },
      { "id": "config:tsconfig", "kind": "config" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:call-metrics-analytics", "to":
"module-node:infrastructure" },
      { "type": "BELONGS_TO", "from": "file:call-metrics-analytics\\tsconfig.json", "to":
"module-node:infrastructure" },
      { "type": "OWNS", "from": "file:call-metrics-analytics\\tsconfig.json", "to": "config:tsconfig"
}
    ]
  }
}
```