Explore Entities and Relationships

September 10, 2026

Explore Entities and Relationships

Use the product model in **Technical Docs** to examine the entities that make up your application and the relationships between them. This helps you validate that technical documentation reflects how your product is actually structured and connected.

Overview

Technical documentation is most useful when it explains more than isolated files or features. The product model helps you explore the important parts of an application—such as components, pages, layouts, services, repositories, stores, hooks, and routes—and understand how they relate to one another.

When you open **Technical Docs**, you can use the entity explorer to find a specific entity and review its connections. For example, you may want to confirm that a component renders another component, that a page uses a hook, that a service calls an API, or that a repository queries a model. Reviewing these relationships is a practical way to check whether generated or imported documentation describes the real technical design.

The model supports several types of entities, including **COMPONENT**, **HOOK**, **PAGE**, **LAYOUT**, **ROUTE**, **CONTEXT**, **PROVIDER**, **STORE**, **CONTROLLER**, **SERVICE**, **REPOSITORY**, and **MIDDLEWARE**. It also organizes findings into categories such as **FRONTEND**, **BACKEND**, **DATABASE**, **API SCHEMA**, **CONFIG**, **DOCUMENTATION**, and **INFRASTRUCTURE**.

> 📷 *Screenshot pending: The Technical Docs screen showing the entity search field and the product model tree.*

Prerequisites

Before you begin, make sure that:

- You have access to the relevant project in Atloria.
- Your project contains technical documentation or product-model information to review.
- You can open the **Technical Docs** area for the project.
- You know the feature, screen, service, or technical concern you want to validate.
- If documentation has not yet been created, you have generated it using **Generate Documentation →** or imported it using **Or import existing documentation**.

Step-by-Step Instructions

1. Open your project's **Technical Docs** area.

   Start from the project where you want to validate documentation. Select **Technical Docs** to work with the project's technical documentation and product model.

   Depending on the state of your project, you may see options such as **Generate Documentation →**, **Or import existing documentation**, or **View Your Documentation →**. Generate or import documentation first if there is no model available to explore.

2. Locate the entity search field.

   In the entity explorer, select the field labeled **Search entities…** or **Search entities...**. The label may appear with either a typographic ellipsis or three dots.

   Enter a name or keyword related to the entity you need to inspect. For example, search for a page name, component name, service, hook, or store. Searching is especially helpful in larger projects where manually expanding categories would take time.

3. Select the entity you want to inspect.

   Choose the matching entity from the available results or from the product model tree. The model may organize entities by technical area, such as **FRONTEND**, **BACKEND**, **DATABASE**, **API SCHEMA**, **CONFIG**, **DOCUMENTATION**, or **INFRASTRUCTURE**.

   Use these categories to narrow your review:

   - Review **FRONTEND** for user-facing components, pages, layouts, hooks, contexts, providers, and stores.
   - Review **BACKEND** for controllers, services, repositories, and middleware.

- Review **DATABASE** when you need to confirm model or data-access relationships.
- Review **API SCHEMA** when validating how APIs are described across the application.

4. Review the entity type.

   Confirm that the selected item is classified correctly. For example, a reusable visual element should normally appear as a **COMPONENT**, while navigation-level content may be a **PAGE** or **LAYOUT**. Application logic may appear as a **HOOK**, **SERVICE**, **CONTROLLER**, or **REPOSITORY**.

   Correct classification matters because it gives readers context. A relationship from a **PAGE** to a **COMPONENT** means something different from a relationship between a **SERVICE** and a **REPOSITORY**.

5. Examine the relationships connected to the entity.

   Review the links shown for the selected entity. The product model can represent relationships such as:

   - **RENDRERS** — a component renders another component.
   - **USES HOOK** — an entity uses a hook.
   - **USES CONTEXT** or **PROVIDES CONTEXT** — an entity consumes or supplies shared context.
   - **USES STORE** or **DISPATCHES ACTION** — an entity interacts with application state.
   - **CALLS API** or **IMPLEMENTS API** — an entity consumes or provides an API.
   - **DEPENDS ON** — an entity relies on another entity.
   - **USES SERVICE** or **INJECTS** — an entity uses a service or dependency.
   - **QUERIES MODEL** — an entity accesses a data model.

   Check that each relationship tells a meaningful story about the selected entity. For example, a frontend component may **USES HOOK** and **CALLS API**, while a controller may **USES SERVICE**, and a repository may **QUERIES MODEL**.

6. Trace the relationship in both directions.

   Do not validate only one connection. Follow the connected entities to check whether the documented flow remains consistent. For instance, if a page is shown as using a component, confirm that the component is connected to the expected hook, context, store, or API behavior.

This is useful for identifying incomplete documentation. A technical explanation may mention an API call, but the model can help you confirm which component, hook, or service initiates that call.

7. Compare the model with your technical documentation.

   Open the relevant technical documentation content and compare its explanations with the entity types and relationships you reviewed. Look for missing dependencies, incorrect ownership, or vague descriptions of how parts work together.

   If you are reviewing generated documentation, use the model as the grounding source for your validation. If you need a different presentation, use the available **StyleSelector** options before generating documentation again. You can also select **Export Documentation** when you need to share the reviewed documentation outside Atloria.

> 📷 *Screenshot pending: A selected entity showing its type and relationship links, such as USES HOOK, CALLS API, or QUERIES MODEL.*

Tips and Best Practices

- **Start with a user-facing entity.** Begin with a **PAGE**, **LAYOUT**, or **COMPONENT** when validating documentation for a feature. This gives you a clear entry point, then lets you trace toward hooks, stores, APIs, services, and data access.
- **Use relationship verbs as validation questions.** Treat each relationship as a question: “Does this component really **RENDRERS** this child component?” “Does this service actually **CALLS API**?” “Does this repository **QUERIES MODEL**?” This makes documentation reviews more specific and repeatable.
- **Check category boundaries.** A complete technical explanation often crosses categories. For example, a **FRONTEND** entity may call an endpoint described in **API SCHEMA**, which is implemented through **BACKEND** services and connected to **DATABASE** entities.
- **Distinguish implementation relationships from general dependencies.** Prefer precise relationships such as **USES HOOK**, **USES SERVICE**, or **QUERIES MODEL** when they are available. Use **DEPENDS ON** for a broader dependency that does not fit a more specific relationship.

- **Review the full chain, not just one link.** A single relationship can look correct while the overall flow is incomplete. For example, a component may call an API, but documentation should also explain the service or controller that implements the corresponding behavior where relevant.
- **Regenerate when the model changes.** If your application structure has changed, generate updated documentation using **Generate Documentation** → before relying on it for a review. A model-based review is most valuable when the documentation represents the current project.

Related Pages

- Generate Technical Documentation
- Import Existing Documentation
- Choose a Documentation Style
- Export Documentation
- Use API Explorer

Troubleshooting

I cannot find the entity I need

Use **Search entities…** and try a shorter or more distinctive part of the name. If no result appears, check whether you are working in the correct project and whether technical documentation has been generated or imported.

The relationships do not match what I expected

Review the connected entities and their types before concluding that the documentation is incorrect. A relationship may be represented through an intermediate entity—for example, a component may use a hook that calls an API rather than calling the API directly. Trace the full chain to validate the documented behavior.

I only see options to generate or import documentation

Your project may not yet have documentation available to explore. Select **Generate Documentation** → to create documentation from the project, or select **Or import existing documentation** to bring in existing material. After documentation is available, return to **Technical Docs** and search for entities.