```
{
  "cognitive_node": {
    "id": "file:cognitive-modules()nestjs-current-understanding.pdf",
    "system": "API-GATEWAY",
    "module": "cognitive-modules",
    "role": "cognitive-module",
    "purpose": "Captures the user’s current understanding of NestJS, focusing on the structure, rules, and patterns needed to implement modules such as call-records, historical-mode, and live-mode.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["rules-of-operation"],
      "exported_symbols": ["nestjs-current-understanding"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Every NestJS module must declare a @Module with imports, providers, and exports as needed.",
        "Resolvers expose GraphQL operations (queries, mutations, subscriptions).",
        "Services encapsulate logic and perform side effects (HTTP calls, PubSub, DB access).",
        "Config modules load environment-dependent keys and are injected into services.",
        "Types and DTOs ensure type safety between GraphQL schema and TypeScript."
      ],
      "security": [
        "Guards and middleware validate access tokens and attach user context.",
        "Decorators allow extraction of user information securely."
      ]
    },
    "why": {
      "associations": [
        "Understanding NestJS requires recognizing the modular structure and the dependency injection container.",
        "Modules like call-records, historical-mode, and live-mode share a repeating design pattern (config + service + resolver + GraphQL schema + types + tests)."
      ],
      "biases": [
```

```
      "The current understanding is project-specific, influenced by API-GATEWAY's architecture."
    ],
    "context": "This cognitive module establishes a baseline of NestJS knowledge, making it possible to plan migrations and measure gaps relative to the fully integrated API-GATEWAY cognitive module."
   },
   "tests": {
    "spec_files": [],
    "key_cases": ["Unit tests for services", "Resolver integration tests", "E2E coverage for GraphQL queries and mutations"]
   },
   "links_hint": [
    "file:src/modules/call-records",
    "file:src/modules/historical-mode",
    "file:src/modules/live-mode",
    "file:cognitive-modules()api-gateway.pdf"
   ]
  },
  "graph_patch": {
   "nodes": [
    { "id": "system:API-GATEWAY", "kind": "system" },
    { "id": "file:cognitive-modules()nestjs-current-understanding.pdf", "kind": "file" },
    { "id": "cognitive-module:nestjs-current-understanding", "kind": "cognitive-module" }
   ],
   "edges": [
    { "type": "BELONGS_TO", "from": "file:cognitive-modules()nestjs-current-understanding.pdf", "to": "system:API-GATEWAY" },
    { "type": "OWNS", "from": "file:cognitive-modules()nestjs-current-understanding.pdf", "to": "cognitive-module:nestjs-current-understanding" }
   ]
  }
}
```


---

src\charts\queries\ChartQuery.ts:

```
import { IsISO8601, IsEnum, IsInt, IsOptional, Min } from 'class-validator';
import { Type } from 'class-transformer';
import { Granularity } from './Granularity';

export class ChartQuery {
  @IsISO8601()
  startDate!: string;

  @IsISO8601()
  endDate!: string;

  @Type(() => Number)
```

```
  @IsInt()
  @Min(1)
  page: number = 1;

  @Type(() => Number)
  @IsInt()
  @Min(1)
  pageSize: number = 100;

  // If provided, it must be one of the three values
  @IsOptional()
  @IsEnum(Granularity)
  granularity?: Granularity;
}
{
  "cognitive_node": {
    "id": "file:src\\charts\\queries\\ChartQuery.ts",
    "system": "API-GATEWAY",
    "module": "charts",
    "role": "dto",
    "purpose": "Defines a query DTO for chart data requests, including date range, pagination,
and granularity validation.",
    "inputs": {
      "di_injections": [],
      "imports": ["class-validator", "class-transformer", "./Granularity"],
      "schema_types": ["Granularity"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["ChartQuery"],
      "exported_symbols": ["ChartQuery"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": ["Validation errors thrown by class-validator decorators when input is
invalid."]
    },
    "contracts": {
      "invariants": [
        "startDate and endDate must be valid ISO8601 strings.",
        "page and pageSize must be integers >= 1.",
        "granularity must be one of the defined enum values if provided."
      ],
      "security": []
    },
    "why": {
      "associations": ["Bridges query inputs from API requests into a validated DTO."],
```

```
      "biases": ["Uses class-validator decorators for runtime validation."],
      "context": "Part of the charts module to standardize query parameters for chart data
retrieval."
    },
    "tests": {
      "spec_files": [],
      "key_cases": [
        "Rejects non-ISO8601 date strings",
        "Rejects page/pageSize less than 1",
        "Accepts valid enum values for granularity"
      ]
    },
    "links_hint": ["enum:Granularity"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:src\\charts\\queries\\ChartQuery.ts", "kind": "file" },
      { "id": "dto:ChartQuery", "kind": "dto" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from": "file:src\\charts\\queries\\ChartQuery.ts", "to":
"module-node:charts" },
      { "type": "OWNS", "from": "file:src\\charts\\queries\\ChartQuery.ts", "to": "dto:ChartQuery"
},
      { "type": "USES_TYPE", "from": "dto:ChartQuery", "to": "enum:Granularity" }
    ]
  }
}
```


