api-gateway.pdf — 1) RULES OF OPERATION FOR THE NODE (v0.1)

PURPOSE OF THIS NODE
- This document acts as the root knowledge node of the “API-GATEWAY” project.
- Its function is to govern creation, integration, and relationships of all NODES (files) via “cognitive memories” and their insertion into the project graph.

FORMAT OF EACH NODE IN THE DOCUMENT
- PATH: exact file path inside the repository.
- CODE: raw file content (verbatim).
- COGNITIVE-MEMORY: output produced by the active cognitive module for memory creation (see “COGNITIVE MODULES”).

COGNITIVE MODULES (pluggable by reference)
- 1) CREATION OF “COGNITIVE MEMORIES”
  When the user requests to create a cognitive memory for code, generate it according to:
  cognitive-modules/cognitive-memory-creation.pdf
  (Input: { path, code, system? }. Single deterministic output: { cognitive_node, graph_patch }. Upsert semantics for the graph. No non-evident inferences. English by default.)
- Future extensions: to change or add modules, replace or add the referenced path. Each module writes into its own semantic space and does not overwrite others unless explicitly ordered.

GRAPH VOCABULARY (allowed)
- Node.kind ∈ { system, module-node, file, class, service, resolver, module, graphql, dto, type, config, guard, decorator, topic }
- Edge.type ∈ { OWNS, BELONGS_TO, IMPORTS, INJECTS, EXPOSES, USES_TYPE, CONFIG, PUBSUB, CALLS }

INTEGRATION CONTRACTS (invariants)
1. Determinism: identical input produces identical output.
2. Upsert: each graph_patch creates or merges; it does not delete without explicit instruction.
3. Safety: do not invent secrets or configuration values; only reference logical keys.
4. Response style: always return a single “code snippet” block, in English by default, with no surrounding prose.
5. No asynchronous promises: perform all work within the current response.

IDENTITY AND NAMING RULES
- Each file is identified as: id = "file:<PATH>".
- Root system id: "system:API-GATEWAY".
- Logical domain modules (when present as directories) use "module-node:<name>".
- Exported classes and services use nodes "class:<Name>", "service:<Name>", etc.

OWNERSHIP AND RELATIONSHIP RULES
- OWNS: system:API-GATEWAY → module-node:<module>.
- BELONGS_TO: file:<path> → module-node:<module> (omit if no module can be inferred; the file belongs only to the system).
- OWNS (file → class/service): file:<path> → service|class:<Name>.

- IMPORTS: file A → file B when clear relative imports exist.
- INJECTS: class/service A → class/service B or token when DI usage is identified.
- EXPOSES: resolver/controller/endpoint → public operations identified.
- USES_TYPE: resolver/service → dto/type/graphql used in signatures.
- CONFIG: class/service → config:<KEY> when configuration keys are read.
- PUBSUB: class/service ↔ topic:<name> with annotation "publish" or "subscribe".
- CALLS: service A → service B when explicit invocation exists.

NODE LIFECYCLE (per file)
1) Intake: { path, code, system? } (default system is "API-GATEWAY").
2) Extraction: the module produces "cognitive_node" (purpose, inputs, outputs, contracts, etc.).
3) Integration: apply "graph_patch" (minimal nodes, ownership, dependencies, public surface).
4) Registry in api-gateway.pdf: store PATH, CODE, and COGNITIVE-MEMORY in that order.

VALIDATION AND AMBIGUITIES
- If no module can be inferred from the path, leave module empty in cognitive_node and do not create a module-node; keep only the system relationship.
- If evidence is insufficient for imports/DI/types/config, keep lists empty and add a hint in links_hint.
- Auto-generated files (for example, GraphQL types) must be marked in “why.context” as “autogenerated” and modeled passively (no inferred logic).
- Do not record secrets or sensitive values; only logical names of config keys.
- On graph-merge conflicts (duplicate nodes), the most specific id prevails (file:… > class:… > module-node:… > system:…).

GOVERNANCE AND VERSIONING
- Version of these rules: v0.1 (default language updated to English). Future changes must declare diffs and rationale.
- Each “cognitive_node” may include “tests.spec_files” and “key_cases” to track coverage.
- The api-gateway.pdf document acts as the editorial source of truth for the project graph.

EXPECTED RESPONSE FORMAT WHEN CREATING NODES (to paste into api-gateway.pdf)
- A single JSON block with exactly these top-level keys: { "cognitive_node": {…}, "graph_patch": {…} }.
- Do not include text outside the JSON block.

2) NODES (Path, code, cognitive memory)
src\modules\call-records\config\call-records.config.ts

```
import { registerAs } from '@nestjs/config';

export type CallRecordsConfig = typeof callRecordsConfig;
const callRecordsConfig = registerAs('call-records', () => {
  return {
    callRecordsBaseURL: process.env.CALL_RECORDS_BASE_URL!,
  };
});
export default callRecordsConfig;
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\config\\call-records.config.ts",
    "kind": "file",
    "system": "API-GATEWAY",
    "module": "call-records",
    "purpose": "Registers a namespaced configuration provider for the Call Records domain via Nest Config, exposing the base URL consumed by downstream services.",
    "exports": {
      "default": "callRecordsConfig",
      "types": ["CallRecordsConfig"]
    },
    "config_namespace": "call-records",
    "defines_config_keys": ["CALL_RECORDS_BASE_URL"],
    "reads_environment": ["CALL_RECORDS_BASE_URL"],
    "contracts": {
      "requires_env": ["CALL_RECORDS_BASE_URL"],
      "deterministic": true,
      "no_side_effects": true
    },
    "links_hint": {
      "imports_external": ["@nestjs/config"],
      "usage_example": "configService.get<string>('call-records.callRecordsBaseURL')"
    },
    "why": {
      "context": "The file calls registerAs('call-records') and returns { callRecordsBaseURL: process.env.CALL_RECORDS_BASE_URL! }, exporting the provider as default and a type alias for convenience."
    }
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
```

    { "id": "file:src\\modules\\call-records\\config\\call-records.config.ts", "kind": "file" },
    { "id": "config:call-records", "kind": "config" },
    { "id": "config:CALL_RECORDS_BASE_URL", "kind": "config" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
    { "type": "BELONGS_TO", "from":
"file:src\\modules\\call-records\\config\\call-records.config.ts", "to":
"module-node:call-records" },
    { "type": "EXPOSES", "from":
"file:src\\modules\\call-records\\config\\call-records.config.ts", "to": "config:call-records" },
    { "type": "EXPOSES", "from":
"file:src\\modules\\call-records\\config\\call-records.config.ts", "to":
"config:CALL_RECORDS_BASE_URL" }
  ]
 }
}

C:\Users\Jorge\Desktop\api-gateway\api-gateway\src\modules\call-records\constants\ call-records.constants.ts

```
export const ONGOING_CALL_STATUSES = ['ongoing-bot', 'ongoing-agent']
as const;
export const TERMINATED_CALL_STATUSES = [
  'terminated-bot',
  'terminated-agent',
  'terminated-caller',
] as const;
export const CALL_STATUSES = [
  ...ONGOING_CALL_STATUSES,
  ...TERMINATED_CALL_STATUSES,
] as const;

export const CALL_OUTCOMES = [
  'scheduled',
  'rescheduled',
  'cancellation',
  'handled-by-agent',
  'not-available',
] as const;

export const SENTIMENT_VALUES = ['negative', 'neutral', 'positive'] as
const;
export const RATING_VALUES = [1, 2, 3, 4, 5] as const;
```

{

```
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\constants\\call-records.constants.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "types",
    "purpose": "Defines constant sets representing call statuses, outcomes, sentiment values,
and rating values for the Call Records domain.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [
        "ONGOING_CALL_STATUSES",
        "TERMINATED_CALL_STATUSES",
        "CALL_STATUSES",
        "CALL_OUTCOMES",
        "SENTIMENT_VALUES",
        "RATING_VALUES"
      ],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Status and outcome arrays must remain aligned with backend or business logic
expectations."
      ],
      "security": []
    },
    "why": {
      "associations": ["Centralized domain constants simplify validation and comparisons."],
      "biases": ["Uses 'as const' to preserve literal union types in TypeScript."],
      "context": "File exists to standardize call-related enums across the call-records module."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": ["Likely referenced by services, DTOs, or resolvers that check call state or
outcome."]
  },
```

  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\constants\\call-records.constants.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\call-records\\constants\\call-records.constants.ts", "to":
"module-node:call-records" }
    ]
  }
}

src\modules\call-records\dto\create-call-record.input.ts

```
import { MinLength } from 'class-validator';
import * as GraphQLTypes from '../../../graphql';

export class CreateCallRecordInput extends
GraphQLTypes.CreateCallRecordInput {}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\dto\\create-call-record.input.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "dto",
    "purpose": "Defines a DTO class for creating a CallRecord, extending the auto-generated
GraphQL input type.",
    "inputs": {
      "di_injections": [],
      "imports": ["class-validator", "../../../graphql"],
      "schema_types": ["GraphQLTypes.CreateCallRecordInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["input:CreateCallRecordInput"],
      "exported_symbols": ["CreateCallRecordInput"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {

    "invariants": ["DTO must align with the GraphQL schema definition of CreateCallRecordInput."],
    "security": ["Validation decorators may be applied to enforce input constraints."]
   },
   "why": {
    "associations": ["Bridges between GraphQL schema and NestJS validation layer."],
    "biases": ["Keeps consistency with generated GraphQL types instead of redefining fields."],
    "context": "Part of DTO layer in call-records module to integrate with GraphQL inputs."
   },
   "tests": {
    "spec_files": [],
    "key_cases": ["Validation of minimal length constraints if applied."]
   },
   "links_hint": ["resolver:CallRecordsResolver", "graphql:CreateCallRecordInput"]
  },
  "graph_patch": {
   "nodes": [
    { "id": "system:API-GATEWAY", "kind": "system" },
    { "id": "module-node:call-records", "kind": "module-node" },
    { "id": "file:src\\modules\\call-records\\dto\\create-call-record.input.ts", "kind": "file" },
    { "id": "dto:CreateCallRecordInput", "kind": "dto" }
   ],
   "edges": [
    { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
    { "type": "BELONGS_TO", "from": "file:src\\modules\\call-records\\dto\\create-call-record.input.ts", "to": "module-node:call-records" },
    { "type": "OWNS", "from": "file:src\\modules\\call-records\\dto\\create-call-record.input.ts", "to": "dto:CreateCallRecordInput" },
    { "type": "USES_TYPE", "from": "dto:CreateCallRecordInput", "to": "graphql:CreateCallRecordInput" }
   ]
  }
}

src\modules\call-records\dto\notify-call-record-created.input.ts

```
import { MinLength } from 'class-validator';
import * as GraphQLTypes from '../../../graphql';

export class NotifyCallRecordCreatedInput extends
GraphQLTypes.NotifyCallRecordCreatedInput {}
```

{
  "cognitive_node": {
   "id": "file:src\\modules\\call-records\\dto\\notify-call-record-created.input.ts",
   "system": "API-GATEWAY",

```
    "module": "call-records",
    "role": "dto",
    "purpose": "Defines a DTO class for notifying when a CallRecord has been created,
extending the auto-generated GraphQL input type.",
    "inputs": {
      "di_injections": [],
      "imports": ["class-validator", "../../../graphql"],
      "schema_types": ["GraphQLTypes.NotifyCallRecordCreatedInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["input:NotifyCallRecordCreatedInput"],
      "exported_symbols": ["NotifyCallRecordCreatedInput"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": ["DTO must remain aligned with the GraphQL schema definition of
NotifyCallRecordCreatedInput."],
      "security": ["Validation decorators may be applied if constraints are introduced."]
    },
    "why": {
      "associations": ["Provides a typed input class for notification events in GraphQL
mutations or subscriptions."],
      "biases": ["Extends generated GraphQL type rather than duplicating structure."],
      "context": "Part of the call-records module DTO layer to integrate GraphQL schema
inputs with NestJS."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": ["resolver:CallRecordsResolver", "graphql:NotifyCallRecordCreatedInput"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\dto\\notify-call-record-created.input.ts", "kind": "file"
},
      { "id": "dto:NotifyCallRecordCreatedInput", "kind": "dto" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
    { "type": "BELONGS_TO", "from":
"file:src\\modules\\call-records\\dto\\notify-call-record-created.input.ts", "to":
"module-node:call-records" },
    { "type": "OWNS", "from":
"file:src\\modules\\call-records\\dto\\notify-call-record-created.input.ts", "to":
"dto:NotifyCallRecordCreatedInput" },
    { "type": "USES_TYPE", "from": "dto:NotifyCallRecordCreatedInput", "to":
"graphql:NotifyCallRecordCreatedInput" }
   ]
  }
}
```

src\modules\call-records\dto\notify-call-record-updated.input.ts

```
import { MinLength } from 'class-validator';
import * as GraphQLTypes from '../../../graphql';

export class NotifyCallRecordUpdatedInput extends
GraphQLTypes.NotifyCallRecordUpdatedInput {}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\dto\\notify-call-record-updated.input.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "dto",
    "purpose": "Defines a DTO class for notifying when a CallRecord has been updated,
extending the auto-generated GraphQL input type.",
    "inputs": {
      "di_injections": [],
      "imports": ["class-validator", "../../../graphql"],
      "schema_types": ["GraphQLTypes.NotifyCallRecordUpdatedInput"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["input:NotifyCallRecordUpdatedInput"],
      "exported_symbols": ["NotifyCallRecordUpdatedInput"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": ["DTO must stay aligned with the GraphQL schema definition of
NotifyCallRecordUpdatedInput."],
      "security": ["Validation decorators may be introduced if stricter input rules are required."]
    },
```

    "why": {
      "associations": ["Supplies a strongly typed input class for notification events in GraphQL mutations or subscriptions."],
      "biases": ["Extends generated GraphQL types directly rather than redefining properties."],
      "context": "Part of the call-records module DTO layer for integrating GraphQL schema inputs with NestJS."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": ["resolver:CallRecordsResolver", "graphql:NotifyCallRecordUpdatedInput"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\dto\\notify-call-record-updated.input.ts", "kind": "file" },
      { "id": "dto:NotifyCallRecordUpdatedInput", "kind": "dto" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\call-records\\dto\\notify-call-record-updated.input.ts", "to": "module-node:call-records" },
      { "type": "OWNS", "from": "file:src\\modules\\call-records\\dto\\notify-call-record-updated.input.ts", "to": "dto:NotifyCallRecordUpdatedInput" },
      { "type": "USES_TYPE", "from": "dto:NotifyCallRecordUpdatedInput", "to": "graphql:NotifyCallRecordUpdatedInput" }
    ]
  }
}

src\modules\call-records\dto\update-call-record.input.ts

```
import { MinLength } from 'class-validator';
import * as GraphQLTypes from '../../../graphql';

export class UpdateCallRecordInput extends
GraphQLTypes.UpdateCallRecordInput {}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\dto\\update-call-record.input.ts",
    "system": "API-GATEWAY",
    "module": "call-records",

"role": "dto",
"purpose": "Defines a DTO class for updating a CallRecord, extending the auto-generated GraphQL input type.",
"inputs": {
"di_injections": [],
"imports": ["class-validator", "../../../graphql"],
"schema_types": ["GraphQLTypes.UpdateCallRecordInput"],
"config_keys": []
},
"outputs": {
"api_surface": ["input:UpdateCallRecordInput"],
"exported_symbols": ["UpdateCallRecordInput"],
"events_pubsub": []
},
"behavior": {
"side_effects": [],
"error_model": []
},
"contracts": {
"invariants": ["DTO must remain aligned with the GraphQL schema definition of UpdateCallRecordInput."],
"security": ["Validation decorators may be added to enforce stricter constraints if needed."]
},
"why": {
"associations": ["Bridges GraphQL schema input for updates into the NestJS application layer."],
"biases": ["Extends generated GraphQL types directly to avoid redundancy."],
"context": "Part of the call-records module DTO layer, integrating GraphQL schema inputs with NestJS services and resolvers."
},
"tests": {
"spec_files": [],
"key_cases": []
},
"links_hint": ["resolver:CallRecordsResolver", "graphql:UpdateCallRecordInput"]
},
"graph_patch": {
"nodes": [
{ "id": "system:API-GATEWAY", "kind": "system" },
{ "id": "module-node:call-records", "kind": "module-node" },
{ "id": "file:src\\modules\\call-records\\dto\\update-call-record.input.ts", "kind": "file" },
{ "id": "dto:UpdateCallRecordInput", "kind": "dto" }
],
"edges": [
{ "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },

      { "type": "BELONGS_TO", "from":
"file:src\\modules\\call-records\\dto\\update-call-record.input.ts", "to":
"module-node:call-records" },
      { "type": "OWNS", "from":
"file:src\\modules\\call-records\\dto\\update-call-record.input.ts", "to":
"dto:UpdateCallRecordInput" },
      { "type": "USES_TYPE", "from": "dto:UpdateCallRecordInput", "to":
"graphql:UpdateCallRecordInput" }
    ]
  }
}

C:\Users\Jorge\Desktop\api-gateway\api-gateway\src\modules\call-records\graphql\call-records.graphql

```
type Mutation {
  createCallRecord(input: CreateCallRecordInput!): CallRecord
  updateCallRecord(input: UpdateCallRecordInput!): CallRecord
  notifyCallRecordCreated(input: NotifyCallRecordCreatedInput!):
CallRecord
  notifyCallRecordUpdated(input: NotifyCallRecordUpdatedInput!):
CallRecord
}

input CreateCallRecordInput {
  callId: String!
  callStartTimeStr: String!
  callerPhoneNumber: String!
}

input UpdateCallRecordInput {
  callId: String!
  callStatus: String
  callTransferredTimeStr: String
  agent: AgentAsInput
  callEndTimeStr: String
  callOutcome: String
}

input AgentAsInput {
  fullName: String
}

input PatientAsInput {
  fullName: String
```

```
}

input SentimentAsInput {
  value: String
  analysis: String
}

input RatingAsInput {
  value: Int
  info: String
}

input NotifyCallRecordCreatedInput {
  callRecord: CallRecordAsInput!
}

input CallRecordAsInput {
  id: ID!
  callId: String!
  callerPhoneNumber: String!
  callStartTime: String!
  callTransferredTime: String
  callEndTime: String
  patient: PatientAsInput!
  agent: AgentAsInput!
  callStatus: String!
  callOutcome: String!
  sentiment: SentimentAsInput!
  botRating: RatingAsInput!
  agentRating: RatingAsInput!
  cleanTranscription: [String!]
  recordingUrl: String
}

input NotifyCallRecordUpdatedInput {
  updateKind: String!
  callRecord: CallRecordAsInput!
}

type CallRecord {
  id: ID!
  callId: String!
  callerPhoneNumber: String!
  callStartTime: String!
  callTransferredTime: String
  callEndTime: String
  patient: Patient!
  agent: Agent!
  callStatus: String!
  callOutcome: String!
  sentiment: Sentiment!
  botRating: Rating!
  agentRating: Rating!
  cleanTranscription: [String!]
  recordingUrl: String
}

type Agent {
  fullName: String
}

type Patient {
  fullName: String
}

type Sentiment {
  value: String
  analysis: String
}

type Rating {
  value: Int
  info: String
}

type Subscription {
  callRecordCreated: CallRecordCreatedEventData
  callRecordUpdated(updateKinds: [String!]): CallRecordUpdatedEventData
}

type CallRecordCreatedEventData {
  callRecord: CallRecord!
}

type CallRecordUpdatedEventData {
  updateKind: String!
  callRecord: CallRecord!
}
```

```
{
  "cognitive_node": {
    "id":
"file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphq
l\\call-records.graphql",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "graphql",
    "purpose": "Defines the GraphQL schema for Call Records, including mutations, inputs,
types, and subscriptions related to call lifecycle events.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [
        "CreateCallRecordInput",
        "UpdateCallRecordInput",
        "NotifyCallRecordCreatedInput",
        "NotifyCallRecordUpdatedInput",
        "AgentAsInput",
        "PatientAsInput",
        "SentimentAsInput",
        "RatingAsInput",
        "CallRecordAsInput"
      ],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "mutation:createCallRecord",
        "mutation:updateCallRecord",
        "mutation:notifyCallRecordCreated",
        "mutation:notifyCallRecordUpdated",
        "subscription:callRecordCreated",
        "subscription:callRecordUpdated"
      ],
      "exported_symbols": [
        "CallRecord",
        "Agent",
        "Patient",
        "Sentiment",
        "Rating",
        "CallRecordCreatedEventData",
        "CallRecordUpdatedEventData"
      ],
      "events_pubsub": [
        "topic:callRecordCreated",
        "topic:callRecordUpdated"
      ]
    },
    "behavior": {
      "side_effects": ["PubSub"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["Schema definitions must align with DTO classes and resolver
implementations."],
      "security": ["Validation of inputs delegated to DTO layer and class-validator decorators."]
    },
    "why": {
      "associations": ["Schema provides the shared contract for API operations related to call
records."],
      "biases": ["Defines types explicitly instead of relying on inference for clarity and
compatibility."],
      "context": "Central schema file of the call-records module to expose mutations and
subscriptions to clients."
    },
    "tests": {
      "spec_files": [],
      "key_cases": ["Subscription events must deliver CallRecordCreatedEventData and
CallRecordUpdatedEventData payloads."]
    },
    "links_hint": [
      "dto:CreateCallRecordInput",
      "dto:UpdateCallRecordInput",
      "dto:NotifyCallRecordCreatedInput",
      "dto:NotifyCallRecordUpdatedInput",
      "resolver:CallRecordsResolver"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id":
"file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphq
l\\call-records.graphql", "kind": "file" },
      { "id": "graphql:CallRecord", "kind": "graphql" },
      { "id": "graphql:CreateCallRecordInput", "kind": "graphql" },
      { "id": "graphql:UpdateCallRecordInput", "kind": "graphql" },
      { "id": "graphql:NotifyCallRecordCreatedInput", "kind": "graphql" },
      { "id": "graphql:NotifyCallRecordUpdatedInput", "kind": "graphql" }
    ],
```

