Tometo

May 10, 2020

Contents

This is the central documentation for the Tometo project. You can use the sidebar to navigate this page!

We provide documentation for a couple of different things:

- **Want to learn how to install Tometo locally** to work on it or to just try it out for yourself? Our *Installation Guide* can help!
- **Interested in contributing to Tometo?** We have a couple of guides related to that *here!*
- **Questions about Phabricator, our internal development platform?** We have some extensive *guides available!*
- We also provide some *technical details about our code structure*.

CHAPTER 1

Installation

Installation can be done either in a Vagrant virtual machine or locally, directly on your machine. The upside to Vagrant is that it uses an automated provisioning script, which means no having to install dependencies or even setting up the project. You run two commands, and then you have a fully functioning setup of Tometo.

The downside is that it *is* a virtual machine – it's not truly on your computer, but running inside an emulated Ubuntu VM. Doing that can stress your RAM and CPU, so if you don't have a computer with modern-day decent specs, you might be better off directly installing Tometo. I would still recommend you to give Vagrant a try though, and if it runs super sluggish, you can still switch to running it directly on your computer.

1.1 Via Vagrant

First, you want to download Vagrant. On most systems, Vagrant uses VirtualBox in the background, so you'll want to install that too, unless you already have another hypervisor installed (don't worry if you don't know what that is - neither do I).

Next, download the source code. If you have Git installed, you can clone the repository like this:

```
git clone https://git.tometo.org/source/tometo.git
```

If you don't want to install Git or have no idea what that is, our GitHub mirror provides a ZIP download here.

Once you have it downloaded and extracted to somewhere you like, open a Terminal (cmd.exe or PowerShell on Windows) in the folder you extracted the code to and run this:

```
vagrant up
```

This **will** take a while, so maybe make some tea in the meantime. Once it's done, you can run this:

```
vagrant ssh
```

You're now in a full Ubuntu Linux virtual machine that just so happens to have Tometo installed into it! The cool thing about this is that all of the changes you make in the directory you downloaded Tometo into will automatically appear on that virtual machine. You can now use the Tometo command-line scripts to start the server, like this:

```
script/run
```

Or, if you want to skip the `vagrant ssh` step, you can directly start it from your computer:

```
script/vagrant-run
```

See also the “Running” section at the bottom of this page.

1.2 Locally

1.2.1 Prerequisites

In order to get the system running on your computer, you’ll need some prerequisites:

- Git
- A PostgreSQL server, and its development headers (sometimes called `libpq-dev` or `postgresql-devel`)
- Elixir (and Erlang, but usually those two are installed side-by-side)
- Node.js, the latest LTS or Stable version should work
- Python 3 and `pip` (plus development headers, sometimes separate as `python3-devel`)
- eSpeak (and its development headers, sometimes separate as `espeak-devel`)
- FFmpeg

Optionally, there’s some more advanced features you can enable, which need some of these dependencies:

- A Google Cloud service account API key that has access to the Text-To-Speech API

Note: These still need to be documented properly.

1.2.2 Automatic Installation

Before doing any installation, make sure you have Elixir’s package manager scripts downloaded locally! Run this command to do so:

```
mix local.hex
```

If you have all of the prerequisites installed, you can try cloning the repository and running our automatic setup script:

```
git clone ssh://vcs@git.tometo.org:2222/source/tometo.git
# or with HTTPS
git clone https://git.tometo.org/source/tometo.git
cd tometo/
script/localsetup
```

This will set up everything for you, the only thing you need to do yourself is fill in the config file for Aph and run the database setup. The config file can be found in `config/dev.exs`. This is where you fill in your database credentials. After you’re done doing that, you can then run the database setup:

```
mix ecto.create
```

1.2.3 Manual Installation

You'll want to install `aeneas`, which parses text for us (this needs ffmpeg and espeak installed and available):

```
pip3 install --user numpy
pip3 install --user aeneas
```

