What I learnt building a web app with Go and HTMX

Introduction

- Haseeb Majid
  - Backend Software Engineer at Nala
- Loves cats 🐱
- Avid cricketer 🏏 ~~#BazBall~~
- 🌱 Amateur Gardener

Who Is This For?



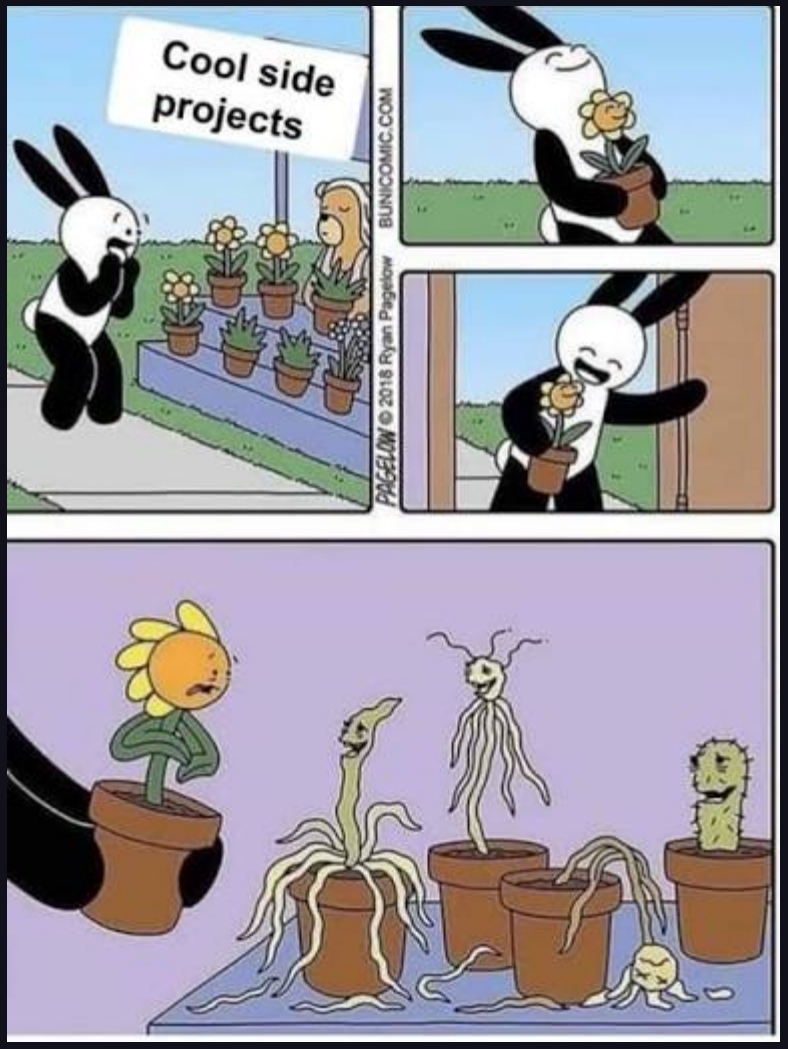
Banter Bus



Enter a nickname and choose an avatar

GracefulBlushOrangutan

PLAY


Cool side projects
BUNICOMIC.COM
PAGELOW © 2018 Ryan Pagelow


English (GB)
Frontend Trace ID: Loading...
Click to copy
Banter Bus
Room Code
5GOKT
HostPlayer
NOT READY
OtherPlayer0
NOT READY
Ready

Banter Bus got me here, LettuceGo made the same lessons easier to show.

Svelte

```svelte
<input bind:value={q} placeholder="Search plants" />
<select bind:value={location}>
  <option value="">All locations</option>
  <option value="indoors">Indoors</option>
  <option value="outdoors">Outdoors</option>
</select>
```

Svelte

```svelte
<input bind:value={q} placeholder="Search plants" />
<select bind:value={location}>
  <option value="">All locations</option>
  <option value="indoors">Indoors</option>
  <option value="outdoors">Outdoors</option>
</select>
```

```svelte
<script lang="ts">
  let q = ''
  let location = ''
  let plants = []

  $: params = new URLSearchParams({ q, location })
</script>
```

```svelte
<script lang="ts">
  let q = ''
  let location = ''
  let plants = []

  $: params = new URLSearchParams({ q, location })
</script>
```

```
<script lang="ts">
  $: if (q || location) {
    loadPlants()
    history.replaceState({}, '', `?${params}`)
  }

  async function loadPlants() {
    const res = await fetch(`/plants?${params}`)
    plants = await res.json()
  }
</script>
```

