

PD-5: Tags on events

Status: draft · Revision 1 · Created by Ivan Blažević <ivan.blazevic@rubycode.co> · 6 October 2026

Overview

What

Organizers can add up to five tags to an event, like ruby, beginners or online. Tags show on the event page and in the events list, and clicking a tag lists the upcoming events that have it.

Why

People look for events on a topic and scroll the whole list today. With tags they find them in one click.

Where

The event form, the event page and the events list.

Who

Team Gather. Ana Kovač reviews the pull requests.

Success metric

Within a month, one in five visits to the events list uses a tag.

Background

Existing Data Structure

Nothing in the schema stores tags, keywords or labels today. There is no `tags` table, no join table and no tag column on `events`. The docs folder has a draft plan for event categories (`docs/plans/pd-2-event-categories/`), but no categories table or column is in the schema, so this plan does not depend on it.

Event (`app/models/event.rb` , `table events`)

Columns: `id` integer not null, `title` string not null, `description` text nullable, `starts_at` datetime not null, `venue` string not null, `capacity` integer not null, `organizer_id` integer not null, `created_at` and `updated_at` datetime not null.

Indexes: `index_events_on_organizer_id`, `index_events_on_starts_at`. Foreign key `organizer_id` points to `users`.

Associations and behaviour this change touches:

- `belongs_to :organizer, class_name: "User"`
- `has_many :rsvps, dependent: :delete_all` and `has_many :comments, dependent: :delete_all`. Child rows are removed with `delete_all`, and the new join table follows the same pattern.
- `has_many :attendees, through: :rsvps`
- `scope :upcoming, -> { where(starts_at: Time.current..).order(:starts_at) }`. The tag listing reuses this scope.
- `organized_by?(user)`. `EventsController#require_organizer` uses it to guard edit and update.

User (`app/models/user.rb`, `table users`)

Columns: `id`, `email_address` string not null (unique index), `password_digest` string not null, `name` string not null (default `""`), timestamps.

This change touches `has_many :organized_events, class_name: "Event", foreign_key: :organizer_id, inverse_of: :organizer, dependent: :destroy`. `EventsController#create` builds through it, and destroying a user destroys their events, so that path must also clean up the events' tag rows.

Rsvp (`app/models/rsvp.rb`, `table rsvps`)

Columns: `id`, `event_id` integer not null, `user_id` integer not null, `status` string not null (check constraint `status IN ('going', 'waitlisted')`), timestamps. It is not changed. It matters here because `EventsController#index` runs one grouped query, `Rsvp.going.where(event: @events).group(:event_id).count`, over the listed events. That query has to keep working on the tag-filtered relation, and the existing request spec asserts it runs exactly once.

Controller and views as they exist today

- `app/controllers/events_controller.rb`: `index` and `show` allow unauthenticated access. `index` loads `Event.upcoming.includes(:organizer)`. `event_params` is `params.expect(event: %i[ title description starts_at venue capacity ])`.
- `config/routes.rb`: `resources :events, except: :destroy`, and `root "events#index"`.
- `app/views/events/index.html.erb`: each card is one `link_to` event, `class: "event-card"`, `id: dom_id(event)` anchor that wraps the date badge, title, time, venue, organizer and seats label.
- `app/views/events/show.html.erb`: the header has the title and time. The `#event-facts` aside is a `<dl>` with `Where`, `Organizer` and `Seats`.
- `app/views/events/_form.html.erb`: fields for title, description, `starts_at`, capacity and venue.

Architectural changes

Target design

Tags are rows in a new `tags` table with one normalized `name` (for example `ruby`, `beginners`, `online`). Events and tags are linked by a new join table, `event_tags`. The same tag row is shared by every event that uses it, which is what lets a click on `ruby` find all events tagged `ruby`.

- `Tag` (`app/models/tag.rb`) has `has_many :event_tags` and `has_many :events, through: :event_tags`.
- `EventTag` (`app/models/event_tag.rb`) has `belongs_to :event` and `belongs_to :tag`.
- `Event` gains `has_many :event_tags, dependent: :delete_all` and `has_many :tags, -> { order(:name) }`, through: `:event_tags`.

