Case study: Mastodon

Basic description of the Mastodon:

Mastodon is a **free, open-source social network server** based on ActivityPub. Follow friends and discover new ones. Publish anything you want: links, pictures, text, video. All servers of Mastodon are **interoperable as a federated network**, i.e. users on one server can seamlessly communicate with users from another one. This includes non-Mastodon software that also implements ActivityPub!

Mastodon is a decentralized solution to commercial platforms; it avoids the risks of a single company monopolizing your communication. Anyone can run Mastodon and participate in the social network seamlessly.

An alternative implementation of the GNU social project. Based on ActivityStreams, Webfinger, PubsubHubbub and Salmon.

1 Technology and Platform used for development

1) Programming language selection:

Mastodon is a Ruby on Rails application with a React.js front-end. It follows standard practices of those frameworks.

Ruby on Rails:

The project focus is a clean REST API and a good user interface. Ruby on Rails is used for the back-end, while React.js and Redux are used for the dynamic front-end. A static front-end for public resources (profiles and statuses) is also provided.

I am not familiar with the Rails, so I google it and found a description of Rails. According to the description, Rails is a web-application framework that includes everything needed to create database-backed web applications according to the Model-View-Controller (MVC) pattern.

Here are are some Pros by using Ruby on Rails:

- Ruby is a more flexible language (this can also be a con - explored later!)
- Faster to spin up new projects

- Less code to make things happen
- Web development popularity edge. Large ruby on rails community, almost entirely web development focused
- Migrations are easier
- Easy to deploy

JavaScript for react.js and node.js:

JavaScript is basically used with react.js to build the front-end. There are tons of discussion about how to choose between React, Vue and angular, here I just list some of the Pros for react:

- Declarative design: React uses a declaration paradigm that makes it easy to describe applications.
- Efficient: React minimizes interaction with the DOM by simulating the DOM.
- Flexibility: React works well with known libraries or frameworks.

By using node.js, it powers the streaming API.

2) Library Used

Ruby

- haml, a templating language
- devise, for authentication
- doorkeeper, for acting as an OAuth 2 provider
- paperclip, for file uploads and attachments
- sidekiq, for background processing

JavaScript

- immutable, for immutable data structures
- react, for rendering the dynamic web application

- react-redux, for managing React state
- react-router-dom, for navigation within React
- react-intl, for localizations within React

3) Features

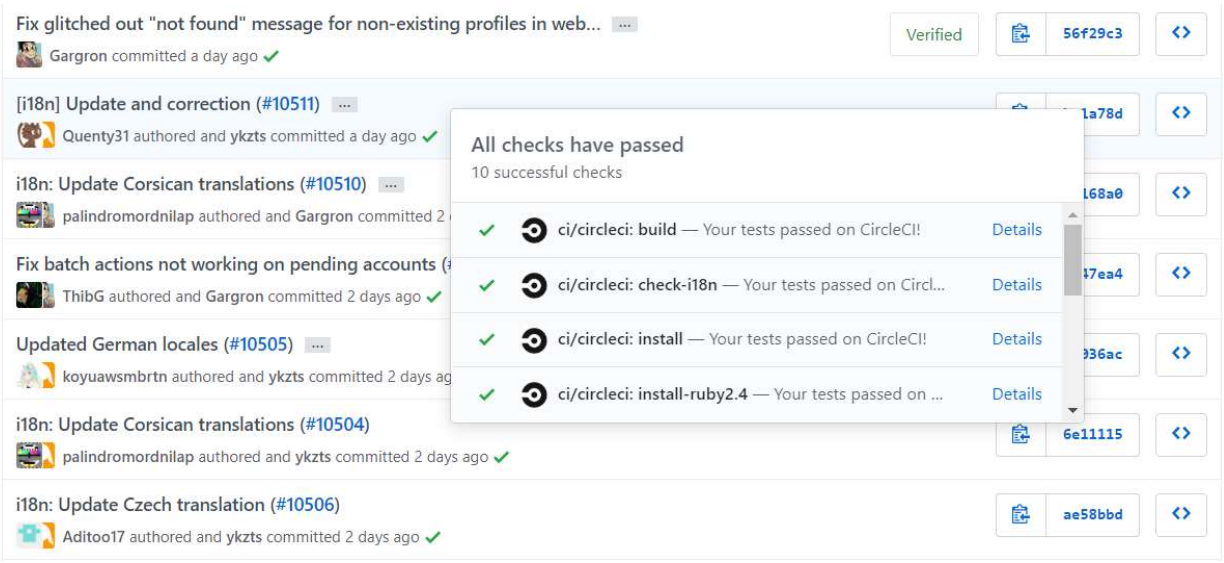
- Fully interoperable with GNU social and any OStatus platform Whatever implements Atom feeds, ActivityStreams, Salmon, PubSubHubbub and Webfinger is part of the network
- Real-time timeline updates See the updates of people you're following appear in real-time in the UI via WebSockets
- Federated thread resolving If someone you follow replies to a user unknown to the server, the server fetches the full thread so you can view it without leaving the UI
- Media attachments like images and WebM Upload and view images and WebM videos attached to the updates
- OAuth2 and a straightforward REST API Mastodon acts as an OAuth2 provider so 3rd party apps can use the API, which is RESTful and simple
- Background processing for long-running tasks Mastodon tries to be as fast and responsive as possible, so all long-running tasks that can be delegated to background processing, are
- Deployable via Docker You don't need to mess with dependencies and configuration if you want to try Mastodon, if you have Docker and Docker Compose the deployment is extremely easy

2 Testing

According to GitHub, Mastodon uses CircleCI as their test framework. CircleCI is Continuous Integration, a development practice which is being used by software teams allowing them to to build, test and deploy applications easier and quicker on multiple platforms.