—--

src\charts\queries\Granularity.ts

```
export enum Granularity {
  Daily = 'daily',
  Monthly = 'monthly',
  Yearly = 'yearly',
}
```


```
{
  "cognitive_node": {
    "id": "file:src\\charts\\queries\\Granularity.ts",
    "system": "API-GATEWAY",
    "module": "charts",
    "role": "enum",
    "purpose": "Defines the granularity levels (daily, monthly, yearly) for chart queries.",
    "inputs": {
```

```
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["Granularity"],
      "exported_symbols": ["Granularity"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "Granularity must be one of 'daily', 'monthly', or 'yearly'."
      ],
      "security": []
    },
    "why": {
      "associations": ["Used in ChartQuery DTO to restrict allowed granularity values."],
      "biases": ["Enum values tied to common reporting intervals."],
      "context": "Provides a controlled set of options for aggregating chart data."
    },
    "tests": {
      "spec_files": [],
      "key_cases": ["Ensures only valid enum values are accepted in queries."]
    },
    "links_hint": ["dto:ChartQuery"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:src\\charts\\queries\\Granularity.ts", "kind": "file" },
      { "id": "enum:Granularity", "kind": "enum" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from": "file:src\\charts\\queries\\Granularity.ts", "to":
"module-node:charts" },
      { "type": "OWNS", "from": "file:src\\charts\\queries\\Granularity.ts", "to":
"enum:Granularity" }
    ]
  }
}
```

---

src\charts\charts.controller.spec.ts:

```
import { Test, TestingModule } from '@nestjs/testing';
import { ChartsController } from './charts.controller';

describe('ChartsController', () => {
  let controller: ChartsController;

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

    controller = module.get<ChartsController>(ChartsController);
  });

  it('should be defined', () => {
    expect(controller).toBeDefined();
  });
});
{
  "cognitive_node": {
    "id": "file:src\\charts\\charts.controller.spec.ts",
    "system": "API-GATEWAY",
    "module": "charts",
    "role": "test",
    "purpose": "Provides a unit test scaffold to ensure ChartsController can be instantiated
within a NestJS testing module.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./charts.controller"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Bootstraps a minimal NestJS testing context."],
      "error_model": ["Fails if ChartsController cannot be instantiated."]
    },
    "contracts": {
      "invariants": ["ChartsController must be defined after test module compilation."],
      "security": []
    },
```

```
    "why": {
      "associations": ["Confirms controller wiring works before adding functional tests."],
      "biases": ["Covers only instantiation, not real chart logic."],
      "context": "Scaffolded Jest test file for ChartsController."
    },
    "tests": {
      "spec_files": ["charts.controller.spec.ts"],
      "key_cases": ["Controller should be defined in Nest testing context."]
    },
    "links_hint": ["controller:ChartsController"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:src\\charts\\charts.controller.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from": "file:src\\charts\\charts.controller.spec.ts", "to":
"module-node:charts" },
      { "type": "IMPORTS", "from": "file:src\\charts\\charts.controller.spec.ts", "to":
"controller:ChartsController" }
    ]
  }
}
```