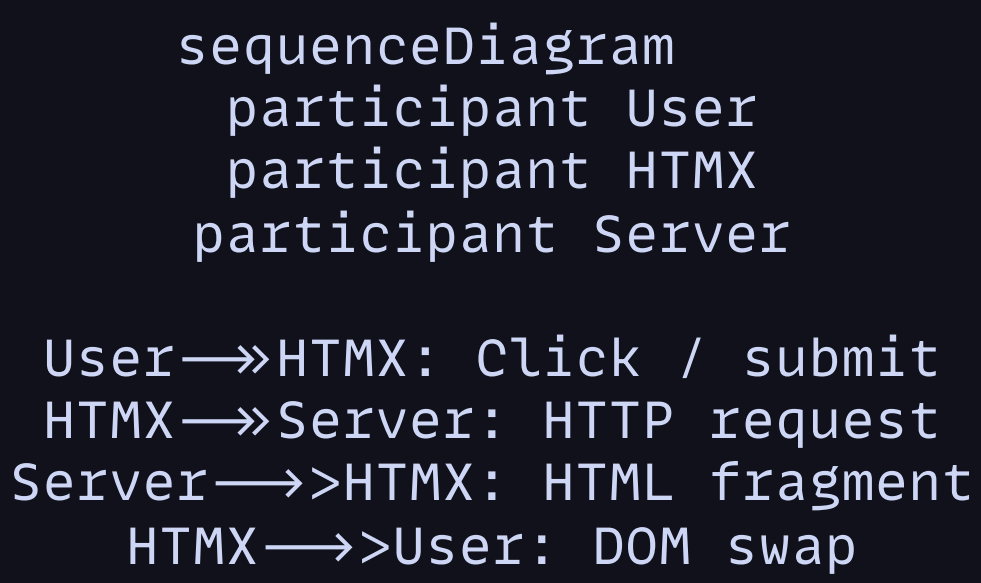
```svelte
<script lang="ts">
  $: if (q || location) {
    loadPlants()
    history.replaceState({}, '', `?${params}`)
  }

  async function loadPlants() {
    const res = await fetch(`/plants?${params}`)
    plants = await res.json()
  }
</script>
```

```
> 2 + 2
<- 4
> "2" + "2"
<- "22"
> 2 + 2 - 2
<- 2
> "2" + "2" - "2"
<- 20
```



What is HTMX?


sequenceDiagram
participant User
participant HTMX
participant Server
User-->>HTMX: Click / submit
HTMX-->>Server: HTTP request
Server-->>HTMX: HTML fragment
HTMX-->>User: DOM swap

```
<form hx-post="/plants/events"
      hx-target="#timeline"
>
   <div class="flex flex-col gap-4">
       <input name="event_type"
              type="radio"
              value="repotted"
              checked />
       <textarea name="notes" rows="3"></textarea>

       <button type="submit">Add Event</button>
   </div>
</form>
```

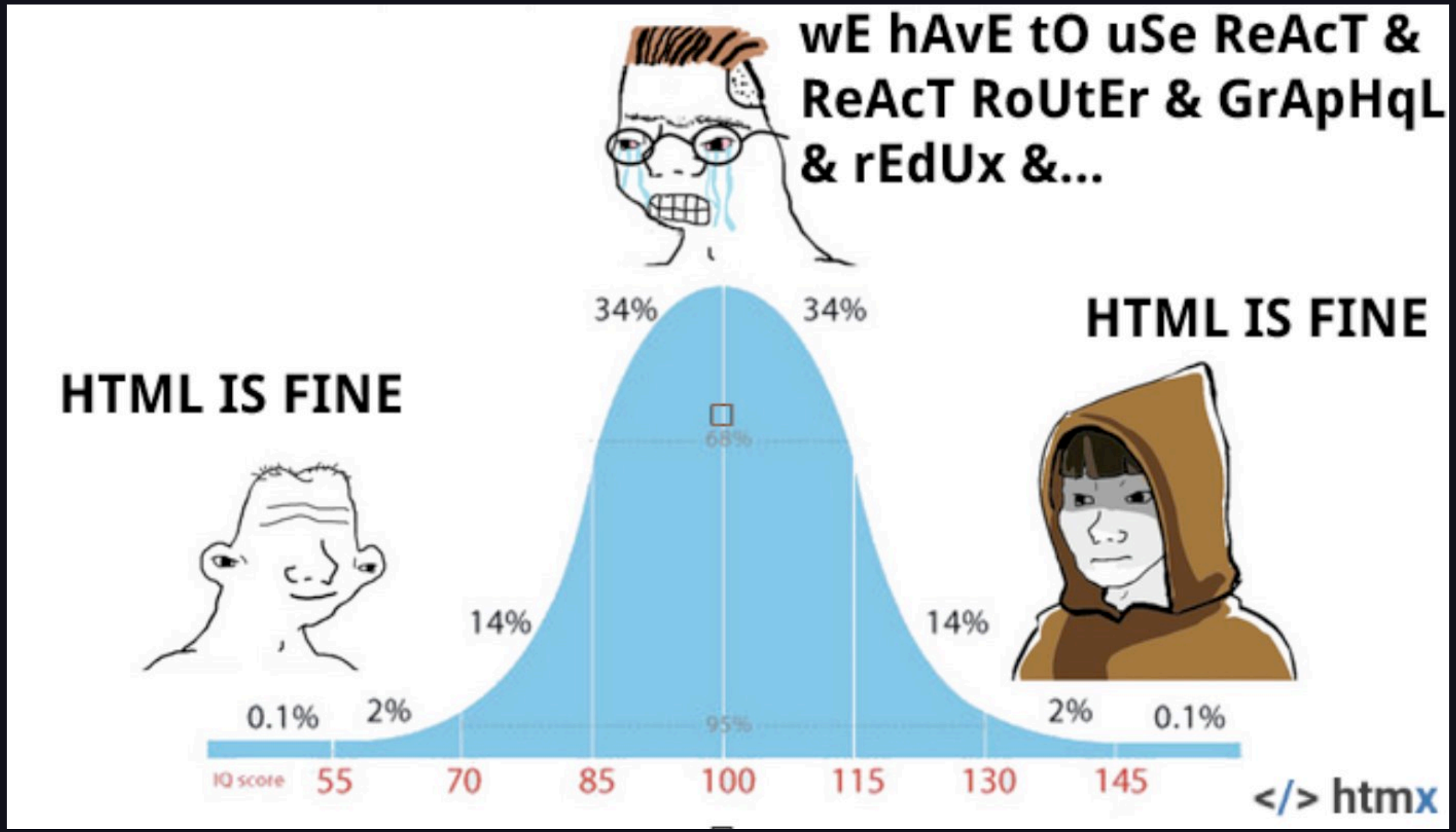
```
<form hx-post="/plants/events"
      hx-target="#timeline"
>
   <div class="flex flex-col gap-4">
       <input name="event_type"
              type="radio"
              value="repotted"
              checked />
       <textarea name="notes" rows="3"></textarea>

       <button type="submit">Add Event</button>
   </div>
</form>
```

```
<form hx-post="/plants/events"
      hx-target="#timeline"
>
   <div class="flex flex-col gap-4">
       <input name="event_type"
              type="radio"
              value="repotted"
              checked />
       <textarea name="notes" rows="3"></textarea>

       <button type="submit">Add Event</button>
   </div>
</form>
```