Check that it's installed correctly:

```
python3 -m aeneas.diagnostics
```

Then, you can clone the repository.

```
git clone https://git.tometo.org/source/tometo.git
```

Once you're in the directoy, you'll want to install the dependencies for the frontend:

```
npm install
```

And the backend:

```
mix deps.get
```

Now you can go ahead and copy the backend configuration file:

```
cp config/dev.example.exs config/dev.exs
```

Next, to create the necessary database tables and configuration, fill in your database configuration in `config/dev.exs` and run this:

```
mix ecto.create
```

1.2.4 Configuration

Configuration is separate for the frontend and the backend, but most likely you won't need to modify the frontend's config at all. The backend config can be found within `config/dev.exs`. For the frontend, you can set the environment variables described in the webpack.config.js file. The easiest way of doing this is by creating a `.env` file in the project root. However, the defaults should work for most development purposes.

Note: TODO: Add production configuration info

1.2.5 Running

We have multiple scripts to provide some common uses if you're planning on working on Tometo. These include:

- `script/build`: Runs a production build
- `script/lint`: Makes sure your code looks nice and is ready to commit
- `script/fix`: Automatically corrects your code based on our linting rules (this modifies your actual code files)
- `script/run`: Runs both the frontend and the backend and watched for changes. This is what you want most of the time.

- `script/run_b`: Runs only the backend
- `script/run_f`: Runs only the frontend

CHAPTER 2

Contributing to Tometo

Thanks for your interest in contributing to Tometo! There is more than one way to contribute to the project, and we appreciate all of them.

If you have any questions, feel free to ask them in the Tometo Discord Server. As a reminder, all contributors, in whichever way, are required to follow the Code of Conduct.

2.1 Feature Requests

If you have ideas or even concrete suggestions for a feature or an enhancement, it's best to discuss it first on Discord before diving into the specifics. If you feel up to implement it, first *create a task for it* and then make a *contribution*!

2.2 Bug Reports

While we wish everything worked perfectly, bugs are natural to occur. This is why we recommend that, even when you're not sure it's a bug, to report it anyways.

Please see *How to report a bug* for more info about bug reports.

2.3 Code contributions

We would welcome your contribution! If you're interested, please first find something to work on, and read the following documents:

- *Phabricator*
- *Code Review*

Feel free to ask in the #development channel on Discord if anything is unclear.

CHAPTER 3

How to report a bug

These guidelines explain how to write a good bug report or feature request (a task) in Tometo's task tracker (see *Phabricator* for more information). Well-written tasks are more likely to be worked on quickly.

3.1 Quick recommendations

- Be clear
- Explain how to reproduce the problem, step by step, so others can reproduce the bug, or understand the request.
- Include only one problem per task
- Include any relevant links and examples

3.2 Before you do anything

3.2.1 Can you reproduce the issue?

Try to reproduce your bug using a recent version of Tometo (most likely a `master` checkout), to see whether it has already been fixed.

3.2.2 Has someone else already reported the issue?

Use the search box of Tometo's bugtracker to see if your bug has already been reported, or the feature requested. You can also perform more advanced searches on the advanced search page.

If you are unsure whether a bug has already been reported, you should report the bug. It is better to have duplicate bugs than it is to have unreported bugs.

3.2.3 Reporting a new bug or feature request

If you have faced a bug in a recent version and no one else appears to have reported it, then:

1. Go to https://git.tometo.org
2. You will have to log in (or register) if you haven't already done so (see *Creating an account*)
3. Click the bookmark button in the upper right corner and choose "Create Tometo task"
4. Fill out at least the following fields:

- **Title**: A short one-sentence summary that explains the problem (not your suggested solution).
  - Good: `Selecting avatar is not functional`
  - Bad: `Software crashes`
- **Description**: Full details of the issue, giving as much detail as possible. This should include:
  - For bugs:
    - Steps to reproduce: Minimized, easy-to-follow steps that will trigger the described problem. Include any special setup steps.
    - Actual Results: What the application did after performing the above steps.
    - Expected Results: What the application should have done, if there was no bug.
  - For feature requests:
    - A description of what you would like to achieve, and why. Explain what you hope the feature will solve (the actual underlying problem) along with specific examples; but do not demand a specific solution, as there might be other/better solutions. A user story might be an effective way of conveying this.
  - Please also provide any other information that may be useful, such as:
    - The web browser, computer systems or hardware you've seen the bug on
    - Links or diffs to one or more pages where you encountered the bug
    - Whether the problem appears every time, only occasionally, only on certain pages, or only under specific circumstances
  - To attach a log file or a screenshot, click the **Upload File** button (a cloud with an arrow) in the toolbar of the Description field.
  - Select the **projects** that are relevant for this bug:
    - The default projects are *triage* (this is important to leave) and *tometo* (you should leave this unless the bug has nothing to do with the core Tometo software).
    - If any other teams might be interested in this, or if the bug appeared in a different component, please attach them to the task. Otherwise, a team member will do it for you once you've created the task.
  - **Subscribers**: If you know any people who will be interested in getting notified of this task, you can add them here. Otherwise, ignore this field.

Check if your report is complete, then press the Create Task button.

The priority for the task will be set by the Triage team, and the people who will be planning to work on this task.

That's all! Thank you so much for helping to improve Tometo!

CHAPTER 4

Releasing Tometo

Releasing software can be an arduous task, which is why we're trying really hard to make it as simple as possible for Tometo. Here's a bunch of information and guides you might appreciate if you're a person who is in charge of releases.

4.1 Basics

Right now, a Tometo release is basically just:

- A tag in the source repository (something like `0.4.2`)
- A corresponding post on the Tometo Release Blog
- For larger releases (such as *0.x* or *x.0*), a post on the main Tometo website at https://tometo.org

4.1.1 Preparing a Release

If you have commit access, you can make one commit that includes the version number changes. That's basically all that's required code-wise. Change the version numbers in these places:

- `package.json`
- `mix.exs`

Call the commit the version number, and create a tag that refers to that commit:

```
git tag x.y.z
```

Then push them:

```
git push origin master
git push origin master --tags
```

4.1.2 Writing a release post

Every release, even minor ones, should have a post on the Release Blog. Assuming you have correct permissions to create blog posts there, the minimum a blog post should contain is a changelog that lists all features, bug fixes and enhancements, ESPECIALLY the ones that were made by people outside of the usual committers. If there is very menial stuff authored by usual committers, such as changing some links in documentation or something, feel free to leave those out, but don't make assumptions.

That being said, it's your blog post! You can put as many fun little things into it as you'd like. Once you're done, let the team know and we'll push the release out onto our social media channels and the Discord. Congrats!!

CHAPTER 5

Phabricator

5.1 Phabricator Help

Phabricator is the software we use for development. It can be found at https://git.tometo.org. This page explains a couple of common things you might want to do in Phabricator.

5.1.1 Creating an account

You can either create an account directly or log in with your GitHub account.

Creating an account directly

To create an account via Phabricator itself, visit this page. You should then get an email with which you can confirm your account.

Creating an account via GitHub

You can also log in via your GitHub account by clicking the GitHub logo on this page.

Troubleshooting

- If you're not getting the verification email, please ask on Discord!
- Similarly, if you created an account with the wrong email address, we can't correct the address, but we can delete the account so you can try again.

5.1.2 Receiving updates and notifications

Phabricator notifies you about relevant activity, including your own actions.

Phabricator offers several tools to receive the notifications you wish to receive.

- If you are interested in a single object (a task, a revision…) just click `Subscribe` in its page. Adding a comment will subscribe you automatically.
- If you are interested in all the activity within a project, you can click Watch Project on the project summary page.

Troubleshooting

If you receive unexpected mail notifications for a task:

- You might be subscribed to the task.
- You might be a member of a project or of a subproject associated to that task. A list of all projects that you are a member of is available.
- You might watch a project associated to that task. A list of all projects that you watch is available.
- In your email preferences under "Maniphest Tasks", you might have enabled "One of a task's subtasks changes status".
- You might have a personal Herald rule set up. Check the "X-Herald-Rules" message header field to see a list of all applied Herald rules.