Organizers type tags into one text field on the event form, separated by commas: `ruby, beginners, online`. This needs no JavaScript and fits the current form. The text goes through a virtual attribute, `Event#tag_list`. It is not a column.

Data flow

1. **Create and edit.** `EventsController#event_params` also permits `:tag_list`. `create` keeps its current shape, `Current.user.organized_events.build(event_params)`, and `update` keeps `require_organizer`. Assigning `tag_list=` only parses and stores the names in memory. It splits on commas, normalizes each name, removes blanks and duplicates, and keeps the result in `@tag_names`. Validation then checks for at most five names and a valid format for each. Only after a successful save does an `after_save` callback, running inside the save transaction, look up or create each `Tag` with `Tag.create_or_find_by!(name:)` and replace the event's `event_tags` rows. An invalid form therefore never creates tags or touches join rows. If `tag_list` was not submitted, the event's tags are left alone.
2. **Events list.** `EventsController#index` still starts from `Event.upcoming`, now `.includes(:organizer, :tags)`. When `params[:tag]` is present, the controller normalizes it the same way `Tag` does and looks it up (`@tag = Tag.find_by(name:)`). It then narrows the list with `.where(id: @tag.event_tags.select(:event_id))`. A subquery is used instead of a join so the results have no duplicate rows and the `includes(:tags)` preload still loads every tag on each card. An unknown tag shows the normal page with an empty state. `@going_counts` stays one grouped query over whatever `@events` is.
3. **Tag links.** Each tag is rendered as a GET link to `events_path(tag: tag.name)`, for example `/events?tag=ruby`. The filtered list is the existing index page, so it can be bookmarked, it needs no new controller, and it counts as a visit to the events list for the success metric. When a tag is active, the page heading says `Upcoming events tagged ruby` and has an `All events` link back to `/events`.
4. **Display.** A shared partial, `app/views/events/_tags.html.erb`, renders an event's tags as a list of links. It is used on the event page and on every card in the list.

Events list card markup

Today the whole card is a single `<a>`. Tag links inside it would be nested anchors, which is invalid HTML and makes clicks unreliable. The card becomes a container element that keeps `class: "event-card"` and `id: dom_id(event)`. Existing specs find cards by `dom_id`. Inside it, the title links to the event and the tag list sits beside that link, not inside it. CSS can keep the whole card clickable by stretching the title link over the card, with the tag links layered above it.

How legacy data coexists during rollout

The change only adds new tables. No existing column changes meaning.

- **Expand.** `tags` and `event_tags` are created empty. Code that is still running during the deploy never reads them, so it keeps working.
- **Dual write.** Not needed, because nothing is moving out of an existing column.
- **Backfill.** Not needed. Existing events have no tags, and `Event#tags` returns an empty list for them. The views render nothing for an event without tags, with no empty wrapper or label.
- **Switch reads.** The new form field, the tag display and the `?tag=` filter ship after the tables exist.
- **Contract.** Nothing to remove.

Suggested ticket split

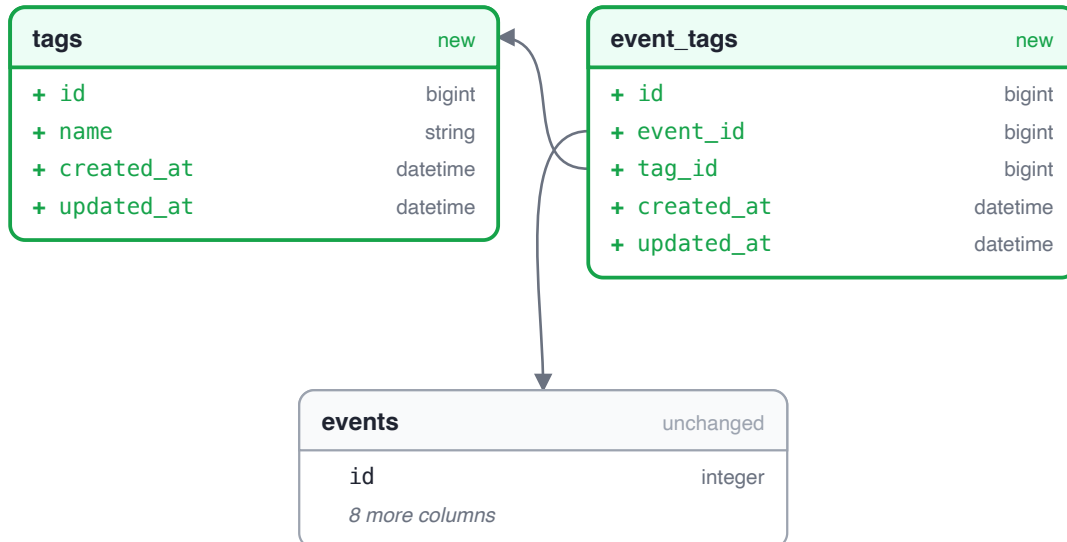
1. Migrations plus the `Tag` and `EventTag` models, with model and database specs.
2. `Event` associations, `tag_list` parsing, validation and saving, plus the form field and the `event_params` change.
3. The `_tags` partial on the event page, plus the restructured cards in the events list with tags shown.
4. The `?tag=` filter on `EventsController#index`, with the heading, the empty state and query-count specs.