Why HTMX?


wE hAvE tO uSe ReAcT &
ReAcT RoUtEr & GrApHqL
& rEdUx &...
HTML IS FINE
HTML IS FINE
34%
34%
68%
14%
14%
0.1%
2%
95%
2%
0.1%
IQ score
55
70
85
100
115
130
145
</> htmx

```
<input
    name="q"
    hx-get="/plants"
    hx-trigger="input changed delay:300ms"
    hx-target="#plants-list"
    hx-push-url="true"
/>
```

```
<input
    name="q"
    hx-get="/plants"
    hx-trigger="input changed delay:300ms"
    hx-target="#plants-list"
    hx-push-url="true"
/>
```

```
<input
    name="q"
    hx-get="/plants"
    hx-trigger="input changed delay:300ms"
    hx-target="#plants-list"
    hx-push-url="true"
/>
```

```
<input
    name="q"
    hx-get="/plants"
    hx-trigger="input changed delay:300ms"
    hx-target="#plants-list"
    hx-push-url="true"
/>
```

```
<div
    hx-get="/plants/plant_1_id/timeline"
    hx-trigger="load"
>
    <p class="text-sm text-muted">Loading...</p>
</div>
```

```
<div
    hx-get="/plants/plant_1_id/timeline"
    hx-trigger="load"
>
    <p class="text-sm text-muted">Loading...</p>
</div>
```

```
<div
    hx-get="/plants/plant_1_id/timeline"
    hx-trigger="load"
>
    <p class="text-sm text-muted">Loading...</p>
</div>
```

```
response.SuccessWithToast(
    w, r, "Health recorded!",
)

_ = healthviews.HealthDotOOB(
    plantID, entry.Rating,
).Render(r.Context(), w)

timeline, _ := h.timeline.GetPlantTimeline(
    r.Context(), plantID,
    defaultHealthTimelineLimit, 0,
)
_ = plants.TimelineOOB(
```

```
response.SuccessWithToast(
    w, r, "Health recorded!",
)

_ = healthviews.HealthDotOOB(
    plantID, entry.Rating,
).Render(r.Context(), w)

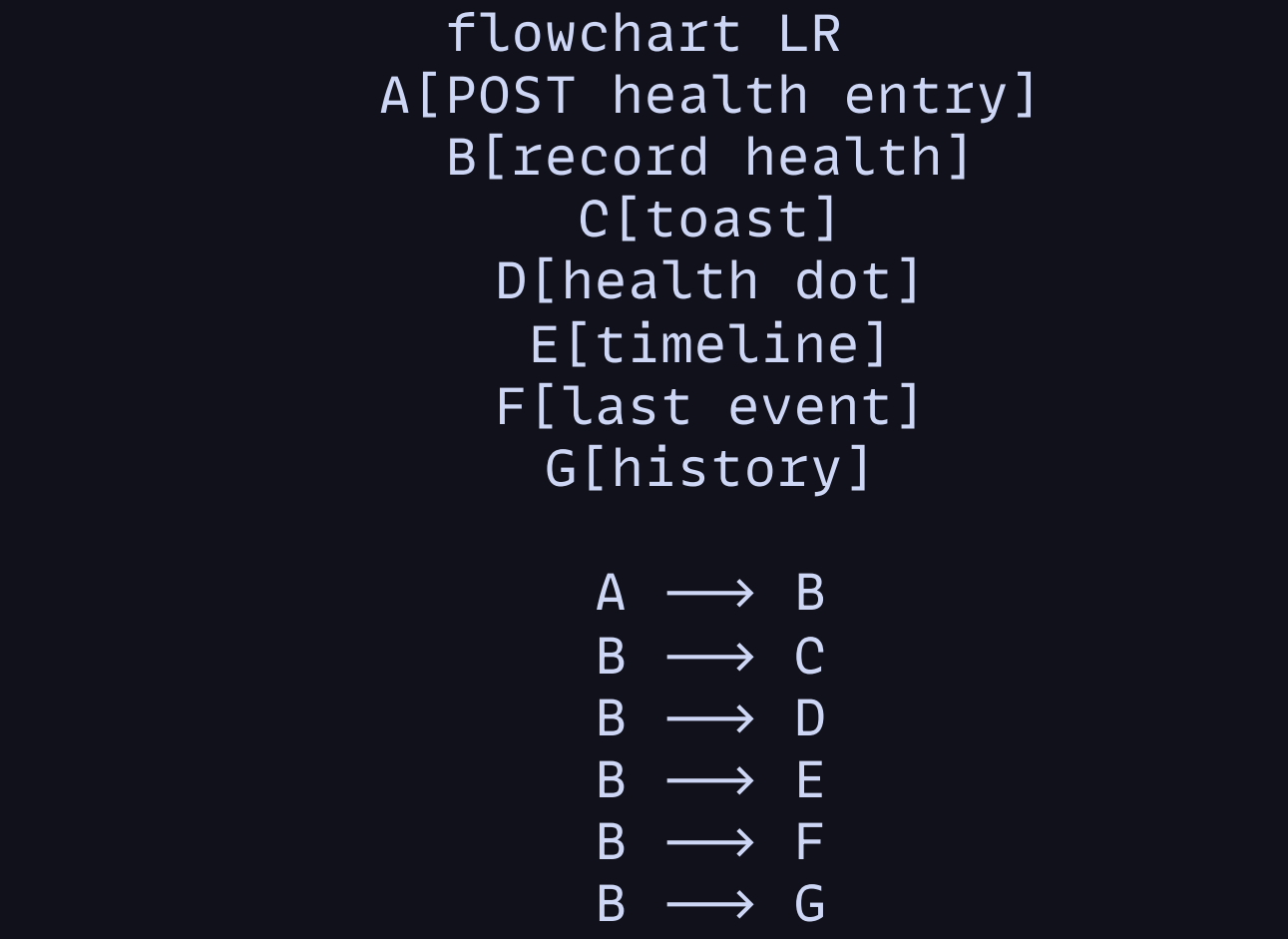
timeline, _ := h.timeline.GetPlantTimeline(
    r.Context(), plantID,
    defaultHealthTimelineLimit, 0,
)
_ = plants.TimelineOOB(
```

```
 4
 5 _ = healthviews.HealthDotOOB(
 6     plantID, entry.Rating,
 7 ).Render(r.Context(), w)
 8
 9 timeline, _ := h.timeline.GetPlantTimeline(
10     r.Context(), plantID,
11     defaultHealthTimelineLimit, 0,
12 )
13 _ = plants.TimelineOOB(
14     plantID, timeline,
15 ).Render(r.Context(), w)
16
17
```

```
    plantID, entry.Rating,
).Render(r.Context(), w)

timeline, _ := h.timeline.GetPlantTimeline(
    r.Context(), plantID,
    defaultHealthTimelineLimit, 0,
)
_ = plants.TimelineOOB(
    plantID, timeline,
).Render(r.Context(), w)


// ...
```


