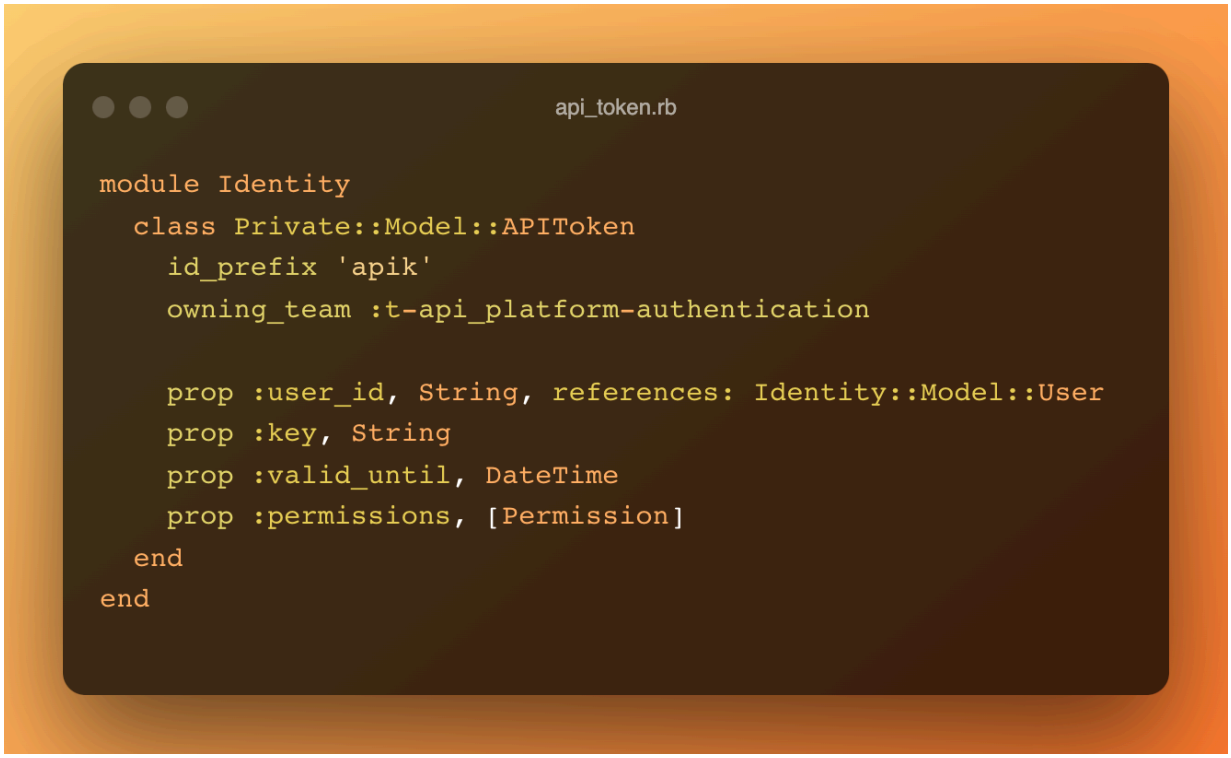
I'm thinking of building a tool to make it easier to browse and understand code in medium-to-large codebases (where it's too big to all fit in one person's head at one time and/or there's multiple people working on a system). At a high-level, the tool would be a code and documentation browser, built around the premise that a codebase is actually a database.

Here's what I mean — consider some code that looks like this:



Conceptually, you can "denormalize" this model into an object with something like the following schema:

api_token.obj

```
{
  type: 'model',
  isPrivate: true,
  name: 'APIToken',
  namespace: ['Identity', 'Private', 'Model'],
  idPrefix: 'apik',
  owningTeam: 't-api_platform-authentication',
  modelFields: [
    {
      name: 'user_id',
      type: 'String',
      references: Identity::Model::User
    },
    {
      name: 'key',
      type: String
    },
    {
      name: 'valid_until',
      type: DateTime
    },
    {
      name: 'permissions',
      type: [Permission]
    }
  ]
}
```

The latter data structure represents a “codebase item”. You could stick it into a database, along with such representations for everything else in a codebase, and do things in the browser with that database.