    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphql\\call-records.graphql", "to": "module-node:call-records" },
      { "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphql\\call-records.graphql", "to": "graphql:CallRecord" },
      { "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphql\\call-records.graphql", "to": "graphql:CreateCallRecordInput" },
      { "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphql\\call-records.graphql", "to": "graphql:UpdateCallRecordInput" },
      { "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphql\\call-records.graphql", "to": "graphql:NotifyCallRecordCreatedInput" },
      { "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\call-records\\graphql\\call-records.graphql", "to": "graphql:NotifyCallRecordUpdatedInput" }
    ]
  }
}

src\modules\call-records\resolvers\call-records.resolver.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { CallRecordsResolver } from './call-records.resolver';

describe('CallRecordsResolver', () => {
  let resolver: CallRecordsResolver;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [CallRecordsResolver],
    }).compile();

    resolver = module.get<CallRecordsResolver>(CallRecordsResolver);
  });

  it('should be defined', () => {
    expect(resolver).toBeDefined();
  });
});
```

{
  "cognitive_node": {

    "id": "file:src\\modules\\call-records\\resolvers\\call-records.resolver.spec.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "test",
    "purpose": "Provides a unit test specification for the CallRecordsResolver, ensuring the resolver can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./call-records.resolver"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["Resolver should be correctly instantiated in a testing module."],
      "security": []
    },
    "why": {
      "associations": ["Basic existence tests are a common scaffold in NestJS projects."],
      "biases": ["Focuses on structural existence, not functional logic."],
      "context": "Scaffolded by NestJS CLI or developer to verify CallRecordsResolver wiring."
    },
    "tests": {
      "spec_files": ["call-records.resolver.spec.ts"],
      "key_cases": ["Should define CallRecordsResolver instance"]
    },
    "links_hint": ["resolver:CallRecordsResolver"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\resolvers\\call-records.resolver.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\call-records\\resolvers\\call-records.resolver.spec.ts", "to": "module-node:call-records" },

    { "type": "IMPORTS", "from":
"file:src\\modules\\call-records\\resolvers\\call-records.resolver.spec.ts", "to":
"resolver:CallRecordsResolver" }
   ]
  }
}

src\modules\call-records\resolvers\call-records.resolver.ts

```
import { Args, Mutation, Resolver, Subscription } from
'@nestjs/graphql';
import { CreateCallRecordInput } from
'../dto/create-call-record.input';
import { CallRecordsService } from '../services/call-records.service';
import { UpdateCallRecordInput } from
'../dto/update-call-record.input';
import { PubSubService } from '../../pub-sub/services/pub-sub.service';
import { NotifyCallRecordCreatedInput } from
'../dto/notify-call-record-created.input';
import { NotifyCallRecordUpdatedInput } from
'../dto/notify-call-record-updated.input';

@Resolver()
export class CallRecordsResolver {
  constructor(
    private readonly callRecordsService: CallRecordsService,
    private readonly pubSubService: PubSubService,
  ) {}
  @Mutation('createCallRecord')
  async createCallRecord(
    @Args('input') createCallRecordInput: CreateCallRecordInput,
  ) {
    const callRecord = await this.callRecordsService.createCallRecord(
      createCallRecordInput,
    );
    return callRecord;
  }
  @Mutation('updateCallRecord')
  updateCallRecord(
    @Args('input') updateCallRecordInput: UpdateCallRecordInput,
  ) {
    return
this.callRecordsService.updateCallRecord(updateCallRecordInput);
  }
  @Mutation('notifyCallRecordCreated')
  notifyCallRecordCreated(
    @Args('input') notifyCallRecordCreatedInput:
NotifyCallRecordCreatedInput,
  ) {
    void this.pubSubService.publish('CALL_RECORD_CREATED', {
      // @ts-expect-error abc
      callRecordCreated: notifyCallRecordCreatedInput,
    });
    return notifyCallRecordCreatedInput.callRecord;
  }
  @Mutation('notifyCallRecordUpdated')
  notifyCallRecordUpdated(
    @Args('input') notifyCallRecordUpdatedInput:
NotifyCallRecordUpdatedInput,
  ) {
    void this.pubSubService.publish('CALL_RECORD_UPDATED', {
      // @ts-expect-error abc
      callRecordUpdated: notifyCallRecordUpdatedInput,
    });
    return notifyCallRecordUpdatedInput.callRecord;
  }
  @Subscription('callRecordCreated')
  subscribeToCallRecordCreatedEvent() {
    return this.pubSubService.subscribe('CALL_RECORD_CREATED');
  }
  @Subscription('callRecordUpdated', {
    filter: (payload, variables) => {
      const updateKinds = (variables.updateKinds || []) as string[];
      if (updateKinds.length === 0) return true;

      return
updateKinds.includes(payload.callRecordUpdated.updateKind);
    },
  })
  subscribeToCallRecordUpdatedEvent() {
    return this.pubSubService.subscribe('CALL_RECORD_UPDATED');
  }
}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\resolvers\\call-records.resolver.ts",
    "system": "API-GATEWAY",

```
  "module": "call-records",
  "role": "resolver",
  "purpose": "GraphQL resolver exposing mutations and subscriptions for CallRecord
lifecycle events.",
  "inputs": {
    "di_injections": ["CallRecordsService", "PubSubService"],
    "imports": [
      "@nestjs/graphql",
      "../dto/create-call-record.input",
      "../dto/update-call-record.input",
      "../dto/notify-call-record-created.input",
      "../dto/notify-call-record-updated.input",
      "../services/call-records.service",
      "../../pub-sub/services/pub-sub.service"
    ],
    "schema_types": [
      "CreateCallRecordInput",
      "UpdateCallRecordInput",
      "NotifyCallRecordCreatedInput",
      "NotifyCallRecordUpdatedInput"
    ],
    "config_keys": []
  },
  "outputs": {
    "api_surface": [
      "mutation:createCallRecord",
      "mutation:updateCallRecord",
      "mutation:notifyCallRecordCreated",
      "mutation:notifyCallRecordUpdated",
      "subscription:callRecordCreated",
      "subscription:callRecordUpdated"
    ],
    "exported_symbols": ["CallRecordsResolver"],
    "events_pubsub": ["topic:CALL_RECORD_CREATED",
"topic:CALL_RECORD_UPDATED"]
  },
  "behavior": {
    "side_effects": ["PubSub", "Service call"],
    "error_model": ["Propagates service errors", "GraphQL validation errors"]
  },
  "contracts": {
    "invariants": [
      "Mutations must return CallRecord or related input type as defined in schema.",
      "Subscriptions must resolve to payloads consistent with schema event types."
    ],
    "security": ["Relies on GraphQL layer for validation; no explicit authn/authz here."]
  },
  "why": {
      "associations": ["Implements GraphQL contract defined in call-records.graphql."],
      "biases": ["Publishes raw input payloads to PubSub topics with minimal transformation."],
      "context": "Acts as the boundary between GraphQL client operations and internal
services for Call Records."
    },
    "tests": {
      "spec_files": ["call-records.resolver.spec.ts"],
      "key_cases": ["createCallRecord returns new record", "Subscriptions deliver published
events"]
    },
    "links_hint": [
      "service:CallRecordsService",
      "service:PubSubService",
      "graphql:CallRecord",
      "graphql:CreateCallRecordInput",
      "graphql:UpdateCallRecordInput"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\resolvers\\call-records.resolver.ts", "kind": "file" },
      { "id": "resolver:CallRecordsResolver", "kind": "resolver" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\call-records\\resolvers\\call-records.resolver.ts", "to":
"module-node:call-records" },
      { "type": "OWNS", "from":
"file:src\\modules\\call-records\\resolvers\\call-records.resolver.ts", "to":
"resolver:CallRecordsResolver" },
      { "type": "INJECTS", "from": "resolver:CallRecordsResolver", "to":
"service:CallRecordsService" },
      { "type": "INJECTS", "from": "resolver:CallRecordsResolver", "to":
"service:PubSubService" },
      { "type": "EXPOSES", "from": "resolver:CallRecordsResolver", "to": "graphql:CallRecord"
},
      { "type": "PUBSUB", "from": "resolver:CallRecordsResolver", "to":
"topic:CALL_RECORD_CREATED", "note": "publish/subscribe" },
      { "type": "PUBSUB", "from": "resolver:CallRecordsResolver", "to":
"topic:CALL_RECORD_UPDATED", "note": "publish/subscribe" }
    ]
  }
}
```

src\modules\call-records\services\call-records.service.spec.ts

```typescript
import { Test, TestingModule } from '@nestjs/testing';
import { CallRecordsService } from './call-records.service';

describe('CallRecordsService', () => {
  let service: CallRecordsService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [CallRecordsService],
    }).compile();

    service = module.get<CallRecordsService>(CallRecordsService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

```json
{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\services\\call-records.service.spec.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "test",
    "purpose": "Provides a unit test specification for the CallRecordsService, ensuring the service can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./call-records.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["Service should be correctly instantiated in a testing module."],
      "security": []
    },
    "why": {
      "associations": ["Basic existence test is a common scaffold generated by NestJS CLI."],
      "biases": ["Checks only definition, not functionality."],
      "context": "Ensures the CallRecordsService provider can be resolved by the NestJS DI
container."
    },
    "tests": {
      "spec_files": ["call-records.service.spec.ts"],
      "key_cases": ["Should define CallRecordsService instance"]
    },
    "links_hint": ["service:CallRecordsService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\services\\call-records.service.spec.ts", "kind": "file"
}
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\call-records\\services\\call-records.service.spec.ts", "to":
"module-node:call-records" },
      { "type": "IMPORTS", "from":
"file:src\\modules\\call-records\\services\\call-records.service.spec.ts", "to":
"service:CallRecordsService" }
    ]
  }
}
```

src\modules\call-records\services\call-records.service.ts

```
import { Inject, Injectable } from '@nestjs/common';
import { CreateCallRecordInput } from '../../../graphql';
import callRecordsConfig, {
  type CallRecordsConfig,
} from '../config/call-records.config';
import { type ConfigType } from '@nestjs/config';
import { CallRecord } from '../types/call-records.types';
import { UpdateCallRecordInput } from
'../dto/update-call-record.input';

@Injectable()
export class CallRecordsService {
  constructor(
    @Inject(callRecordsConfig.KEY)
    private readonly callRecordsConfig: ConfigType<CallRecordsConfig>,
  ) {}
  async createCallRecord(createCallRecordInput: CreateCallRecordInput)
{
    const response = await
fetch(this.callRecordsConfig.callRecordsBaseURL, {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(createCallRecordInput),
    });
    const {
      data: { callRecord },
    } = (await response.json()) as { data: { callRecord: CallRecord }
};
    return callRecord;
  }
  async updateCallRecord(updateCallRecordInput: UpdateCallRecordInput)
{
    const response = await
fetch(this.callRecordsConfig.callRecordsBaseURL, {
      method: 'PATCH',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify(updateCallRecordInput),
    });
    const {
      data: { callRecord },
    } = (await response.json()) as { data: { callRecord: CallRecord }
};
    return callRecord;
  }
}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\services\\call-records.service.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "service",
    "purpose": "Provides methods to create and update call records by forwarding requests to the external Call Records service API.",

```
  "inputs": {
    "di_injections": ["ConfigType<CallRecordsConfig>"],
    "imports": [
      "@nestjs/common",
      "../../../graphql",
      "../config/call-records.config",
      "@nestjs/config",
      "../types/call-records.types",
      "../dto/update-call-record.input"
    ],
    "schema_types": ["CreateCallRecordInput", "UpdateCallRecordInput", "CallRecord"],
    "config_keys": ["CALL_RECORDS_BASE_URL"]
  },
  "outputs": {
    "api_surface": ["createCallRecord", "updateCallRecord"],
    "exported_symbols": ["CallRecordsService"],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": ["HTTP"],
    "error_model": ["Throws if fetch fails or response is not valid JSON"]
  },
  "contracts": {
    "invariants": [
      "Must send JSON payloads with Content-Type application/json.",
      "Must return parsed CallRecord objects from backend responses."
    ],
    "security": [
      "Depends on backend endpoint being properly secured; service itself does not add auth
headers."
    ]
  },
  "why": {
    "associations": ["Acts as a bridge between GraphQL resolver layer and external REST
backend."],
    "biases": ["Implements direct fetch calls instead of abstracting HTTP client logic."],
    "context": "Supports resolver operations for Call Records mutations by delegating
persistence to the backend."
  },
  "tests": {
    "spec_files": ["call-records.service.spec.ts"],
    "key_cases": ["createCallRecord forwards valid payload", "updateCallRecord applies
PATCH correctly"]
  },
  "links_hint": [
    "resolver:CallRecordsResolver",
    "config:call-records",
    "dto:UpdateCallRecordInput",
      "graphql:CreateCallRecordInput",
      "type:CallRecord"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\services\\call-records.service.ts", "kind": "file" },
      { "id": "service:CallRecordsService", "kind": "service" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\call-records\\services\\call-records.service.ts", "to": "module-node:call-records" },
      { "type": "OWNS", "from": "file:src\\modules\\call-records\\services\\call-records.service.ts", "to": "service:CallRecordsService" },
      { "type": "INJECTS", "from": "service:CallRecordsService", "to": "config:call-records" },
      { "type": "USES_TYPE", "from": "service:CallRecordsService", "to": "graphql:CreateCallRecordInput" },
      { "type": "USES_TYPE", "from": "service:CallRecordsService", "to": "dto:UpdateCallRecordInput" },
      { "type": "USES_TYPE", "from": "service:CallRecordsService", "to": "type:CallRecord" }
    ]
  }
}
```

src\modules\call-records\types\call-records.types.ts

```
import { Maybe } from 'graphql/jsutils/Maybe';
import {
  CALL_STATUSES,
  CALL_OUTCOMES,
  SENTIMENT_VALUES,
  RATING_VALUES,
} from '../constants/call-records.constants';

export type CallStatus = (typeof CALL_STATUSES)[number];
export type CallOutcome = (typeof CALL_OUTCOMES)[number];
export type SentimentValue = (typeof SENTIMENT_VALUES)[number];
export type RatingValue = (typeof RATING_VALUES)[number];

export type CallRecord = {
  id: string;
  callId: string;
  callerPhoneNumber: string;
  callStartTime: string;
  callTransferredTime: Maybe<string>;
  callEndTime: Maybe<string>;
  callStatus: CallStatus;
  callOutcome: Maybe<CallOutcome>;
  agent: {
    fullName: Maybe<string>;
  };
  patient: {
    fullName: Maybe<string>;
  };
  sentiment: {
    analysis: Maybe<string>;
    value: Maybe<SentimentValue>;
  };
  botRating: {
    value: Maybe<number>;
    info: Maybe<string>;
  };
  agentRating: {
    value: Maybe<number>;
    info: Maybe<string>;
  };
  cleanTranscription: Maybe<string[]>;
  recordingUrl: Maybe<string>;
};
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\types\\call-records.types.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "types",
    "purpose": "Defines strongly typed TypeScript representations for call records, including status, outcome, sentiment, rating values, and the CallRecord entity.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "graphql/jsutils/Maybe",
        "../constants/call-records.constants"
      ],
      "schema_types": [
        "CallStatus",
        "CallOutcome",

```
      "SentimentValue",
      "RatingValue",
      "CallRecord"
    ],
    "config_keys": []
  },
  "outputs": {
    "api_surface": [],
    "exported_symbols": [
      "CallStatus",
      "CallOutcome",
      "SentimentValue",
      "RatingValue",
      "CallRecord"
    ],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": [],
    "error_model": []
  },
  "contracts": {
    "invariants": [
      "CallRecord must use domain constants to restrict allowed values.",
      "Optional fields are wrapped in Maybe<> to align with GraphQL nullability."
    ],
    "security": []
  },
  "why": {
    "associations": [
      "Provides type safety and consistency between GraphQL schema and backend logic."
    ],
    "biases": [
      "Uses GraphQL Maybe utility to preserve nullability semantics."
    ],
    "context": "Supports the CallRecordsService and resolvers with strongly typed data
models."
  },
  "tests": {
    "spec_files": [],
    "key_cases": []
  },
  "links_hint": [
    "service:CallRecordsService",
    "resolver:CallRecordsResolver",
    "constants:call-records.constants.ts"
  ]
 },
```

  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\types\\call-records.types.ts", "kind": "file" },
      { "id": "type:CallRecord", "kind": "type" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\call-records\\types\\call-records.types.ts", "to": "module-node:call-records" },
      { "type": "OWNS", "from": "file:src\\modules\\call-records\\types\\call-records.types.ts", "to": "type:CallRecord" },
      { "type": "IMPORTS", "from": "file:src\\modules\\call-records\\types\\call-records.types.ts", "to": "file:src\\modules\\call-records\\constants\\call-records.constants.ts" }
    ]
  }
}

src\modules\call-records\call-records.module.ts

```
import { Module } from '@nestjs/common';
import { CallRecordsResolver } from
'./resolvers/call-records.resolver';
import { CallRecordsService } from './services/call-records.service';
import callRecordsConfig from './config/call-records.config';
import { ConfigModule } from '@nestjs/config';
import { PubSubModule } from '../pub-sub/pub-sub.module';

@Module({
  imports: [ConfigModule.forFeature(callRecordsConfig), PubSubModule],
  providers: [CallRecordsResolver, CallRecordsService],
})
export class CallRecordsModule {}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\call-records\\call-records.module.ts",
    "system": "API-GATEWAY",
    "module": "call-records",
    "role": "module",
    "purpose": "Defines the CallRecordsModule, wiring together resolver, service, config, and PubSub dependencies for the call-records domain.",
    "inputs": {
      "di_injections": [],
      "imports": [

```
        "@nestjs/common",
        "@nestjs/config",
        "./resolvers/call-records.resolver",
        "./services/call-records.service",
        "./config/call-records.config",
        "../pub-sub/pub-sub.module"
      ],
      "schema_types": [],
      "config_keys": ["CALL_RECORDS_BASE_URL"]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": ["CallRecordsModule"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Resolver and Service must be registered as providers.",
        "ConfigModule.forFeature must load callRecordsConfig correctly."
      ],
      "security": []
    },
    "why": {
      "associations": ["Aggregates all call-records functionality into a cohesive NestJS
module."],
      "biases": ["Follows NestJS idiomatic pattern of separating module, resolver, service."],
      "context": "Entry point module for the call-records domain within API-GATEWAY."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "resolver:CallRecordsResolver",
      "service:CallRecordsService",
      "config:call-records",
      "module:PubSubModule"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:call-records", "kind": "module-node" },
      { "id": "file:src\\modules\\call-records\\call-records.module.ts", "kind": "file" },
```

      { "id": "module:CallRecordsModule", "kind": "module" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:call-records" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\call-records\\call-records.module.ts", "to": "module-node:call-records" },
      { "type": "OWNS", "from": "file:src\\modules\\call-records\\call-records.module.ts", "to": "module:CallRecordsModule" },
      { "type": "IMPORTS", "from": "module:CallRecordsModule", "to": "config:call-records" },
      { "type": "IMPORTS", "from": "module:CallRecordsModule", "to": "module:PubSubModule" },
      { "type": "INJECTS", "from": "module:CallRecordsModule", "to": "resolver:CallRecordsResolver" },
      { "type": "INJECTS", "from": "module:CallRecordsModule", "to": "service:CallRecordsService" }
    ]
  }
}

src\modules\common\type-utils\common.type-utils.ts

```
export type Maybe<T> = T | null;
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\common\\type-utils\\common.type-utils.ts",
    "system": "API-GATEWAY",
    "module": "common",
    "role": "types",
    "purpose": "Provides a generic utility type Maybe<T> to express optional values that can be null.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": ["Maybe"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": ["Maybe<T> must only represent either a value of type T or null."],

      "security": []
    },
    "why": {
      "associations": ["Common utility type for aligning with GraphQL nullability semantics."],
      "biases": ["Provides a simplified representation instead of importing from GraphQL library."],
      "context": "Placed in common/type-utils to be reused across multiple modules."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": ["graphql:CallRecord", "type:CallRecord"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:common", "kind": "module-node" },
      { "id": "file:src\\modules\\common\\type-utils\\common.type-utils.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:common" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\common\\type-utils\\common.type-utils.ts", "to": "module-node:common" },
      { "type": "EXPOSES", "from": "file:src\\modules\\common\\type-utils\\common.type-utils.ts", "to": "type:Maybe" }
    ]
  }
}

src\modules\historical-mode\config\historical-mode.config.ts

```
import { registerAs } from '@nestjs/config';

export type HistoricalModeConfig = typeof historicalModeConfig;
const historicalModeConfig = registerAs('historical-mode', () => {
  return {
    terminatedCallRecordsBaseURL:
process.env.TERMINATED_CALL_RECORDS_BASE_URL!,
  };
});
export default historicalModeConfig;
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\historical-mode\\config\\historical-mode.config.ts",
    "system": "API-GATEWAY",

```
    "module": "historical-mode",
    "role": "config",
    "purpose": "Registers a configuration namespace for historical-mode, exposing the base
URL for terminated call records.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/config"],
      "schema_types": [],
      "config_keys": ["TERMINATED_CALL_RECORDS_BASE_URL"]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": ["historicalModeConfig", "HistoricalModeConfig"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Requires environment variable TERMINATED_CALL_RECORDS_BASE_URL to be
defined."
      ],
      "security": []
    },
    "why": {
      "associations": ["Provides typed config integration for historical mode operations."],
      "biases": ["Uses registerAs to create namespaced config for better modularization."],
      "context": "Supports HistoricalMode module by supplying the backend endpoint for
terminated call records."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": ["module:HistoricalModeModule", "service:TerminatedCallRecordsService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:historical-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\historical-mode\\config\\historical-mode.config.ts", "kind": "file" },
      { "id": "config:historical-mode", "kind": "config" },
      { "id": "config:TERMINATED_CALL_RECORDS_BASE_URL", "kind": "config" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode"
},
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\historical-mode\\config\\historical-mode.config.ts", "to":
"module-node:historical-mode" },
      { "type": "EXPOSES", "from":
"file:src\\modules\\historical-mode\\config\\historical-mode.config.ts", "to":
"config:historical-mode" },
      { "type": "EXPOSES", "from":
"file:src\\modules\\historical-mode\\config\\historical-mode.config.ts", "to":
"config:TERMINATED_CALL_RECORDS_BASE_URL" }
    ]
  }
}
```