Database changes

Data model

Tables this plan changes: 2 new. Read from the migration code in Database changes.

■ New ■ Changed ■ Removed ■ Referenced, unchanged



Two new tables. Nothing existing is changed, renamed or removed, so no `ignored_columns` step is needed. The database is SQLite (`config/database.yml`). The check constraint below uses `length()` and `lower()`, which SQLite supports. SQLite's `lower()` only folds ASCII letters, which fits the ASCII-only tag format in the assumptions.

Step 1: Expand, table `tags`

- `name` string, **not null**, no default. Stored already normalized: lowercase, trimmed, with inner spaces turned into hyphens.
- `created_at`, `updated_at` datetime, not null.
- Unique index `index_tags_on_name` on `name`. It makes `create_or_find_by!` safe when two organizers add the same new tag at the same moment, and it serves the `?tag=` lookup.
- Check constraint `tags_name_check`: length 1 to 30 and already lowercase. This backs up the model validation.

```
class CreateTags < ActiveRecord::Migration[8.1]
  def change
    create_table :tags do |t|
      t.string :name, null: false
      t.timestamps
    end
    add_index :tags, :name, unique: true
    add_check_constraint :tags, "length(name) BETWEEN 1 AND 30 AND name = lower(name)"
  end
end
```

```
end
end
```

Step 1: Expand, table `event_tags`

- `event_id` integer, **not null**. Foreign key to `events` with `on_delete: :cascade`, so a deleted event never leaves join rows behind, even when the delete skips callbacks.
- `tag_id` integer, **not null**. Foreign key to `tags` without cascade, so deleting a tag that is still in use fails loudly instead of silently untagging events.
- `created_at`, `updated_at` datetime, not null.
- Unique index `index_event_tags_on_event_id_and_tag_id` on `[event_id, tag_id]`. It stops the same tag being attached twice and serves the preload of tags by `event_id` on the list and the event page. Because it covers `event_id` lookups, no separate `event_id` index is created.
- Index `index_event_tags_on_tag_id` on `tag_id`. It serves the `?tag=` filter subquery.

```
class CreateEventTags < ActiveRecord::Migration[8.1]
  def change
    create_table :event_tags do |t|
      t.references :event, null: false, index: false, foreign_key: { on_delete: :cascade }
      t.references :tag, null: false, foreign_key: true
      t.timestamps
    end
    add_index :event_tags, %i[ event_id tag_id ], unique: true
  end
end
```

Later steps

- **Dual write and backfill:** none. Existing events start with no tags.
- **Switch reads:** application code ships after both migrations have run. See Infrastructure changes.
- **Contract:** none.

The five-tags-per-event limit lives in the model only. It would take a trigger to enforce it in SQLite, and the only writer is the organizer's own form. `db/schema.rb` is regenerated by the migrations and committed with them.

Application changes

Tag (`app/models/tag.rb`, **new**)

- `has_many :event_tags, dependent: :restrict_with_exception` and `has_many :events, through: :event_tags`.
- `NAME_FORMAT = /\A[a-z0-9]+(-[a-z0-9]+)*\z/` and `MAX_LENGTH = 30`.