5.1.3 Creating a Task

There are several ways to create a task, depending on the information you want to carry:

- A new task: click the Bookmark button toward the right side of the top navigation bar. From the dropdown menu, choose Create Tometo Task. You will get a blank form.
- A security problem: click the Bookmark button toward the right side of the top navigation bar. From the dropdown menu, choose Report Tometo Security Issue. You will get a form pre-tagged with Security, and with a link to special instructions for filing security bugs.
- A subtask of an existing task: click **Edit Related Tasks… > Create Subtask** located in the right column of the current task. The dependency between both tasks will be set, and some values of the parent task will be carried by default (Assigned To, Subscribers, Priority, Projects). Subtasks will be listed in the parent task under "Task Graph", sorted by most recently updated.

Fill the form, leaving the fields you are not sure about unchanged. Use the live preview at the end of the page to check whether your text looks as you expect.

5.1.4 Commenting and editing a task

To reply, you need an account as well.

Phabricator allows you to post and edit comments and descriptions using text formatting and inserting images or other files. To insert a file, just drag and drop, paste or select it from the File Upload button. You can use toolbar at the top of the input text area and you can use Phabricator's own formatting.

To edit the description of a task, select "Edit Task" in the side bar.

To close a task as a duplicate of another task, select "Edit Related Tasks… > Close As Duplicate" in the side bar.

5.1.5 Project Management in Phabricator

More detailed information about this can be found in *Phabricator/Projects*.

Parent tasks and subtasks

Tasks can be a parent task or a subtask of any number of other tasks. If Task A is not solved until Task B is solved, then Task A is the parent task, and Task B is the subtask. Such relations can be set via "Edit Related Tasks…". Parent tasks and subtasks are displayed under "Task Graph" in the task. This feature can be used to accomplish a few different things:

- Blockers and Subtasks. A (parent) task might simply be blocked by another (sub)task, representing a dependency.
- Tracking. A "workless" (parent) task blocked by several (sub)tasks might be tracking a collection of (subtasks within a release or other time period.)

5.1.6 More advanced features

Restricting access to tasks

Access to a specific task can be changed via "Edit Task" and then changing the "Visible To" field to something else than "Public". This option is only available to some users.

Note that tasks filed as Security issues are not publicly visible.

5.2 Project Management

This guide focuses on productively managing a Tometo project. A project here is defined as a unit of what makes up Tometo. There's the base `tometo` project, but a project can also be a milestone, a component, a team, and basically anything else that can be separated by hierarchy.

5.2.1 Tasks

Tasks are the basic building block of Phabricator. A Phabricator task is something that can define things like bugs, user stories, feature requests, etc. While a task can represent different kinds of things, all tasks have certain properties, like title, author (creator), assignee, description, projects/tags, etc. Unlike many other project management and issue tracking tools, a task can have a one-to-many relationship with 'projects' - that is, a single task can be associated with more than one 'project'.

Scope of Tasks

Tasks need to describe an achievable objective. Ideally, tasks are defined with a scope that can be resolved by one person with a decent amount of effort. Huge tasks that take several people and several days will be more manageable when you identify the subtasks required to complete them. Trivial subtasks that one person can complete in a moment but should be documented can be just added as a checklist in a description of the main task. A good principle to follow is that a task should be completeable in ~3 days maximum. If a task is larger than this, it should be broken down into smaller tasks. Also consider this piece of advice: does your task require more than a couple of hours of work? Then you may want to add it to Phabricator. Would there be nasty consequences if you forgot executing that task? Avoid this by adding it to Phabricator.

Team-level goals are often called “Epics”. Some projects have an “Epic” column in their workboard, which is used for tracking larger goals that require months worth of work.