C:\Users\Jorge\Desktop\api-gateway\api-gateway\src\modules\historical-mode\graphql\historical-mode.graphql

```
type Query {
  terminatedCallRecordsPage(
    queryParameters: QueryParameters
  ): TerminatedCallRecordsPage
  test: String
}

input QueryParameters {
  search: String
  sort: String
  callStartTime: Range
  botDuration: Range
  agentDuration: Range
  status: [String]
  outcome: [String]
  sentiment: [String]
  botRating: Range
  agentRating: Range
  page: String
  limit: String
}

input Range {
  from: String
  to: String
}

type TerminatedCallRecordsPage {
  meta: Meta!
  terminatedCallRecords: [CallRecord!]!
}

type Meta {
  totalDocuments: Int!
  totalPages: Int!
  page: Int!
  limit: Int!
  pageSize: Int!
}

type Subscription {
  test: Int
}
```

{
  "cognitive_node": {
    "id":
"file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\historical-mode\\graphql\\historical-mode.graphql",
    "system": "API-GATEWAY",
    "module": "historical-mode",
    "role": "graphql",
    "purpose": "Defines the GraphQL schema for historical-mode queries and subscriptions, including terminated call records pagination and test hooks.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [
        "QueryParameters",
        "Range",
        "TerminatedCallRecordsPage",
        "Meta"
      ],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "query:terminatedCallRecordsPage",
        "query:test",
        "subscription:test"
      ],
      "exported_symbols": [
        "TerminatedCallRecordsPage",
        "Meta",

```
      "QueryParameters",
      "Range"
    ],
    "events_pubsub": ["topic:test"]
  },
  "behavior": {
    "side_effects": ["PubSub"],
    "error_model": []
  },
  "contracts": {
    "invariants": [
      "Queries must return paginated terminated call records data aligned with Meta
information."
    ],
    "security": ["Filtering and pagination inputs are trusted but should be validated at resolver
layer."]
  },
  "why": {
    "associations": ["Provides API contract for accessing terminated call records with flexible
query parameters."],
    "biases": ["Includes a test query and subscription likely for debugging or scaffolding."],
    "context": "Core schema for HistoricalMode module, supporting analytics or archived
calls retrieval."
  },
  "tests": {
    "spec_files": [],
    "key_cases": ["Query terminatedCallRecordsPage returns Meta and array of
CallRecords."]
  },
  "links_hint": [
    "resolver:HistoricalModeResolver",
    "service:TerminatedCallRecordsService",
    "type:CallRecord"
  ]
 },
 "graph_patch": {
  "nodes": [
    { "id": "system:API-GATEWAY", "kind": "system" },
    { "id": "module-node:historical-mode", "kind": "module-node" },
    { "id":
"file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\historical-mode\\gra
phql\\historical-mode.graphql", "kind": "file" },
    { "id": "graphql:TerminatedCallRecordsPage", "kind": "graphql" },
    { "id": "graphql:QueryParameters", "kind": "graphql" },
    { "id": "graphql:Range", "kind": "graphql" },
    { "id": "graphql:Meta", "kind": "graphql" }
  ],
  "edges": [
```

{ "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode" },
{ "type": "BELONGS_TO", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\historical-mode\\graphql\\historical-mode.graphql", "to": "module-node:historical-mode" },
{ "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\historical-mode\\graphql\\historical-mode.graphql", "to": "graphql:TerminatedCallRecordsPage" },
{ "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\historical-mode\\graphql\\historical-mode.graphql", "to": "graphql:QueryParameters" },
{ "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\historical-mode\\graphql\\historical-mode.graphql", "to": "graphql:Range" },
{ "type": "EXPOSES", "from": "file:C:\\Users\\Jorge\\Desktop\\api-gateway\\api-gateway\\src\\modules\\historical-mode\\graphql\\historical-mode.graphql", "to": "graphql:Meta" }
  ]
 }
}

src\modules\historical-mode\resolvers\historical-mode.resolver.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { HistoricalModeResolver } from './historical-mode.resolver';

describe('HistoricalModeResolver', () => {
  let resolver: HistoricalModeResolver;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [HistoricalModeResolver],
    }).compile();

    resolver =
module.get<HistoricalModeResolver>(HistoricalModeResolver);
  });

  it('should be defined', () => {
    expect(resolver).toBeDefined();
  });
});
```

{
 "cognitive_node": {
  "id": "file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.spec.ts",

    "system": "API-GATEWAY",
    "module": "historical-mode",
    "role": "test",
    "purpose": "Provides a unit test specification for the HistoricalModeResolver, ensuring the resolver can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./historical-mode.resolver"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["Resolver should be correctly instantiated within the NestJS DI container during tests."],
      "security": []
    },
    "why": {
      "associations": ["Basic existence tests are a common scaffold in NestJS projects."],
      "biases": ["Focuses on structural definition rather than functional behavior."],
      "context": "Ensures the HistoricalModeResolver is properly wired for further tests of its query and subscription logic."
    },
    "tests": {
      "spec_files": ["historical-mode.resolver.spec.ts"],
      "key_cases": ["Should define HistoricalModeResolver instance"]
    },
    "links_hint": ["resolver:HistoricalModeResolver"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:historical-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode" },

      { "type": "BELONGS_TO", "from":
"file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.spec.ts", "to":
"module-node:historical-mode" },
      { "type": "IMPORTS", "from":
"file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.spec.ts", "to":
"resolver:HistoricalModeResolver" }
    ]
  }
}

src\modules\historical-mode\resolvers\historical-mode.resolver.ts

```
import * as GraphQLTypes from '../../../graphql';
import { Args, Context, Query, Resolver, Subscription } from
'@nestjs/graphql';
import { HistoricalModeService } from
'../services/historical-mode.service';
import { QueryParameters } from '../utils/historical-mode.utils';
import { PubSubService } from
'src/modules/pub-sub/services/pub-sub.service';

@Resolver()
export class HistoricalModeResolver {
  constructor(
    private readonly historicalModeService: HistoricalModeService,
    private readonly pubSubService: PubSubService,
  ) {}
  @Query('terminatedCallRecordsPage')
  getTerminatedCallRecords(
    @Args() args: { queryParameters?: QueryParameters },
  ): Promise<GraphQLTypes.TerminatedCallRecordsPage> {
    const queryParameters = args.queryParameters || {};
    return
this.historicalModeService.getTerminatedCallRecords(queryParameters);
  }
  @Query('test')
  test(@Context() context) {
    console.log('[test:query-resolver]', context.userInfo);
    return 'TEST';
  }
  @Subscription('test')
  naturals(@Context() context) {
    console.log('[test:subscription-resolver]', context.userInfo);
    return (async function* () {
      for (let current = 1; current <= 10; current += 1) {
        yield { test: current };
        await new Promise((r) => setTimeout(r, 3 * 1000));
      }
    })();
  }
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.ts",
    "system": "API-GATEWAY",
    "module": "historical-mode",
    "role": "resolver",
    "purpose": "GraphQL resolver for HistoricalMode, exposing queries and subscriptions for terminated call records and a test subscription.",
    "inputs": {
      "di_injections": ["HistoricalModeService", "PubSubService"],
      "imports": [
        "../../../graphql",
        "@nestjs/graphql",
        "../services/historical-mode.service",
        "../utils/historical-mode.utils",
        "src/modules/pub-sub/services/pub-sub.service"
      ],
      "schema_types": [
        "GraphQLTypes.TerminatedCallRecordsPage",
        "QueryParameters"
      ],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "query:terminatedCallRecordsPage",
        "query:test",
        "subscription:test"
      ],
      "exported_symbols": ["HistoricalModeResolver"],
      "events_pubsub": ["topic:test"]
    },
    "behavior": {
      "side_effects": ["Console logging", "PubSub subscription"],
      "error_model": ["Propagates errors from HistoricalModeService"]
    },
    "contracts": {
      "invariants": [
        "terminatedCallRecordsPage query must return TerminatedCallRecordsPage consistent with schema.",
      "test query must return a string.",
      "test subscription yields an async sequence of Int values."
    ],
    "security": ["Context includes userInfo, logged for visibility but not validated here."]
  },
  "why": {
    "associations": ["Acts as boundary between GraphQL client operations and
HistoricalModeService."],
    "biases": ["Includes a scaffolded test query and subscription, likely for debugging or
demonstration."],
    "context": "Implements HistoricalMode GraphQL contract defined in
historical-mode.graphql."
  },
  "tests": {
    "spec_files": ["historical-mode.resolver.spec.ts"],
    "key_cases": [
      "terminatedCallRecordsPage returns valid page data",
      "test query returns TEST",
      "test subscription yields incrementing values"
    ]
  },
  "links_hint": [
    "service:HistoricalModeService",
    "service:PubSubService",
    "graphql:TerminatedCallRecordsPage"
  ]
 },
 "graph_patch": {
  "nodes": [
    { "id": "system:API-GATEWAY", "kind": "system" },
    { "id": "module-node:historical-mode", "kind": "module-node" },
    { "id": "file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.ts", "kind":
"file" },
    { "id": "resolver:HistoricalModeResolver", "kind": "resolver" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode"
},
    { "type": "BELONGS_TO", "from":
"file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.ts", "to":
"module-node:historical-mode" },
    { "type": "OWNS", "from":
"file:src\\modules\\historical-mode\\resolvers\\historical-mode.resolver.ts", "to":
"resolver:HistoricalModeResolver" },
    { "type": "INJECTS", "from": "resolver:HistoricalModeResolver", "to":
"service:HistoricalModeService" },
    { "type": "INJECTS", "from": "resolver:HistoricalModeResolver", "to":
"service:PubSubService" },
```

      { "type": "EXPOSES", "from": "resolver:HistoricalModeResolver", "to": "graphql:TerminatedCallRecordsPage" },
      { "type": "PUBSUB", "from": "resolver:HistoricalModeResolver", "to": "topic:test", "note": "subscription" }
    ]
  }
}

src\modules\historical-mode\services\historical-mode.service.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { HistoricalModeService } from './historical-mode.service';

describe('HistoricalModeService', () => {
  let service: HistoricalModeService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [HistoricalModeService],
    }).compile();

    service = module.get<HistoricalModeService>(HistoricalModeService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\historical-mode\\services\\historical-mode.service.spec.ts",
    "system": "API-GATEWAY",
    "module": "historical-mode",
    "role": "test",
    "purpose": "Provides a unit test specification for the HistoricalModeService, ensuring the service can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./historical-mode.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []

```
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["Service should be correctly instantiated in a NestJS testing context."],
      "security": []
    },
    "why": {
      "associations": ["Ensures service wiring is correct before adding functional tests."],
      "biases": ["Covers only service definition, not logic."],
      "context": "Generated or scaffolded test file for HistoricalModeService."
    },
    "tests": {
      "spec_files": ["historical-mode.service.spec.ts"],
      "key_cases": ["Should define HistoricalModeService instance"]
    },
    "links_hint": ["service:HistoricalModeService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:historical-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\historical-mode\\services\\historical-mode.service.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\historical-mode\\services\\historical-mode.service.spec.ts", "to": "module-node:historical-mode" },
      { "type": "IMPORTS", "from": "file:src\\modules\\historical-mode\\services\\historical-mode.service.spec.ts", "to": "service:HistoricalModeService" }
    ]
  }
}
```

src\modules\historical-mode\services\historical-mode.service.ts

```
import { Injectable, Inject } from '@nestjs/common';
import { type ConfigType } from '@nestjs/config';
import historicalModeConfig, {
  type HistoricalModeConfig,
} from '../config/historical-mode.config';
import {
  QueryParameters,
  turnAPIFeaturesIntoQueryString,
  turnQueryParametersIntoAPIFeatures,
} from '../utils/historical-mode.utils';
import { type TerminatedCallRecordsPage } from
'../types/historical-mode.types';

@Injectable()
export class HistoricalModeService {
  constructor(
    @Inject(historicalModeConfig.KEY)
    private readonly historicalModeConfig:
ConfigType<HistoricalModeConfig>,
  ) {}
  async getTerminatedCallRecords(queryParameters: QueryParameters) {
    const apiFeatures =
turnQueryParametersIntoAPIFeatures(queryParameters);
    const queryString = turnAPIFeaturesIntoQueryString(apiFeatures);
    const response = await fetch(

`${this.historicalModeConfig.terminatedCallRecordsBaseURL}?${queryStrin
g}`,
    );
    const {
      data: { meta, callRecords: terminatedCallRecords },
    } = (await response.json()) as {
      status: 'SUCCESS';
      data: TerminatedCallRecordsPage;
    };

    return {
      meta,
      terminatedCallRecords,
    };
  }
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\historical-mode\\services\\historical-mode.service.ts",
    "system": "API-GATEWAY",
    "module": "historical-mode",
    "role": "service",
  "purpose": "Provides methods for fetching terminated call records from the HistoricalMode
backend service, transforming query parameters into API-compatible queries.",
  "inputs": {
    "di_injections": ["ConfigType<HistoricalModeConfig>"],
    "imports": [
      "@nestjs/common",
      "@nestjs/config",
      "../config/historical-mode.config",
      "../utils/historical-mode.utils",
      "../types/historical-mode.types"
    ],
    "schema_types": ["QueryParameters", "TerminatedCallRecordsPage"],
    "config_keys": ["TERMINATED_CALL_RECORDS_BASE_URL"]
  },
  "outputs": {
    "api_surface": ["getTerminatedCallRecords"],
    "exported_symbols": ["HistoricalModeService"],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": ["HTTP fetch"],
    "error_model": ["Throws if fetch fails or response JSON is invalid"]
  },
  "contracts": {
    "invariants": [
      "Must always construct valid query string from input QueryParameters.",
      "Must return meta and terminatedCallRecords as structured in
TerminatedCallRecordsPage."
    ],
    "security": [
      "Relies on environment variable TERMINATED_CALL_RECORDS_BASE_URL for
backend endpoint."
    ]
  },
  "why": {
    "associations": ["Acts as a bridge between GraphQL resolver and external REST API for
terminated calls."],
    "biases": ["Implements fetch directly rather than abstracting into an HTTP client utility."],
    "context": "Supports HistoricalModeResolver queries for paginated terminated call
records."
  },
  "tests": {
    "spec_files": ["historical-mode.service.spec.ts"],
    "key_cases": ["Correctly builds query string", "Parses meta and terminatedCallRecords
from response"]
  },
  "links_hint": [
    "resolver:HistoricalModeResolver",
      "config:historical-mode",
      "type:TerminatedCallRecordsPage",
      "util:historical-mode.utils"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:historical-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\historical-mode\\services\\historical-mode.service.ts", "kind": "file" },
      { "id": "service:HistoricalModeService", "kind": "service" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\historical-mode\\services\\historical-mode.service.ts", "to": "module-node:historical-mode" },
      { "type": "OWNS", "from": "file:src\\modules\\historical-mode\\services\\historical-mode.service.ts", "to": "service:HistoricalModeService" },
      { "type": "INJECTS", "from": "service:HistoricalModeService", "to": "config:historical-mode" },
      { "type": "USES_TYPE", "from": "service:HistoricalModeService", "to": "graphql:QueryParameters" },
      { "type": "USES_TYPE", "from": "service:HistoricalModeService", "to": "type:TerminatedCallRecordsPage" },
      { "type": "USES_TYPE", "from": "service:HistoricalModeService", "to": "util:historical-mode.utils" }
    ]
  }
}
```

src\modules\historical-mode\types\historical-mode.types.ts

```
import { Maybe } from 'graphql/jsutils/Maybe';
import { TERMINATED_CALL_STATUSES } from
'src/modules/call-records/constants/call-records.constants';
import {
  CallOutcome,
  SentimentValue,
} from 'src/modules/call-records/types/call-records.types';

export type TerminatedCallStatus = (typeof
TERMINATED_CALL_STATUSES)[number];
export type TerminatedCallRecord = {
  id: string;
  callId: string;
  callerPhoneNumber: string;
  callStartTime: string;
  callTransferredTime: Maybe<string>;
  callEndTime: string;
  callStatus: TerminatedCallStatus;
  callOutcome: CallOutcome;
  agent: {
    fullName: Maybe<string>;
  };
  patient: {
    fullName: Maybe<string>;
  };
  sentiment: {
    analysis: Maybe<string>;
    value: Maybe<SentimentValue>;
  };
  botRating: {
    value: Maybe<number>;
    info: Maybe<string>;
  };
  agentRating: {
    value: Maybe<number>;
    info: Maybe<string>;
  };
  cleanTranscription: Maybe<string[]>;
  recordingUrl: Maybe<string>;
};

export type TerminatedCallRecordsPage = {
  meta: Record<
    'totalDocuments' | 'totalPages' | 'page' | 'limit' | 'pageSize',
    number
  >;
  callRecords: TerminatedCallRecord[];
};
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\historical-mode\\types\\historical-mode.types.ts",
    "system": "API-GATEWAY",
    "module": "historical-mode",

    "role": "types",
    "purpose": "Defines TypeScript types for terminated call records, including TerminatedCallStatus, TerminatedCallRecord, and TerminatedCallRecordsPage with pagination metadata.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "graphql/jsutils/Maybe",
        "src/modules/call-records/constants/call-records.constants",
        "src/modules/call-records/types/call-records.types"
      ],
      "schema_types": [
        "TerminatedCallStatus",
        "TerminatedCallRecord",
        "TerminatedCallRecordsPage"
      ],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [
        "TerminatedCallStatus",
        "TerminatedCallRecord",
        "TerminatedCallRecordsPage"
      ],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "TerminatedCallStatus must only be values from TERMINATED_CALL_STATUSES.",
        "TerminatedCallRecord must include required call properties and typed nested structures.",
        "TerminatedCallRecordsPage meta must include totalDocuments, totalPages, page, limit, and pageSize."
      ],
      "security": []
    },
    "why": {
      "associations": ["Provides strong typing to ensure compatibility with HistoricalMode GraphQL schema."],
      "biases": ["Uses GraphQL Maybe type to model nullable fields."],
      "context": "Supports HistoricalModeService and resolvers with consistent typing for terminated call records."
    },

```
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "service:HistoricalModeService",
      "resolver:HistoricalModeResolver",
      "graphql:TerminatedCallRecordsPage"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:historical-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\historical-mode\\types\\historical-mode.types.ts", "kind": "file" },
      { "id": "type:TerminatedCallRecord", "kind": "type" },
      { "id": "type:TerminatedCallRecordsPage", "kind": "type" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode"
},
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\historical-mode\\types\\historical-mode.types.ts", "to":
"module-node:historical-mode" },
      { "type": "OWNS", "from":
"file:src\\modules\\historical-mode\\types\\historical-mode.types.ts", "to":
"type:TerminatedCallRecord" },
      { "type": "OWNS", "from":
"file:src\\modules\\historical-mode\\types\\historical-mode.types.ts", "to":
"type:TerminatedCallRecordsPage" },
      { "type": "USES_TYPE", "from": "type:TerminatedCallRecord", "to":
"constants:call-records.constants.ts" },
      { "type": "USES_TYPE", "from": "type:TerminatedCallRecord", "to": "type:CallOutcome" },
      { "type": "USES_TYPE", "from": "type:TerminatedCallRecord", "to":
"type:SentimentValue" }
    ]
  }
}
```