- `normalizes :name, with: ->(name) { name.strip.downcase.gsub(/\s+/, "-") }`. The controller reuses this through `Tag.normalize_value_for(:name, params[:tag])`, so ? `tag=Ruby` finds `ruby`.
- `validates :name, presence: true, length: { maximum: MAX_LENGTH }, format: { with: NAME_FORMAT }, uniqueness: true`.

`EventTag ( app/models/event_tag.rb , new)`

- `belongs_to :event` and `belongs_to :tag`.
- `validates :tag_id, uniqueness: { scope: :event_id }`. This gives a friendly error, and the unique index is the real guarantee.

`Event ( app/models/event.rb )`

- `MAX_TAGS = 5`.
- `has_many :event_tags, dependent: :delete_all` and `has_many :tags, -> { order(:name) }, through: :event_tags`.
- `tag_list`: returns `@tag_names.join(", ")` when the form has assigned names, otherwise `tags.map(&:name).join(", ")`.
- `tag_list=(value)`: splits on commas, normalizes each part with `Tag.normalize_value_for(:name, part)`, drops blanks, removes duplicates, and stores the result in `@tag_names`. It does not touch the database.
- `validate :tag_names_are_valid`, which runs only when `@tag_names` is set. It adds an error on `:tag_list` when there are more than `MAX_TAGS` names ("can't have more than 5 tags"). It also adds one for any name that does not match `Tag::NAME_FORMAT` or is longer than `Tag::MAX_LENGTH`, naming the bad tag. Checking count and format before saving keeps a bad submission from creating any `Tag` rows.
- `after_save :save_tags, if: -> { @tag_names } : self.tags = @tag_names.map { Tag.create_or_find_by!(name: it) }, then @tag_names = nil`. It runs inside the save transaction, so a failure rolls back the event change too.
- `scope :tagged, ->(tag) { where(id: tag.event_tags.select(:event_id)) }`.
- `reload` already clears `@seats_taken`, and it should also clear `@tag_names`.

`EventsController ( app/controllers/events_controller.rb )`

- `event_params: params.expect(event: %i[ title description starts_at venue capacity tag_list ])`.
- `index:`
- `@events = Event.upcoming.includes(:organizer, :tags)`.
- When `params[:tag].present?`, set `@tag_name = Tag.normalize_value_for(:name, params[:tag].to_s)` and `@tag = Tag.find_by(name: @tag_name)`. Then narrow with `@events = @tag ? @events.tagged(@tag) : @events.none`.
- `@going_counts` is unchanged.
- `show`: no change. The view reads `@event.tags`.
- `create`, `update`, `require_organizer` and authentication rules are unchanged. Only the organizer can change an event's tags, because only they can submit the form.

Routes (config/routes.rb)

No change. The filter is a query parameter on the existing `events#index` route.

Helper (app/helpers/application_helper.rb)

Add `tag_link(tag)`, which returns `link_to tag.name, events_path(tag: tag.name), class: "tag"`. It gets `aria-current="page"` when `tag.name == @tag&.name`.

Views

- `app/views/events/_tags.html.erb` (new): takes `tags:` as a local. It renders `<ul class="tag-list">` with one `tag_link` per tag, or nothing when the list is empty.
- `app/views/events/_form.html.erb`: add a `tag_list` field after Venue. Use `form.label :tag_list, "Tags"`, and `form.text_field :tag_list, placeholder: "ruby, beginners, online"` with a hint "Up to 5, separated by commas". Errors appear in the existing error box.
- `app/views/events/show.html.erb`: render `events/tags` in the header under the time, with `tags: @event.tags`.
- `app/views/events/index.html.erb`:
- Restructure each card so it is no longer one `<a>`. The outer element keeps `class="event-card"` and `id=dom_id(event)`. The title becomes `link_to event.title, event.events/tags` is rendered with `tags: event.tags`, outside the title link.
- When `@tag_name` is set, the hero heading reads Upcoming events tagged `<name>`, with an All events link to `events_path`.
- The empty state says No upcoming events tagged `<name>`. instead of No upcoming events yet.
- Stylesheet: add `.tag-list` and `.tag` styles, and keep the whole card clickable.