flowchart LR
A[POST health entry]
B[record health]
C[toast]
D[health dot]
E[timeline]
F[last event]
G[history]
A ⟶ B
B ⟶ C
B ⟶ D
B ⟶ E
B ⟶ F
B ⟶ G

```go
func ErrorWithToast(
    w http.ResponseWriter,
    r *http.Request,
    statusCode int,
    message string,
) {
    w.Header().Set("HX-Retarget", "#toast-container")
    w.Header().Set("HX-Reswap", "outerHTML")
    w.WriteHeader(statusCode)
    _ = components.Toast(
        message,
        components.ToastError,
    ).Render(r.Context(), w)
```

```
    w http.ResponseWriter,
    r *http.Request,
    statusCode int,
    message string,
) {
    w.Header().Set("HX-Retarget", "#toast-container")
    w.Header().Set("HX-Reswap", "outerHTML")
    w.WriteHeader(statusCode)
    _ = components.Toast(
        message,
        components.ToastError,
    ).Render(r.Context(), w)
}
```

```
    w http.ResponseWriter,
    r *http.Request,
    statusCode int,
    message string,
) {
    w.Header().Set("HX-Retarget", "#toast-container")
    w.Header().Set("HX-Reswap", "outerHTML")
    w.WriteHeader(statusCode)
    _ = components.Toast(
        message,
        components.ToastError,
    ).Render(r.Context(), w)
}
```