The test process is divided into parts: Build -> check-i18n -> install -> install-ruby -> test-ruby -> test webui.

According to the official tutorial, the installation should be set on Ubuntu 18.04.



3 Software architecture

1) How to edit

The configuration of Mastodon can be extract as the following diagram. If you want to add / edit functionality, just go to the target repo and make some alteration.

Ruby:

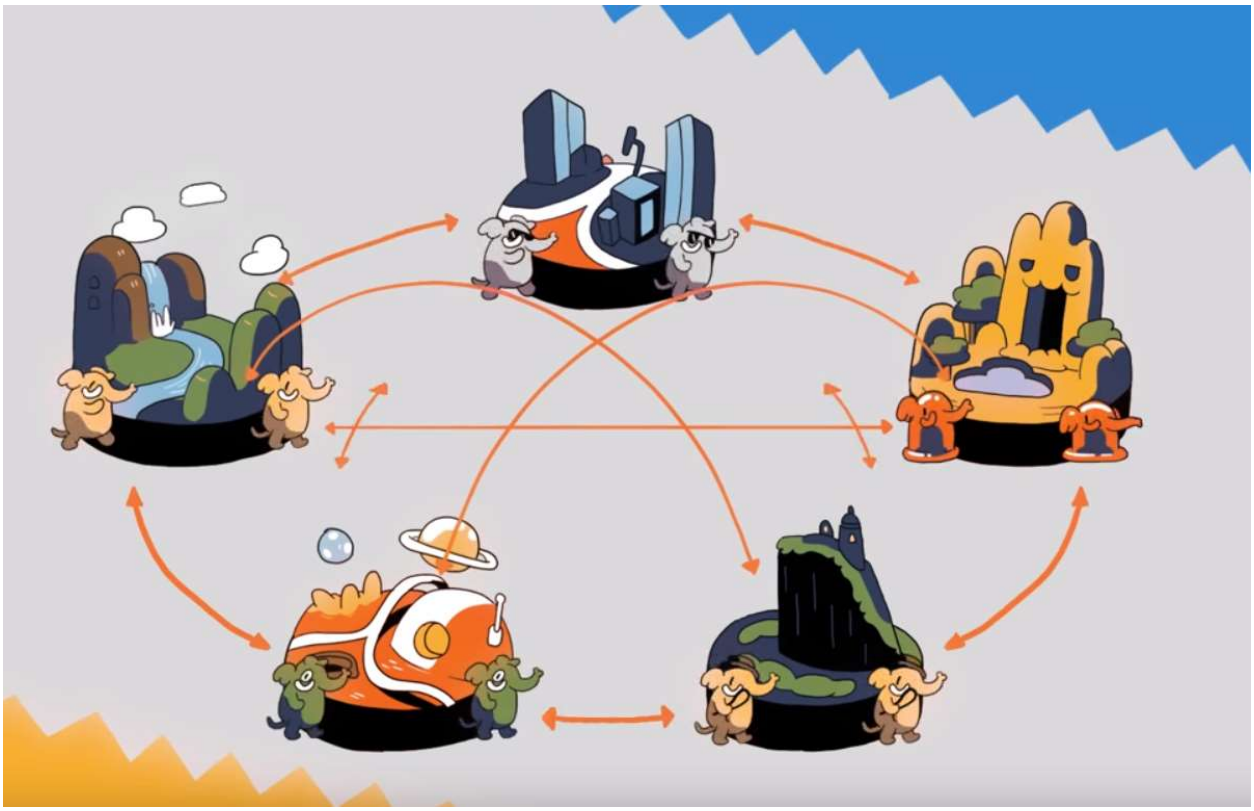
Path	Description
`app/controllers`	Code that binds business logic to templates
`app/helpers`	Code that can be used from views, i.e. common operations
`app/lib`	Code that doesn't fit in the other categories
`app/models`	Code that represents data entities
`app/serializers`	Code that generates JSON from models
`app/services`	Complex logical operations involving multiple models
`app/views`	Templates for generating HTML or other output
`app/workers`	Code that executes outside the request-response cycle
`spec`	Automated test suite

JS:

Path	Description
`app/javascript/mastodon`	Code for the multi-column React.js application
`app/javascript/packs`	Code for non-React.js pages

For example, system service files can read environment variables from an `EnvironmentFile` or from inline definitions with `Environment`, so you can have different configuration parameters for specific services. They can also be specified when calling Mastodon from the command line.

2) System Diagram



I found a really interesting picture that could basically interpret the working mechanism of Mastodon.

The main function is to expose your status on the web, just like most of the social network especially Tweeter. However, its main feature is "distributed and decentralized". Not all users are concentrated on one server. Anyone can have their own servers, which are connected to each other and eventually form a huge network. Visually speaking, the entire Mastodon network consists of thousands of instances, which are connected to each other and eventually form a huge network, like a huge universe. Each server is a galaxy in the universe, and the user is the planet in each galaxy. Obviously, the most straight forward advantage of the decentralized network is its safety. Unlike Tweeter, the social media is not controlled by a single corporation who has full control over everything.

3) System Architecture(for front-end)

For the frontend development, the developer simply use the MVC pattern. The Model-View-Controller (MVC) is an architectural pattern that separates an application into three main logical components: the model, the view, and the controller. Each of these components are built to handle specific development aspects of an application.