---

src\charts\charts.controller.ts:

```
import { Controller, Get, Query } from '@nestjs/common';
import { ChartsService } from './charts.service';
import { ChartQuery } from './queries/ChartQuery';

@Controller('charts')
export class ChartsController {
  constructor(private readonly chartsService: ChartsService) {}

  @Get('median-duration')
  medianDuration(@Query() query: ChartQuery) {
    return this.chartsService.medianDuration(query);
  }

  @Get('handling-overview')
  handlingOverview(@Query() query: ChartQuery) {
    return this.chartsService.handlingOverview(query);
  }
```

```
  @Get('ai-operation-breakdown')
  aiOperationBreakdown(@Query() query: ChartQuery) {
    return this.chartsService.aiOperationBreakdown(query);
  }

  @Get('call-volume')
  callVolume(@Query() query: ChartQuery) {
    return this.chartsService.callVolume(query);
  }
}
{
  "cognitive_node": {
    "id": "file:src\\charts\\charts.controller.ts",
    "system": "API-GATEWAY",
    "module": "charts",
    "role": "controller",
    "purpose": "Exposes HTTP endpoints under /charts to provide analytical data (median
duration, handling overview, AI operation breakdown, call volume).",
    "inputs": {
      "di_injections": ["ChartsService"],
      "imports": ["@nestjs/common", "./charts.service", "./queries/ChartQuery"],
      "schema_types": ["ChartQuery"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "GET /charts/median-duration",
        "GET /charts/handling-overview",
        "GET /charts/ai-operation-breakdown",
        "GET /charts/call-volume"
      ],
      "exported_symbols": ["ChartsController"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Delegates execution to ChartsService methods."],
      "error_model": ["Validation errors from ChartQuery DTO.", "Unhandled service
exceptions."]
    },
    "contracts": {
      "invariants": [
        "Each endpoint must accept query parameters validated by ChartQuery.",
        "Each endpoint delegates to a corresponding ChartsService method."
      ],
      "security": ["Endpoints depend on global middleware/guards for access control."]
    },
    "why": {
      "associations": [
```

```
      "Acts as an interface layer between incoming HTTP requests and the ChartsService business logic.",
      "Groups all chart-related analytical endpoints under the same controller."
     ],
     "biases": ["Uses RESTful GET endpoints rather than GraphQL."],
     "context": "Provides analytics API surface for front-end dashboards or external integrations."
    },
    "tests": {
     "spec_files": ["charts.controller.spec.ts"],
     "key_cases": [
      "Controller should route to ChartsService.medianDuration with query.",
      "Controller should route to ChartsService.handlingOverview with query.",
      "Controller should route to ChartsService.aiOperationBreakdown with query.",
      "Controller should route to ChartsService.callVolume with query."
     ]
    },
    "links_hint": ["service:ChartsService", "dto:ChartQuery"]
   },
   "graph_patch": {
    "nodes": [
     { "id": "system:API-GATEWAY", "kind": "system" },
     { "id": "module-node:charts", "kind": "module-node" },
     { "id": "file:src\\charts\\charts.controller.ts", "kind": "file" },
     { "id": "controller:ChartsController", "kind": "controller" }
    ],
    "edges": [
     { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:charts" },
     { "type": "BELONGS_TO", "from": "file:src\\charts\\charts.controller.ts", "to": "module-node:charts" },
     { "type": "OWNS", "from": "file:src\\charts\\charts.controller.ts", "to": "controller:ChartsController" },
     { "type": "USES", "from": "controller:ChartsController", "to": "service:ChartsService" },
     { "type": "USES_TYPE", "from": "controller:ChartsController", "to": "dto:ChartQuery" }
    ]
   }
}
```