If a task fails defining an actionable objective (e.g. never-ending tasks, support questions, generic complaints…) they might be resolved as Invalid. Users resolving a task as Invalid should explain their reasons in a comment (also see the Bug report life cycle). Tasks are fully editable, and if the causes for the Invalid resolution have been addressed, they can be reopened.

Assigning Tasks

Each Task may be assigned to one person. As with Priority, this is inherent to the Task, and affects every Project that Task is in. So it is impossible to have a Task assigned to Lucy in one Project, but to Kim in another. The assigned person is displayed as part of the Task card in each Workboard.

In principle, a task gets assigned to whomever will take ownership of it. Some teams might choose to assign Tasks to people while the Tasks are in a TODO column. Others would have people assign Tasks to themselves only at the moment that they are moving them from TODO to DOING. Try to avoid assigning a task to yourself before you start working on it.

Setting Task priorities

Each Task has a Priority field, which is reflected in the sidebar color of Tasks that appear in Workboards. Note that this Priority is inherent to the Task, and thus will be the same in every Project and Workboard that task appears in.

One task can have only one set priority at a time. Priority should normally be set by the Triage team, the Core team, or committers who plan to work on the task, or by the Bugwrangler or experienced community members, not by the reporter filing the bug report or by outside observers. When in doubt, do not change the Priority field value, but add a comment suggesting the change and convincing reasons for it.

Within a Workboard, Tasks can be grouped by priority within a column. Choose “Natural” to have no grouping. Drag and drop Tasks up or down within a column. This allows a groomed backlog to be sequenced by priority, or could indicate the urgency of items in a “Needs Review” column. Note that Workboard columns can be grouped by several criteria, so while discussing a Workboard with someone not in the room, it is best to agree on and use the same sort order to avoid confusion.

We offer these priority levels on Phabricator:

- Needs Triage - Default option, this indicates that the Triage team or someone else with enough experience has to prioritize this task.
- Unbreak Now! - Something is broken or needs to be fixed immediately, setting anything else aside.
- High - Someone is working or planning to work on this task soon.
- Medium - Less priority than High, but someone is still planning on working on it.
- Low - Less priority than Medium, but someone is still working on it. This doesn’t necessarily mean the task is not important, it just means that nobody has this on their to-do list right now.
- Lowest - Nobody plans to work on this task, but we would be happy if somebody does.

Tasks that cannot be worked on yet

Certain tasks might not be actionable until an action has been performed outside of the task itself.

If a task depends on another task in Phabricator, you should set the number of that other task under “Edit Related Tasks… > Edit Subtasks”. Afterwards, the other (sub)task will be displayed under “Task Graph”.

Closing a Task

A task can be closed as Resolved, Declined, or Invalid. It can also be merged as a duplicate of another task (details). When a task is associated with multiple projects, care should be taken to coordinate the closing of a task with all involved parties. Anyone marking a task as declined or invalid should include a comment explaining why.

5.2.2 Projects

Projects are the basic organizational method in Phabricator; they are a way to organize tasks and manage workflow or to "tag" for queries and general organization.

Types of Projects

There are several types of top-level projects in our Phabricator. Each type has to follow the purpose, color and icon defined in these guidelines.

- Types you can request to be created directly:
  - **Component** corresponds to a distinct and recognizable piece of software, service, event or any set of periodically occuring tasks of the same type (i.e. access or requests). If you're unsure, go for this. *Icon: Briefcase, Color: Blue*
    - **Umbrella** could be used for larger Projects that don't have a distinct codebase and instead consist of several smaller projects. *Icon: Umbrella, Color: Blue*
  - **Group** corresponds to an existing team. *Icon: Group, Color: Violet*
  - **Goal** can be used for goals that will be met at some point in time, such as feature implementations. *Icon: Goal, Color: Checkered*
  - **User** allow you to track progress of personal tasks. *Icon: User, Color: Orange*
- Types that should be discussed before being created:
  - **Tag** is used as a cross-component keyword (like "translation"), a "never-ending" project. *Icon: Tag, Color: Yellow*
  - **ACL** are projects that regulate access. This should be used instead of locking down resources to projects like Groups, or locking projects themselved. *Icon: Policy, Color: Red*

Archiving a Project

If/when your Project is complete, abandoned, or otherwise no longer active, it should be archived. This prevents clutter and signals to others that the Project is inactive.