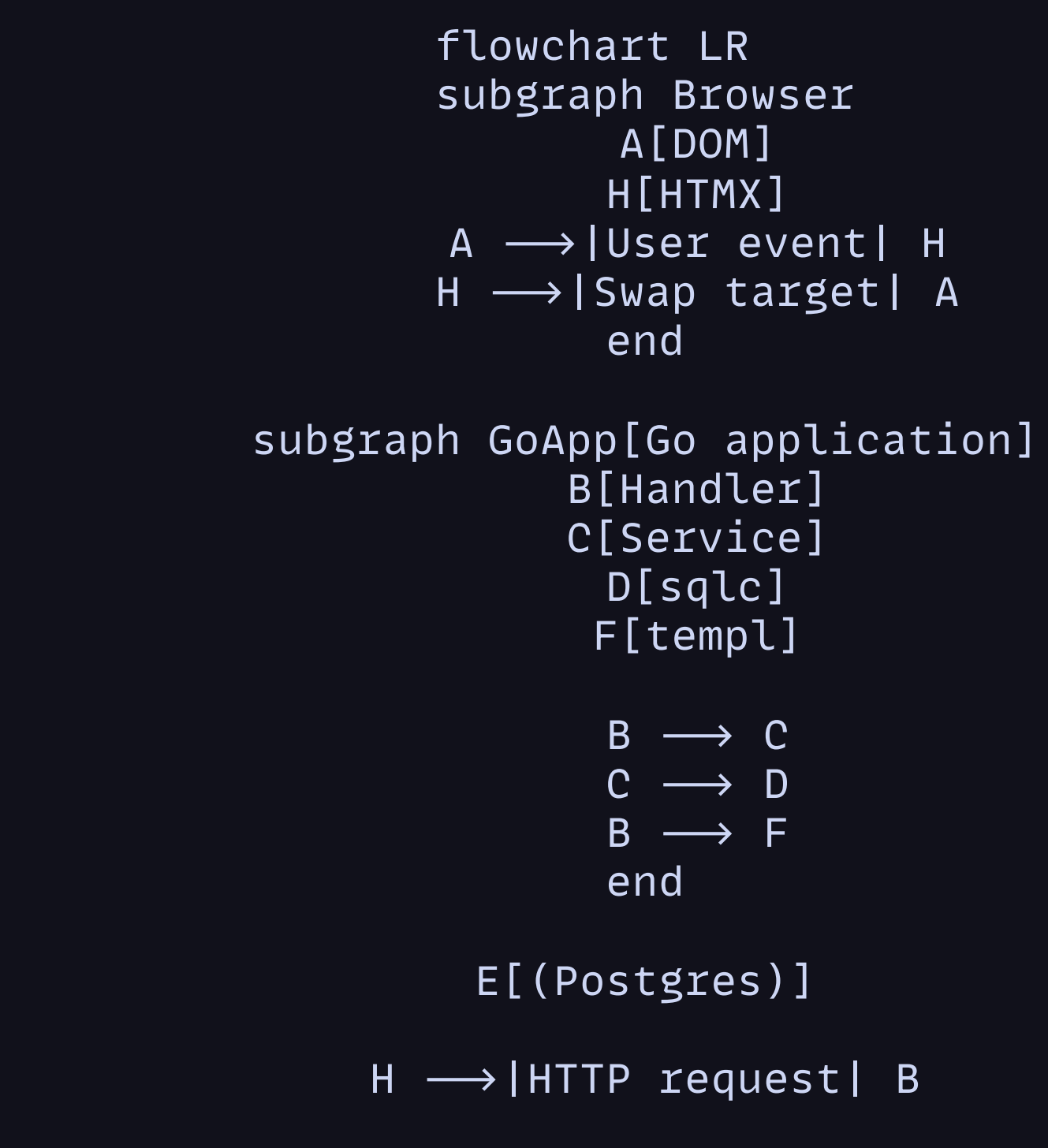
What About JSON?

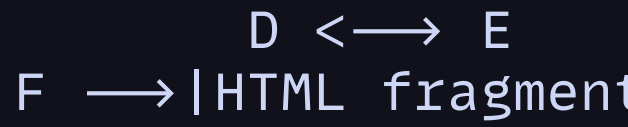
- Separate API
- Mobile app
- More glue code

When not to use HTMX?

- Lots of frontend reactivity
- Separate frontend/backend teams
- Design System

Backend


flowchart LR
subgraph Browser
A[DOM]
H[HTMX]
A ⟶|User event| H
H ⟶|Swap target| A
end
subgraph GoApp[Go application]
B[Handler]
C[Service]
D[sqlc]
F[templ]
B ⟶ C
C ⟶ D
B ⟶ F
end
E[(Postgres)]
H ⟶|HTTP request| B


D <⟶ E
F ⟶|HTML fragment| H

Handler

```
 1 func (h *EventHandler) UpdateTimeline(
 2     w http.ResponseWriter,
 3     r *http.Request,
 4 ) {
 5     // ...
 6
 7     timeline, err := h.service.UpdatePlantTimeline(
 8         r.Context(),
 9         plantID,
10     )
11     if err != nil {
12         response.ErrorWithToast(
13             w, r,
```

Handler

```
 2      w http.ResponseWriter,
 3      r *http.Request,
 4  ) {
 5      // ...
 6
 7      timeline, err := h.service.UpdatePlantTimeline(
 8          r.Context(),
 9          plantID,
10      )
11      if err != nil {
12          response.ErrorWithToast(
13              w, r,
14              http.StatusInternalServerError,
```