---

src\charts\charts.module.ts

```
import { Module } from '@nestjs/common';
import { ChartsService } from './charts.service';
import { ChartsController } from './charts.controller';

@Module({
  providers: [ChartsService],
  controllers: [ChartsController]
```

```
})
export class ChartsModule {}
{
  "cognitive_node": {
    "id": "file:src\\charts\\charts.module.ts",
    "system": "API-GATEWAY",
    "module": "charts",
    "role": "module",
    "purpose": "Registers the Charts module, wiring the ChartsService provider with the
ChartsController.",
    "inputs": {
      "di_injections": ["ChartsService"],
      "imports": ["@nestjs/common", "./charts.service", "./charts.controller"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["ChartsModule"],
      "exported_symbols": ["ChartsModule"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Encapsulates charts-related logic into a self-contained NestJS
module."],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "ChartsService must be available for injection within this module.",
        "ChartsController must be registered as a controller of this module."
      ],
      "security": []
    },
    "why": {
      "associations": ["Binds controller and service into the NestJS dependency injection
system."],
      "biases": ["No external configuration or imports yet, limited to internal service and
controller."],
      "context": "Enables charts-related HTTP endpoints by connecting controller with service
logic."
    },
    "tests": {
      "spec_files": [],
      "key_cases": ["Module compiles with ChartsService and ChartsController available."]
    },
    "links_hint": ["service:ChartsService", "controller:ChartsController"]
  },
  "graph_patch": {
```

```
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:src\\charts\\charts.module.ts", "kind": "file" },
      { "id": "module:ChartsModule", "kind": "module" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from": "file:src\\charts\\charts.module.ts", "to":
"module-node:charts" },
      { "type": "OWNS", "from": "file:src\\charts\\charts.module.ts", "to": "module:ChartsModule"
},
      { "type": "REGISTERS", "from": "module:ChartsModule", "to": "service:ChartsService" },
      { "type": "REGISTERS", "from": "module:ChartsModule", "to":
"controller:ChartsController" }
    ]
  }
}
```


—————————————————————————————

src\charts\charts.service.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { ChartsService } from './charts.service';

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

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

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

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


```
{
  "cognitive_node": {
    "id": "file:src\\charts\\charts.service.spec.ts",
    "system": "API-GATEWAY",
    "module": "charts",
    "role": "test",
    "purpose": "Provides a basic unit test scaffold to verify ChartsService can be instantiated
within a NestJS testing module.",
```

```
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/testing", "./charts.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Bootstraps a minimal NestJS testing context."],
      "error_model": ["Fails if ChartsService cannot be instantiated."]
    },
    "contracts": {
      "invariants": ["ChartsService must be defined after test module compilation."],
      "security": []
    },
    "why": {
      "associations": ["Ensures ChartsService is properly registered as a provider."],
      "biases": ["Covers only instantiation, no functional logic."],
      "context": "Scaffolded Jest test file for ChartsService."
    },
    "tests": {
      "spec_files": ["charts.service.spec.ts"],
      "key_cases": ["ChartsService should be defined in Nest testing context."]
    },
    "links_hint": ["service:ChartsService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:src\\charts\\charts.service.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from": "file:src\\charts\\charts.service.spec.ts", "to":
"module-node:charts" },
      { "type": "IMPORTS", "from": "file:src\\charts\\charts.service.spec.ts", "to":
"service:ChartsService" }
    ]
  }
}
```


