Web apps in go

Story of a small social network

Dmitrii (Dima) Petrov

WHY?

Should I build apps

Big
Product

Your
impact

Decisionmaking
is
a skill

PCOM

https://github.com/can3p/pcom

https://ididnteatthatcow.com

Xinstabook

Hate amplifier

Self-promotion

Big scale

=

Tech problems

Limited visibility by default

Not e2e encrypted

Markdown all the way

No private posts

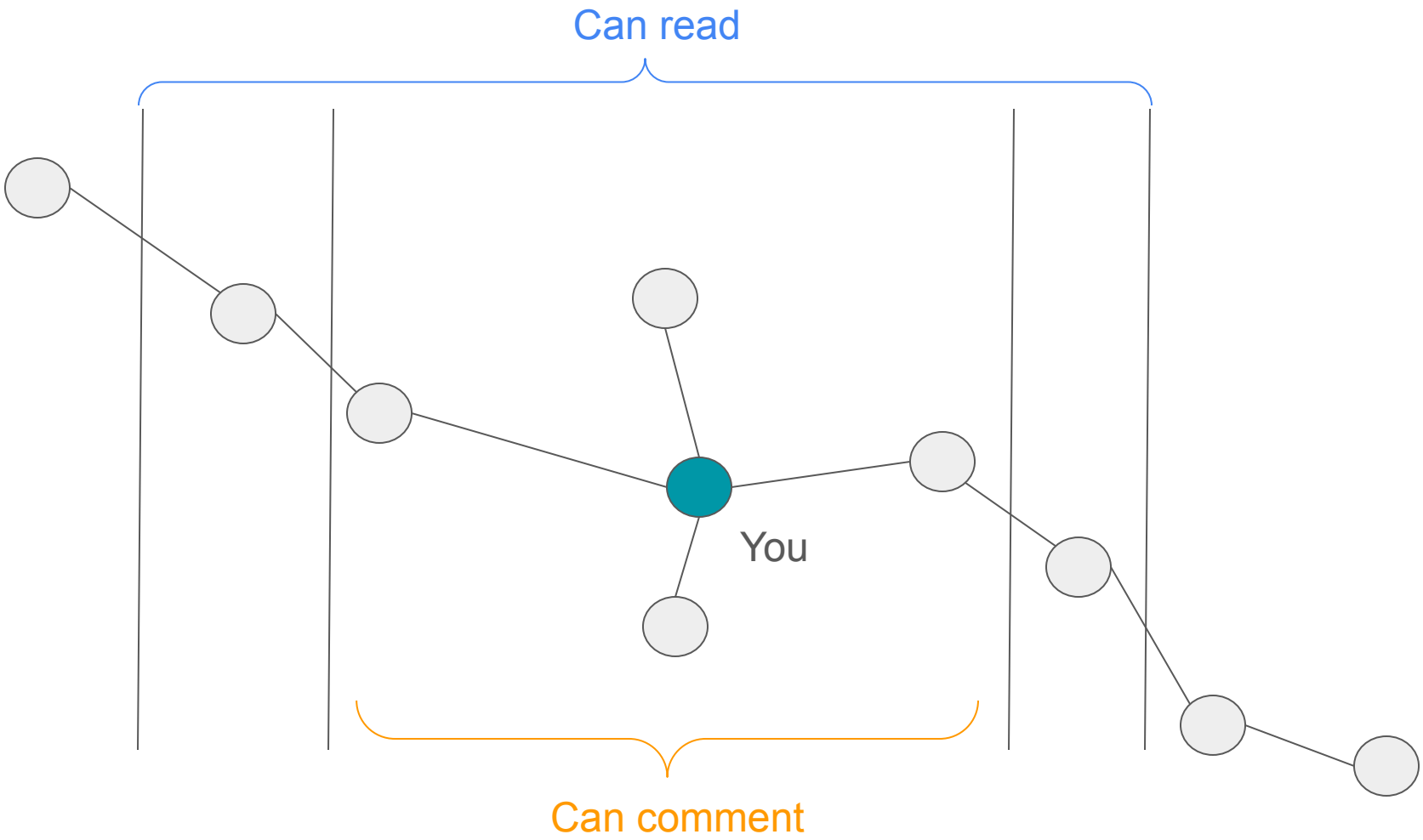
pcom

Other formats

Open platform

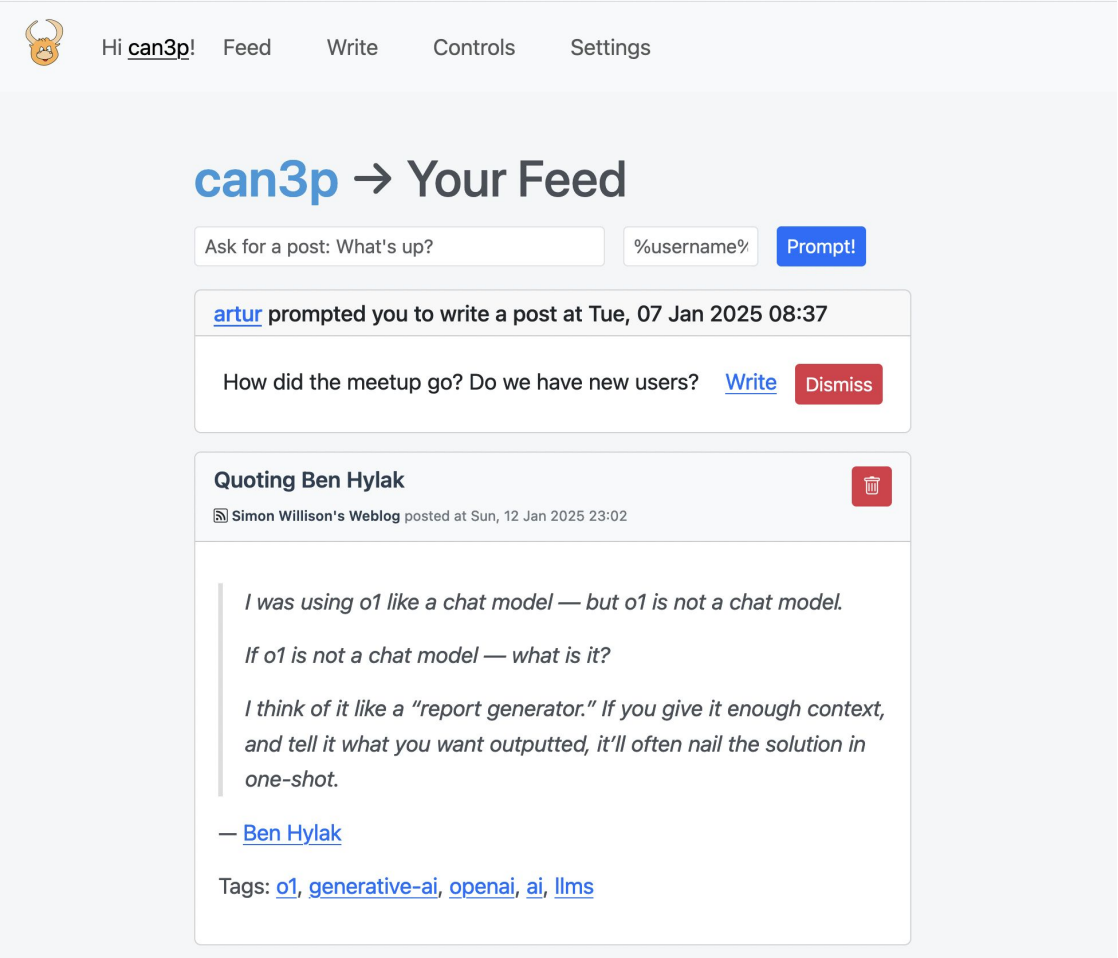
Not federated

Client friendly


Can read
You
Can comment

Feed

- Posts
- Comments
- Rss Feeds
- Prompts



New connections: introduction or direct invite

can3p → Latest Posts

One of your direct connections has a connection with can3p Ask for introduction

One of your direct connections has a connection with can3p *You've requested mediation request with note* ***I'm bob, we've met on the meetup, let's follow each other!*** Revoke request

Mediation Requests

- User bob has asked you to introduce them to can3p with note **I'm bob, we've met on the meetup, let's follow each other!** Sign Dismiss