src\modules\historical-mode\utils\historical-mode.utils.ts

```
import dayjs from 'dayjs';

import { Maybe } from '../../common/type-utils/common.type-utils';
import {
  CALL_STATUSES,
  CALL_OUTCOMES,
  SENTIMENT_VALUES,
} from 'src/modules/call-records/constants/call-records.constants';
import {
  CallOutcome,
  SentimentValue,
  RatingValue,
} from 'src/modules/call-records/types/call-records.types';
import { TerminatedCallStatus } from '../types/historical-mode.types';

export type OrderBy =
  | 'callStartTime'
  | 'botCallDuration'
  | 'agentCallDuration'
  | 'patientName'
  | 'agentName'
  | 'sentimentValue'
  | 'botRatingValue'
  | 'agentRatingValue';

export type Direction = 'asc' | 'desc';
export type APIFeatures = {
  search: { term: Maybe<string> };
  filter: {
    callStartTimeRange: {
      from: Maybe<Date>;
      to: Maybe<Date>;
    };
    botCallDurationRange: {
      from: number;
      to: number;
    };
    agentCallDurationRange: {
      from: number;
      to: number;
    };
    callStatusList: TerminatedCallStatus[];
    callOutcomeList: CallOutcome[];
    sentimentValueList: SentimentValue[];
    botRatingValueRange: {
      from: RatingValue;
      to: RatingValue;
    };
    agentRatingValueRange: {
      from: RatingValue;
      to: RatingValue;
    };
  };
  sort: {
    orderBy: OrderBy;
    direction: Direction;
  };
  pagination: {
    pageNumber: number;
    perPage: number;
  };
};

export function turnAPIFeaturesIntoQueryString(apiFeatures:
APIFeatures) {
  const { search, filter, sort, pagination } = apiFeatures;
  const queryParams = new URLSearchParams();
  if (search.term !== null && search.term !== '') {
    queryParams.append('search', search.term);
  }

  if (
    filter.callStartTimeRange.from !== null &&
    filter.callStartTimeRange.to !== null
  ) {
    queryParams.append(
      'callStartTime[from]',
      dayjs(
        new Date(filter.callStartTimeRange.from.setHours(0, 0, 0)),
      ).toISOString(),
    );
    console.log(
      '[from]',
      new Date(filter.callStartTimeRange.from.setHours(0, 0, 0)),
    );
    queryParams.append(
      'callStartTime[to]',
      dayjs(
        new Date(filter.callStartTimeRange.to.setHours(23, 59, 59)),
      ).toISOString(),
    );
    console.log(
      '[to]',
      new Date(filter.callStartTimeRange.to.setHours(23, 59, 59)),
    );
  }
  if (
    filter.botCallDurationRange.from !== 0 ||
    filter.botCallDurationRange.to !== 15
  ) {
    queryParams.append(
      'botDuration[from]',
      String(filter.botCallDurationRange.from),
    );
    queryParams.append(
      'botDuration[to]',
      String(filter.botCallDurationRange.to),
    );
  }
  if (
    filter.agentCallDurationRange.from !== 0 ||
    filter.agentCallDurationRange.to !== 15
  ) {
    queryParams.append(
      'agentDuration[from]',
      String(filter.agentCallDurationRange.from),
    );
    queryParams.append(
      'agentDuration[to]',
      String(filter.agentCallDurationRange.to),
    );
  }
  if (filter.callStatusList.length !== CALL_STATUSES.length) {
    filter.callStatusList.forEach((callStatus) => {
      queryParams.append('status[]', callStatus);
    });
  }
  if (filter.callOutcomeList.length !== CALL_OUTCOMES.length) {
    filter.callOutcomeList.forEach((callOutcome) => {
      queryParams.append('outcome[]', callOutcome);
    });
  }
  if (filter.sentimentValueList.length !== SENTIMENT_VALUES.length) {
    filter.sentimentValueList.forEach((sentimentValue) => {
      queryParams.append('sentiment[]', sentimentValue);
    });
  }
  if (
    filter.botRatingValueRange.from !== 1 ||
    filter.botRatingValueRange.to !== 5
  ) {
    queryParams.append(
      'botRating[from]',
      String(filter.botRatingValueRange.from),
    );
    queryParams.append('botRating[to]',
String(filter.botRatingValueRange.to));
  }
  if (
    filter.agentRatingValueRange.from !== 1 ||
    filter.agentRatingValueRange.to !== 5
  ) {
    queryParams.append(
      'agentRating[from]',
      String(filter.agentRatingValueRange.from),
    );
    queryParams.append(
      'agentRating[to]',
      String(filter.agentRatingValueRange.to),
    );
  }

  if (sort.orderBy !== 'callStartTime' || sort.direction !== 'desc') {
    queryParams.append(
      'sort',
      `${sort.direction === 'desc' ? '-' : ''}${sort.orderBy}`,
    );
  }

  if (pagination.pageNumber !== 1) {
    queryParams.append('page', String(pagination.pageNumber));
  }

  if (pagination.perPage !== 5) {
    queryParams.append('limit', String(pagination.perPage));
  }

  const queryString = queryParams.toString();
  return queryString;
```

```
}

export type SearchParameters = {
  search?: string;
  'callStartTime[from]'?: string;
  'callStartTime[to]'?: string;
  'botDuration[from]'?: string;
  'botDuration[to]'?: string;
  'agentDuration[from]'?: string;
  'agentDuration[to]'?: string;
  'status[]'?: string[];
  'outcome[]'?: string[];
  'sentiment[]'?: string[];
  'botRating[from]'?: string;
  'botRating[to]'?: string;
  'agentRating[from]'?: string;
  'agentRating[to]'?: string;
  sort?: string;
  page?: string;
  limit?: string;
};
export function turnSearchParametersIntoQueryParameters(
  searchParameters: SearchParameters,
): QueryParameters {
  return {
    search: searchParameters.search,
    callStartTime: {
      from: searchParameters['callStartTime[from]'],
      to: searchParameters['callStartTime[to]'],
    },
    botDuration: {
      from: searchParameters['botDuration[from]'],
      to: searchParameters['botDuration[to]'],
    },
    agentDuration: {
      from: searchParameters['agentDuration[from]'],
      to: searchParameters['agentDuration[to]'],
    },
    status: searchParameters['status[]'],
    outcome: searchParameters['outcome[]'],
    sentiment: searchParameters['sentiment[]'],
    botRating: {
      from: searchParameters['botRating[from]'],
      to: searchParameters['botRating[to]'],
    },
    agentRating: {
      from: searchParameters['agentRating[from]'],
      to: searchParameters['agentRating[to]'],
    },
    sort: searchParameters.sort,
    page: searchParameters.page,
    limit: searchParameters.limit,
  };
}

export type QueryParameters = {
  search?: string;
  sort?: string;
  callStartTime?: {
    from?: string;
    to?: string;
  };
  botDuration?: {
    from?: string;
    to?: string;
  };
  agentDuration?: {
    from?: string;
    to?: string;
  };
  status?: string[];
  outcome?: string[];
  sentiment?: string[];
  botRating?: {
    from?: string;
    to?: string;
  };
  agentRating?: {
    from?: string;
    to?: string;
  };
  page?: string;
  limit?: string;
};
export function turnQueryParametersIntoAPIFeatures(
  queryParameters: QueryParameters,
): APIFeatures {
  return {
    search: {
      term:
        queryParameters.search === undefined ? null :
queryParameters.search,
    },
    filter: {
      callStartTimeRange: {
        from:
          queryParameters.callStartTime === undefined ||
          queryParameters.callStartTime.from === undefined
            ? null
            : new Date(queryParameters.callStartTime.from),
        to:
          queryParameters.callStartTime === undefined ||
          queryParameters.callStartTime.to === undefined
            ? null
            : new Date(queryParameters.callStartTime.to),
      },
      botCallDurationRange: {
        from:
          queryParameters.botDuration === undefined ||
          queryParameters.botDuration.from === undefined
            ? 0
            : Number(queryParameters.botDuration.from),
        to:
          queryParameters.botDuration === undefined ||
          queryParameters.botDuration.to === undefined
            ? 15
            : Number(queryParameters.botDuration.to),
      },
      agentCallDurationRange: {
        from:
          queryParameters.agentDuration === undefined ||
          queryParameters.agentDuration.from === undefined
            ? 0
            : Number(queryParameters.agentDuration.from),
        to:
          queryParameters.agentDuration === undefined ||
          queryParameters.agentDuration.to === undefined
            ? 15
            : Number(queryParameters.agentDuration.to),
      },
      callStatusList: (queryParameters.status === undefined
        ? CALL_STATUSES
        : queryParameters.status) as TerminatedCallStatus[],
      callOutcomeList: (queryParameters.outcome === undefined
        ? CALL_OUTCOMES
        : queryParameters.outcome) as CallOutcome[],
      sentimentValueList: (queryParameters.sentiment === undefined
        ? SENTIMENT_VALUES
        : queryParameters.sentiment) as SentimentValue[],
      botRatingValueRange: {
        from:
          queryParameters.botRating === undefined ||
          queryParameters.botRating.from === undefined
            ? 1
            : (Number(queryParameters.botRating.from) as RatingValue),
        to:
          queryParameters.botRating === undefined ||
          queryParameters.botRating.to === undefined
            ? 5
            : (Number(queryParameters.botRating.to) as RatingValue),
      },
      agentRatingValueRange: {
        from:
          queryParameters.agentRating === undefined ||
          queryParameters.agentRating.from === undefined
            ? 1
            : (Number(queryParameters.agentRating.from) as
RatingValue),
        to:
          queryParameters.agentRating === undefined ||
          queryParameters.agentRating.to === undefined
            ? 5
            : (Number(queryParameters.agentRating.to) as RatingValue),
      },
    },
    sort: {
      orderBy:
        queryParameters.sort === undefined
          ? 'callStartTime'
          : queryParameters.sort.startsWith('-')
            ? (queryParameters.sort.slice(1) as OrderBy)
            : (queryParameters.sort as OrderBy),
      direction:
        queryParameters.sort === undefined
          ? 'desc'
          : queryParameters.sort.startsWith('-')
            ? 'desc'
            : 'asc',
    },
    pagination: {
      pageNumber:
        queryParameters.page === undefined ? 1 :
Number(queryParameters.page),
      perPage:
        queryParameters.limit === undefined ? 5 :
Number(queryParameters.limit),
    },
  };
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\historical-mode\\utils\\historical-mode.utils.ts",
    "system": "API-GATEWAY",
    "module": "historical-mode",
    "role": "util",
    "purpose": "Transforms HistoricalMode query parameters to and from API features and
builds a URL query string for backend requests; declares shared sorting, filtering, and
pagination types.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "dayjs",
        "../../common/type-utils/common.type-utils",
        "src/modules/call-records/constants/call-records.constants",
        "src/modules/call-records/types/call-records.types",
        "../types/historical-mode.types"
      ],
      "schema_types": [
        "OrderBy",
        "Direction",
        "APIFeatures",
        "SearchParameters",
        "QueryParameters",
        "TerminatedCallStatus",
        "CallOutcome",
        "SentimentValue",
        "RatingValue"
    ],
    "config_keys": []
  },
  "outputs": {
    "api_surface": [
      "fn:turnAPIFeaturesIntoQueryString",
      "fn:turnSearchParametersIntoQueryParameters",
      "fn:turnQueryParametersIntoAPIFeatures"
    ],
    "exported_symbols": [
      "OrderBy",
      "Direction",
      "APIFeatures",
      "SearchParameters",
      "QueryParameters",
      "turnAPIFeaturesIntoQueryString",
      "turnSearchParametersIntoQueryParameters",
      "turnQueryParametersIntoAPIFeatures"
    ],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": ["console.log in date-range normalization within
turnAPIFeaturesIntoQueryString"],
    "error_model": [
      "Relies on valid input types; may generate invalid query strings if provided malformed
values (e.g., NaN coercions from Number())"
    ]
  },
  "contracts": {
    "invariants": [
      "Date filters: when both from and to exist, normalize to [00:00:00, 23:59:59] local time
before ISO serialization via dayjs().",
      "Defaults: bot/agent duration ranges default to [0, 15]; rating ranges default to [1, 5];
sort defaults to -callStartTime; pagination defaults to page=1, limit=5.",
      "Enumerations: omit status/outcome/sentiment params when lists include all domain
values (to avoid redundant filtering).",
      "Query string must be stable and only include parameters that differ from defaults."
    ],
    "security": [
      "No secrets handled; outputs plain query strings.",
      "Callers must sanitize/validate incoming QueryParameters if sourced from user input."
    ]
  },
  "why": {
    "associations": [
      "Centralizes cross-layer translation between UI/GraphQL-friendly QueryParameters and
backend API query syntax."
```

```
      ],
      "biases": [
        "Prefers explicit defaults to minimize emitted query params; uses dayjs for ISO
formatting."
      ],
      "context": "Used by HistoricalModeService to construct backend request URLs and to
interpret inbound search parameters consistently."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Emits no params when all values equal defaults",
        "Correctly serializes full date range with start/end of day normalization",
        "Applies descending sort when string starts with '-'",
        "Builds list params (status/outcome/sentiment) with repeated keys",
        "Pagination maps page/limit strings to numbers with defaults"
      ]
    },
    "links_hint": [
      "service:HistoricalModeService",
      "type:TerminatedCallRecordsPage",
      "constants:call-records.constants.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:historical-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\historical-mode\\utils\\historical-mode.utils.ts", "kind": "file" },
      { "id": "util:historical-mode.utils", "kind": "util" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode"
},
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\historical-mode\\utils\\historical-mode.utils.ts", "to":
"module-node:historical-mode" },
      { "type": "OWNS", "from":
"file:src\\modules\\historical-mode\\utils\\historical-mode.utils.ts", "to":
"util:historical-mode.utils" },
      { "type": "IMPORTS", "from": "util:historical-mode.utils", "to":
"file:src\\modules\\common\\type-utils\\common.type-utils.ts" },
      { "type": "IMPORTS", "from": "util:historical-mode.utils", "to":
"file:src\\modules\\call-records\\constants\\call-records.constants.ts" },
      { "type": "IMPORTS", "from": "util:historical-mode.utils", "to":
"file:src\\modules\\call-records\\types\\call-records.types.ts" },
      { "type": "IMPORTS", "from": "util:historical-mode.utils", "to":
"file:src\\modules\\historical-mode\\types\\historical-mode.types.ts" }
```

```
    ]
  }
}
```

src\modules\historical-mode\historical-mode.module.ts

```typescript
import { Module } from '@nestjs/common';
import { HistoricalModeService } from
'./services/historical-mode.service';
import { HistoricalModeResolver } from
'./resolvers/historical-mode.resolver';
import { ConfigModule } from '@nestjs/config';
import historicalModeConfig from './config/historical-mode.config';
import { PubSubModule } from '../pub-sub/pub-sub.module';

@Module({
  imports: [ConfigModule.forFeature(historicalModeConfig),
PubSubModule],
  providers: [HistoricalModeService, HistoricalModeResolver],
})
export class HistoricalModeModule {}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\historical-mode\\historical-mode.module.ts",
    "system": "API-GATEWAY",
    "module": "historical-mode",
    "role": "module",
    "purpose": "Declares HistoricalModeModule, wiring the resolver and service, loading
namespaced configuration, and attaching PubSub dependencies.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "@nestjs/common",
        "@nestjs/config",
        "./services/historical-mode.service",
        "./resolvers/historical-mode.resolver",
        "./config/historical-mode.config",
        "../pub-sub/pub-sub.module"
      ],
      "schema_types": [],
      "config_keys": ["TERMINATED_CALL_RECORDS_BASE_URL"]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": ["HistoricalModeModule"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "HistoricalModeService and HistoricalModeResolver are registered as providers.",
        "ConfigModule.forFeature(historicalModeConfig) must expose the historical-mode
namespace."
      ],
      "security": []
    },
    "why": {
      "associations": ["Aggregates Historical Mode functionality and exposes it to the
application graph."],
      "biases": ["Uses idiomatic NestJS module composition with feature-scoped config."],
      "context": "Entry point for historical-mode domain, enabling queries/subscriptions and
backend access."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "service:HistoricalModeService",
      "resolver:HistoricalModeResolver",
      "config:historical-mode",
      "module:PubSubModule"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:historical-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\historical-mode\\historical-mode.module.ts", "kind": "file" },
      { "id": "module:HistoricalModeModule", "kind": "module" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:historical-mode"
},
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\historical-mode\\historical-mode.module.ts", "to":
"module-node:historical-mode" },
      { "type": "OWNS", "from": "file:src\\modules\\historical-mode\\historical-mode.module.ts",
"to": "module:HistoricalModeModule" },
      { "type": "IMPORTS", "from": "module:HistoricalModeModule", "to":
"config:historical-mode" },
    { "type": "IMPORTS", "from": "module:HistoricalModeModule", "to":
"module:PubSubModule" },
    { "type": "INJECTS", "from": "module:HistoricalModeModule", "to":
"service:HistoricalModeService" },
    { "type": "INJECTS", "from": "module:HistoricalModeModule", "to":
"resolver:HistoricalModeResolver" }
   ]
  }
}
```

src\modules\iam\authentication\decorators\user.decorator.ts

```
import { createParamDecorator, ExecutionContext } from
'@nestjs/common';
import { GqlExecutionContext } from '@nestjs/graphql';
import { REQUEST_USER_INFO_KEY } from '../../constants';

export const User = createParamDecorator((field, context:
ExecutionContext) => {
  let request;
  if (context.getType() === 'http') {
    request = context.switchToHttp().getRequest();
  } else if (context.getType() === ('graphql' as any)) {
    request = GqlExecutionContext.create(context).getContext().req;
  }

  const user = request[REQUEST_USER_INFO_KEY];
  return field ? user?.[field] : user;
});
```

```
{
  "cognitive_node": {
   "id": "file:src/modules/iam/authentication/decorators/user.decorator.ts",
   "system": "API-GATEWAY",
   "module": "iam",
   "role": "decorator",
   "purpose": "Provides a custom @User decorator to extract user information from the
HTTP or GraphQL execution context.",
   "inputs": {
    "di_injections": [],
    "imports": [
     "@nestjs/common",
     "@nestjs/graphql",
     "../../constants"
    ],
    "schema_types": [],
    "config_keys": []
    },
    "outputs": {
      "api_surface": ["decorator:@User"],
      "exported_symbols": ["User"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Reads request context"],
      "error_model": ["Returns undefined if user info is absent"]
    },
    "contracts": {
      "invariants": [
        "Always return either the full user object or the specified field if it exists.",
        "Fallback is undefined when user information is not present."
      ],
      "security": [
        "Relies on prior middleware/guards to populate REQUEST_USER_INFO_KEY."
      ]
    },
    "why": {
      "associations": [
        "Simplifies resolver and controller signatures by injecting user context directly."
      ],
      "biases": ["Assumes user info is consistently attached to request."],
      "context": "Decorator pattern in NestJS to integrate authentication layer into GraphQL
and HTTP handlers."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Extraction from HTTP request",
        "Extraction from GraphQL context",
        "Return of specific field when provided",
        "Undefined if user info missing"
      ]
    },
    "links_hint": ["guard:AccessTokenGuard", "middleware:AuthenticationMiddleware"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src/modules/iam/authentication/decorators/user.decorator.ts", "kind": "file" },
      { "id": "decorator:User", "kind": "decorator" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
```

    { "type": "BELONGS_TO", "from": "file:src/modules/iam/authentication/decorators/user.decorator.ts", "to": "module-node:iam" },
    { "type": "OWNS", "from": "file:src/modules/iam/authentication/decorators/user.decorator.ts", "to": "decorator:User" },
    { "type": "IMPORTS", "from": "file:src/modules/iam/authentication/decorators/user.decorator.ts", "to": "file:src/modules/iam/constants" }
   ]
  }
}

src\modules\iam\authentication\guards\access-token.guard.spec.ts

```
import { AccessTokenGuard } from './access-token.guard';

describe('AccessTokenGuard', () => {
  it('should be defined', () => {
    expect(new AccessTokenGuard()).toBeDefined();
  });
});
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\authentication\\guards\\access-token.guard.spec.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "test",
    "purpose": "Provides a unit test scaffold for the AccessTokenGuard, verifying that the guard can be instantiated.",
    "inputs": {
      "di_injections": [],
      "imports": ["./access-token.guard"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["AccessTokenGuard should be constructible."],
      "security": []
    },

    "why": {
      "associations": ["Ensures guard definition exists before functional tests are added."],
      "biases": ["Covers only instantiation, not authorization logic."],
      "context": "Generated or placeholder Jest test for AccessTokenGuard."
    },
    "tests": {
      "spec_files": ["access-token.guard.spec.ts"],
      "key_cases": ["Should define AccessTokenGuard instance"]
    },
    "links_hint": ["guard:AccessTokenGuard"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src\\modules\\iam\\authentication\\guards\\access-token.guard.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\iam\\authentication\\guards\\access-token.guard.spec.ts", "to": "module-node:iam" },
      { "type": "IMPORTS", "from": "file:src\\modules\\iam\\authentication\\guards\\access-token.guard.spec.ts", "to": "guard:AccessTokenGuard" }
    ]
  }
}

src\modules\iam\authentication\guards\access-token.guard.ts

```
import {
  CanActivate,
  ExecutionContext,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { Request } from 'express';
import { GqlExecutionContext } from '@nestjs/graphql';
import { REQUEST_USER_INFO_KEY } from '../../constants';
import { IdpTokenService } from '../services/idp-token.service';

@Injectable()
export class AccessTokenGuard implements CanActivate {
  constructor(private readonly idpTokenService: IdpTokenService) {}
  async canActivate(context: ExecutionContext): Promise<boolean> {
    let request;
    if (context.getType() === 'http') {
      request = context.switchToHttp().getRequest(); //
eslint-disable-line
    } else if (context.getType() === ('graphql' as any)) {
      request = GqlExecutionContext.create(context).getContext<{
        req: Request & { subscriptions?: string };
      }>().req;
      if (
        request.headers === undefined &&
        request.subscriptions !== undefined
      ) {
        return true;
      }
    }

    const accessToken = this.extractAccessTokenFromHeader(request);
    if (accessToken === null) {
      throw new UnauthorizedException();
    }
    try {
      const payload = await
this.idpTokenService.verifyAsync(accessToken);
      console.log('[payload}', payload);
      // const { user } = await this.cognitoService.adminGetUser(
      //   payload.username as string,
      // );
      request[REQUEST_USER_INFO_KEY] = payload;
    } catch {
      throw new UnauthorizedException();
    }
    return true;
  }

  private extractAccessTokenFromHeader(request: Request): string | null
{
    const [authType, accessToken] =
      request.headers.authorization?.split(' ') || [];
    return authType === 'Bearer' ? accessToken || null : null;
  }
}
```
{

```
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\authentication\\guards\\access-token.guard.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "guard",
    "purpose": "Protects HTTP and GraphQL routes by validating Bearer access tokens and
attaching user information to the request context.",
    "inputs": {
      "di_injections": ["IdpTokenService"],
      "imports": [
        "@nestjs/common",
        "express",
        "@nestjs/graphql",
        "../../constants",
        "../services/idp-token.service"
      ],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["AccessTokenGuard"],
      "exported_symbols": ["AccessTokenGuard"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Reads headers",
        "Logs payload",
        "Throws UnauthorizedException on invalid/missing token"
      ],
      "error_model": [
        "UnauthorizedException when no token or invalid token is provided"
      ]
    },
    "contracts": {
      "invariants": [
        "For HTTP, extracts Bearer token from Authorization header.",
        "For GraphQL, extracts token from GqlExecutionContext, allowing bypass for
subscriptions without headers.",
        "Attaches verified payload to request under REQUEST_USER_INFO_KEY."
      ],
      "security": [
        "Strictly requires Bearer tokens unless GraphQL subscription without headers.",
        "Relies on IdpTokenService.verifyAsync for token validation."
      ]
    },
    "why": {
      "associations": [
        "Links authentication layer with NestJS route protection.",
        "Provides consistent request decoration with user info across HTTP and GraphQL."
      ],
      "biases": [
        "Logs payload to console, which may expose sensitive information if not removed."
      ],
      "context": "Guard for IAM module, ensuring that only authenticated requests reach
resolvers or controllers."
    },
    "tests": {
      "spec_files": ["access-token.guard.spec.ts"],
      "key_cases": [
        "Allows request with valid Bearer token",
        "Rejects request without Authorization header",
        "Rejects request with invalid token",
        "Bypasses subscription when headers are absent"
      ]
    },
    "links_hint": [
      "decorator:User",
      "middleware:AuthenticationMiddleware",
      "service:IdpTokenService"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src\\modules\\iam\\authentication\\guards\\access-token.guard.ts", "kind": "file"
},
      { "id": "guard:AccessTokenGuard", "kind": "guard" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\iam\\authentication\\guards\\access-token.guard.ts", "to":
"module-node:iam" },
      { "type": "OWNS", "from":
"file:src\\modules\\iam\\authentication\\guards\\access-token.guard.ts", "to":
"guard:AccessTokenGuard" },
      { "type": "INJECTS", "from": "guard:AccessTokenGuard", "to": "service:IdpTokenService"
}
    ]
  }
}
```

src\modules\iam\authentication\middleware\authentication.middleware.ts