Assuming you have appropriate permissions, to archive a Project:

1. go to your Project page;
2. click the **Manage** item in the navigation bar on the left;
3. click **Archive Project**.

Make sure to handle the open tasks of your archived Project: Either associate the tasks with at least one other active Project, or close the tasks as declined in combination with an explanatory comment.

5.2.3 Workboards

The workboard is the primary user interface for viewing and manipulating tasks that belong to a project. Projects which are used solely as supplemental “flags”, for example i18n, may not use their boards. Boards are useful to follow the development status of tasks within a ‘project’. Keep in mind that boards are not just a means of managing the flow of work. It’s a communication tool both for your team/project/etc as well as to the public/etc. Boards should be used and maintained with this in mind; the simpler and clearer the language used, as well as the clarity of organization of your board, the better for communicating to a broader audience.

Every ‘project’ has one single board. You can access it by clicking the Workboard icon in the left-side menu of the project. If there is no such icon, the board is disabled. You can enable it via “Manage” in the left-side menu and then choosing “Edit Menu” on the right.

Columns

Project boards may display the tasks in the project in multiple columns. A project that is itself used as a tag (i18n, Need-volunteer) may not use its board and thus will not have columns.

When you first set up the board for a project, you can choose “New Empty Board” which starts with a single “Backlog” column; or you can choose “Import board columns from another project” to use another board’s set of columns as a starting point. After that you can create arbitrary columns in the board such as In development, Blocked with questions, Code review, and Done. Then much like other ‘project’ management tools you can drag and drop tasks within and between columns to recategorize them. The order of tasks within each column is maintained, and you can drag and drop tasks up and down, so it is common to treat this as another kind of prioritization or stack ranking. A task can belong to only one column within each board.

Typical uses of workboards

- Workflow - Typical columns track the state of tasks, such as Todo, Doing and Done - This supports both Scrum-type work and Kanban-style work - Tasks are moved from column to column regularly
- Categories - Columns might group Tasks by component, functional area or even complexity

Examples

- Backlog, Doing, Review, Blocked, (Done)
- To Do, In Progress, In Review, (Done)
- Backlog, Needs Plan, Needs Code, Non-Code, In Development, Needs Review

“Done”

Each Task has a Status field, which includes the state of Resolved. Resolved tasks are by default hidden from Workboards. When they are displayed, they appear grayed out and struck out. Other statuses include Open and Stalled.

It is important not to mark a task Resolved until it is considered done by ALL of the projects it is in. It’s fine to move a Task to your own Workboard’s DONE column, but before you mark a task Resolved, ensure that no other project still wants to keep tracking its progress.

5.3 Code Review

Differential is the code review tool in Phabricator. It's used to land most significant code-changes into Tometo codebases. Along with Differential, *Arcanist* is used to push your local changes to Differential, opening a new *revision* (basically a pull request) as a result.

5.3.1 Who can create revisions?

Anyone can create revisions. You may not have push access to the project you're contributing a revision to, but Arcanist can also operate off of local commits (and in fact, this is what you should do if you're a first-time contributor!). Tometo Committers can push their work to a feature branch, creating revisions off them works the same way.

5.3.2 Install Arcanist

While you don't *need* to install Arcanist, the fact is that it makes repeat contributions much easier once you wrap your head around it. Arcanist has a great quick-start guide that should help you get it set up on your PC. You can also check your distribution's package manager and install it from there (saves you the trouble of installing PHP by yourself).

Don't forget to run `arc install-certificate`!