Seeds (db/seeds.rb)

Give the three demo events tags through `tag_list`. For example, `ruby, rails` for Ruby Zagreb Meetup, `hotwire, workshop` for the Hotwire workshop, and `rails, online` for the Rails upgrade clinic.

Factories (spec/factories/tags.rb, spec/factories/event_tags.rb, new)

- `tag` with `sequence(:name) { "tag-#{it}" }`.
- `event_tag` with an event and a tag.
- A `:tagged` trait on `spec/factories/events.rb` that sets `tag_list`.

Infrastructure changes

None of these are needed:

- no new queues or jobs
- no external services
- no new gems
- no environment variables

The Gemfile has no feature-flag library, and this plan does not add one.

Rollout order:

1. Deploy the two migrations (`CreateTags` , `CreateEventTags`) together with the `Tag` and `EventTag` models. They only create empty tables, so app processes still running the old code are unaffected.
2. Deploy the `Event` associations, the form field, the tag display and the `?tag=` filter. This can be the same deploy as step 1, provided migrations finish before the new code serves requests. If the deploy process cannot guarantee that, ship step 1 on its own first.
3. Run `db/seeds.rb` in development only. Production events get tags only from their organizers.

Rollback: revert the application code first. The tables can stay, because old code ignores them. Drop them only if the feature is abandoned.

Work overview

Out of Scope

Tag suggestions, following a tag, renaming or merging tags.

Risks

- **Nested links on the events list.** Putting tag links inside today's whole-card `<a>` would produce invalid HTML, and clicks could open the event instead of the tag. *Mitigation:* restructure the card so the title link and tag links are siblings. Keep `id=dom_id(event)` on the card so existing specs still find it. Add a request spec asserting no `a a` in the list.
- **Tags created by an invalid form.** Creating `Tag` rows in the `tag_list=` setter would leave junk tags when the event fails validation. Assigning `has_many :through` on a saved event would also write join rows before validation. *Mitigation:* the setter only stores names. Tags and join rows are written in `after_save`, inside the save transaction.
- **Two organizers adding the same new tag at once.** *Mitigation:* the unique index on `tags.name`, plus `Tag.create_or_find_by!`, which retries as a find when the insert hits the unique index.
- **Near-duplicate tags such as** `Ruby`, `ruby`, `ruby on rails` **and** `ruby-on-rails`. These split one topic across several tags and weaken the one-click search. *Mitigation:* normalize to lowercase, trim, and turn spaces into hyphens before validating and storing. Synonyms like `ror` versus `rails` remain possible, because merging tags is out of scope.
- **Offensive or spam tags on public pages.** Any signed-in organizer can create any tag, and anyone can see it. *Mitigation:* tags only appear on events the author organizes, and the organizer can remove them. Moderation is listed under Outstanding questions.
- **Breaking the one-query** `@going_counts` **on the list.** *Mitigation:* filter with a subquery on `events.id`, not a join, so `Rsvp.going.where(event: @events)` stays one grouped query. The existing spec covers this, and a filtered variant is added.

- **Editing an event without the tag field clears its tags.** For example, an older form posting after deploy. *Mitigation:* tags are only replaced when `tag_list` is present in the params. A missing key leaves them untouched.

Performance

- **Events list** (`EventsController#index`). The query count stays fixed however many events are listed: one for `events`, one for `users` (organizer preload), one for `rsvps` (grouped going counts), one for `event_tags` and one for `tags` (both from the `includes(:tags)` preload). With `?tag=`, one more query looks up the tag by name, and the filter becomes a subquery inside the `events` query, not an extra round trip.
- **Indexes.** The tag lookup uses the unique index on `tags.name`. The filter subquery `SELECT event_id FROM event_tags WHERE tag_id = ?` uses `index_event_tags_on_tag_id`. The preload `WHERE event_id IN (...)` uses the leading column of `index_event_tags_on_event_id_and_tag_id`. The outer query keeps using `index_events_on_starts_at` for upcoming.
- **N+1 risk.** Calling `event.tags` on a card without the preload would cost a query per card. The plan adds a request spec that counts `FROM "event_tags"` and `FROM "tags"` queries with 1 and with 10 tagged events, in the style of the existing comment and RSVP query-count specs. `features/step_definitions/events_index_steps.rb` already has a "queries ran against" step that the Cucumber features can reuse.
- **Event page.** It adds two queries (`event_tags` and `tags`) for at most five tags.
- **Saving an event.** It costs up to five `create_or_find_by!` calls plus the join-row replacement, all on very small tables, and only on create or edit.
- **Table size.** Both new tables start empty and grow with the number of events. `event_tags` has at most five rows per event.
- **Backfill.** None, so no batch size applies.
- The events list is not paginated today. A popular tag returns no more rows than the unfiltered list already does.