Here are some ideas for things you could do with this database. For each idea, do you think you:

- Would definitely want to use it if you could (“I need this”)?
- Might play around with it if you could (“This might be cool”)?
- Wouldn’t be interested (“I don’t get it”)?

Using items:

- You can browse a codebase where items are grouped into “models”, “events”, “event consumers”, “crons”, etc. These groups are automatically generated from your codebase, but you can also define your own groups, maybe using something like SQL over the whole database. You can share these queries/groups with your coworkers. The browser might look like this.

- You can customize the data schema for codebase items (adding custom fields to codebase items) by extracting properties from existing code (e.g. parsing out DSL fields or specific syntax patterns, perhaps using regex). You can write complex SQL queries against these properties. You can collaboratively build custom schemas with your coworkers.

- You can add custom properties for codebase items that aren’t actually based on anything in the source code (e.g. `isDeprecated`, `replacedBy`). These properties can also be part of the schema you build collaboratively with your coworkers (it’s like an annotation layer over the code that’s shared with your team).

- You can install an editor plugin that overlays these custom properties as you navigate or write code. For example, you can see if your team considers a class or method to be deprecated, and what its replacement should be if it is.

- You can write complex queries that automatically stay up-to-date with the codebase. For example, you could write something like `SELECT name, scheduledAt FROM crons WHERE owningTeam = 't-your-team'` to get a table of all the crons your team owns and when they’re scheduled. You can get a permalink to this query and even embed it in documentation. Clicking on a row in the table would take you directly to the relevant code (or let you preview it inline).

Codebase insights:

- You can generate charts by writing queries over codebase items — for example, to track the progress of a migration or adoption/removal of some code.
- You can generate charts that join codebase data to runtime data — for example, if you're an infrastructure team, you can see which teams are generating the most errors for you based on ownership or authorship of erroring code.
- You can draw diagrams (e.g. GraphViz, Drawbot, Mermaid) from codebase items — think whiteboard diagrams, but automatically stays in sync with the code. Diagrams bidirectionally link to the code: you can click into the diagram to go to the actual underlying code, and when you're navigating the code itself, you can see the diagrams in which the code shows up.
- Code items can be joined to your company's org chart. If you're looking at some code, you can browse a directory of the owning team and see contact info for them. If you're looking at a team directory, you can see a list of the specific code they own or have changed a bunch. This data automatically stays up-to-date across re-orgs.
- You can assign code ownership, and update them to hand off ownership as people move around. If you're a manager implementing a re-org, you can see everything your team owns and make sure all the ownership is covered in the new team structure.

Documentation:

- You can "attach" notes to code items (functions, classes, events, etc). They're visible in the codebase browser, and also while you navigate around the code (e.g. if you have a note for a particular function, it's visible everywhere that function is called). Notes can be private to you or shared with coworkers.
- You can attach Q&A (long-form like StackOverflow) or discussion threads (like Slack) to code items. From your code editor, you can highlight some code and send a question about it to a coworker. You can also browse Q&A and discussion threads directly in the browser.
- In those notes, Q&A, and discussion threads, there's a notebook-style editor that lets you easily reference code items (e.g. using `[[item name]]` syntax), embed complex query results, or even runnable code blocks. Code references automatically update if the underlying code is moved or renamed. You can collaboratively edit this documentation, like Google Docs.

Avoiding boilerplate:

- Codebase items automatically get turned into a RESTful, read-only API, which you can use to build additional tools (e.g. you could build an admin page of all the crons your team owns and filter them based on a priority annotation).
- Codebase items automatically get turned into a RESTful API with write support, which you can use to build additional tools (e.g. you could build an admin UI to update data model descriptions or change the scheduled time of a cron), and changes can get automatically committed/PR'd to the source code.
- You can define custom, wizard-style code generators (e.g. with multiple steps; with possible options listed in a GUI), and other developers can use those generators (either via the browser GUI or a CLI) to generate boilerplate code.