—----------

---

src\charts\charts.service.ts

```
import { Injectable } from '@nestjs/common';
import { ChartQuery } from './queries/ChartQuery';

@Injectable()
export class ChartsService {
  private readonly backendBaseUrl = 'http://localhost:5000';

  private toQueryRecord(query: ChartQuery): Record<string, string> {
    const record: Record<string, string> = {
      startDate: query.startDate,
      endDate: query.endDate,
      page: String(query.page),
      pageSize: String(query.pageSize),
    };
    if (query.granularity) {
      record.granularity = String(query.granularity);
    }
    return record;
  }

  private async forwardToBackend(
    path: string,
    chartQuery: ChartQuery,
  ): Promise<any> {
    const url = new URL(path, this.backendBaseUrl);
    const params = this.toQueryRecord(chartQuery);

    for (const [key, value] of Object.entries(params)) {
      url.searchParams.set(key, value);
    }

    const response = await fetch(url.toString());
    if (!response.ok) {
      const text = await response.text().catch(() => '');
      throw new Error(`Backend returned ${response.status}: ${text}`);
    }
    return response.json();
  }

  medianDuration(query: ChartQuery) {
    return this.forwardToBackend(
      '/api/call-records/agreggated/median-duration',
      query,
    );
  }

  handlingOverview(query: ChartQuery) {
```

```
    return this.forwardToBackend(
      '/api/call-records/agreggated/handling-overview',
      query,
    );
  }

  aiOperationBreakdown(query: ChartQuery) {
    return this.forwardToBackend(
      '/api/call-records/agreggated/ai-operation-breakdown',
      query,
    );
  }

  callVolume(query: ChartQuery) {
    return this.forwardToBackend(
      '/api/call-records/agreggated/call-volume',
      query,
    );
  }
}
{
  "cognitive_node": {
    "id": "file:src\\charts\\charts.service.ts",
    "system": "API-GATEWAY",
    "module": "charts",
    "role": "service",
    "purpose": "Acts as a proxy between the charts controller and the backend analytics
service, forwarding validated queries and returning backend responses.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/common", "./queries/ChartQuery"],
      "schema_types": ["ChartQuery"],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [
        "medianDuration(query: ChartQuery)",
        "handlingOverview(query: ChartQuery)",
        "aiOperationBreakdown(query: ChartQuery)",
        "callVolume(query: ChartQuery)"
      ],
      "exported_symbols": ["ChartsService"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Performs HTTP requests to backend service at http://localhost:5000."],
      "error_model": [
        "Throws Error when backend responds with non-2xx status.",
```

```
        "Propagates network errors from fetch."
      ]
    },
    "contracts": {
      "invariants": [
        "startDate and endDate must be valid ISO8601 strings (validated in ChartQuery).",
        "Backend endpoint paths must remain stable for analytics endpoints.",
        "Responses are expected to be JSON."
      ],
      "security": ["Relies on backend service for authentication/authorization if required."]
    },
    "why": {
      "associations": [
        "Abstracts query forwarding to backend APIs so that controllers remain thin.",
        "Converts DTO (ChartQuery) into URLSearchParams for backend compatibility."
      ],
      "biases": ["Backend base URL is hardcoded, limiting configurability."],
      "context": "Enables dashboard analytics by exposing service methods called by
ChartsController."
    },
    "tests": {
      "spec_files": ["charts.service.spec.ts"],
      "key_cases": [
        "Successfully forwards valid queries to backend and returns JSON.",
        "Throws error when backend responds with non-2xx status.",
        "Correctly serializes granularity when provided."
      ]
    },
    "links_hint": ["controller:ChartsController", "dto:ChartQuery"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:charts", "kind": "module-node" },
      { "id": "file:src\\charts\\charts.service.ts", "kind": "file" },
      { "id": "service:ChartsService", "kind": "service" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:charts" },
      { "type": "BELONGS_TO", "from": "file:src\\charts\\charts.service.ts", "to":
"module-node:charts" },
      { "type": "OWNS", "from": "file:src\\charts\\charts.service.ts", "to": "service:ChartsService"
},
      { "type": "USES_TYPE", "from": "service:ChartsService", "to": "dto:ChartQuery" }
    ]
  }
}
```