Connection Requests

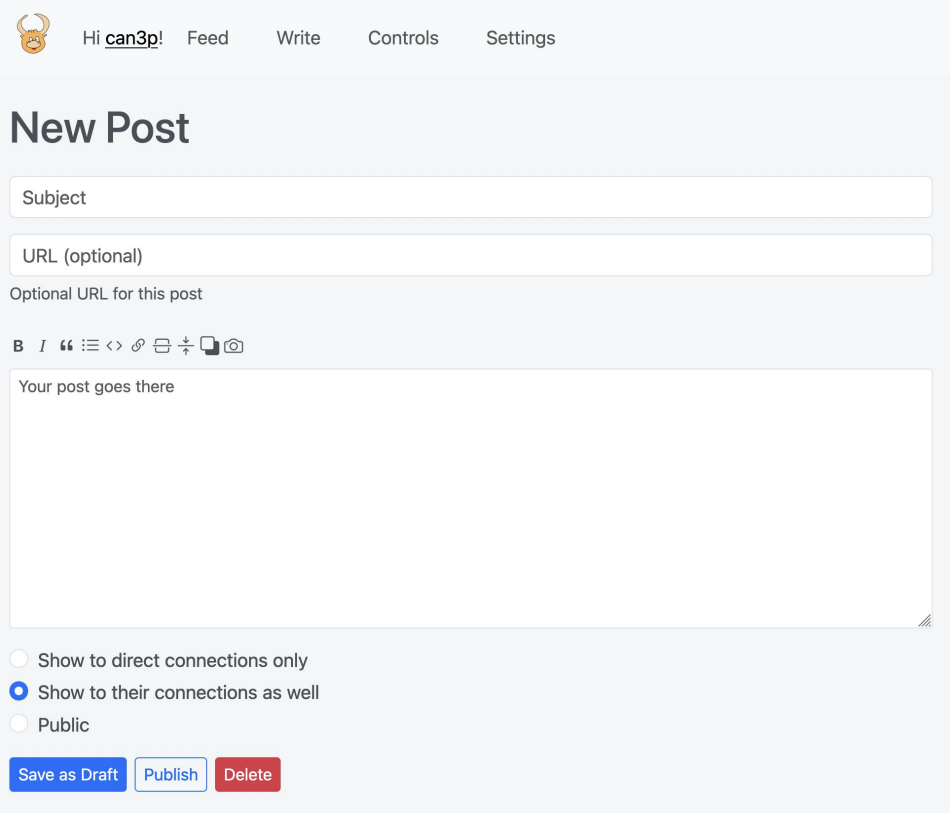
User bob for a connection with you and the following users have signed for him

- karl (Yeah, I know Bob, he's cool)