Security

- **Authorization.**
- Tags are only changed through the event form. `create` builds the event on `Current.user.organized_events`, and `update` sits behind the existing `require_organizer` check, so only the organizer can add or remove an event's tags.
- The list and the event page, including tags and the `?tag=` filter, stay public, matching today's `allow_unauthenticated_access only: %i[ index show ]`.
- No new routes are added.
- **Input validation.**
- `tag_list` is permitted as one scalar string through `params.expect`.
- Each name is normalized, then checked against `Tag::NAME_FORMAT` (letters, digits and hyphens) and `Tag::MAX_LENGTH` (30). The list is capped at `Event::MAX_TAGS` (5). Both checks run before any row is written.

- The `tags_name_check` constraint backs up the length and lowercase rules in the database.
- **Filter parameter.** `params[:tag]` is normalized and used only as a bound value in `Tag.find_by(name:)`, so it carries no SQL injection risk. Unknown or malformed values just show an empty list.
- **Output.** Tag names are rendered through ERB escaping. Links are built with `events_path(tag: tag.name)`, which URL-encodes the query string. The heading echoes the normalized `@tag_name` through escaping, so a crafted `?tag=` cannot inject markup.
- **Data exposure.** Tags are public by design. No user data is added to any page.
- **Content.** Tags are free text visible to everyone. Moderation is an open question, not a security control in this plan.

Risk Level: LOW. The change adds public, escaped, format-restricted text that only an event's organizer can set, through the existing authorization checks.

Monitoring

- **Errors.** Watch the error tracker or production log for exceptions from `EventsController#create`, `#update` and `#index` after release. Pay particular attention to `ActiveRecord::RecordNotUnique` and `ActiveRecord::StatementInvalid` from `tags` and `event_tags` (constraint failures), and to `ActiveRecord::RecordInvalid` raised from `Event#save_tags`.
- **Form failures.** Watch the rate of 422 responses on event create and update compared with before release. A jump suggests the tag format rules are too strict.
- **Success metric (one in five visits to the events list uses a tag).** Measure `GET /events` and `GET /` requests handled by `EventsController#index` that include a `tag` parameter, as a share of all `EventsController#index` requests. Rails already logs the parameters of each request. Check weekly for the first month. The app has no analytics today, so the source of this number is an outstanding question.
- **Adoption.** Check weekly from the console:
 - the share of upcoming events with at least one tag: `Event.upcoming.where(id: EventTag.select(:event_id)).count` against `Event.upcoming.count`
 - the total `Tag.count`
 - the list of tags used by only one event, which is a quick way to spot typos, near-duplicates and abusive tags
- **Performance.** Watch the response time of `EventsController#index` with and without `tag`. It should not change noticeably.

Outstanding questions

1. **Allowed characters.** Must ship before the form and migration tickets. Should tags allow Croatian letters (č, ć, đ, š, ž), spaces or other punctuation? Is 30 characters the right maximum? The plan assumes lowercase ASCII letters, digits and hyphens, with spaces turned into hyphens. Allowing non-ASCII letters changes both the validation and the database check constraint, because SQLite's `lower()` does not fold them.

2. **Unknown tag in the URL.** Must ship before the filter ticket. Should `/events?tag=nonexistent` show an empty list (`200` , as planned) or return `404` ?
3. **Measuring the success metric.** Must ship before release. Where are production request logs kept, and who reports the monthly number? Does a visit to `/` (the root, which is also `events#index`) count as a visit to the events list?
4. **Moderation.** Must ship before release. Can anyone other than the organizer remove an offensive tag, for example an admin? Should a list of banned words exist? The app has no admin role in the schema today.
5. **Relation to the event categories plan.** Must ship before the filter ticket. `docs/plans/pd-2-event-categories/` plans a `?category=` filter on the same page, and nothing from it is in the schema yet. If both ship, should `?category=` and `?tag=` combine, and which ships first?
6. **Tag order.** Should tags show in the order the organizer typed them, or alphabetically (as planned)? Keeping typed order would need a `position` column on `event_tags` .
7. **Unused tags.** When the last event using a tag drops it, should the `tags` row be deleted? The plan keeps it. It does no harm, because tags are only displayed through events.