—--

src\app.controller.spec.ts

```
import { Test, TestingModule } from '@nestjs/testing';
import { AppController } from './app.controller';
import { AppService } from './app.service';

describe('AppController', () => {
  let appController: AppController;

  beforeEach(async () => {
    const app: TestingModule = await Test.createTestingModule({
      controllers: [AppController],
      providers: [AppService],
    }).compile();

    appController = app.get<AppController>(AppController);
  });

  describe('root', () => {
    it('should return "Hello World!"', () => {
      expect(appController.getHello()).toBe('Hello World!');
    });
  });
});
{
  "cognitive_node": {
    "id": "file:src\\app.controller.spec.ts",
    "system": "API-GATEWAY",
    "module": "app",
    "role": "test",
    "purpose": "Provides a unit test to verify that AppController returns the expected 'Hello
World!' string.",
    "inputs": {
      "di_injections": ["AppService"],
      "imports": ["@nestjs/testing", "./app.controller", "./app.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": [],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Bootstraps a minimal NestJS test module."],
      "error_model": ["Fails test if AppController.getHello() does not return expected string."]
    },
```

```
    "contracts": {
      "invariants": ["AppController.getHello() must return 'Hello World!'."],
      "security": []
    },
    "why": {
      "associations": ["Confirms baseline controller and service wiring works in NestJS."],
      "biases": ["Covers only trivial output, no business logic."],
      "context": "Default scaffold test generated by NestJS starter project."
    },
    "tests": {
      "spec_files": ["app.controller.spec.ts"],
      "key_cases": ["AppController.getHello should return 'Hello World!'."]
    },
    "links_hint": ["controller:AppController", "service:AppService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:app", "kind": "module-node" },
      { "id": "file:src\\app.controller.spec.ts", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:app" },
      { "type": "BELONGS_TO", "from": "file:src\\app.controller.spec.ts", "to":
"module-node:app" },
      { "type": "IMPORTS", "from": "file:src\\app.controller.spec.ts", "to":
"controller:AppController" },
      { "type": "IMPORTS", "from": "file:src\\app.controller.spec.ts", "to": "service:AppService" }
    ]
  }
}
```


—--

src\app.controller.ts

```
import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }
}
{
```

```
  "cognitive_node": {
    "id": "file:src\\app.controller.ts",
    "system": "API-GATEWAY",
    "module": "app",
    "role": "controller",
    "purpose": "Defines the root controller exposing a GET / endpoint that returns a simple
greeting.",
    "inputs": {
      "di_injections": ["AppService"],
      "imports": ["@nestjs/common", "./app.service"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["GET /"],
      "exported_symbols": ["AppController"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Delegates call to AppService.getHello()."],
      "error_model": []
    },
    "contracts": {
      "invariants": ["GET / must return the string provided by AppService.getHello()."],
      "security": []
    },
    "why": {
      "associations": ["Acts as entry point controller for root path of application."],
      "biases": ["Implements default NestJS starter endpoint."],
      "context": "Provides a baseline endpoint to verify application setup and connectivity."
    },
    "tests": {
      "spec_files": ["app.controller.spec.ts"],
      "key_cases": ["AppController.getHello returns 'Hello World!' when service is correctly
wired."]
    },
    "links_hint": ["service:AppService"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:app", "kind": "module-node" },
      { "id": "file:src\\app.controller.ts", "kind": "file" },
      { "id": "controller:AppController", "kind": "controller" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:app" },
      { "type": "BELONGS_TO", "from": "file:src\\app.controller.ts", "to": "module-node:app" },
```

```
      { "type": "OWNS", "from": "file:src\\app.controller.ts", "to": "controller:AppController" },
      { "type": "USES", "from": "controller:AppController", "to": "service:AppService" }
    ]
  }
}
```