```
// auth.middleware.ts
```

```
import {
  Injectable,
  NestMiddleware,
  UnauthorizedException,
} from '@nestjs/common';
import { Request, Response, NextFunction } from 'express';
import { IdpTokenService } from '../services/idp-token.service';
import { REQUEST_USER_INFO_KEY } from '../../constants';

@Injectable()
export class AuthenticationMiddleware implements NestMiddleware {
  constructor(private readonly idpTokenService: IdpTokenService) {}
  async use(req: Request, res: Response, next: NextFunction) {
    console.log('[headers}', req.headers);
    if (req.headers.authorization === '12345678') {
      return next();
    }
    console.log('[headersauth}', req.headers.authorization);

    const accessToken = this.extractAccessTokenFromHeader(req);
    if (accessToken === null) {
      throw new UnauthorizedException();
    }
    try {
      const payload = await
this.idpTokenService.verifyAsync(accessToken);
      req[REQUEST_USER_INFO_KEY] = payload;
    } catch {
      console.log('UNAUTHORIZED');
      throw new UnauthorizedException();
    }

    return next();
  }

  private extractAccessTokenFromHeader(request: Request): string | null
{
    const [authType, accessToken] =
      request.headers.authorization?.split(' ') || [];
    return authType === 'Bearer' ? accessToken || null : null;
  }
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\authentication\\middleware\\authentication.middleware.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "middleware",
    "purpose": "Intercepts incoming HTTP requests to validate Authorization headers, verify
access tokens, and attach user information to the request object.",
    "inputs": {
      "di_injections": ["IdpTokenService"],
      "imports": [
        "@nestjs/common",
        "express",
        "../services/idp-token.service",
        "../../constants"
      ],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["AuthenticationMiddleware"],
      "exported_symbols": ["AuthenticationMiddleware"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Logs request headers",
        "Modifies request object by attaching user payload",
        "Throws UnauthorizedException on failure"
      ],
      "error_model": ["UnauthorizedException if token is missing or invalid"]
    },
    "contracts": {
      "invariants": [
        "Allows bypass if Authorization header is exactly '12345678' (likely for testing).",
        "Otherwise requires Bearer token in Authorization header.",
        "Attaches verified payload to request under REQUEST_USER_INFO_KEY."
      ],
      "security": [
        "Bearer token must be validated by IdpTokenService.verifyAsync.",
        "Hardcoded bypass '12345678' should be removed or restricted in production."
      ]
    },
    "why": {
      "associations": ["Executes before controllers/resolvers to ensure only authenticated
requests proceed."],
      "biases": ["Includes debug logging and a test bypass path."],
      "context": "Middleware used at global application level to authenticate incoming HTTP requests."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Accepts request with bypass value '12345678'",
        "Accepts request with valid Bearer token",
        "Rejects request with missing Authorization header",
        "Rejects request with invalid token"
      ]
    },
    "links_hint": [
      "guard:AccessTokenGuard",
      "decorator:User",
      "service:IdpTokenService"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src\\modules\\iam\\authentication\\middleware\\authentication.middleware.ts", "kind": "file" },
      { "id": "middleware:AuthenticationMiddleware", "kind": "middleware" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\iam\\authentication\\middleware\\authentication.middleware.ts", "to": "module-node:iam" },
      { "type": "OWNS", "from": "file:src\\modules\\iam\\authentication\\middleware\\authentication.middleware.ts", "to": "middleware:AuthenticationMiddleware" },
      { "type": "INJECTS", "from": "middleware:AuthenticationMiddleware", "to": "service:IdpTokenService" }
    ]
  }
}
```

src\modules\iam\authentication\services\idp-token.service.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { IdpTokenService } from './idp-token.service';

describe('IdpTokenService', () => {
  let service: IdpTokenService;
  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [IdpTokenService],
    }).compile();

    service = module.get<IdpTokenService>(IdpTokenService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\authentication\\services\\idp-token.service.spec.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "test",
    "purpose": "Provides a unit test scaffold for IdpTokenService, ensuring that the service can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./idp-token.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["IdpTokenService should be correctly instantiated in a NestJS DI container during tests."],
      "security": []
    },
    "why": {
      "associations": ["Ensures that the service provider wiring is correct before adding functional tests."],
      "biases": ["Covers only instantiation, not actual token verification logic."],
      "context": "Scaffold or placeholder Jest test for IdpTokenService."

    },
    "tests": {
      "spec_files": ["idp-token.service.spec.ts"],
      "key_cases": ["Should define IdpTokenService instance"]
    },
    "links_hint": ["service:IdpTokenService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src\\modules\\iam\\authentication\\services\\idp-token.service.spec.ts", "kind":
"file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\iam\\authentication\\services\\idp-token.service.spec.ts", "to":
"module-node:iam" },
      { "type": "IMPORTS", "from":
"file:src\\modules\\iam\\authentication\\services\\idp-token.service.spec.ts", "to":
"service:IdpTokenService" }
    ]
  }
}

src\modules\iam\authentication\services\idp-token.service.ts

```
import { Injectable } from '@nestjs/common';
import { UserInfo } from '../../types/iam.types';

@Injectable()
export class IdpTokenService {
  async verifyAsync(token: string) {
    const response = await fetch(
      'http://localhost:3001/authentication/user-info',
      {
        headers: {
          Authorization: `Bearer ${token}`,
        },
      },
    );
    const payload = (await response.json()) as {
      status?: 'SUCCESS';
      data: UserInfo;
    };
    if (payload.status !== 'SUCCESS') {
      throw new Error('Unauthorized: Bad Accesss Token');
    }
    return payload.data;
  }
}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\authentication\\services\\idp-token.service.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "service",
    "purpose": "Handles validation of access tokens by delegating verification to an external identity provider service and returning associated user information.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "@nestjs/common",
        "../../types/iam.types"
      ],
      "schema_types": ["UserInfo"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["verifyAsync"],
      "exported_symbols": ["IdpTokenService"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["HTTP fetch"],
      "error_model": [
        "Throws Error('Unauthorized: Bad Accesss Token') if response status is not SUCCESS"
      ]
    },
    "contracts": {
      "invariants": [
        "Always calls identity provider endpoint with Bearer token.",
        "Returns payload.data typed as UserInfo if verification succeeds."
      ],
      "security": [
        "Depends on secure communication with IdP endpoint at http://localhost:3001/authentication/user-info.",
        "Error handling ensures invalid tokens are rejected."
      ]
    },
    "why": {
      "associations": ["Central component for token verification in IAM module."],

    "biases": ["Hardcodes IdP URL to localhost, which may need environment-driven configuration for deployment."],
    "context": "Used by AccessTokenGuard and AuthenticationMiddleware to validate and attach user information to requests."
  },
  "tests": {
    "spec_files": ["idp-token.service.spec.ts"],
    "key_cases": [
      "Valid token returns UserInfo",
      "Invalid token throws Unauthorized error",
      "Handles malformed JSON responses gracefully"
    ]
  },
  "links_hint": [
    "guard:AccessTokenGuard",
    "middleware:AuthenticationMiddleware",
    "decorator:User",
    "type:UserInfo"
  ]
 },
 "graph_patch": {
  "nodes": [
    { "id": "system:API-GATEWAY", "kind": "system" },
    { "id": "module-node:iam", "kind": "module-node" },
    { "id": "file:src\\modules\\iam\\authentication\\services\\idp-token.service.ts", "kind": "file" },
    { "id": "service:IdpTokenService", "kind": "service" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
    { "type": "BELONGS_TO", "from": "file:src\\modules\\iam\\authentication\\services\\idp-token.service.ts", "to": "module-node:iam" },
    { "type": "OWNS", "from": "file:src\\modules\\iam\\authentication\\services\\idp-token.service.ts", "to": "service:IdpTokenService" },
    { "type": "USES_TYPE", "from": "service:IdpTokenService", "to": "type:UserInfo" }
  ]
 }
}

src\modules\iam\types\iam.types.ts

```
export type UserInfo = {
  username: string;
  roles: string[];
};
```

```json
{
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\types\\iam.types.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "types",
    "purpose": "Defines UserInfo type representing authenticated user identity and associated roles.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["UserInfo"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": ["UserInfo"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "UserInfo.username must be a string.",
        "UserInfo.roles must be an array of strings representing assigned roles."
      ],
      "security": [
        "Represents security-critical user identity attributes; must be sourced from verified tokens only."
      ]
    },
    "why": {
      "associations": ["Provides the contract for identity payloads returned by IdpTokenService."],
      "biases": ["Minimal shape defined, leaving additional claims to future extension."],
      "context": "Central type for IAM module, linking identity verification to request-scoped user info."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "service:IdpTokenService",
      "guard:AccessTokenGuard",
      "middleware:AuthenticationMiddleware",
```

      "decorator:User"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src\\modules\\iam\\types\\iam.types.ts", "kind": "file" },
      { "id": "type:UserInfo", "kind": "type" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\iam\\types\\iam.types.ts", "to":
"module-node:iam" },
      { "type": "OWNS", "from": "file:src\\modules\\iam\\types\\iam.types.ts", "to":
"type:UserInfo" }
    ]
  }
}

src\modules\iam\constants.ts

```
export const REQUEST_USER_INFO_KEY = 'userInfo';
export const REQUEST_ACCESS_TOKEN_KEY = 'accessToken';
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\constants.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "constants",
    "purpose": "Defines shared constants used for attaching user information and access
tokens to request objects.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [
        "REQUEST_USER_INFO_KEY",
        "REQUEST_ACCESS_TOKEN_KEY"
      ],
      "events_pubsub": []
    },
    "behavior": {

```
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "REQUEST_USER_INFO_KEY must always map to the property storing verified user
information on request objects.",
        "REQUEST_ACCESS_TOKEN_KEY must always map to the property storing raw
access token if attached."
      ],
      "security": [
        "Keys should only be used internally to attach sensitive data; must not be leaked
externally."
      ]
    },
    "why": {
      "associations": [
        "Provides stable, centralized keys for attaching identity and token information in
middleware, guards, and decorators."
      ],
      "biases": ["Uses simple string keys to reduce complexity."],
      "context": "Part of IAM module's shared constants, ensuring consistency across
authentication utilities."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "guard:AccessTokenGuard",
      "middleware:AuthenticationMiddleware",
      "decorator:User",
      "service:IdpTokenService"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src\\modules\\iam\\constants.ts", "kind": "file" },
      { "id": "constant:REQUEST_USER_INFO_KEY", "kind": "constant" },
      { "id": "constant:REQUEST_ACCESS_TOKEN_KEY", "kind": "constant" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\iam\\constants.ts", "to":
"module-node:iam" },
```

    { "type": "OWNS", "from": "file:src\\modules\\iam\\constants.ts", "to": "constant:REQUEST_USER_INFO_KEY" },
    { "type": "OWNS", "from": "file:src\\modules\\iam\\constants.ts", "to": "constant:REQUEST_ACCESS_TOKEN_KEY" }
   ]
  }
}

src\modules\iam\iam.module.ts

```
import { Module } from '@nestjs/common';
import { IdpTokenService } from
'./authentication/services/idp-token.service';
import { APP_GUARD } from '@nestjs/core';
import { AccessTokenGuard } from
'./authentication/guards/access-token.guard';

@Module({
  providers: [
    IdpTokenService,
    // {
    //   provide: APP_GUARD,
    //   useClass: AccessTokenGuard,
    // },
  ],
  exports: [IdpTokenService],
})
export class IamModule {}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\iam\\iam.module.ts",
    "system": "API-GATEWAY",
    "module": "iam",
    "role": "module",
    "purpose": "Defines the IAM module, registering identity provider token verification service and optionally enabling global access token guard.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "@nestjs/common",
        "@nestjs/core",
        "./authentication/services/idp-token.service",
        "./authentication/guards/access-token.guard"
      ],
      "schema_types": [],
      "config_keys": []

```
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": ["IamModule"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "IdpTokenService is always registered as a provider.",
        "AccessTokenGuard can be enabled globally by uncommenting the APP_GUARD
provider."
      ],
      "security": [
        "When enabled globally, AccessTokenGuard enforces authentication across all
resolvers/controllers."
      ]
    },
    "why": {
      "associations": [
        "Centralizes IAM-related providers for reuse across API-GATEWAY."
      ],
      "biases": [
        "Leaves guard commented out, likely to enable incremental testing before enforcing
global authentication."
      ],
      "context": "IAM module integrates authentication services and optional guards into the
NestJS dependency injection system."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "service:IdpTokenService",
      "guard:AccessTokenGuard",
      "decorator:User",
      "middleware:AuthenticationMiddleware"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:iam", "kind": "module-node" },
      { "id": "file:src\\modules\\iam\\iam.module.ts", "kind": "file" },
    { "id": "module:IamModule", "kind": "module" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:iam" },
    { "type": "BELONGS_TO", "from": "file:src\\modules\\iam\\iam.module.ts", "to":
"module-node:iam" },
    { "type": "OWNS", "from": "file:src\\modules\\iam\\iam.module.ts", "to":
"module:IamModule" },
    { "type": "INJECTS", "from": "module:IamModule", "to": "service:IdpTokenService" },
    { "type": "CAN_ENABLE", "from": "module:IamModule", "to": "guard:AccessTokenGuard",
"note": "optional APP_GUARD" }
  ]
 }
}
```

src\modules\live-mode\config\live-mode.config.ts

```
import { registerAs } from '@nestjs/config';

export type LiveModeConfig = typeof liveModeConfig;
const liveModeConfig = registerAs('live-mode', () => {
  return {
    freshCallRecordsBaseURL: process.env.FRESH_CALL_RECORDS_BASE_URL!,
  };
});
export default liveModeConfig;
```

```
{
 "cognitive_node": {
  "id": "file:src\\modules\\live-mode\\config\\live-mode.config.ts",
  "system": "API-GATEWAY",
  "module": "live-mode",
  "role": "config",
  "purpose": "Registers namespaced configuration for live-mode, exposing the base URL for
fresh call records.",
  "inputs": {
    "di_injections": [],
    "imports": ["@nestjs/config"],
    "schema_types": [],
    "config_keys": ["FRESH_CALL_RECORDS_BASE_URL"]
  },
  "outputs": {
    "api_surface": [],
    "exported_symbols": ["liveModeConfig", "LiveModeConfig"],
    "events_pubsub": []
  },
  "behavior": {
    "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Requires environment variable FRESH_CALL_RECORDS_BASE_URL to be defined."
      ],
      "security": []
    },
    "why": {
      "associations": ["Provides typed config integration for live mode operations."],
      "biases": ["Uses registerAs to create namespaced config for modular design."],
      "context": "Supports LiveMode module by supplying backend endpoint for fresh call
records."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": ["module:LiveModeModule", "service:FreshCallRecordsService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\config\\live-mode.config.ts", "kind": "file" },
      { "id": "config:live-mode", "kind": "config" },
      { "id": "config:FRESH_CALL_RECORDS_BASE_URL", "kind": "config" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\live-mode\\config\\live-mode.config.ts", "to": "module-node:live-mode" },
      { "type": "EXPOSES", "from": "file:src\\modules\\live-mode\\config\\live-mode.config.ts",
"to": "config:live-mode" },
      { "type": "EXPOSES", "from": "file:src\\modules\\live-mode\\config\\live-mode.config.ts",
"to": "config:FRESH_CALL_RECORDS_BASE_URL" }
    ]
  }
}
```

src\modules\live-mode\graphql\live-mode.graphql

```graphql
type Query {
  freshCallRecords(interval: Int): [FreshCallRecord!]
}


type FreshCallRecord {
  id: String!
  callId: String!
  callerPhoneNumber: String!
  callStartTime: String!
  callEndTime: String
  agent: Agent!
  patient: Patient!
  callStatus: String!
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\live-mode\\graphql\\live-mode.graphql",
    "system": "API-GATEWAY",
    "module": "live-mode",
    "role": "graphql",
    "purpose": "Defines the GraphQL schema for live-mode queries, exposing fresh call
records and their structure.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": ["FreshCallRecord"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["query:freshCallRecords"],
      "exported_symbols": ["FreshCallRecord"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "freshCallRecords query must return a list of FreshCallRecord objects.",
        "FreshCallRecord type must include id, callId, callerPhoneNumber, callStartTime, and
callStatus as non-nullable."
      ],
      "security": ["Resolvers must enforce access control for live call data."]
    },
    "why": {
      "associations": ["Schema acts as the contract between client queries and live call records
service."],
      "biases": ["Keeps schema minimal by focusing only on live call essentials."],
      "context": "Used by LiveModeResolver to expose fresh call records to clients in near
real-time."
    },
```

    "tests": {
      "spec_files": [],
      "key_cases": ["freshCallRecords query returns expected structure"]
    },
    "links_hint": [
      "resolver:LiveModeResolver",
      "service:FreshCallRecordsService",
      "type:Agent",
      "type:Patient"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\graphql\\live-mode.graphql", "kind": "file" },
      { "id": "graphql:FreshCallRecord", "kind": "graphql" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\live-mode\\graphql\\live-mode.graphql", "to": "module-node:live-mode" },
      { "type": "EXPOSES", "from": "file:src\\modules\\live-mode\\graphql\\live-mode.graphql",
"to": "graphql:FreshCallRecord" }
    ]
  }
}

src\modules\live-mode\resolvers\live-mode.resolver.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { LiveModeResolver } from './live-mode.resolver';

describe('LiveModeResolver', () => {
  let resolver: LiveModeResolver;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [LiveModeResolver],
    }).compile();

    resolver = module.get<LiveModeResolver>(LiveModeResolver);
  });

  it('should be defined', () => {
    expect(resolver).toBeDefined();
  });
});
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.spec.ts",
    "system": "API-GATEWAY",
    "module": "live-mode",
    "role": "test",
    "purpose": "Provides a unit test scaffold for LiveModeResolver, ensuring it can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./live-mode.resolver"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["LiveModeResolver should be correctly instantiated during test setup."],
      "security": []
    },
    "why": {
      "associations": ["Ensures the resolver provider is properly wired in NestJS before functional tests are added."],
      "biases": ["Tests only resolver definition, not query logic."],
      "context": "Scaffolded Jest test file for LiveModeResolver."
    },
    "tests": {
      "spec_files": ["live-mode.resolver.spec.ts"],
      "key_cases": ["Should define LiveModeResolver instance"]
    },
    "links_hint": ["resolver:LiveModeResolver"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.spec.ts", "kind": "file" }
    ],
```

    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.spec.ts", "to":
"module-node:live-mode" },
      { "type": "IMPORTS", "from":
"file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.spec.ts", "to":
"resolver:LiveModeResolver" }
    ]
  }
}

src\modules\live-mode\resolvers\live-mode.resolver.ts

```
import { Args, Query, Resolver } from '@nestjs/graphql';
import { LiveModeService } from '../services/live-mode.service';

@Resolver()
export class LiveModeResolver {
  constructor(private readonly liveModeService: LiveModeService) {}
  @Query('freshCallRecords')
  getFreshCallRecords(@Args() args: { interval?: number }) {
    return this.liveModeService.getFreshCallRecords(args.interval);
  }
}
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.ts",
    "system": "API-GATEWAY",
    "module": "live-mode",
    "role": "resolver",
    "purpose": "GraphQL resolver for live-mode, exposing the freshCallRecords query that
retrieves recent call records from the service layer.",
    "inputs": {
      "di_injections": ["LiveModeService"],
      "imports": ["@nestjs/graphql", "../services/live-mode.service"],
      "schema_types": ["FreshCallRecord"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["query:freshCallRecords"],
      "exported_symbols": ["LiveModeResolver"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Delegates call to LiveModeService"],

      "error_model": ["Propagates errors from service layer"]
    },
    "contracts": {
      "invariants": [
        "Always delegates retrieval of fresh call records to LiveModeService.",
        "Returns results consistent with GraphQL FreshCallRecord type."
      ],
      "security": ["Resolver must rely on guards/middleware for authentication if required."]
    },
    "why": {
      "associations": ["Acts as boundary between GraphQL schema and LiveModeService
logic."],
      "biases": ["Keeps resolver minimal, focusing on delegation."],
      "context": "Implements GraphQL query defined in live-mode.graphql."
    },
    "tests": {
      "spec_files": ["live-mode.resolver.spec.ts"],
      "key_cases": ["Delegates freshCallRecords query to service"]
    },
    "links_hint": [
      "service:LiveModeService",
      "graphql:FreshCallRecord"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.ts", "kind": "file" },
      { "id": "resolver:LiveModeResolver", "kind": "resolver" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.ts", "to": "module-node:live-mode"
},
      { "type": "OWNS", "from": "file:src\\modules\\live-mode\\resolvers\\live-mode.resolver.ts",
"to": "resolver:LiveModeResolver" },
      { "type": "INJECTS", "from": "resolver:LiveModeResolver", "to":
"service:LiveModeService" }
    ]
  }
}

src\modules\live-mode\services\live-mode.service.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { LiveModeService } from './live-mode.service';
```