Handler

```
15              "failed to load timeline",
16          )
17          return
18      }
19
20      err = plantcomponents.Timeline(
21          plantID,
22          timeline,
23      ).Render(r.Context(), w)
24      if err != nil {
25          h.logger.ErrorContext(
26              r.Context(),
27              "failed to render timeline content",
28              slog.Any("error", err),
```

Lettuce Go Home Plants Graveyard Locations Logout

🪴 Repotted
· Aug 2, 2026
Bigger pot today

📷 Photo uploaded
· Jul 16, 2026



Munched
· Jul 14, 2026

📷 Photo uploaded
· Jul 14, 2026



🩺 Health check: Great
· Jun 23, 2026

📷 proper leaves now
· Jun 23, 2026



proper leaves now

📷 sprouted
· Jun 23, 2026



Templ

- HTML Templates
- LSP
- Components

```
package components

import ".../service"

templ HealthHistory(entries []service.HealthEntry) {
    for _, entry := range entries {
        <div class="flex gap-3 items-center py-2">
            @Health(entry.Rating)
            <span class="text-sm font-medium text-text
                { entry.Rating }
            </span>
        </div>
    }
```

```
1  package components
2
3  import ".../service"
4
5  templ HealthHistory(entries []service.HealthEntry) {
6      for _, entry := range entries {
7          <div class="flex gap-3 items-center py-2">
8              @Health(entry.Rating)
9              <span class="text-sm font-medium text-text
10                 { entry.Rating }
11             </span>
12         </div>
13     }
```

```
package components

import ".../service"

templ HealthHistory(entries []service.HealthEntry) {
    for _, entry := range entries {
        <div class="flex gap-3 items-center py-2">
            @Health(entry.Rating)
            <span class="text-sm font-medium text-text
                { entry.Rating }
            </span>
        </div>
    }
```

```
package components

import ".../service"

templ HealthHistory(entries []service.HealthEntry) {
    for _, entry := range entries {
        <div class="flex gap-3 items-center py-2">
            @Health(entry.Rating)
            <span class="text-sm font-medium text-text
                { entry.Rating }
            </span>
        </div>
    }
```

```
 2
 3 import ".../service"
 4
 5 templ HealthHistory(entries []service.HealthEntry) {
 6     for _, entry := range entries {
 7         <div class="flex gap-3 items-center py-2">
 8             @Health(entry.Rating)
 9             <span class="text-sm font-medium text-text
10                 { entry.Rating }
11             </span>
12         </div>
13     }
14 }
```

```
 2
 3 import ".../service"
 4
 5 templ HealthHistory(entries []service.HealthEntry) {
 6     for _, entry := range entries {
 7         <div class="flex gap-3 items-center py-2">
 8             @Health(entry.Rating)
 9             <span class="text-sm font-medium text-text
10                 { entry.Rating }
11             </span>
12         </div>
13     }
14 }
```

```
<div class="flex gap-3 items-center py-2">
    <span class="inline-block w-3 h-3 rounded-full bg-hea
    <span class="text-sm font-medium text-text">
        3
    </span>
</div>
```

```
package layouts

import "gitlab.com/.../http/views/components"

templ Base(title string, environment string) {
  <!DOCTYPE html>
  <html lang="en">
    <head>
    </head>
    <body class="bg-base-200 text-neutral">
      { children... }
    </body>
  </html>
```

```
 2
 3 import "gitlab.com/.../http/views/components"
 4
 5 templ Base(title string, environment string) {
 6   <!DOCTYPE html>
 7   <html lang="en">
 8     <head>
 9     </head>
10     <body class="bg-base-200 text-neutral">
11       { children... }
12     </body>
13   </html>
14 }
```

```
templ PlantPage(title string, environment string) {
  @Base(title, environment) {
        <div class="drawer lg:drawer-open">
          // ...
        </div>
    }
}
```

```
templ PlantPage(title string, environment string) {
  @Base(title, environment) {
        <div class="drawer lg:drawer-open">
          // ...
        </div>
    }
}
```

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Basil</title>
  </head>
  <body class="bg-base-200 text-neutral">
    <div class="drawer lg:drawer-open">
      <!-- page content -->
    </div>
  </body>
</html>
```

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <title>Basil</title>
  </head>
  <body class="bg-base-200 text-neutral">
    <div class="drawer lg:drawer-open">
      <!-- page content -->
    </div>
  </body>
</html>
```

Postgres

SQLC

```sql
-- name: ListHealthByPlant :many
SELECT id, plant_id, rating, recorded_at
FROM health_entries
WHERE plant_id = $1
ORDER BY recorded_at DESC
LIMIT $2 OFFSET $3;
```

