TemplateController

September 4, 2026

TemplateController

Kind: Controller

Source: `atloria-monorepo/apps/api/src/template/template.controller.ts`

Controller for document template operations.
Provides endpoints for browsing, using, and creating templates.

`TemplateController` is a NestJS controller responsible for document template operations in the API. It exposes HTTP endpoints to browse available templates, create/manage templates, and apply a template to generate documents. It sits at the edge of the backend, validating request inputs and delegating business logic to the underlying template service layer.

Diagram

```
graph LR
  Client[API Client] -->|HTTP request| TC[TemplateController]
  TC -->|validate/parse DTOs| Pipes[Nest Pipes/Guards]
  TC -->|delegate| TS[TemplateService]
  TS -->|read/write| DB[(Database)]
  TS -->|load/store| FS[(Template Storage)]
  TS -->|generate document| DOC[Document Builder/Renderer]
  TS -->|response DTO| TC
  TC -->|HTTP response| Client
```

Usage

```
// Example: calling TemplateController endpoints from a client (Node.js / browser).
// Adjust baseUrl and routes to match your actual API paths.

async function listTemplates(baseUrl: string, token: string) {
  const res = await fetch(`${baseUrl}/templates`, {
    method: "GET",
    headers: { Authorization: `Bearer ${token}` },
```

```
  });

  if (!res.ok) throw new Error(`Failed to list templates: ${res.status}`);
  return res.json(); // e.g., [{ id, name, category, ... }]
}

async function createTemplate(baseUrl: string, token: string) {
  const res = await fetch(`${baseUrl}/templates`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({
      name: "NDA v1",
      description: "Standard NDA template",
      // ...other fields expected by your CreateTemplateDto
    }),
  });

  if (!res.ok) throw new Error(`Failed to create template: ${res.status}`);
  return res.json(); // e.g., { id, ... }
}

async function applyTemplate(baseUrl: string, token: string, templateId: string) {
  const res = await fetch(`${baseUrl}/templates/${templateId}/use`, {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
      Authorization: `Bearer ${token}`,
    },
    body: JSON.stringify({
      variables: { companyName: "Atloria, Inc." },
      // ...other fields expected by your "use template" DTO
    }),
  });

  if (!res.ok) throw new Error(`Failed to use template: ${res.status}`);

  // Could be JSON metadata or a file stream depending on implementation.
  const contentType = res.headers.get("content-type") ?? "";
  return contentType.includes("application/json") ? res.json() : res.blob();
}

// Example runner
(async () => {
  const baseUrl = "https://api.example.com";
  const token = process.env.API_TOKEN!;

  const templates = await listTemplates(baseUrl, token);
  const created = await createTemplate(baseUrl, token);
  const output = await applyTemplate(baseUrl, token, created.id);
```

```
  console.log({ templatesCount: templates.length, createdId: created.id, outputType: typeof out
})();
```

AI Coding Instructions

- Keep controller methods thin: validate/transform request DTOs and delegate all business logic to the template service; avoid embedding persistence or rendering logic in the controller.
- Use NestJS DTOs + `ValidationPipe` consistently for all inputs (query/params/body) to prevent template injection and malformed payloads from reaching the service.
- Maintain stable route patterns for template listing/creation vs. “use/apply” actions (e.g., `POST /templates/:id/use`) and document the response type (JSON vs file/stream) to avoid client breakages.
- Ensure authentication/authorization guards are applied uniformly, especially for template creation and any endpoint that can access private templates or generate documents.
- When adding new endpoints, align return shapes with existing response DTOs and handle errors with Nest exceptions (`BadRequestException` , `NotFoundException`) rather than raw errors.

Relationships

- MODULE_DECLARES → `list`
- MODULE_DECLARES → `getById`
- MODULE_DECLARES → `create`
- MODULE_DECLARES → `useTemplate`
- MODULE_DECLARES → `saveAsTemplate`
- MODULE_DECLARES → `update`
- MODULE_DECLARES → `delete`
- DEPENDS_ON → `TemplateService`

Referenced By

- `TemplateModule` (MODULE_DECLARES)