```
describe('LiveModeService', () => {
  let service: LiveModeService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [LiveModeService],
    }).compile();

    service = module.get<LiveModeService>(LiveModeService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\live-mode\\services\\live-mode.service.spec.ts",
    "system": "API-GATEWAY",
    "module": "live-mode",
    "role": "test",
    "purpose": "Provides a unit test scaffold for LiveModeService, ensuring it can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./live-mode.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["LiveModeService should be correctly instantiated during test setup."],
      "security": []
    },
    "why": {
      "associations": ["Ensures service wiring is correct before adding functional tests."],

```
      "biases": ["Covers only service definition, not its business logic."],
      "context": "Scaffolded Jest test file for LiveModeService."
    },
    "tests": {
      "spec_files": ["live-mode.service.spec.ts"],
      "key_cases": ["Should define LiveModeService instance"]
    },
    "links_hint": ["service:LiveModeService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\services\\live-mode.service.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\live-mode\\services\\live-mode.service.spec.ts", "to":
"module-node:live-mode" },
      { "type": "IMPORTS", "from":
"file:src\\modules\\live-mode\\services\\live-mode.service.spec.ts", "to":
"service:LiveModeService" }
    ]
  }
}
```

src\modules\live-mode\services\live-mode.service.ts

```
import { Inject, Injectable } from '@nestjs/common';
import liveModeConfig, { LiveModeConfig } from
'../config/live-mode.config';
import { type ConfigType } from '@nestjs/config';
import { FreshCallRecord } from '../types/live-mode.types';

@Injectable()
export class LiveModeService {
  constructor(
    @Inject(liveModeConfig.KEY)
    private readonly liveModeConfig: ConfigType<LiveModeConfig>,
  ) {}
  async getFreshCallRecords(interval?: number) {
    const queryParams = new URLSearchParams();
    if (interval !== undefined) {
      queryParams.append('interval', String(interval));
    }
    const queryString = queryParams.toString();
    const url =
      this.liveModeConfig.freshCallRecordsBaseURL +
      (queryString !== '' ? queryString : '');
    console.log('[url]', url);
    const response = await fetch(url);
    const {
      data: { callRecords: freshCallRecords },
    } = (await response.json()) as {
      status: 'SUCCESS';
      data: {
        callRecords: FreshCallRecord[];
      };
    };

    return freshCallRecords;
  }
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\live-mode\\services\\live-mode.service.ts",
    "system": "API-GATEWAY",
    "module": "live-mode",
    "role": "service",
    "purpose": "Provides methods to fetch fresh call records from the live-mode backend service, optionally scoped by time interval.",
    "inputs": {
      "di_injections": ["ConfigType<LiveModeConfig>"],
      "imports": [
        "@nestjs/common",
        "@nestjs/config",
        "../config/live-mode.config",
        "../types/live-mode.types"
      ],
      "schema_types": ["FreshCallRecord"],
      "config_keys": ["FRESH_CALL_RECORDS_BASE_URL"]
    },
    "outputs": {
      "api_surface": ["getFreshCallRecords"],
      "exported_symbols": ["LiveModeService"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["HTTP fetch", "Console logging"],
      "error_model": [
        "Throws if fetch fails or response is malformed",
        "Assumes backend always responds with status SUCCESS"
      ]
    },
    "contracts": {
      "invariants": [
        "Always calls backend using configured base URL.",
        "Appends ?interval query param only when interval is provided.",
        "Must return an array of FreshCallRecord."
      ],
      "security": [
        "Relies on trusted backend for validation and access control."
      ]
    },
    "why": {
      "associations": [
        "Acts as a bridge between GraphQL resolvers and external REST API for live call data."
      ],
      "biases": ["Logs URL for debugging, which may reveal backend details."],
      "context": "Supports LiveModeResolver queries for retrieving fresh call records."
    },
    "tests": {
      "spec_files": ["live-mode.service.spec.ts"],
      "key_cases": [
        "Returns freshCallRecords when backend responds with SUCCESS",
        "Appends interval query when provided",
        "Throws on invalid/malformed backend response"
      ]
    },
    "links_hint": [
      "resolver:LiveModeResolver",
      "config:live-mode",
      "type:FreshCallRecord"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\services\\live-mode.service.ts", "kind": "file" },
      { "id": "service:LiveModeService", "kind": "service" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\live-mode\\services\\live-mode.service.ts", "to": "module-node:live-mode"
},
      { "type": "OWNS", "from": "file:src\\modules\\live-mode\\services\\live-mode.service.ts",
"to": "service:LiveModeService" },
    { "type": "INJECTS", "from": "service:LiveModeService", "to": "config:live-mode" },
    { "type": "USES_TYPE", "from": "service:LiveModeService", "to": "type:FreshCallRecord"
}
   ]
  }
}
```

src\modules\live-mode\types\live-mode.types.ts

```
import { CallStatus } from
'src/modules/call-records/types/call-records.types';
import { Maybe } from
'src/modules/common/type-utils/common.type-utils';

export type FreshCallRecord = {
  id: string;
  callId: string;
  callerPhoneNumber: string;
  callStartTime: string;
  callEndTime: Maybe<string>;
  agent: {
    fullName: Maybe<string>;
  };
  patient: {
    fullName: Maybe<string>;
  };
  callStatus: CallStatus;
};
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\live-mode\\types\\live-mode.types.ts",
    "system": "API-GATEWAY",
    "module": "live-mode",
    "role": "types",
    "purpose": "Defines the FreshCallRecord type for live-mode, representing active or
recently completed call records with nullable fields aligned to GraphQL schema.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "src/modules/call-records/types/call-records.types",
        "src/modules/common/type-utils/common.type-utils"
      ],
      "schema_types": ["CallStatus", "Maybe"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": ["FreshCallRecord"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "FreshCallRecord must always include id, callId, callerPhoneNumber, callStartTime,
and callStatus.",
        "Nullable fields (end time, agent/patient names) must be wrapped with Maybe<> to
reflect GraphQL nullability."
      ],
      "security": []
    },
    "why": {
      "associations": ["Provides strong typing for results returned by LiveModeService and
LiveModeResolver."],
      "biases": ["Keeps type minimal, excluding derived fields like sentiment or ratings present
in historical records."],
      "context": "Supports live-mode GraphQL schema for fetching fresh call records."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "service:LiveModeService",
      "resolver:LiveModeResolver",
      "graphql:FreshCallRecord"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\types\\live-mode.types.ts", "kind": "file" },
      { "id": "type:FreshCallRecord", "kind": "type" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from":
"file:src\\modules\\live-mode\\types\\live-mode.types.ts", "to": "module-node:live-mode" },
    { "type": "OWNS", "from": "file:src\\modules\\live-mode\\types\\live-mode.types.ts", "to": "type:FreshCallRecord" },
    { "type": "USES_TYPE", "from": "type:FreshCallRecord", "to": "type:CallStatus" },
    { "type": "USES_TYPE", "from": "type:FreshCallRecord", "to": "type:Maybe" }
  ]
 }
}
```

src\modules\live-mode\live-mode.module.ts

```
import { Module } from '@nestjs/common';
import { LiveModeService } from './services/live-mode.service';
import { LiveModeResolver } from './resolvers/live-mode.resolver';
import { ConfigModule } from '@nestjs/config';
import liveModeConfig from './config/live-mode.config';

@Module({
  imports: [ConfigModule.forFeature(liveModeConfig)],
  providers: [LiveModeService, LiveModeResolver],
})
export class LiveModeModule {}
```

```
{
 "cognitive_node": {
  "id": "file:src\\modules\\live-mode\\live-mode.module.ts",
  "system": "API-GATEWAY",
  "module": "live-mode",
  "role": "module",
  "purpose": "Defines the LiveModeModule, wiring together service, resolver, and configuration for fetching and exposing fresh call records.",
  "inputs": {
   "di_injections": [],
   "imports": [
    "@nestjs/common",
    "@nestjs/config",
    "./services/live-mode.service",
    "./resolvers/live-mode.resolver",
    "./config/live-mode.config"
   ],
   "schema_types": [],
   "config_keys": ["FRESH_CALL_RECORDS_BASE_URL"]
  },
  "outputs": {
   "api_surface": [],
   "exported_symbols": ["LiveModeModule"],
   "events_pubsub": []
  },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "LiveModeService and LiveModeResolver must always be registered as providers.",
        "ConfigModule.forFeature must properly expose live-mode namespace."
      ],
      "security": []
    },
    "why": {
      "associations": ["Aggregates all live-mode functionality under a single NestJS module."],
      "biases": ["Minimal composition, only includes service, resolver, and config integration."],
      "context": "Entry point for live-mode domain in API-GATEWAY, exposing fresh call
records through GraphQL."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "service:LiveModeService",
      "resolver:LiveModeResolver",
      "config:live-mode"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:live-mode", "kind": "module-node" },
      { "id": "file:src\\modules\\live-mode\\live-mode.module.ts", "kind": "file" },
      { "id": "module:LiveModeModule", "kind": "module" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:live-mode" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\live-mode\\live-mode.module.ts",
"to": "module-node:live-mode" },
      { "type": "OWNS", "from": "file:src\\modules\\live-mode\\live-mode.module.ts", "to":
"module:LiveModeModule" },
      { "type": "INJECTS", "from": "module:LiveModeModule", "to": "service:LiveModeService"
},
      { "type": "INJECTS", "from": "module:LiveModeModule", "to":
"resolver:LiveModeResolver" },
      { "type": "IMPORTS", "from": "module:LiveModeModule", "to": "config:live-mode" }
    ]
  }
}
```

src\modules\pub-sub\services\gql-subscriptions.service.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { GqlSubscriptionsService } from './gql-subscriptions.service';

describe('GqlSubscriptionsService', () => {
  let service: GqlSubscriptionsService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [GqlSubscriptionsService],
    }).compile();

    service =
module.get<GqlSubscriptionsService>(GqlSubscriptionsService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

{
  "cognitive_node": {
    "id": "file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.spec.ts",
    "system": "API-GATEWAY",
    "module": "pub-sub",
    "role": "test",
    "purpose": "Provides a unit test scaffold for GqlSubscriptionsService, ensuring it can be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./gql-subscriptions.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []

    },
    "contracts": {
      "invariants": ["GqlSubscriptionsService should be correctly instantiated during test setup."],
      "security": []
    },
    "why": {
      "associations": ["Ensures provider wiring is correct before functional tests are added."],
      "biases": ["Only validates instantiation, not functional behavior."],
      "context": "Scaffolded Jest test file for GqlSubscriptionsService."
    },
    "tests": {
      "spec_files": ["gql-subscriptions.service.spec.ts"],
      "key_cases": ["Should define GqlSubscriptionsService instance"]
    },
    "links_hint": ["service:GqlSubscriptionsService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:pub-sub", "kind": "module-node" },
      { "id": "file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:pub-sub" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.spec.ts", "to": "module-node:pub-sub" },
      { "type": "IMPORTS", "from": "file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.spec.ts", "to": "service:GqlSubscriptionsService" }
    ]
  }
}

src\modules\pub-sub\services\gql-subscriptions.service.ts

```
import { Injectable } from '@nestjs/common';
import {
  EventName,
  EventNameToPayload,
  PubSubService,
} from './pub-sub.service';

import { PubSub } from 'graphql-subscriptions';
import { PubSubAsyncIterableIterator } from
'graphql-subscriptions/dist/pubsub-async-iterable-iterator';
```

```
@Injectable()
export class GqlSubscriptionsService implements PubSubService {
  private pubSub: PubSub<EventNameToPayload>;

  onModuleInit() {
    this.pubSub = new PubSub<EventNameToPayload>();
  }
  publish<E extends EventName>(
    eventName: E,
    payload: EventNameToPayload[E],
  ): Promise<void> {
    return this.pubSub.publish(eventName, payload);
  }
  subscribe<E extends EventName>(
    eventName: E,
  ): PubSubAsyncIterableIterator<EventNameToPayload[E]> {
    return
this.pubSub.asyncIterableIterator<EventNameToPayload[E]>(eventName);
  }
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.ts",
    "system": "API-GATEWAY",
    "module": "pub-sub",
    "role": "service",
    "purpose": "Implements a GraphQL-compatible PubSub service using
graphql-subscriptions to manage publish/subscribe patterns for events.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "@nestjs/common",
        "./pub-sub.service",
        "graphql-subscriptions"
      ],
      "schema_types": ["EventName", "EventNameToPayload"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["publish", "subscribe"],
      "exported_symbols": ["GqlSubscriptionsService"],
      "events_pubsub": ["GraphQL subscription events"]
    },
   "behavior": {
     "side_effects": ["Creates in-memory PubSub broker", "Manages async iterators for
subscriptions"],
     "error_model": ["Relies on graphql-subscriptions internal handling of errors and delivery
guarantees"]
   },
   "contracts": {
     "invariants": [
       "pubSub must be initialized in onModuleInit before publish/subscribe can be used.",
       "publish resolves after event dispatch.",
       "subscribe returns an AsyncIterator for event consumption."
     ],
     "security": [
       "No authentication or filtering at service level; relies on resolvers and guards."
     ]
   },
   "why": {
     "associations": [
       "Provides low-level event delivery mechanism for resolvers handling GraphQL
subscriptions."
     ],
     "biases": [
       "Uses in-memory PubSub, which may not scale horizontally without external broker."
     ],
     "context": "Part of pub-sub module, enabling event-driven updates in GraphQL API."
   },
   "tests": {
     "spec_files": ["gql-subscriptions.service.spec.ts"],
     "key_cases": [
       "Service initializes PubSub on module init",
       "publish successfully dispatches event",
       "subscribe returns async iterator"
     ]
   },
   "links_hint": [
     "service:PubSubService",
     "resolver:CallRecordsResolver",
     "resolver:HistoricalModeResolver"
   ]
  },
  "graph_patch": {
   "nodes": [
     { "id": "system:API-GATEWAY", "kind": "system" },
     { "id": "module-node:pub-sub", "kind": "module-node" },
     { "id": "file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.ts", "kind": "file" },
     { "id": "service:GqlSubscriptionsService", "kind": "service" }
   ],
   "edges": [
    { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:pub-sub" },
    { "type": "BELONGS_TO", "from":
"file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.ts", "to":
"module-node:pub-sub" },
    { "type": "OWNS", "from":
"file:src\\modules\\pub-sub\\services\\gql-subscriptions.service.ts", "to":
"service:GqlSubscriptionsService" },
    { "type": "IMPLEMENTS", "from": "service:GqlSubscriptionsService", "to":
"service:PubSubService" }
   ]
  }
}
```

src\modules\pub-sub\services\pub-sub.service.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { PubSubService } from './pub-sub.service';

describe('PubSubService', () => {
  let service: PubSubService;

  beforeEach(async () => {
    const module: TestingModule = await Test.createTestingModule({
      providers: [PubSubService],
    }).compile();

    service = module.get<PubSubService>(PubSubService);
  });

  it('should be defined', () => {
    expect(service).toBeDefined();
  });
});
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\pub-sub\\services\\pub-sub.service.spec.ts",
    "system": "API-GATEWAY",
    "module": "pub-sub",
    "role": "test",
    "purpose": "Provides a unit test scaffold for PubSubService, checking that the service can
be instantiated within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./pub-sub.service"],
      "schema_types": [],
      "config_keys": []
```

    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Testing framework execution"],
      "error_model": []
    },
    "contracts": {
      "invariants": ["PubSubService should be correctly instantiated in a NestJS DI context during tests."],
      "security": []
    },
    "why": {
      "associations": ["Confirms provider wiring is correct before real event-driven logic is implemented."],
      "biases": ["Covers only definition, does not validate actual publish/subscribe operations."],
      "context": "Scaffold Jest test file for PubSubService."
    },
    "tests": {
      "spec_files": ["pub-sub.service.spec.ts"],
      "key_cases": ["Should define PubSubService instance"]
    },
    "links_hint": ["service:PubSubService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:pub-sub", "kind": "module-node" },
      { "id": "file:src\\modules\\pub-sub\\services\\pub-sub.service.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:pub-sub" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\pub-sub\\services\\pub-sub.service.spec.ts", "to": "module-node:pub-sub" },
      { "type": "IMPORTS", "from": "file:src\\modules\\pub-sub\\services\\pub-sub.service.spec.ts", "to": "service:PubSubService" }
    ]
  }
}

src\modules\pub-sub\services\pub-sub.service.ts

```
import { Injectable } from '@nestjs/common';
import { PubSubAsyncIterableIterator } from
'graphql-subscriptions/dist/pubsub-async-iterable-iterator';
import { CallRecord } from
'src/modules/call-records/types/call-records.types';

export type EventNameToPayload = {
  CALL_RECORD_CREATED: { callRecordCreated: { callRecord: CallRecord }
};
  CALL_RECORD_UPDATED: {
    callRecordUpdated: {
      updateKind: 'call-status' | 'call-recording' |
'call-sentiment-analysis';
      callRecord: CallRecord;
    };
  };
};
export type EventName = keyof EventNameToPayload;
@Injectable()
export abstract class PubSubService {
  abstract publish<E extends EventName>(
    eventName: E,
    payload: EventNameToPayload[E],
  ): Promise<void>;
  abstract subscribe<E extends EventName>(
    eventName: E,
  ): PubSubAsyncIterableIterator<EventNameToPayload[E]>;
}
```

```
{
  "cognitive_node": {
    "id": "file:src\\modules\\pub-sub\\services\\pub-sub.service.ts",
    "system": "API-GATEWAY",
    "module": "pub-sub",
    "role": "abstract-service",
    "purpose": "Defines an abstract PubSubService contract for publishing and subscribing to
call record-related events, strongly typed with event payloads.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "@nestjs/common",
        "graphql-subscriptions/dist/pubsub-async-iterable-iterator",
        "src/modules/call-records/types/call-records.types"
      ],
      "schema_types": ["CallRecord"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["publish", "subscribe"],
      "exported_symbols": ["PubSubService", "EventName", "EventNameToPayload"],
      "events_pubsub": ["CALL_RECORD_CREATED", "CALL_RECORD_UPDATED"]
    },
    "behavior": {
      "side_effects": ["Delegates to underlying PubSub implementation"],
      "error_model": ["Relies on implementation to manage delivery, errors, and async
iteration"]
    },
    "contracts": {
      "invariants": [
        "publish must accept event names and payloads consistent with EventNameToPayload
mapping.",
        "subscribe must return PubSubAsyncIterableIterator for given event."
      ],
      "security": [
        "Events may contain sensitive call record data; must only be consumed by authorized
subscribers."
      ]
    },
    "why": {
      "associations": [
        "Provides abstraction to decouple event-driven infrastructure from business logic.",
        "Allows different implementations (e.g., in-memory PubSub, distributed brokers) while
keeping resolvers stable."
      ],
      "biases": ["Currently tailored to call record lifecycle events only."],
      "context": "Base service implemented by GqlSubscriptionsService and consumed by
resolvers handling GraphQL subscriptions."
    },
    "tests": {
      "spec_files": ["pub-sub.service.spec.ts"],
      "key_cases": [
        "Implementations must satisfy publish/subscribe contracts",
        "Typed event payloads enforced at compile-time"
      ]
    },
    "links_hint": [
      "service:GqlSubscriptionsService",
      "resolver:CallRecordsResolver",
      "resolver:HistoricalModeResolver",
      "type:CallRecord"
    ]
  },
  "graph_patch": {
    "nodes": [
```

    { "id": "system:API-GATEWAY", "kind": "system" },
    { "id": "module-node:pub-sub", "kind": "module-node" },
    { "id": "file:src\\modules\\pub-sub\\services\\pub-sub.service.ts", "kind": "file" },
    { "id": "service:PubSubService", "kind": "abstract-service" },
    { "id": "event:CALL_RECORD_CREATED", "kind": "event" },
    { "id": "event:CALL_RECORD_UPDATED", "kind": "event" }
  ],
  "edges": [
    { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:pub-sub" },
    { "type": "BELONGS_TO", "from":
"file:src\\modules\\pub-sub\\services\\pub-sub.service.ts", "to": "module-node:pub-sub" },
    { "type": "OWNS", "from": "file:src\\modules\\pub-sub\\services\\pub-sub.service.ts", "to":
"service:PubSubService" },
    { "type": "EMITS", "from": "service:PubSubService", "to":
"event:CALL_RECORD_CREATED" },
    { "type": "EMITS", "from": "service:PubSubService", "to":
"event:CALL_RECORD_UPDATED" },
    { "type": "USES_TYPE", "from": "event:CALL_RECORD_CREATED", "to":
"type:CallRecord" },
    { "type": "USES_TYPE", "from": "event:CALL_RECORD_UPDATED", "to":
"type:CallRecord" }
  ]
 }
}

src\modules\pub-sub\pub-sub.module.ts

```
import { Module } from '@nestjs/common';
import { GqlSubscriptionsService } from
'./services/gql-subscriptions.service';
import { PubSubService } from './services/pub-sub.service';

@Module({
  providers: [
    {
      provide: PubSubService,
      useClass: GqlSubscriptionsService,
    },
  ],
  exports: [PubSubService],
})
export class PubSubModule {}
```

{
 "cognitive_node": {
  "id": "file:src\\modules\\pub-sub\\pub-sub.module.ts",
  "system": "API-GATEWAY",