Accept Reject

Github style editor / comments

- Markdown
- Images
- Drafts
- Preview
- Spoilers
- Cuts
- Gallery



Public posts

Link posts

Mobile friendly

FAST

API

Share private posts

Custom styles

Threaded comments

RSS feeds

ididnteatthatcow.com

That’s a lot of

Forms

Emails

Permissions

Interactions

Code

Rails

Requirements

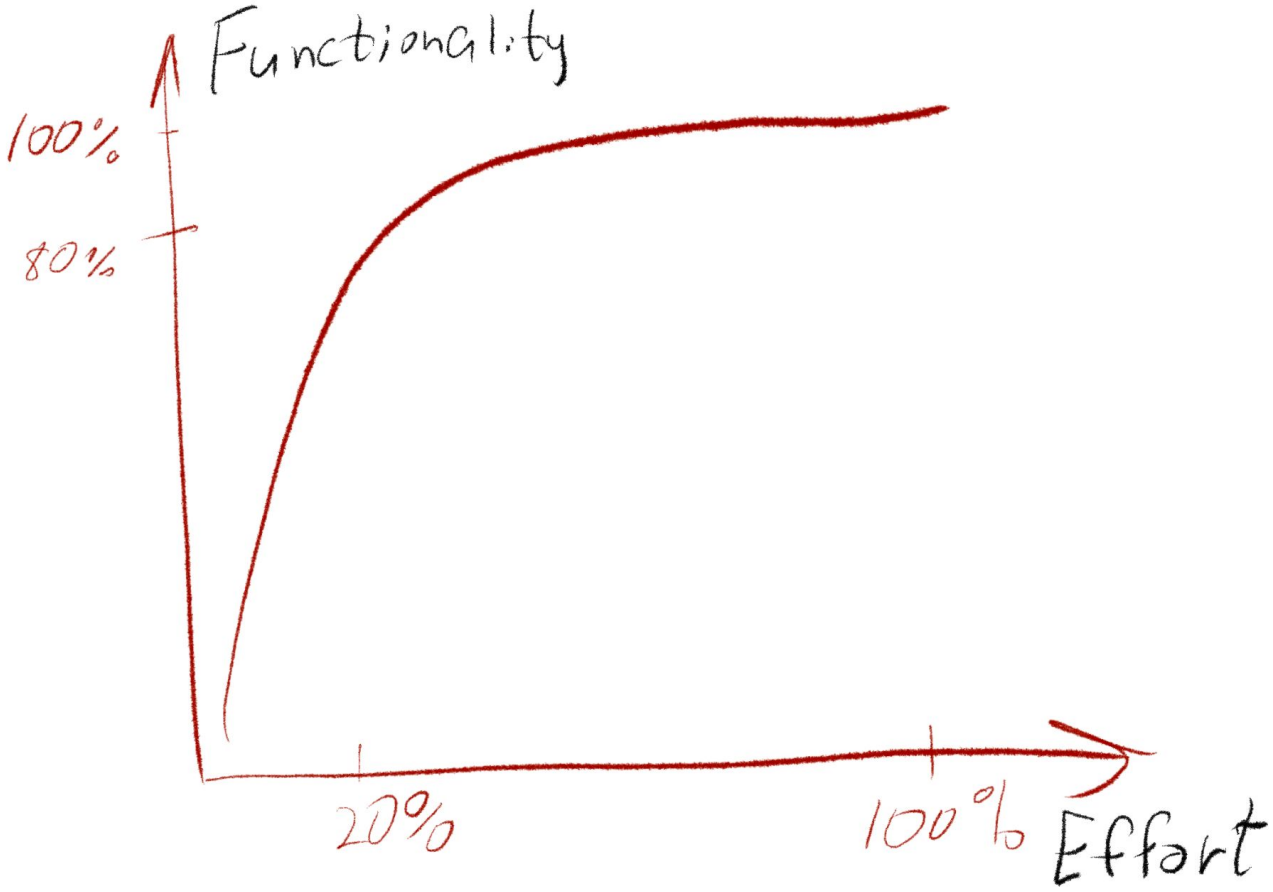
Cheap infra

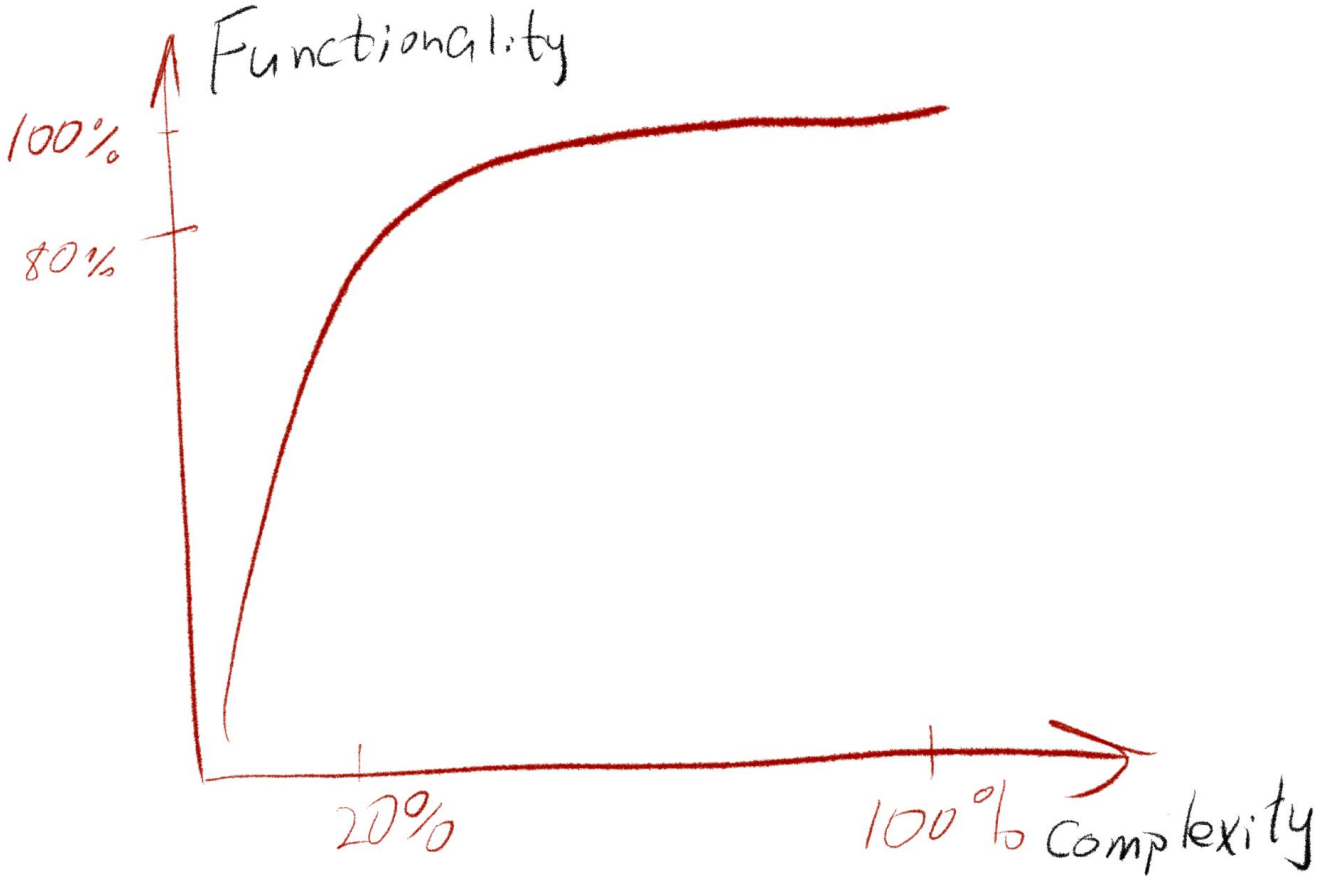
Fast iterations

Streamlined ops

Minimum deps

Reusable components


Functionality
100%
80%
20%
100%
Effort


Functionality
100%
80%
20%
100%
Complexity

DB:	Postgres
BE:	Golang
ORM:	sqlboiler, sqlmigrate
Web:	gin
FE:	HTMX, stimulus.js

Hosting:	Fly.io (ams cluster)
Logs, metrics:	Fly.io
RDS:	Fly.io (bigger machine) (any would do)
CDN:	Bunny.net (any would do)
S3:	Cloudflare R2 (any would do)
Emails:	Mailjet (any would do*)

Deployment

- Local development
- Releases
- Secrets

- One Dockerfile
- App is configured with env vars
- Postgres is the only local dependency
- One Makefile
  - Tunnel
  - Shell
  - Lint
  - Test
- run.sh, env.pl, generate.sh, sqlmigrate.sh
- fly secrets set SENDER_ADDRESS=info@coolwebsite.com / .env file

fly deploy

```
$ make tunnel
$ ./env.sh
$ ./sqlmigrate.sh up
```

Code bits

Fundamentals: controlled

Last mile code: messy

Routing

```go
func funcmap() template.FuncMap {
    return template.FuncMap{
        "link": Link,

        "abslink": AbsLink,
    }
}
```

```go
func DefaultAuthorizedHome() string {
    return Link("feed")
}

func Link(name string, args ...string) string {
    builder := links.NewArgBuilder(args...)

    var out string
    var fragment string

    switch name {
    case "default_authorized_home":
        out = DefaultAuthorizedHome()
    case "terms_of_service":
        out = "/articles/terms_of_service"
    case "post":
        out = "/posts/" + builder.Shift()
    }

    l := out + builder.BuildQueryString()

    if fragment != "" {
        l = l + "#" + fragment
    }

    return l
}

func AbsLink(name string, args ...string) string {
    return util.SiteRoot() + Link(name, args...)
}
```