5.3.3 The Code Review Workflow

Usually, our workflow is somewhat like this:

For new contributors

- You locally set up Tometo
- You make your changes, and commit the result (either on `master` or on your own branch, it doesn't matter)
- You run `arc diff`, which creates a revision with your local changes
- Someone reviews your revision, if you need to make changes you run `arc diff` again after making them to update the revision
- Once it's accepted, the reviewer commandeers (takes control of) the revision and lands it into the source repository
- You're done!

For committers

- You make changes and push them to a feature branch
- You run `arc diff` to create a revision
- After the revision has been reviewed and accepted, you run `arc land` locally
- You're done!

For reviewers

- Someone creates a revision
- To locally test it, run `arc patch <revision number>` to make Arcanist locally download the revision diff
- You review accordingly
- After accepting the revision, optionally land it if the author doesn't have push access

5.3.4 Troubleshooting

Arcanist can be a delicate beast, and you should feel free to ask in the #development channel on Discord if you're unsure about something or if something doesn't work. In the worst case, just recreate your revision.

CHAPTER 6

API Details

Tometo's API, called Aph, takes care of everything in the backend, such as interfacing with the database, generating statuses, and so forth. We have a couple of rules when working with Aph that we try to follow in order to make a consistent effort at keeping the codebase clean and understandable.

6.1 API Interface

We follow the RESTful API standards. This means a couple of things:

- Use proper HTTP statuses (i.e. `201` on a successful `POST`)
- Routes should be: - Plural unless referring to a singleton object (`/api/statuses`, `/api/users`)
  - Usable by multiple methods (`GET /api/users`, `POST /api/users`)
  - Used for specific cases only when it's unavoidable (`/api/users/:id/invitations`)
- When returning an error, put it in this format:

```
{
  "error": true,
  "id": "unique-error-id",
  "message": "Helpful error message"
}
```

- Always return full resource objects, not only subsections of them. For example, when creating a user, don't only return the username, return the entire thing, even if it might seem superfluous.

6.2 General Elixir/Phoenix recommendations

6.2.1 Phoenix Views

Views can either use preloaded data, render different views, render many items, or not render much at all. This can get confusing quickly, which is why we have some standard naming practices for the `render` function in our views:

Say we've got an example entity called `Book` that has a 1-to-many relation with both `Page` and `Word`. The `book_view.ex` could include the following:

- `book.json` for a MINIMAL version of the entity with no preloads, but all keys (including timestamps).
- `book_with_page.json` for `book.json` but with a `Page` preloaded.
- `book_full.json` for `book.json` with every foreign resource preloaded.

We don't use a `data` key in views, like the Phoenix generators would like you to believe. Most things in our JSON returns are top-level.

6.2.2 Other Things

`AphWeb.ErrorView` contains the most common error codes. Some of them accept custom messages, some don't — feel free to modify one to include a message, just be sure to check where else the error is being used from first!

When you want to make sure that only logged in users can access a route, you can put your route in one of the authorization scopes in `lib/aph_web/router.ex`. The scopes that exist are as follows:

- `:redirect_if_user_is_authenticated`: Makes sure the user is **not** logged in
- `:required_authenticated_user`: Makes sure the user is logged in
- `:require_admin_or_mod_user`: Makes sure the user is logged in and at least a mod

The other API pipeline is generic, and can be used without regard to any authorization.

Always use brackets around method calls, it's cleaner and easier to understand. Piping is usually not worth it when it's only a single pipe (unless you're working with a `conn` struct).

Linting can be done via `script/lint`, or, if you want to directly fix potential formatting mishaps, `script/fix`.

CHAPTER 7

Code of Conduct

7.1 Our Pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

7.2 Our Standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.3 Our Responsibilities

The Tometo team is responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

The Tometo team has the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

7.4 Scope

This Code of Conduct applies within all project spaces, and it also applies when an individual is representing the project or its community in public spaces. This applies to the project GitLab instance, the Discord server, but also the official Twitter account. Representation of a project may be further defined and clarified by project maintainers.

7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project lead at mokou@posteo.de. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The Tometo team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

7.6 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

For answers to common questions about this code of conduct, see https://www.contributor-covenant.org/faq