```
  "module": "pub-sub",
  "role": "module",
  "purpose": "Defines the PubSubModule, binding the abstract PubSubService contract to
the GqlSubscriptionsService implementation and exporting it for use across the application.",
  "inputs": {
    "di_injections": [],
    "imports": [
      "@nestjs/common",
      "./services/gql-subscriptions.service",
      "./services/pub-sub.service"
    ],
    "schema_types": [],
    "config_keys": []
  },
  "outputs": {
    "api_surface": [],
    "exported_symbols": ["PubSubModule"],
    "events_pubsub": ["CALL_RECORD_CREATED", "CALL_RECORD_UPDATED"]
  },
  "behavior": {
    "side_effects": [],
    "error_model": []
  },
  "contracts": {
    "invariants": [
      "Always provides PubSubService using GqlSubscriptionsService as its
implementation.",
      "Exported PubSubService is injectable in other modules."
    ],
    "security": [
      "Module itself does not enforce filtering or access control; relies on resolvers and
guards."
    ]
  },
  "why": {
    "associations": ["Centralizes event-driven messaging by exposing PubSubService to
other modules."],
    "biases": ["Currently bound to GraphQL in-memory subscriptions; could be swapped for
distributed broker."],
    "context": "Acts as the provider hub for PubSub functionality in the API-GATEWAY
system."
  },
  "tests": {
    "spec_files": [],
    "key_cases": []
  },
  "links_hint": [
    "service:PubSubService",
      "service:GqlSubscriptionsService",
      "resolver:CallRecordsResolver",
      "resolver:HistoricalModeResolver"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:pub-sub", "kind": "module-node" },
      { "id": "file:src\\modules\\pub-sub\\pub-sub.module.ts", "kind": "file" },
      { "id": "module:PubSubModule", "kind": "module" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:pub-sub" },
      { "type": "BELONGS_TO", "from": "file:src\\modules\\pub-sub\\pub-sub.module.ts", "to":
"module-node:pub-sub" },
      { "type": "OWNS", "from": "file:src\\modules\\pub-sub\\pub-sub.module.ts", "to":
"module:PubSubModule" },
      { "type": "BINDS", "from": "module:PubSubModule", "to": "service:PubSubService",
"note": "Implemented by GqlSubscriptionsService" }
    ]
  }
}
```

src\app.module.ts

```
import {
  ConsoleLogger,
  MiddlewareConsumer,
  Module,
  RequestMethod,
} from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { ApolloDriverConfig, ApolloDriver } from '@nestjs/apollo';
import { GraphQLModule } from '@nestjs/graphql';
import { HistoricalModeModule } from
'./modules/historical-mode/historical-mode.module';
import { PubSubModule } from './modules/pub-sub/pub-sub.module';
import { ApolloServerPluginLandingPageLocalDefault } from
'@apollo/server/plugin/landingPage/default';
import { IamModule } from './modules/iam/iam.module';
import { AuthenticationMiddleware } from
'./modules/iam/authentication/middleware/authentication.middleware';
import { IdpTokenService } from
'./modules/iam/authentication/services/idp-token.service';
```

```
import { CallRecordsModule } from
'./modules/call-records/call-records.module';
import { LiveModeModule } from './modules/live-mode/live-mode.module';

@Module({
  imports: [
    // eslint-disable-next-line
    ConfigModule.forRoot(),
    GraphQLModule.forRootAsync<ApolloDriverConfig>({
      driver: ApolloDriver,
      imports: [IamModule],
      inject: [IdpTokenService],
      useFactory: (idpTokenService: IdpTokenService) => ({
        typePaths: ['./**/*.graphql'],
        subscriptions: {
          'graphql-ws': {
            onConnect: async (context) => {
              const accessToken = context.connectionParams?.accessToken
as
                | string
                | undefined;
              if (accessToken === undefined) return false;

              try {
                const userInfo = await
idpTokenService.verifyAsync(accessToken);
                // @ts-expect-error abc
                context.extra.userInfo = userInfo;
                return true;
              } catch {
                return false;
              }
            },
            onDisconnect: (context) => {
              console.log('Client disconnected',
context.connectionParams);
            },
          },
        },
        playground: false,
        plugins: [ApolloServerPluginLandingPageLocalDefault()],
        context: async ({ req, extra }) => {
          if (extra) {
            console.log('[extra]', extra);

            return extra;
          }
          const userInfo = req?.userInfo;
          return {
            userInfo,
          };
        },
      }),
    }),
    HistoricalModeModule,
    PubSubModule,
    IamModule,
    CallRecordsModule,
    LiveModeModule,
  ],
  providers: [
    // {
    //   provide: APP_GUARD,
    //   useClass: AccessTokenGuard,
    // },
  ],
})
export class AppModule {
  configure(consumer: MiddlewareConsumer) {
    consumer.apply(AuthenticationMiddleware).forRoutes('*');
  }
}
```

{
  "cognitive_node": {
    "id": "file:src\\app.module.ts",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "root-module",
    "purpose": "Defines the root NestJS application module for API-GATEWAY, integrating configuration, GraphQL setup, IAM, PubSub, CallRecords, HistoricalMode, and LiveMode modules.",
    "inputs": {
      "di_injections": ["IdpTokenService"],
      "imports": [
        "@nestjs/common",
        "@nestjs/config",

```
      "@nestjs/apollo",
      "@nestjs/graphql",
      "@apollo/server/plugin/landingPage/default",
      "./modules/historical-mode/historical-mode.module",
      "./modules/pub-sub/pub-sub.module",
      "./modules/iam/iam.module",
      "./modules/iam/authentication/middleware/authentication.middleware",
      "./modules/iam/authentication/services/idp-token.service",
      "./modules/call-records/call-records.module",
      "./modules/live-mode/live-mode.module"
    ],
    "schema_types": [],
    "config_keys": ["all environment variables loaded via ConfigModule"]
  },
  "outputs": {
    "api_surface": [],
    "exported_symbols": ["AppModule"],
    "events_pubsub": ["GraphQL subscriptions for call records"]
  },
  "behavior": {
    "side_effects": [
      "Bootstraps global GraphQL schema and resolvers",
      "Handles WebSocket subscriptions for GraphQL events",
      "Applies AuthenticationMiddleware globally"
    ],
    "error_model": [
      "Subscription connection rejected if access token is invalid or missing",
      "GraphQL context creation fails if user info cannot be resolved"
    ]
  },
  "contracts": {
    "invariants": [
      "All feature modules (HistoricalMode, CallRecords, LiveMode, PubSub, Iam) must be
imported for system functionality.",
      "GraphQLModule must load all *.graphql schemas under project.",
      "AuthenticationMiddleware applies to all routes."
    ],
    "security": [
      "Subscription authentication enforced at connection level via IdpTokenService.",
      "HTTP authentication enforced via AuthenticationMiddleware.",
      "Optional APP_GUARD for global AccessTokenGuard available but commented out."
    ]
  },
  "why": {
    "associations": [
      "Acts as composition root of the API-GATEWAY system.",
      "Centralizes GraphQL server configuration, middleware, and authentication setup."
    ],
      "biases": [
        "Uses in-memory PubSub, limiting horizontal scalability.",
        "Leaves global guard commented out, likely for development flexibility."
      ],
      "context": "Main entry point wiring together all modules and providers into a single
NestJS application."
    },
    "tests": {
      "spec_files": ["test/app.e2e-spec.ts"],
      "key_cases": [
        "AppModule loads without errors",
        "GraphQL subscriptions reject invalid tokens",
        "Middleware attaches userInfo to requests"
      ]
    },
    "links_hint": [
      "module:HistoricalModeModule",
      "module:CallRecordsModule",
      "module:LiveModeModule",
      "module:PubSubModule",
      "module:IamModule",
      "middleware:AuthenticationMiddleware",
      "service:IdpTokenService",
      "guard:AccessTokenGuard"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:src\\app.module.ts", "kind": "file" },
      { "id": "module:AppModule", "kind": "module" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:src\\app.module.ts", "to": "system:API-GATEWAY"
},
      { "type": "OWNS", "from": "file:src\\app.module.ts", "to": "module:AppModule" },
      { "type": "IMPORTS", "from": "module:AppModule", "to": "module:HistoricalModeModule"
},
      { "type": "IMPORTS", "from": "module:AppModule", "to": "module:CallRecordsModule" },
      { "type": "IMPORTS", "from": "module:AppModule", "to": "module:LiveModeModule" },
      { "type": "IMPORTS", "from": "module:AppModule", "to": "module:PubSubModule" },
      { "type": "IMPORTS", "from": "module:AppModule", "to": "module:IamModule" },
      { "type": "INJECTS", "from": "module:AppModule", "to": "service:IdpTokenService" },
      { "type": "APPLIES", "from": "module:AppModule", "to":
"middleware:AuthenticationMiddleware" },
      { "type": "CAN_ENABLE", "from": "module:AppModule", "to":
"guard:AccessTokenGuard", "note": "optional global guard" }
    ]
```

 }
}

src\generate-types.ts

```
import { GraphQLDefinitionsFactory } from '@nestjs/graphql';
import { join } from 'path';

const definitionsFactory = new GraphQLDefinitionsFactory();
void definitionsFactory.generate({
  typePaths: ['./**/*.graphql'],
  path: join(process.cwd(), 'src/graphql.ts'),
  outputAs: 'class',
  watch: true,
  skipResolverArgs: true,
});
```

{
 "cognitive_node": {
  "id": "file:src\\generate-types.ts",
  "system": "API-GATEWAY",
  "module": "root",
  "role": "utility",
  "purpose": "Generates TypeScript type definitions from GraphQL schema files to ensure type safety between schema and code.",
  "inputs": {
   "di_injections": [],
   "imports": [
    "@nestjs/graphql",
    "path"
   ],
   "schema_types": ["All *.graphql files under project"],
   "config_keys": []
  },
  "outputs": {
   "api_surface": [],
   "exported_symbols": [],
   "events_pubsub": []
  },
  "behavior": {
   "side_effects": ["Writes src/graphql.ts file with generated classes", "Watches for schema changes and regenerates types"],
   "error_model": ["Process may fail if GraphQL schema files contain syntax errors"]
  },
  "contracts": {
   "invariants": [
    "Always scans all .graphql files in project for type generation.",

        "Generated file is written to src/graphql.ts.",
        "watch: true ensures regeneration on schema change."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Bridges GraphQL schema definitions with TypeScript code to maintain type alignment."
      ],
      "biases": [
        "Outputs all definitions into a single file (src/graphql.ts)."
      ],
      "context": "Developer tool to keep GraphQL schema and application code synchronized."
    },
    "tests": {
      "spec_files": [],
      "key_cases": ["Successful type generation from valid schema", "Error handling for malformed schema"]
    },
    "links_hint": [
      "file:src\\graphql.ts",
      "graphql:schemas"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:src\\generate-types.ts", "kind": "file" },
      { "id": "utility:GraphQLDefinitionsFactory", "kind": "utility" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:src\\generate-types.ts", "to": "system:API-GATEWAY" },
      { "type": "USES", "from": "file:src\\generate-types.ts", "to": "utility:GraphQLDefinitionsFactory" },
      { "type": "GENERATES", "from": "utility:GraphQLDefinitionsFactory", "to": "file:src\\graphql.ts" }
    ]
  }
}

src\graphql.ts

```
/*
 * -------------------------------------------------------
 * THIS FILE WAS AUTOMATICALLY GENERATED (DO NOT MODIFY)
 * -------------------------------------------------------
 */

/* tslint:disable */
/* eslint-disable */

export class CreateCallRecordInput {
    callId: string;
    callStartTimeStr: string;
    callerPhoneNumber: string;
}

export class UpdateCallRecordInput {
    callId: string;
    callStatus?: Nullable<string>;
    callTransferredTimeStr?: Nullable<string>;
    agent?: Nullable<AgentAsInput>;
    callEndTimeStr?: Nullable<string>;
    callOutcome?: Nullable<string>;
}

export class AgentAsInput {
    fullName?: Nullable<string>;
}

export class PatientAsInput {
    fullName?: Nullable<string>;
}

export class SentimentAsInput {
    value?: Nullable<string>;
    analysis?: Nullable<string>;
}

export class RatingAsInput {
    value?: Nullable<number>;
    info?: Nullable<string>;
}

export class NotifyCallRecordCreatedInput {
    callRecord: CallRecordAsInput;
}

export class CallRecordAsInput {
    id: string;
    callId: string;
    callerPhoneNumber: string;
    callStartTime: string;
    callTransferredTime?: Nullable<string>;
    callEndTime?: Nullable<string>;
    patient: PatientAsInput;
    agent: AgentAsInput;
    callStatus: string;
    callOutcome: string;
    sentiment: SentimentAsInput;
    botRating: RatingAsInput;
    agentRating: RatingAsInput;
    cleanTranscription?: Nullable<string[]>;
    recordingUrl?: Nullable<string>;
}

export class NotifyCallRecordUpdatedInput {
    updateKind: string;
    callRecord: CallRecordAsInput;
}

export class QueryParameters {
    search?: Nullable<string>;
    sort?: Nullable<string>;
    callStartTime?: Nullable<Range>;
    botDuration?: Nullable<Range>;
    agentDuration?: Nullable<Range>;
    status?: Nullable<Nullable<string>[]>;
    outcome?: Nullable<Nullable<string>[]>;
    sentiment?: Nullable<Nullable<string>[]>;
    botRating?: Nullable<Range>;
    agentRating?: Nullable<Range>;
    page?: Nullable<string>;
    limit?: Nullable<string>;
}

export class Range {
    from?: Nullable<string>;
    to?: Nullable<string>;
}

export abstract class IMutation {
    createCallRecord?: Nullable<CallRecord>;
    updateCallRecord?: Nullable<CallRecord>;
    notifyCallRecordCreated?: Nullable<CallRecord>;
    notifyCallRecordUpdated?: Nullable<CallRecord>;
}

export class CallRecord {
    id: string;
    callId: string;
    callerPhoneNumber: string;
    callStartTime: string;
    callTransferredTime?: Nullable<string>;
    callEndTime?: Nullable<string>;
    patient: Patient;
    agent: Agent;
    callStatus: string;
    callOutcome: string;
    sentiment: Sentiment;
    botRating: Rating;
    agentRating: Rating;
    cleanTranscription?: Nullable<string[]>;
    recordingUrl?: Nullable<string>;
}

export class Agent {
    fullName?: Nullable<string>;
}

export class Patient {
    fullName?: Nullable<string>;
}

export class Sentiment {
    value?: Nullable<string>;
    analysis?: Nullable<string>;
}

export class Rating {
    value?: Nullable<number>;
    info?: Nullable<string>;
}

export abstract class ISubscription {
    callRecordCreated?: Nullable<CallRecordCreatedEventData>;
    callRecordUpdated?: Nullable<CallRecordUpdatedEventData>;
    test?: Nullable<number>;
}

export class CallRecordCreatedEventData {
    callRecord: CallRecord;
}

export class CallRecordUpdatedEventData {
    updateKind: string;
    callRecord: CallRecord;
}

export abstract class IQuery {
    terminatedCallRecordsPage?: Nullable<TerminatedCallRecordsPage>;
    test?: Nullable<string>;
    freshCallRecords?: Nullable<FreshCallRecord[]>;
}

export class TerminatedCallRecordsPage {
    meta: Meta;
    terminatedCallRecords: CallRecord[];
}

export class Meta {
    totalDocuments: number;
    totalPages: number;
    page: number;
    limit: number;
    pageSize: number;
}

export class FreshCallRecord {
    id: string;
    callId: string;
    callerPhoneNumber: string;
    callStartTime: string;
    callEndTime?: Nullable<string>;
    agent: Agent;
    patient: Patient;
    callStatus: string;
}
```

```
type Nullable<T> = T | null;
```

{
  "cognitive_node": {
    "id": "file:src\\graphql.ts",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "generated-types",
    "purpose": "Holds automatically generated TypeScript classes and interfaces corresponding to the GraphQL schema definitions, ensuring type alignment between schema and application code.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [
        "CreateCallRecordInput",
        "UpdateCallRecordInput",
        "AgentAsInput",
        "PatientAsInput",
        "SentimentAsInput",
        "RatingAsInput",
        "NotifyCallRecordCreatedInput",
        "CallRecordAsInput",
        "NotifyCallRecordUpdatedInput",
        "QueryParameters",
        "Range",
        "IMutation",
        "CallRecord",
        "Agent",
        "Patient",
        "Sentiment",
        "Rating",
        "ISubscription",
        "CallRecordCreatedEventData",
        "CallRecordUpdatedEventData",
        "IQuery",
        "TerminatedCallRecordsPage",
        "Meta",
        "FreshCallRecord"
      ],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "GraphQL input/output classes",
        "GraphQL abstract query/mutation/subscription interfaces"

```
    ],
    "exported_symbols": [
      "All schema-mapped TypeScript classes"
    ],
    "events_pubsub": [
      "callRecordCreated",
      "callRecordUpdated",
      "test"
    ]
  },
  "behavior": {
    "side_effects": ["Generated file is overwritten automatically"],
    "error_model": ["Manual edits will be lost on regeneration"]
  },
  "contracts": {
    "invariants": [
      "File must always stay in sync with GraphQL schema files (*.graphql).",
      "Classes directly map to schema definitions, including nullability semantics via
Nullable<T>."
    ],
    "security": []
  },
  "why": {
    "associations": [
      "Provides strongly typed API surface for resolvers and services consuming GraphQL
schema."
    ],
    "biases": [
      "Single-file generation may reduce modular clarity but ensures simplicity."
    ],
    "context": "Generated by GraphQLDefinitionsFactory from src/generate-types.ts as part
of developer tooling."
  },
  "tests": {
    "spec_files": [],
    "key_cases": []
  },
  "links_hint": [
    "utility:GraphQLDefinitionsFactory",
    "file:src\\generate-types.ts",
    "resolvers:CallRecordsResolver",
    "resolvers:HistoricalModeResolver",
    "resolvers:LiveModeResolver"
  ]
 },
 "graph_patch": {
  "nodes": [
    { "id": "system:API-GATEWAY", "kind": "system" },
```

      { "id": "file:src\\graphql.ts", "kind": "file" },
      { "id": "generated:GraphQLTypes", "kind": "generated" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:src\\graphql.ts", "to": "system:API-GATEWAY" },
      { "type": "GENERATED_BY", "from": "file:src\\graphql.ts", "to":
"utility:GraphQLDefinitionsFactory" },
      { "type": "CONTAINS", "from": "file:src\\graphql.ts", "to": "generated:GraphQLTypes" }
    ]
  }
}

src\main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  await app.listen(process.env.PORT ?? 4000);
}
void bootstrap();
```

{
  "cognitive_node": {
    "id": "file:src\\main.ts",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "entrypoint",
    "purpose": "Bootstraps the NestJS application by creating and starting the root
AppModule.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/core", "./app.module"],
      "schema_types": [],
      "config_keys": ["PORT"]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Creates NestJS application instance",
        "Starts HTTP server"
      ],
      "error_model": [

```
        "Fails if AppModule cannot be instantiated",
        "Fails if configured port is unavailable"
      ]
    },
    "contracts": {
      "invariants": [
        "Application must start using AppModule as the composition root.",
        "Defaults to port 4000 if PORT environment variable is undefined."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Acts as the system entrypoint where execution begins."
      ],
      "biases": [
        "Relies on environment variable PORT; defaults may not suit all deployment
environments."
      ],
      "context": "Standard NestJS bootstrap file to launch the API-GATEWAY."
    },
    "tests": {
      "spec_files": ["test/app.e2e-spec.ts"],
      "key_cases": [
        "Application starts successfully on default port",
        "Application starts on custom port when PORT is defined"
      ]
    },
    "links_hint": [
      "module:AppModule",
      "file:src\\app.module.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:src\\main.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:src\\main.ts", "to": "system:API-GATEWAY" },
      { "type": "BOOTSTRAPS", "from": "file:src\\main.ts", "to": "module:AppModule" }
    ]
  }
}
```

test\app.e2e-spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { INestApplication } from '@nestjs/common';
```

```typescript
import * as request from 'supertest';
import { App } from 'supertest/types';
import { AppModule } from './../src/app.module';

describe('AppController (e2e)', () => {
  let app: INestApplication<App>;

  beforeEach(async () => {
    const moduleFixture: TestingModule = await
Test.createTestingModule({
      imports: [AppModule],
    }).compile();

    app = moduleFixture.createNestApplication();
    await app.init();
  });

  it('/ (GET)', () => {
    return request(app.getHttpServer())
      .get('/')
      .expect(200)
      .expect('Hello World!');
  });
});
```

```
{
  "cognitive_node": {
    "id": "file:test\\app.e2e-spec.ts",
    "system": "API-GATEWAY",
    "module": "tests",
    "role": "e2e-test",
    "purpose": "Provides an end-to-end test for the API-GATEWAY application, ensuring that
the AppModule initializes correctly and responds to HTTP requests.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "@nestjs/testing",
        "@nestjs/common",
        "supertest",
        "./../src/app.module"
      ],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Bootstraps full application context for testing", "Executes real HTTP
requests against in-memory server"],
      "error_model": ["Fails if AppModule cannot initialize", "Fails if expected route '/' does not
return 'Hello World!'"]
    },
    "contracts": {
      "invariants": [
        "Application must initialize without runtime errors.",
        "GET / must return HTTP 200 and body 'Hello World!'"
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Validates that AppModule wiring works end-to-end, including HTTP server bootstrap."
      ],
      "biases": [
        "Tests only the default GET / route; does not cover GraphQL or domain modules."
      ],
      "context": "Serves as a smoke test for the application to confirm basic operability."
    },
    "tests": {
      "spec_files": ["test\\app.e2e-spec.ts"],
      "key_cases": ["GET / returns 200 with 'Hello World!'"]
    },
    "links_hint": [
      "module:AppModule",
      "file:src\\main.ts",
      "file:src\\app.module.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:test\\app.e2e-spec.ts", "kind": "file" },
      { "id": "test:e2e-AppModule", "kind": "test" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:test\\app.e2e-spec.ts", "to":
"system:API-GATEWAY" },
      { "type": "TESTS", "from": "test:e2e-AppModule", "to": "module:AppModule" }
    ]
  }
```

}

test\jest-e2e.json

```json
{
  "moduleFileExtensions": ["js", "json", "ts"],
  "rootDir": ".",
  "testEnvironment": "node",
  "testRegex": ".e2e-spec.ts$",
  "transform": {
    "^.+\\.(t|j)s$": "ts-jest"
  }
}
```