Pages

- Shared fields (title)
- Everything else is unique

```go
func routes(r *gin.RouterGroup) {
    controls := r.Group("/controls", auth.EnforceAuth)
    actions := controls.Group("/action", csrf.CheckCSRF)
    controlsForms := controls.Group("/form", csrf.CheckCSRF)

    controls.GET("/settings", func(c *gin.Context) {
        userData := auth.GetUserData(c)

        ginhelpers.HTML(c, "settings.html", web.Settings(c, db, &userData))
    })
}
```

```go
var ErrNotFound = errors.Errorf("not found")
var ErrNeedsLogin = errors.Errorf("needs login")
var ErrForbidden = errors.Errorf("forbidden")
var ErrBadRequest = errors.Errorf("invalid input")

func HTML[T any](c *gin.Context, templateName string, result mo.Result[T]) {
>---if result.IsOk() {
>---->---c.HTML(http.StatusOK, templateName, result.MustGet())
>---->---return
>---}

>---var httpCode int = http.StatusInternalServerError

>---switch result.Error() {
>---case ErrNotFound:
>---->---httpCode = http.StatusNotFound
>---case ErrForbidden:
>---->---httpCode = http.StatusForbidden
>---case ErrBadRequest:
>---->---httpCode = http.StatusBadRequest
>---case ErrNeedsLogin:
>---->---auth.RedirectToLogin(c)
>---->---c.Abort()
>---->---return
>---}

>---if util.InCluster() {
>---->---c.Status(httpCode)
>---->---return
>---}

>---c.String(httpCode, result.Error().Error())
}
```

```go
type SettingsPage struct {
    *BasePage
    AvailableInvites int64
    UsedInvites      core.UserInvitationSlice
    ActiveAPIKey     *core.UserAPIKey
    GeneralSettings  *forms.SettingsGeneralForm
    UserStyles       *forms.SettingsUserStyles
    Feeds            []*feedops.RssFeed
}
```

```go
type BasePage struct {
    ProjectName string
    Name        string
    User        *auth.UserData
    StyleNonce  *string
    ScriptNonce *string
    RSSFeed     string
}
```

```go
func Settings(c *gin.Context, db boil.ContextExecutor, userData *auth.UserData) mo.Result[*SettingsPage] {
    totalInvites, err := core.UserInvitations(
        core.UserInvitationWhere.UserID.EQ(userData.DBUser.ID),
    ).Count(c, db)

    if err != nil {
        return mo.Err[*SettingsPage](err)
    }
```

Forms

- Validation
- Different actions on success

```
controlsForms.POST("/add_user_feed", func(c *gin.Context) {
    userData := auth.GetUserData(c)
    dbUser := userData.DBUser

    form := forms.NewAddFeedForm(dbUser)

    gogoForms.DefaultHandler(c, db, form)
})
```

```go
type AddFeedFormInput struct {
    URL string `form:"url"`
}

type AddFeedForm struct {
    *forms.FormBase[AddFeedFormInput]
    User *core.User
}

func NewAddFeedForm(u *core.User) *AddFeedForm {
    return &AddFeedForm{
        FormBase: &forms.FormBase[AddFeedFormInput]{
            Name:         "add_rss_feed",
            FormTemplate: "form--settings-feeds.html",
            Input:        &AddFeedFormInput{},
        },
        User: u,
    }
}
```

```go
func (f *AddFeedForm) Validate(c *gin.Context, db boil.ContextExecutor) error {
    if err := validation.ValidateURL(f.Input.URL); err != nil {
        f.AddError("url", err.Error())
    }

    if !strings.HasPrefix(f.Input.URL, "http://") && !strings.HasPrefix(f.Input.URL, "https://") {
        f.AddError("url", "url should have http or https protocol")
    }

    return f.Errors.PassedValidation()
}

func (f *AddFeedForm) Save(c context.Context, exec boil.ContextExecutor) (forms.FormSaveAction, error) {
    url := strings.TrimSpace(f.Input.URL)

    if err := feedops.SubscribeToFeed(c, exec, f.User.ID, url); err != nil {
        return nil, err
    }

    return forms.FormSaveFullReload, nil
}
```

```go
type FormSaveAction func(*gin.Context, Form)

func ReplaceHistory(action forms.FormSaveAction, url string) forms.FormSaveAction {
    return func(c *gin.Context, f forms.Form) {
        c.Header("HX-Replace-Url", url)
        action(c, f)
    }
}
```