---

src\app.module.ts

```
import { Module } from '@nestjs/common';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { ChartsModule } from './charts/charts.module';

@Module({
  imports: [ChartsModule],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}
{
  "cognitive_node": {
    "id": "file:src\\app.module.ts",
    "system": "API-GATEWAY",
    "module": "app",
    "role": "module",
    "purpose": "Root NestJS module that wires together AppController, AppService, and
imports ChartsModule.",
    "inputs": {
      "di_injections": ["AppService"],
      "imports": ["@nestjs/common", "./app.controller", "./app.service",
"./charts/charts.module"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["AppModule"],
      "exported_symbols": ["AppModule"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": ["Acts as the application root module for bootstrapping NestJS."],
      "error_model": []
    },
    "contracts": {
      "invariants": [
        "AppController must be registered as a controller.",
        "AppService must be available as a provider.",
```

```
        "ChartsModule must be imported for chart-related functionality."
      ],
      "security": []
    },
    "why": {
      "associations": ["Serves as the composition root of the application."],
      "biases": ["Minimal root module that directly imports one feature module."],
      "context": "Defines the top-level wiring of the API-GATEWAY application."
    },
    "tests": {
      "spec_files": [],
      "key_cases": ["Application bootstraps with ChartsModule, AppController, and
AppService."]
    },
    "links_hint": ["controller:AppController", "service:AppService", "module:ChartsModule"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:app", "kind": "module-node" },
      { "id": "file:src\\app.module.ts", "kind": "file" },
      { "id": "module:AppModule", "kind": "module" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:app" },
      { "type": "BELONGS_TO", "from": "file:src\\app.module.ts", "to": "module-node:app" },
      { "type": "OWNS", "from": "file:src\\app.module.ts", "to": "module:AppModule" },
      { "type": "IMPORTS", "from": "module:AppModule", "to": "module:ChartsModule" },
      { "type": "REGISTERS", "from": "module:AppModule", "to": "controller:AppController" },
      { "type": "REGISTERS", "from": "module:AppModule", "to": "service:AppService" }
    ]
  }
}
```


—————————————————————————————

src\app.service.ts

```
import { Injectable } from '@nestjs/common';

@Injectable()
export class AppService {
  getHello(): string {
    return 'Hello World!';
  }
}
```


```
{
  "cognitive_node": {
    "id": "file:src\\app.service.ts",
```

```
    "system": "API-GATEWAY",
    "module": "app",
    "role": "service",
    "purpose": "Provides a simple service method returning the greeting message 'Hello
World!'.",
    "inputs": {
      "di_injections": [],
      "imports": ["@nestjs/common"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["getHello(): string"],
      "exported_symbols": ["AppService"],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [],
      "error_model": []
    },
    "contracts": {
      "invariants": ["getHello() must consistently return the string 'Hello World!'."],
      "security": []
    },
    "why": {
      "associations": ["Acts as the backing provider for AppController."],
      "biases": ["Implements only trivial logic as a NestJS starter service."],
      "context": "Default service demonstrating how controllers delegate to services."
    },
    "tests": {
      "spec_files": ["app.controller.spec.ts"],
      "key_cases": ["AppService.getHello returns 'Hello World!'."]
    },
    "links_hint": ["controller:AppController"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:app", "kind": "module-node" },
      { "id": "file:src\\app.service.ts", "kind": "file" },
      { "id": "service:AppService", "kind": "service" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:app" },
      { "type": "BELONGS_TO", "from": "file:src\\app.service.ts", "to": "module-node:app" },
      { "type": "OWNS", "from": "file:src\\app.service.ts", "to": "service:AppService" },
      { "type": "USED_BY", "from": "service:AppService", "to": "controller:AppController" }
    ]
```

```
  }
}
```

—--

src\main.ts

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

async function bootstrap() {
  const app = await NestFactory.create(AppModule);

  // Turn on validation
  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
      whitelist: true,
      forbidNonWhitelisted: false,
    }),
  );

  await app.listen(3000);
}
bootstrap();
{
  "cognitive_node": {
    "id": "file:src\\main.ts",
    "system": "API-GATEWAY",
    "module": "app",
    "role": "bootstrap",
    "purpose": "Bootstraps the NestJS application, enabling global validation and starting the
HTTP server.",
    "inputs": {
      "di_injections": ["AppModule"],
      "imports": ["@nestjs/core", "./app.module", "@nestjs/common"],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["Nest application instance"],
      "exported_symbols": [],
      "events_pubsub": []
    },
    "behavior": {
      "side_effects": [
        "Creates NestJS application from AppModule.",
        "Registers global ValidationPipe with transformation and whitelisting.",
```