SQLC

```sql
-- name: ListHealthByPlant :many
SELECT id, plant_id, rating, recorded_at
FROM health_entries
WHERE plant_id = $1
ORDER BY recorded_at DESC
LIMIT $2 OFFSET $3;
```

```go
type ListHealthByPlantParams struct {
    PlantID uuid.UUID `json:"plant_id"`
    Limit   int32     `json:"limit"`
    Offset  int32     `json:"offset"`
}

func (q *Queries) ListHealthByPlant(
    ctx context.Context,
    arg ListHealthByPlantParams,
) ([]HealthEntry, error) {
    rows, err := q.db.Query(
        ctx, listHealthByPlant,
        arg.PlantID, arg.Limit, arg.Offset,
```

```
    PlantID uuid.UUID `json:"plant_id"`
    Limit   int32     `json:"limit"`
    Offset  int32     `json:"offset"`
}

func (q *Queries) ListHealthByPlant(
    ctx context.Context,
    arg ListHealthByPlantParams,
) ([]HealthEntry, error) {
    rows, err := q.db.Query(
        ctx, listHealthByPlant,
        arg.PlantID, arg.Limit, arg.Offset,
    )
```

```
 8      ctx context.Context,
 9      arg ListHealthByPlantParams,
10  ) ([]HealthEntry, error) {
11      rows, err := q.db.Query(
12          ctx, listHealthByPlant,
13          arg.PlantID, arg.Limit, arg.Offset,
14      )
15      if err != nil {
16          return nil, err
17      }
18      defer rows.Close()
19
20      var items []HealthEntry
```

```
18      defer rows.Close()
19
20      var items []HealthEntry
21      for rows.Next() {
22          var i HealthEntry
23          if err := rows.Scan(
24              &i.ID, &i.PlantID, &i.UserID,
25              &i.Rating, &i.Notes,
26              &i.RecordedAt, &i.CreatedAt,
27              &i.UpdatedAt,
28          ); err != nil {
29              return nil, err
30          }
```

DevEx

Taskfile

```yaml
version: "3"

tasks:
  dev:
    cmds:
      - docker compose up -d
      - concurrently "air" "task watch"

  watch:
    cmds:
      - templ generate -watch &
      - tailwindcss -i ./static/css/tailwind.css \
          -o ./static/css/styles.css --minify --watch
```

Taskfile

```yaml
version: "3"

tasks:
  dev:
    cmds:
      - docker compose up -d
      - concurrently "air" "task watch"

  watch:
    cmds:
      - templ generate -watch &
      - tailwindcss -i ./static/css/tailwind.css \
          -o ./static/css/styles.css --minify --watch
```

Taskfile

```yaml
version: "3"

tasks:
  dev:
    cmds:
      - docker compose up -d
      - concurrently "air" "task watch"

  watch:
    cmds:
      - templ generate -watch &
      - tailwindcss -i ./static/css/tailwind.css \
          -o ./static/css/styles.css --minify --watch
```

Live Reload

```
import (
    "github.com/aarol/reload"
)

func (s *Server) wrapWithReload(handler http.Handler) h
    reloader := reload.New(
        "internal/transport/http/views",
        "static/css",
    )
    s.logger.Debug("starting reload server")
    return reloader.Handle(handler)
}
```

Live Reload

```
import (
    "github.com/aarol/reload"
)

func (s *Server) wrapWithReload(handler http.Handler) h
    reloader := reload.New(
        "internal/transport/http/views",
        "static/css",
    )
    s.logger.Debug("starting reload server")
    return reloader.Handle(handler)
}
```

Live Reload

```
import (
    "github.com/aarol/reload"
)

func (s *Server) wrapWithReload(handler http.Handler) h
    reloader := reload.New(
        "internal/transport/http/views",
        "static/css",
    )
    s.logger.Debug("starting reload server")
    return reloader.Handle(handler)
}
```

Live Reload

```
import (
    "github.com/aarol/reload"
)

func (s *Server) wrapWithReload(handler http.Handler) h
    reloader := reload.New(
        "internal/transport/http/views",
        "static/css",
    )
    s.logger.Debug("starting reload server")
    return reloader.Handle(handler)
}
```

flake.nix

```
myPackages = with pkgs; [
  go
  goose
  air
  golangci-lint
  gofumpt
  go-task
  playwright-driver
  playwright-cli
  templ
  sqlc
  tailwindcss_4
];
```

flake.nix

```
myPackages = with pkgs; [
  go
  goose
  air
  golangci-lint
  gofumpt
  go-task
  playwright-driver
  playwright-cli
  templ
  sqlc
  tailwindcss_4
];
```

flake.nix

```
 5    golangci-lint
 6    gofumpt
 7    go-task
 8    playwright-driver
 9    playwright-cli
10    templ
11    sqlc
12    tailwindcss_4
13  ];
14
15  devShellPackages =
16    with pkgs;
17    myPackages;
```

flake.nix

```
 5    golangci-lint
 6    gofumpt
 7    go-task
 8    playwright-driver
 9    playwright-cli
10    templ
11    sqlc
12    tailwindcss_4
13  ];
14
15  devShellPackages =
16    with pkgs;
17    myPackages;
```

Nix

```
example on main via 🐹 v1.22.8
› which golangci-lint

example on main via 🐹 v1.22.8
› nix develop

example on main via 🐹 v1.22.8 ❄️ impure (nix-shell-env)
› which golangci-lint
/nix/store/kcd...golangci-lint-1.56.2/bin/golangci-lint
```

Nix

```
example on main via 🐹 v1.22.8
› which golangci-lint

example on main via 🐹 v1.22.8
› nix develop

example on main via 🐹 v1.22.8 ❄️ impure (nix-shell-env)
› which golangci-lint
/nix/store/kcd...golangci-lint-1.56.2/bin/golangci-lint
```