Testing

Model: `spec/models/tag_spec.rb`

- Normalizes `Ruby` , `RUBY` and `ruby on rails` to `ruby` , `ruby` and `ruby-on-rails` .
- Rejects blank names, names over 30 characters, and names with characters outside letters, digits and hyphens.
- Rejects a duplicate name.
- Cannot be destroyed while an event uses it.

Model: `spec/models/event_spec.rb`

- `tag_list = "ruby, beginners, online"` saves three tags and reuses an existing `Tag` row instead of creating a duplicate.
- Duplicates and blanks are removed: `"ruby, Ruby, , ruby "` gives one tag.
- Six tags make the event invalid, with an error on `tag_list` . No `Tag` or `EventTag` rows are created.
- One malformed tag makes the event invalid, the error names that tag, and no rows are written.
- Editing from `ruby, online` to `ruby, beginners` removes `online` from the event, adds `beginners` , and keeps the `online` tag row.
- An empty `tag_list` removes all tags.
- Updating other attributes without assigning `tag_list` leaves tags unchanged.
- An existing event without tags (legacy data) is valid, returns no tags, and saves without a `tag_list` .
- `Event.tagged(tag)` returns only events with that tag, and combined with `upcoming` excludes past events.

- Deleting an event removes its `event_tags` rows, both through `dependent: :delete_all` and through the database cascade.
- Destroying the organizer `User` removes their events' `event_tags` rows.

Database: `spec/db/event_tags_table_spec.rb` and `spec/db/tags_table_spec.rb`

Follow the existing `spec/db/*_table_spec.rb` pattern.

- The unique index on `[event_id, tag_id]` rejects a duplicate pair.
- The unique index on `tags.name` rejects a duplicate name.
- `tags_name_check` rejects an uppercase or a 31-character name inserted with `insert_all`.
- The foreign keys reject an unknown `event_id` or `tag_id`.

Request: `spec/requests/events_spec.rb`

Permissions:

- A guest cannot create a tagged event and is redirected to sign in.
- A signed-in non-organizer's `PATCH` with `tag_list` changes nothing.
- The organizer's `PATCH` updates tags.

Create and edit:

- `POST` with `tag_list: "ruby, online"` creates the event with both tags.
- `POST` with six tags returns `422` and shows the error.

Event page:

- Shows each tag as a link to `/events?tag=<name>`.
- Shows no tag list for an untagged event.

Events list:

- Each card shows its own tags, found through the card's `dom_id`.
- The page has no nested anchors (`a a`).
- `?tag=ruby` lists only upcoming events tagged `ruby`, in `starts_at` order, and excludes past events tagged `ruby`.
- `?tag=Ruby` behaves like `?tag=ruby`.
- An unknown tag shows `No upcoming events tagged ...` with status `200`.
- `?tag=<script>` is rendered escaped.
- The heading and `All events` link appear only when filtering.
- The seats label is still correct on a filtered list.

Query counts:

- `FROM "rsvps"` is still one query when filtering.
- `FROM "event_tags"` and `FROM "tags"` counts are the same with 1 and with 10 tagged events.

System and Cucumber: `features/pd-5-tags-on-events/*.feature`

- An organizer adds `ruby`, `beginners` on the new-event form and sees both tags on the event page.
- The organizer edits the event, removes a tag, and sees it gone.
- A guest on the events list clicks `ruby` on a card and sees only upcoming `ruby` events, with the heading `Upcoming events tagged ruby`, then clicks `All events` to return.
- A guest on an event page clicks a tag and lands on the filtered list.
- Reuse the existing "I visit the events page while counting queries" step to assert a fixed number of queries against `event_tags` and `tags`.

Sign-off

Role	Decision	By	When
Review	pending		