```json
{
  "cognitive_node": {
    "id": "file:test\\jest-e2e.json",
    "system": "API-GATEWAY",
    "module": "tests",
    "role": "config",
    "purpose": "Provides Jest configuration for running end-to-end tests in the API-GATEWAY project.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": [
        "moduleFileExtensions",
        "rootDir",
        "testEnvironment",
        "testRegex",
        "transform"
      ]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Controls how Jest runs and compiles test files"],
      "error_model": ["Misconfiguration can prevent tests from running or compiling"]
    },
    "contracts": {
      "invariants": [
        "Only files ending with .e2e-spec.ts are matched for execution.",
        "ts-jest must transform TypeScript files before execution."
      ],
```

      "security": []
    },
    "why": {
      "associations": [
        "Defines the runtime environment and file matching for Jest’s e2e test suite."
      ],
      "biases": [
        "Configuration is minimal, assumes ts-jest handles all transformations."
      ],
      "context": "Supports running test\\app.e2e-spec.ts and any other e2e tests in the project."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "file:test\\app.e2e-spec.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:test\\jest-e2e.json", "kind": "file" },
      { "id": "config:jest-e2e", "kind": "config" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:test\\jest-e2e.json", "to": "system:API-GATEWAY" },
      { "type": "OWNS", "from": "file:test\\jest-e2e.json", "to": "config:jest-e2e" },
      { "type": "CONFIGURES", "from": "config:jest-e2e", "to": "test:e2e-AppModule" }
    ]
  }
}

.prettierrc

```
{
  "singleQuote": true,
  "trailingComma": "all"
}
```

{
  "cognitive_node": {
    "id": "file:.prettierrc",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "config",

```
    "purpose": "Defines Prettier formatting rules for the API-GATEWAY codebase to ensure
consistent style across files.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": ["singleQuote", "trailingComma"]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Affects automatic code formatting and lint integrations"],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "All string literals should use single quotes.",
        "Trailing commas should be added where valid (objects, arrays, etc.)."
      ],
      "security": []
    },
    "why": {
      "associations": ["Improves code readability and enforces consistent formatting across
contributors."],
      "biases": ["Minimal config—leaves many Prettier defaults unchanged."],
      "context": "Part of developer tooling and DX (developer experience) setup."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "config:eslint",
      "script:format"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:.prettierrc", "kind": "file" },
      { "id": "config:prettier", "kind": "config" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:.prettierrc", "to": "system:API-GATEWAY" },
      { "type": "OWNS", "from": "file:.prettierrc", "to": "config:prettier" }
    ]
  }
}
```

eslint.config.mjs

```
// @ts-check
import eslint from '@eslint/js';
import eslintPluginPrettierRecommended from
'eslint-plugin-prettier/recommended';
import globals from 'globals';
import tseslint from 'typescript-eslint';

export default tseslint.config(
  {
    ignores: ['eslint.config.mjs'],
  },
  eslint.configs.recommended,
  ...tseslint.configs.recommendedTypeChecked,
  eslintPluginPrettierRecommended,
  {
    languageOptions: {
      globals: {
        ...globals.node,
        ...globals.jest,
      },
      sourceType: 'commonjs',
      parserOptions: {
        projectService: true,
        tsconfigRootDir: import.meta.dirname,
      },
    },
  },
  {
    rules: {
      '@typescript-eslint/no-explicit-any': 'off',
      '@typescript-eslint/no-floating-promises': 'warn',
      '@typescript-eslint/no-unsafe-argument': 'warn',
      'no-restricted-syntax': [
        'error',
        {
          selector:
            'ImportDeclaration[source.value=/graphql-types/] >
ImportSpecifier',
          message:
            'Named imports are not allowed for "graphql-types" imports.
Use "import * as GraphQLTypes [...]" instead.',
        },
      ],
    },
  },
);
```

{
  "cognitive_node": {
    "id": "file:eslint.config.mjs",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "config",
    "purpose": "Defines ESLint configuration for the API-GATEWAY project, integrating TypeScript, Prettier, and custom rules for GraphQL imports.",
    "inputs": {
      "di_injections": [],
      "imports": [
        "@eslint/js",
        "eslint-plugin-prettier/recommended",
        "globals",
        "typescript-eslint"
      ],
      "schema_types": [],
      "config_keys": [
        "ignores",
        "languageOptions",
        "rules"
      ]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Affects linting behavior across project files"],
      "error_model": ["Lint errors if rules are violated"]
    },
    "contracts": {
      "invariants": [
        "All TypeScript files must adhere to recommended eslint and @typescript-eslint rules, except explicitly disabled ones.",
        "No named imports allowed from graphql-types; only namespace imports permitted."
      ],

```
      "security": []
    },
    "why": {
      "associations": [
        "Ensures consistent linting with TypeScript awareness and Prettier integration.",
        "Protects against unsafe TypeScript practices while balancing developer ergonomics."
      ],
      "biases": [
        "Relaxes strictness on 'any' type by turning off no-explicit-any.",
        "Allows floating promises and unsafe arguments but only warns."
      ],
      "context": "Part of developer tooling, enforces code quality and consistency in
collaboration with Prettier."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "config:prettier",
      "script:lint"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:eslint.config.mjs", "kind": "file" },
      { "id": "config:eslint", "kind": "config" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:eslint.config.mjs", "to": "system:API-GATEWAY" },
      { "type": "OWNS", "from": "file:eslint.config.mjs", "to": "config:eslint" },
      { "type": "COOPERATES_WITH", "from": "config:eslint", "to": "config:prettier" }
    ]
  }
}
```

nest-cli.json

```
{
  "$schema": "https://json.schemastore.org/nest-cli",
  "collection": "@nestjs/schematics",
  "sourceRoot": "src",
  "compilerOptions": {
    "deleteOutDir": true
  }
}
```

```
{
  "cognitive_node": {
    "id": "file:nest-cli.json",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "config",
    "purpose": "Configures NestJS CLI behavior for the API-GATEWAY project, defining
schematics collection, source root, and compiler options.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": ["$schema", "collection", "sourceRoot", "compilerOptions.deleteOutDir"]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Controls how Nest CLI generates and compiles project artifacts"],
      "error_model": ["Misconfiguration could break Nest CLI commands or compilation"]
    },
    "contracts": {
      "invariants": [
        "All CLI commands must resolve paths relative to src as the source root.",
        "deleteOutDir ensures the dist folder is cleared before each build."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Provides essential CLI metadata for scaffolding, building, and maintaining NestJS
project."
      ],
      "biases": [
        "Minimal configuration, relies on defaults for most CLI features."
      ],
      "context": "Supports developer workflows by guiding the Nest CLI to correct project paths
and behaviors."
    },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "script:build",
      "script:start",
      "file:src\\app.module.ts"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:nest-cli.json", "kind": "file" },
      { "id": "config:nest-cli", "kind": "config" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:nest-cli.json", "to": "system:API-GATEWAY" },
      { "type": "OWNS", "from": "file:nest-cli.json", "to": "config:nest-cli" }
    ]
  }
}
```

package.json

```
{
  "name": "api-gateway",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r
ts-node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json",
    "gen-types": "node ./dist/generate-types.js"
  },
  "dependencies": {
    "@apollo/server": "^4.12.2",
    "@nestjs/apollo": "^13.1.0",
    "@nestjs/common": "^11.0.1",
    "@nestjs/config": "^4.0.2",
    "@nestjs/core": "^11.0.1",
    "@nestjs/graphql": "^13.1.0",
    "@nestjs/platform-express": "^11.0.1",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.2",
    "dayjs": "^1.11.17",
    "graphql": "^16.11.0",
    "graphql-subscriptions": "^3.0.0",
    "reflect-metadata": "^0.2.2",
    "rxjs": "^7.8.1",
    "ts-morph": "^25.0.1"
  },
  "devDependencies": {
    "@eslint/eslintrc": "^3.2.0",
    "@eslint/js": "^9.18.0",
    "@nestjs/cli": "^11.0.0",
    "@nestjs/schematics": "^11.0.0",
    "@nestjs/testing": "^11.0.1",
    "@types/express": "^5.0.0",
    "@types/jest": "^30.0.0",
    "@types/node": "^22.10.7",
    "@types/supertest": "^6.0.2",
    "eslint": "^9.18.0",
    "eslint-config-prettier": "^10.0.1",
    "eslint-plugin-prettier": "^5.2.2",
    "globals": "^16.0.0",
    "jest": "^30.0.0",
    "prettier": "^3.4.2",
    "source-map-support": "^0.5.21",
    "supertest": "^7.0.0",
    "ts-jest": "^29.2.5",
    "ts-loader": "^9.5.2",
    "ts-node": "^10.9.2",
    "tsconfig-paths": "^4.2.0",
    "typescript": "^5.7.3",
    "typescript-eslint": "^8.20.0"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "json",
      "ts"
    ],
    "rootDir": "src",
    "testRegex": ".*\\.spec\\.ts$",
    "transform": {
      "^.+\\.(t|j)s$": "ts-jest"
    },
    "collectCoverageFrom": [
      "**/*.(t|j)s"
    ],
    "coverageDirectory": "../coverage",
    "testEnvironment": "node"
  }
}
```

```
{
  "cognitive_node": {
    "id": "file:package.json",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "config",
    "purpose": "Defines project metadata, scripts, dependencies, and Jest configuration for
the API-GATEWAY project.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": [
        "name",
        "version",
        "scripts",
        "dependencies",
        "devDependencies",
        "jest"
      ]
    },
    "outputs": {
      "api_surface": ["npm scripts for build, start, lint, test, and type generation"],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Controls package installation, test execution, code formatting, and build
lifecycle"],
      "error_model": ["Misconfigured scripts or missing dependencies can break development
workflow"]
    },
    "contracts": {
      "invariants": [
        "Project name must remain api-gateway to align with ecosystem tooling.",
        "All scripts must resolve within project structure (dist, src, test).",
        "Dependencies define runtime requirements; devDependencies define build/test/lint
tools."
      ],
      "security": [
        "Marked as private and UNLICENSED, ensuring it is not published to npm."
      ]
    },
    "why": {
      "associations": [
        "Central manifest for Node.js ecosystem, used by NestJS CLI, TypeScript, Jest, and
ESLint."
      ],
      "biases": [
        "Minimal metadata (no description/author), focuses only on functional requirements."
      ],
      "context": "Defines the build, lint, and test lifecycle for the API-GATEWAY project."
    },
    "tests": {
      "spec_files": ["*.spec.ts", "*.e2e-spec.ts"],
      "key_cases": ["Unit tests via Jest", "E2E tests via jest-e2e.json config"]
    },
    "links_hint": [
      "file:nest-cli.json",
      "file:eslint.config.mjs",
      "file:.prettierrc",
      "file:test\\jest-e2e.json"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:package.json", "kind": "file" },
      { "id": "config:npm", "kind": "config" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:package.json", "to": "system:API-GATEWAY" },
      { "type": "OWNS", "from": "file:package.json", "to": "config:npm" },
      { "type": "CONFIGURES", "from": "config:npm", "to": "config:jest" },
      { "type": "CONFIGURES", "from": "config:npm", "to": "config:eslint" },
      { "type": "CONFIGURES", "from": "config:npm", "to": "config:prettier" }
    ]
  }
}
```

tsconfig.build.json

```json
{
  "extends": "./tsconfig.json",
  "exclude": ["node_modules", "test", "dist", "**/*spec.ts"]
}
```

{
  "cognitive_node": {
    "id": "file:tsconfig.build.json",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "config",
    "purpose": "Specifies TypeScript configuration for building the project, extending the base tsconfig.json while excluding non-buildable files and directories.",
    "inputs": {
      "di_injections": [],
      "imports": ["./tsconfig.json"],
      "schema_types": [],
      "config_keys": ["extends", "exclude"]
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Controls which files are compiled during production builds"],
      "error_model": ["If excludes are misconfigured, important files may be skipped from build"]
    },
    "contracts": {
      "invariants": [
        "Always extends the root tsconfig.json for base settings.",
        "Excludes node_modules, test files, dist outputs, and spec files from compilation."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Separates build configuration from development/test configuration."
      ],
      "biases": [
        "Excludes all spec.ts files to keep build output clean of tests."
      ],
      "context": "Ensures only application code is compiled into dist during build."
    },
    "tests": {
      "spec_files": [],

```
    "key_cases": []
  },
  "links_hint": [
    "file:tsconfig.json",
    "script:build"
  ]
 },
 "graph_patch": {
  "nodes": [
    { "id": "system:API-GATEWAY", "kind": "system" },
    { "id": "file:tsconfig.build.json", "kind": "file" },
    { "id": "config:tsconfig-build", "kind": "config" }
  ],
  "edges": [
    { "type": "BELONGS_TO", "from": "file:tsconfig.build.json", "to": "system:API-GATEWAY"
},
    { "type": "OWNS", "from": "file:tsconfig.build.json", "to": "config:tsconfig-build" },
    { "type": "EXTENDS", "from": "config:tsconfig-build", "to": "file:tsconfig.json" }
  ]
 }
}
```

tsconfig.json

```
{
  "compilerOptions": {
    "module": "nodenext",
    "moduleResolution": "nodenext",
    "resolvePackageJsonExports": true,
    "esModuleInterop": true,
    "isolatedModules": true,
    "declaration": true,
    "removeComments": true,
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
    "allowSyntheticDefaultImports": true,
    "target": "ES2023",
    "sourceMap": true,
    "outDir": "./dist",
    "baseUrl": "./",
    "incremental": true,
    "skipLibCheck": true,
    "strictNullChecks": true,
    "forceConsistentCasingInFileNames": true,
    "noImplicitAny": false,
    "strictBindCallApply": false,
    "noFallthroughCasesInSwitch": false
  }
}
```

```
{
 "cognitive_node": {
  "id": "file:tsconfig.json",
  "system": "API-GATEWAY",
  "module": "root",
  "role": "config",
  "purpose": "Provides the base TypeScript compiler configuration for the API-GATEWAY project, controlling module resolution, language features, and build output settings.",
  "inputs": {
   "di_injections": [],
   "imports": [],
   "schema_types": [],
   "config_keys": ["compilerOptions"]
  },
  "outputs": {
   "api_surface": [],
   "exported_symbols": [],
   "events_pubsub": []
  },
  "behavior": {
   "side_effects": ["Controls TypeScript transpilation behavior across the project"],
   "error_model": ["Misconfiguration may cause build failures or runtime incompatibility"]
  },
  "contracts": {
   "invariants": [
    "Target JavaScript version must be ES2023.",
    "Output directory for compiled code must be ./dist.",
    "Decorators and metadata must be emitted for NestJS compatibility.",
    "Module resolution strategy is NodeNext."
   ],
   "security": []
  },
  "why": {
   "associations": ["Central TS config ensures consistent compiler settings across builds and tooling."],
   "biases": [
    "Disables some strictness rules (noImplicitAny, strictBindCallApply) for flexibility.",
    "Enables experimental features (decorators, emitDecoratorMetadata) required by NestJS."
   ],
   "context": "Foundation for project-wide TypeScript compilation; extended by tsconfig.build.json for production builds."
  },
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "file:tsconfig.build.json",
      "script:build"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:tsconfig.json", "kind": "file" },
      { "id": "config:tsconfig", "kind": "config" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:tsconfig.json", "to": "system:API-GATEWAY" },
      { "type": "OWNS", "from": "file:tsconfig.json", "to": "config:tsconfig" }
    ]
  }
}
```

README.md

```
<p align="center">
  <a href="http://nestjs.com/" target="blank"><img
src="https://nestjs.com/img/logo-small.svg" width="120" alt="Nest Logo"
/></a>
</p>

[circleci-image]:
https://img.shields.io/circleci/build/github/nestjs/nest/master?token=a
bc123def456
[circleci-url]: https://circleci.com/gh/nestjs/nest

  <p align="center">A progressive <a href="http://nodejs.org"
target="_blank">Node.js</a> framework for building efficient and
scalable server-side applications.</p>
    <p align="center">
<a href="https://www.npmjs.com/~nestjscore" target="_blank"><img
src="https://img.shields.io/npm/v/@nestjs/core.svg" alt="NPM Version"
/></a>
<a href="https://www.npmjs.com/~nestjscore" target="_blank"><img
src="https://img.shields.io/npm/l/@nestjs/core.svg" alt="Package
License" /></a>
```

~~~
<a href="https://www.npmjs.com/~nestjscore" target="_blank"><img
src="https://img.shields.io/npm/dm/@nestjs/common.svg" alt="NPM
Downloads" /></a>
<a href="https://circleci.com/gh/nestjs/nest" target="_blank"><img
src="https://img.shields.io/circleci/build/github/nestjs/nest/master"
alt="CircleCI" /></a>
<a href="https://discord.gg/G7Qnnhy" target="_blank"><img
src="https://img.shields.io/badge/discord-online-brightgreen.svg"
alt="Discord"/></a>
<a href="https://opencollective.com/nest#backer" target="_blank"><img
src="https://opencollective.com/nest/backers/badge.svg" alt="Backers on
Open Collective" /></a>
<a href="https://opencollective.com/nest#sponsor" target="_blank"><img
src="https://opencollective.com/nest/sponsors/badge.svg" alt="Sponsors
on Open Collective" /></a>
  <a href="https://paypal.me/kamilmysliwiec" target="_blank"><img
src="https://img.shields.io/badge/Donate-PayPal-ff3f59.svg" alt="Donate
us"/></a>
    <a href="https://opencollective.com/nest#sponsor"
target="_blank"><img
src="https://img.shields.io/badge/Support%20us-Open%20Collective-41B883
.svg" alt="Support us"></a>
  <a href="https://twitter.com/nestframework" target="_blank"><img
src="https://img.shields.io/twitter/follow/nestframework.svg?style=soci
al&label=Follow" alt="Follow us on Twitter"></a>
</p>
  <!--[![Backers on Open
Collective](https://opencollective.com/nest/backers/badge.svg)](https:/
/opencollective.com/nest#backer)
  [![Sponsors on Open
Collective](https://opencollective.com/nest/sponsors/badge.svg)](https:
//opencollective.com/nest#sponsor)-->

## Description

[Nest](https://github.com/nestjs/nest) framework TypeScript starter
repository.

## Project setup

```bash
$ npm install
```
~~~

~~~markdown
## Compile and run the project

```bash
# development
$ npm run start

# watch mode
$ npm run start:dev

# production mode
$ npm run start:prod
```

## Run tests

```bash
# unit tests
$ npm run test

# e2e tests
$ npm run test:e2e

# test coverage
$ npm run test:cov
```

## Deployment

When you're ready to deploy your NestJS application to production,
there are some key steps you can take to ensure it runs as efficiently
as possible. Check out the [deployment
documentation](https://docs.nestjs.com/deployment) for more
information.

If you are looking for a cloud-based platform to deploy your NestJS
application, check out [Mau](https://mau.nestjs.com), our official
platform for deploying NestJS applications on AWS. Mau makes deployment
straightforward and fast, requiring just a few simple steps:

```bash
$ npm install -g @nestjs/mau
$ mau deploy
~~~

~~~markdown
```

With Mau, you can deploy your application in just a few clicks,
allowing you to focus on building features rather than managing
infrastructure.

## Resources

Check out a few resources that may come in handy when working with
NestJS:

- Visit the [NestJS Documentation](https://docs.nestjs.com) to learn
more about the framework.
- For questions and support, please visit our [Discord
channel](https://discord.gg/G7Qnnhy).
- To dive deeper and get more hands-on experience, check out our
official video [courses](https://courses.nestjs.com/).
- Deploy your application to AWS with the help of [NestJS
Mau](https://mau.nestjs.com) in just a few clicks.
- Visualize your application graph and interact with the NestJS
application in real-time using [NestJS
Devtools](https://devtools.nestjs.com).
- Need help with your project (part-time to full-time)? Check out our
official [enterprise support](https://enterprise.nestjs.com).
- To stay in the loop and get updates, follow us on
[X](https://x.com/nestframework) and
[LinkedIn](https://linkedin.com/company/nestjs).
- Looking for a job, or have a job to offer? Check out our official
[Jobs board](https://jobs.nestjs.com).

## Support

Nest is an MIT-licensed open source project. It can grow thanks to the
sponsors and support by the amazing backers. If you'd like to join
them, please [read more here](https://docs.nestjs.com/support).

## Stay in touch

- Author - [Kamil Myśliwiec](https://twitter.com/kammysliwiec)
- Website - [https://nestjs.com](https://nestjs.com/)
- Twitter - [@nestframework](https://twitter.com/nestframework)

## License
~~~

```
Nest is [MIT
licensed](https://github.com/nestjs/nest/blob/master/LICENSE).
```

{
  "cognitive_node": {
    "id": "file:README.md",
    "system": "API-GATEWAY",
    "module": "root",
    "role": "documentation",
    "purpose": "Provides project overview, setup instructions, testing commands, deployment guidance, and resources for working with the NestJS-based API-GATEWAY.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["README content for developers and contributors"],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Sets expectations for developers using or contributing to the repository"],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Project setup always requires npm install.",
        "Standard scripts (start, test, deploy) are documented and must align with package.json."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Acts as the main entry point for human readers to understand and use the project.",
        "Links to NestJS official documentation and resources."
      ],
      "biases": [
        "Content is boilerplate NestJS starter README, not yet customized for this specific API-GATEWAY domain."
      ],
      "context": "Helps developers onboard quickly and follow best practices for building, testing, and deploying NestJS apps."
    },

```
    "tests": {
      "spec_files": [],
      "key_cases": []
    },
    "links_hint": [
      "script:build",
      "script:start",
      "script:test",
      "script:deploy",
      "config:npm"
    ]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "file:README.md", "kind": "file" },
      { "id": "doc:readme", "kind": "documentation" }
    ],
    "edges": [
      { "type": "BELONGS_TO", "from": "file:README.md", "to": "system:API-GATEWAY" },
      { "type": "OWNS", "from": "file:README.md", "to": "doc:readme" }
    ]
  }
}
```