Mathematical Content Flow in the Turn-Formal Application

Overview

The Turn-Formal application uses a sophisticated pipeline to handle mathematical content:

1. **Math Formalism in Rust**: Mathematics is formally defined in Rust data structures
2. **Export Process**: Rust code converts these structures to JSON representations
3. **Frontend Loading**: The TypeScript frontend loads and renders these JSON files

Backend (Rust) Components

Math Content Definition

- Mathematical concepts are defined in Rust structs in structured theory files
- Organized by mathematical domains (set_theory, group_theory, topology, etc.)
- JSON files for each domain are stored in `/subjects/math/theories/{domain}/`

Export Pipeline

The export system consists of these key components:

1. **`export_math_content.rs`**: Entry point binary that:
    - Runs example generators to create sample files
    - Provides a CLI for manual exports
2. **`turn_render` module**: Handles conversion of Rust formalism to renderable structures:
    - `models.rs`: Defines output data structures like `TheoremRender`, `ProofRender`
    - `exporter.rs`: Implements conversion and file output logic
    - `expression_converter.rs`: Converts math expressions to renderable nodes
    - `cli.rs`: Command-line interface for the export process
3. **MathNode System**: A tree structure that represents mathematical expressions:
    - `MathNode`: Contains an ID and `MathNodeContent`
    - `MathNodeContent`: Enum of various mathematical constructs (operations, relationships, functions)
    - Supports rich representation of complex math expressions

JSON Output

- Each domain has a metadata file with overview information
- Individual theorem files contain full theorem details
- Expression trees are encoded as nested JSON structures

Frontend (TypeScript) Components

Content Loading

- `mathService.ts`: Primary service that manages loading of math content:
  - Uses `refreshTheoryCache()` to load JSON files from specific patterns
  - Tries multiple file paths to locate content
  - Maintains a cache of loaded theories

Data Structures

- TypeScript interfaces mirror the Rust output structures:
  - `MathContent`: Top-level structure for mathematical content
  - `Definition`, `Theorem`, `ProofStep`: Structured math elements
  - `MathNode` and `MathNodeContent`: TypeScript equivalents to Rust structures

Rendering Components

- The frontend uses specialized components to render mathematical expressions
- Type definitions are auto-generated using `ts-rs` to ensure compatibility

Integration Flow

1. Rust code defines formal mathematics
2. Export tools convert to JSON with renderable structures
3. Frontend loads JSON into TypeScript objects
4. Components render the expressions using the `MathNode` tree structures

File Locations

- **Backend Definition**: `/subjects/math/formalism/`
- **Export Code**: `/subjects/math/export/turn_render/`
- **Output JSON**: `/subjects/math/theories/{domain}/`
- **Frontend Loading**: `/frontend/src/services/mathService.ts`
- **Frontend Components**: `/frontend/src/pages/MathPage/components/`