Retarget
Trigger
NoContent
SuccessBadge

FormSaveDefault
FormSaveFullReload
FormSaveRedirect

```
<div class="card">
  <h5 class="card-header">General</h5>
  <div class="card-body">
    {{ template "form--settings-general.html" .GeneralSettings.TemplateData }}
  </div>
</div>
```

```
{{ if .FormSaved }}
  {{ template "partial--success-message.html" toMap "Message" "Settings have been saved" }}
{{ end }}

<form method="POST"
  action="{{ link "form_save_settings" }}"
  hx-post="{{ link "form_save_settings" }}"
  hx-swap="outerHTML"
  hx-disabled-elt="this"
  >

  <div class="mb-3">
    <label for="settingsTimeZone" class="form-label">Profile Visibility</label>
    <select name="profile_visibility"
            class="form-control {{ if (.Errors.HasError "profile_visibility") }}is-invalid{{ end }}"
            >
        {{ $selected := .User.ProfileVisibility }}
        {{ if (and .Input .Input.ProfileVisibility) }}
          {{ $selected = .Input.ProfileVisibility }}
        {{ end }}

        {{ range .ProfileVisibility }}
          <option value="{{ .Value }}" {{ if eq .Value $selected }}selected{{ end }}>{{ .Label }}</option>
        {{ end }}
    </select>
    {{ if (.Errors.HasError "profile_visibility") }}
    <div class="invalid-feedback">{{ .Errors.profile_visibility }}</div>
    {{ end }}
  </div>

  <button type="submit" class="btn btn-primary">Save Settings</button>
</form>
```

https://github.com/can3p/gogo

https://github.com/can3p/gogo-cli

HTMX

Page transitions

Forms

Stimulus.js

Actions

Everything else

```
<button type="button"
        class="btn btn-sm btn-danger"
        data-controller="action"
        data-action="action#run"
        data-action-action-value="remove_rss_subscription"
        data-action-prompt-value="Do you want to unsubcribe from {{ .URL }}?"
        data-id="{{ .ID }}"
        ><i class="bi-trash"></i></button>
```

- Page reload on success
- That’s it really
- No state needed
- Doesn’t play well with replication lag
- We can easily improve

```go
r.POST("/remove_rss_subscription", func(c *gin.Context) {
    userData := auth.GetUserData(c)

    var input struct {
        SubscriptionID string `json:"id"`
    }

    if err := c.BindJSON(&input); err != nil {
        reportError(c, fmt.Sprintf("Bad input: %s", err.Error()))
        return
    }

    if input.SubscriptionID == "" {
        reportError(c, "No subscription found")
        return
    }

    // in case feedops make more than one query at some point
    err := transact.Transact(db, func(tx *sql.Tx) error {
        return feedops.UnsubscribeFromFeed(c, tx, userData.DBUser.ID, input.SubscriptionID)
    })

    if err != nil {
        reportError(c, fmt.Sprintf("Operation Failed: %s", err.Error()))
        return
    }

    reportSuccess(c)
})
```

That’s it!

But I can talk about

- Markdown processing
- Command line client
- Media server
- Templates
- Template helpers
- Code structure
- Emails

https://github.com/can3p/pcom

gophers #pcom

Emails

```go
var sender sender.Sender
var mediaStorage server.MediaStorage

if shouldUseRealSender {
    sender = mailjet.NewSender()
} else {
    sender = console.NewSender()
}

dbSender := dbsender.NewSender(db, sender)
sender = dbSender

go dbSender.RunPoller(ctx)
```

```go
type Sender interface {
    Send(
        ctx context.Context,
        exec boil.ContextExecutor,
        uniqueID string,
        emailType string,
        mail *Mail) error
}

type Mail struct {
    From    mail.Address
    To      []mail.Address
    Cc      []mail.Address
    Bcc     []mail.Address
    Subject string
    Text    string
    Html    string
}
```

User images

```
if shouldUseS3 {
    var err error
    mediaStorage, err = s3.NewS3Server()

    if err != nil {
        panic(err)
    }
} else {
    var err error
    mediaStorage, err = local.NewLocalServer("user_media")

    if err != nil {
        panic(err)
    }
}

mediaServer := server.New(mediaStorage,
    server.WithClass("thumb", server.ClassParams{Width: 720}),
    server.WithPermaCache(util.InCluster()),
)
```