

can take a look at PR <https://github.com/choonkeat/tiny-form-fields/pull/60>

17.5s

Fetching PR #60 details...

1m 57s

PR #60 review — "fix validation skipping"

What it does: Adds a `selectedFieldIsValid` helper and gates `SelectField`, `DragStart`, and `DragStartNew` on it, so users can't navigate away from a selected field whose label is empty/whitespace or duplicates another. Also tightens the HTML5 pattern from `".*"` to `".+"` and adds a "Question title cannot be empty" inline error.

Looks good

- Focused, minimal diff in `src/Main.elm`.
- `String.trim` correctly catches whitespace-only labels in the new check.
- Pattern change `".*" → ".+"` matches the "label is required" intent.
- Empty-label error mirrors the existing duplicate-label error styling.

Worth a look

1. **Possible gap: `AddFormField` is not guarded.** While an invalid field is selected, clicking "Add field" (line 3244 / handler at line 687) still appends a new field. It doesn't change `selectedFieldIndex`, so the invalid field stays selected and its error remains visible — but the user can keep stacking fields on top of an invalid one. Probably lower severity than the `SelectField/Drag` bypass, but worth deciding if it should also be blocked for consistency.

2. **Silent no-op UX.** Guarded actions return `(model, Cmd.none)` with no feedback. If a user clicks another field to switch away, *nothing happens* — they may not realize the inline error a few rows up is the reason. Consider scrolling to the invalid field, flashing its error, or animating it (you already have `AnimateYellowFade`).

3. **Pre-existing inconsistency surfaced by this change.** `hasDuplicateLabel` (line 3641-ish) compares `f.label == newLabel` without trimming, while the new `isEmptyLabel` trims. So `"foo"` and `" foo"` are *not* treated as duplicates. Not introduced by this PR, but the new code makes the asymmetry more visible. Optional follow-up.

4. **No tests.** Given this is framed as a security-relevant fix ("circumvent validation"), a regression test asserting that `SelectField / DragStart / DragStartNew` are no-ops when the selected field has an empty/duplicate label would be valuable. Without it, future refactors could silently regress.

5. **Compiled `dist/*` files are bundled in the diff.** That's consistent with this repo's style (they were committed in prior merges), so just confirming it's intentional rather than build noise.

Verdict

Solid fix for the stated bug. I'd merge after deciding on points 1 & 2 — both are quick.

Approve and merge

Address points 1 & 2 first

Add a regression test

Look at something else

I doubt scattering `selectedFieldIsValid model` then in various places is the best way to address this issue; susceptible to leaving gaps where bug reappears

Is this a Editor mode only problem? The submit button is not part of our form builder right? What mechanism is currently controlling whether we can submit a form that is currently being built/edited?

6.5s

Good push-back. Investigating the submission/persistence flow to find where the "form is valid" decision actually happens...

1m 20s

You're right — and the investigation backs you up. Here's the actual mechanism:

How submission really works

1. Tiny-form-fields **doesn't have its own submit**. The Elm app is rendered *inside* the host page's `<form method="post">` (see `index.html:54`).
2. The "Submit form config" button (`index.html:80-86`) is a plain `<button type="submit">` belonging to the host form. The Elm app emits `formFields` JSON to the host on every change via the outgoing port (`Main.elm:710`, etc.) — that emission is **not gated by validation**, it just keeps the host's hash/URL in sync.
3. What actually blocks a bad submit is **HTML5 native form validation**: the browser walks every input inside the `<form>`, checks `required / pattern / etc.`, and refuses to submit if any fails. That's the only gate.

Why the bug exists

- `viewFormFieldBuilder` (the panel with the `<input pattern="...">` for the label) is only rendered for `selectedFieldIndex` (`Main.elm:3189-3196`). Other fields render a non-input summary.
- So when you blank out field A's label and click field B, A's `<input pattern>` leaves the DOM. The browser sees no invalid inputs → submit succeeds → host saves a config with an empty label.
- This is **Editor-only**. `CollectData` mode renders its own input set per field and is unaffected.

Why the PR's approach is fragile (you're correct)

Guarding `SelectField / DragStart / DragStartNew` is plugging holes in an Elm-side router for a problem that lives in the DOM. It misses every other escape hatch: clicking the `×` close button (line 3184 — also calls `SelectField Nothing`, so happens to be guarded transitively, but only by accident), `AddFormField`, `DeleteFormField`, `MoveFormFieldUp/Down`, future actions, hash navigation, the host page calling Elm's `init` with new flags, etc. Every new `Msg` becomes a place where the bug could regress.

The cleaner fix

Make every field's invariants live permanently in the DOM, not just when selected. For each `FormField`, render an always-present hidden/visually-suppressed `<input>` with `pattern="."` and `value=field.label` (and similar for any other constraint that matters). Then the host form's native submit gate sees all of them at all times, regardless of selection or editor state.

Concretely, somewhere in the Editor view (e.g. alongside the field list, before `viewFormFieldBuilder`), emit:

```
formFields
  |> Array.toList
  |> List.indexedMap (\i f ->
    input
      [ type_ "text"
      , Attr.attribute "aria-hidden" True
      , Attr.tabindex -1
      , class "tff-validation-ignored"
      , Attr.value f.label
      , Attr.required True
      , Attr.pattern (if hasDuplicate f.label "" ".")
      , Attr.title (if hasDuplicate f.label "" f.label)
      ]
  )
```

Then **delete** the three `if selectedFieldIsValid model` guards in `update`. The native browser validation does the work, with one source of truth, and it's not bypassable by adding new `Msgs` in the future. The `pattern="."` change in the visible label input is still good — that's the inline UX feedback when the user is actively editing.