```
      "Starts server on port 3000."
    ],
    "error_model": [
      "Bootstrap errors if AppModule fails to compile.",
      "Runtime error if port 3000 is unavailable."
    ]
  },
  "contracts": {
    "invariants": [
      "ValidationPipe must always sanitize and transform incoming requests.",
      "Application must listen on defined port (3000)."
    ],
    "security": [
      "Whitelist property ensures only validated DTO fields are processed."
    ]
  },
  "why": {
    "associations": [
      "Acts as the application entry point and orchestrator of startup logic.",
      "Global pipes enforce DTO validation across all controllers."
    ],
    "biases": ["Validation settings are globally enforced, not module-specific."],
    "context": "Ensures consistent request validation for the entire API-GATEWAY project."
  },
  "tests": {
    "spec_files": ["app.controller.spec.ts", "app.e2e-spec.ts"],
    "key_cases": [
      "Application bootstraps successfully.",
      "ValidationPipe transforms and rejects invalid input."
    ]
  },
  "links_hint": ["module:AppModule"]
 },
 "graph_patch": {
  "nodes": [
   { "id": "system:API-GATEWAY", "kind": "system" },
   { "id": "module-node:app", "kind": "module-node" },
   { "id": "file:src\\main.ts", "kind": "file" }
  ],
  "edges": [
   { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:app" },
   { "type": "BELONGS_TO", "from": "file:src\\main.ts", "to": "module-node:app" },
   { "type": "BOOTSTRAPS", "from": "file:src\\main.ts", "to": "module:AppModule" }
  ]
 }
}
```

—--

package.json

```
{
  "name": "charts-proxy",
  "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"
  },
  "dependencies": {
    "@nestjs/common": "^11.0.1",
    "@nestjs/core": "^11.0.1",
    "@nestjs/platform-express": "^11.0.1",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.2",
    "reflect-metadata": "^0.2.2",
    "rxjs": "^7.8.1"
  },
  "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": "manifest",
    "purpose": "Defines project metadata, scripts, dependencies, and testing configuration for
the charts-proxy NestJS application.",
    "inputs": {
      "di_injections": [],
      "imports": [],
      "schema_types": [],
      "config_keys": []
    },
    "outputs": {
      "api_surface": ["npm scripts", "jest config"],
      "exported_symbols": ["project metadata"],
      "events_pubsub": []
    },
```

```
    "behavior": {
      "side_effects": [
        "Enables build, start, lint, and test workflows.",
        "Configures Jest for unit and e2e tests."
      ],
      "error_model": ["Build or test failures if misconfigured."]
    },
    "contracts": {
      "invariants": [
        "Dependencies must match required NestJS and TypeScript versions.",
        "Scripts provide consistent development and testing commands."
      ],
      "security": []
    },
    "why": {
      "associations": [
        "Central node manifest enabling dependency management and project lifecycle
commands."
      ],
      "biases": ["Project named charts-proxy, scoped as a NestJS service."],
      "context": "Root configuration required to run, build, and test the API-GATEWAY charts
proxy service."
    },
    "tests": {
      "spec_files": [],
      "key_cases": ["Jest runs with ts-jest transformer.", "Prettier and ESLint formatting works
as expected."]
    },
    "links_hint": ["file:tsconfig.json", "file:jest-e2e.json"]
  },
  "graph_patch": {
    "nodes": [
      { "id": "system:API-GATEWAY", "kind": "system" },
      { "id": "module-node:root", "kind": "module-node" },
      { "id": "file:package.json", "kind": "file" }
    ],
    "edges": [
      { "type": "OWNS", "from": "system:API-GATEWAY", "to": "module-node:root" },
      { "type": "BELONGS_TO", "from": "file:package.json", "to": "module-node:root" }
    ]
  }
}
```