decidim docs

AI tools

In order to help the moderator communities to manage better their Decidim installations, we have shipped the first version of the AI tools. This is a set of tools that will help the moderators to detect and manage the content that is being published.

Functionalities

Spam detection

This service allows you to install and configure a spam detection service so that any suspicious content get reported at the very early possible step.

This service can be trained either using the datasets that we provide, or using your own content. This way the engine learns your own business domain and it is able to detect spam or things that are not relevant for your activity.

Installation

In order to install the module, you need to run the following command:

```
bundle add decidim-ai
```

Configuration

The AI tool allows you to configure most of the parameters using either defaults, or the initializer. In order to get control of your AI installation, you may need to create an initializer in your application.

You can do it by creating a file in `config/initializers/decidim_ai.rb` with the following content:

```
Decidim::Ai::SpamDetection.resource_score_threshold = 0.75 # default
# The entry must be a hash with the following keys:
# - name: the name of the analyzer
# - strategy: the class of the strategy to use
# - options: a hash with the options to pass to the strategy
# Example:
# Decidim::Ai.registered_analyzers = [
#   {
#     name: :bayes,
#     strategy: Decidim::Ai::SpamContent::BayesStrategy,
```

```
#     options: {
#       adapter: :redis,
#       params: {
#         url:                lambda { ENV["REDIS_URL"] }
#       }
#     }
#   }
# ]
Decidim::Ai::SpamDetection.resource_analyzers = [
  {
    name: :bayes,
    strategy: Decidim::Ai::SpamDetection::Strategy::Bayes,
    options: {
      adapter: ENV.fetch("DECIDIM_SPAM_DETECTION_BACKEND_RESOURCE", "redis"),
      params: { url:
ENV.fetch("DECIDIM_SPAM_DETECTION_BACKEND_RESOURCE_REDIS_URL",
"redis://localhost:6379/2") }
    }
  }
]

Decidim::Ai::SpamDetection.reporting_user_email = "your-admin@example.org"

# If you want to use a different spam detection service,
# you can use a class service having the following contract
#
Decidim::Ai::SpamDetection.resource_detection_service =
"Decidim::Ai::SpamDetection::Service"

# Customize here what are the analyzed models. You may want to use this to
# override what we register by default, or to register your own resources.

Decidim::Ai::SpamDetection.resource_models = {
  "Decidim::Comments::Comment" =>
"Decidim::Ai::SpamDetection::Resource::Comment",
  "Decidim::Initiative" => "Decidim::Ai::SpamDetection::Resource::Initiative",
  "Decidim::Debates::Debate" => "Decidim::Ai::SpamDetection::Resource::Debate",
  "Decidim::Meetings::Meeting" =>
"Decidim::Ai::SpamDetection::Resource::Meeting",
  "Decidim::Proposals::Proposal" =>
"Decidim::Ai::SpamDetection::Resource::Proposal",
  "Decidim::Proposals::CollaborativeDraft" =>
"Decidim::Ai::SpamDetection::Resource::CollaborativeDraft",
  "Decidim::User" => "Decidim::Ai::SpamDetection::Resource::UserBaseEntity"
}

Decidim::Ai::SpamDetection.user_score_threshold = 0.75 # default

# The entry must be a hash with the following keys:
# - name: the name of the analyzer
# - strategy: the class of the strategy to use
```

```
# - options: a hash with the options to pass to the strategy
# Example:
# Decidim::Ai::SpamDetection.user_analyzers = [
#   {
#     name: :bayes,
#     strategy: Decidim::Ai::SpamContent::BayesStrategy,
#     options: {
#       adapter: :redis,
#       params: {
#         url:                lambda { ENV["REDIS_URL"] }
#       }
#     }
#   }
# ]
Decidim::Ai::SpamDetection.user_analyzers = [
  {
    name: :bayes,
    strategy: Decidim::Ai::SpamDetection::Strategy::Bayes,
    options: {
      adapter: ENV.fetch("DECIDIM_SPAM_DETECTION_BACKEND_USER", "redis"),
      params: { url: ENV.fetch("DECIDIM_SPAM_DETECTION_BACKEND_USER_REDIS_URL",
"redis://localhost:6379/3") }
    }
  }
]

# Customize here what are the analyzed models. You may want to use this to
# override what we register by default, or to register your own resources.
# Follow the documentation on how to trail more resources
Decidim::Ai::SpamDetection.user_models = {
  "Decidim::User" => "Decidim::Ai::SpamDetection::Resource::UserBaseEntity"
}

# If you want to use a different spam detection service, you can define your own
service.
# Refer to documentation for more details.
#
Decidim::Ai::SpamDetection.user_detection_service =
"Decidim::Ai::SpamDetection::Service"
```

Commands

Decidim Ai provides a set of commands that you can use to manage the engine.

Create reporting user

In order to preserve the database integrity, you need to configure a system user that could be used to report content in the application. Use the following command to create an user for each one of the or-

ganizations you may have. The email address defined by `Decidim::Ai::SpamDetection.reporting_user_email` will be used to find or create the user.

```
bin/rails decidim:ai:spam:create_reporting_user
```

Load custom model

In some cases, when you manage multiple installations, you may want to share the same model between them. You can use the following command to load a simple CSV.

```
bin/rails decidim:ai:spam:load_application_dataset[/path/to/file.csv]
```

Load the data from your server

In some cases, like an upgrade, you may want to train your model using your existing data, so you can use:

```
bin/rails decidim:ai:spam:train_application_database
```

Reset the model

If the trained model becomes corrupt, you could use the below command to reinitialize the model. On you do this, you would need to train the model again. using any of the above commands.

```
bin/rails decidim:ai:spam:reset
```

Sidekiq

Decidim Ai comes with a new queue that is aimed to be ran to analyze the content of the platform. We have decided to have it in a separate queue to avoid blocking other events that your sidekiq may use.

We start to provide the `spam_analysis` queue name.