One caveat

A shadow input needs to be focusable enough that the browser's "scroll to first invalid" works on submit, but not part of tab order. `tabindex="-1"` plus a CSS `display:none`, which excludes it from validation tooltip will anchor to the shadow input — if you want the tooltip to appear on the visible field instead, that's a small extra step (intercept the form's `invalid` event and `focus()` the field row, or switch to a JS-side check on the form's `submit` event seeing the `formFields`-derived state).

Want me to draft the fix as a follow-up commit/PR comment?

Draft the shadow-input fix

Post this analysis as a PR review comment

Try a JS submit-event approach instead

Leave the PR as-is and follow up later

Good background and principles. Make every field's invariants live permanently in the DOM, not just when selected. I'm wary about performance for this. In the `CollectData` mode, I recall we use hidden invalid field to make the form invalid when there's the number of checkboxes selected does not satisfy the user defined rule (e.g. minimum 2 checkboxes, maximum 3 checkboxes)

1. locate this mechanism
2. I'm thinking of using something like this to power our logic of "this form is still invalid"

discuss

2.7s

Locating the checkbox min/max invalid-input mechanism in `CollectData` mode...

1m 22s

Found it — and you're right, this is the correct lever.

The existing mechanism (CollectData)

`src/Main.elm:2309-2339` in the `ChooseMultiple` branch:

```
-- Add validation element for CollectData
validationElement =
  let
    effectiveMin = ...
    needsValidation = effectiveMin > selectedCount
  in
    if not disabledMode && needsValidation
    [ input
      [ type_ "number"
      , required True
      , attribute "value" (String.fromInt selectedCount)
      , attribute "min" (Maybe.fromInt effectiveMin)
      , attribute "max" (Maybe.fromInt maxCount)
      , attribute "class" "tff-validation-ignored"
      ]
    ]
  else
    []
```

CSS at `dist/tiny-form-fields.min.css`:

```
.tff-visually-hidden{position:absolute; left:-9999px; top:-9999px; width:1px; height:1px; border:1px solid black; border-radius:3px; background-color:black; opacity:0.5; pointer-events:none; z-index:-1}
```

Why this works: the browser validates all inputs in the host `<form>` regardless of CSS positioning, *as long as they aren't `display:none / [hidden] / disabled`*. Off-screen still counts. So this single number-input is the entire "the form is in an unsubmittable state because checkbox count is out of range" signal — no Elm-side guards anywhere.

Performance properties of this pattern

- Cheap on purpose:
- **Conditional emission** — only when the field actually has constraints. Zero DOM nodes when not needed.
 - **One node per constrained field, not per choice**. So $O(K)$ where K = constrained fields, typically 0 or 1.
 - **Reuses already-computed `selectedCount`** — no extra traversal.
 - **No diff churn on unrelated edits** — the input's attrs only change when the count or the constraint changes.

Applying it to the Editor-mode label-validation problem

Same shape, even cheaper. The "form is invalid" condition reduces to a single boolean computed in one $O(N)$ pass over `formFields`. Emit **one** hidden input total, only when invalid:

```
-- single-pass validity check
editorFormValidity : Array FormField -> EditorFormValidity
editorFormValidity formFields =
  let
    labels = Array.toList formFields
    labelCounts = List.foldl (\l -> Dict.update l (+1)) Dict.empty labels
    firstIssue = formFields
      |> Array.toList
      |> List.filterMap (\f ->
        if String.isEmpty f.label
        Just "Question title cannot be empty"
        else if (Dict.get f.label labelCounts) > 1
        Just ("Duplicate label: " <f.label> ".")
        Nothing)
      |> List.head
  in
    firstIssue
```

```
-- somewhere inside the Editor view,
viewEditorValidationGate : Array FormField -> EditorFormValidity -> Maybe FormField
viewEditorValidationGate formFields editorFormValidity formField =
  case editorFormValidity formFields of
    Nothing ->
      text ""
    Just msg ->
      input
        [ type "text"
        , required True
        , Attr.attribute "value" f.label
        , Attr.attribute "class" "tff-validation-ignored"
        , Attr.tabindex -1
        ]
```

Then **delete**:

- the three `if selectedFieldIsValid model` then `(model, Cmd.none)` else `...` blocks in `update`,
- the `selectedFieldIsValid` helper itself.

Keep the `pattern="."` change on the visible label input in `viewFormFieldBuilder` — it's the right inline UX while editing the selected field. The hidden gate just guarantees the *whole-form* invariant survives across selection changes.

Why this answers the original concerns

- **Performance:** one `Dict` build ($O(N)$) + one DOM node only when invalid. The common case (valid form) emits zero extra DOM. Cheaper than my earlier per-field-shadow-input idea.
- **No gap-prone scattering:** there is no list of "Msgs that need a guard". Any future `Msg`, hash navigation, port replay, or weird flow path goes through the host form's submit, which is gated by the DOM.
- **Single source of truth:** add a new field-level invariant later? Append a clause to `editorFormValidity`. Done.
- **Editor-only:** matches the problem scope. `CollectData` has its own per-field gates already (the existing `pattern`, plus `required`, plus the checkbox min/max one) and is unaffected.

One detail worth deciding

Browser tooltips on a `position:absolute; left:-9999px` element anchor at the off-screen location — the user sees a generic "Please fill in this field" without scroll. If you want the tooltip to surface near the offending row, two cheap options:

1. Render the gate input inline with the offending row instead of as a single global node (back to $O(K)$ emission, still cheap).
2. Listen for the form's `invalid` event in JS and `scrollIntoView` on the corresponding row.

Want me to draft this as a follow-up commit on top of #60 (or as a comment on the PR proposing the alternative)?

Draft the follow-up commit

Post as a PR comment first

Also handle the per-row tooltip detail

Discuss further before coding