Nix

```
example on main via 🐹 v1.22.8
› which golangci-lint

example on main via 🐹 v1.22.8
› nix develop

example on main via 🐹 v1.22.8 ❄️ impure (nix-shell-env)
› which golangci-lint
/nix/store/kcd...golangci-lint-1.56.2/bin/golangci-lint
```

Playwright

```
func TestTimelineRefreshesAfterAddEvent(t *testing.T)
    // setup ...

    page.Locator(
        "a:has-text('Timeline Refresh Plant')",
    ).First().Click()

    timelineSection := page.Locator(
        "#timeline-section",
    )
    expect.Locator(timelineSection).ToContainText(
        "Repotted",
    )
```

Playwright

```
func TestTimelineRefreshesAfterAddEvent(t *testing.T)
    // setup ...

    page.Locator(
        "a:has-text('Timeline Refresh Plant')",
    ).First().Click()

    timelineSection := page.Locator(
        "#timeline-section",
    )
    expect.Locator(timelineSection).ToContainText(
        "Repotted",
    )
```

Playwright

```
 3
 4      page.Locator(
 5          "a:has-text('Timeline Refresh Plant')",
 6      ).First().Click()
 7
 8      timelineSection := page.Locator(
 9          "#timeline-section",
10      )
11      expect.Locator(timelineSection).ToContainText(
12          "Repotted",
13      )
14
15      lastEventOOB := page.Locator("#last-event-oob")
```

Playwright

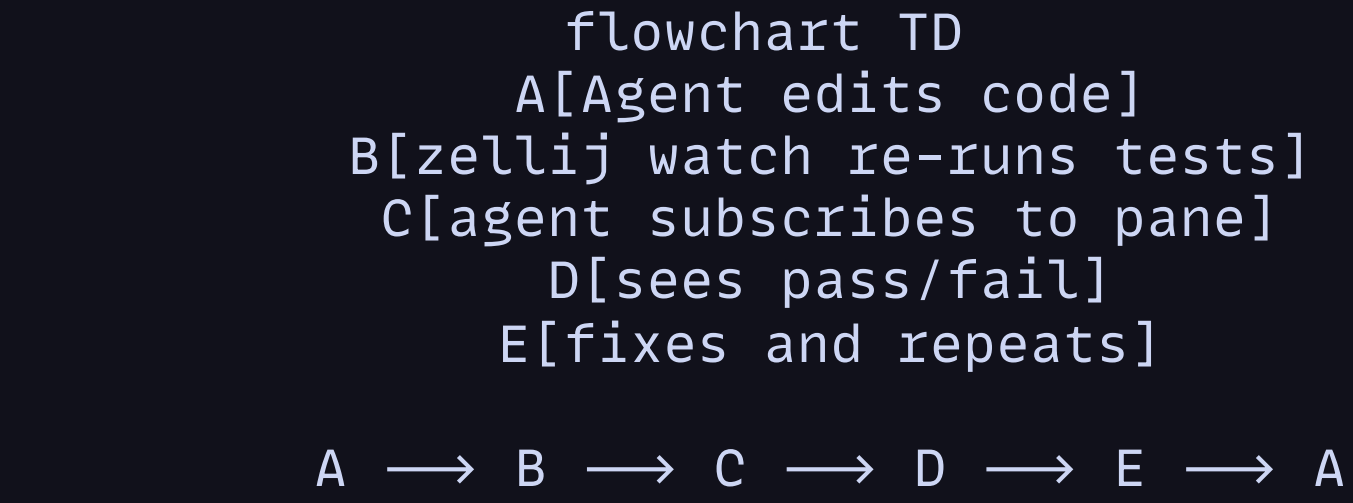
```
 5          "a:has-text('Timeline Refresh Plant')",
 6      ).First().Click()
 7
 8      timelineSection := page.Locator(
 9          "#timeline-section",
10      )
11      expect.Locator(timelineSection).ToContainText(
12          "Repotted",
13      )
14
15      lastEventOOB := page.Locator("#last-event-oob")
16      expect.Locator(lastEventOOB).ToContainText("—")
17  }
```

Playwright

```
 5          "a:has-text('Timeline Refresh Plant')",
 6      ).First().Click()
 7
 8      timelineSection := page.Locator(
 9          "#timeline-section",
10      )
11      expect.Locator(timelineSection).ToContainText(
12          "Repotted",
13      )
14
15      lastEventOOB := page.Locator("#last-event-oob")
16      expect.Locator(lastEventOOB).ToContainText("—")
17  }
```

LLM Tips

```
playwright-cli open http://localhost:9090
playwright-cli fill \
  input[name=username] "haseeb" --submit
playwright-cli snapshot
```


flowchart TD
A[Agent edits code]
B[zellij watch re-runs tests]
C[agent subscribes to pane]
D[sees pass/fail]
E[fixes and repeats]
A ⟶ B ⟶ C ⟶ D ⟶ E ⟶ A

- `zellij run -- go test ./...`
- `zellij subscribe <pane-id>`
- `zellij dump_pane <pane-id>`

Takeaways

- HTMX
  - Backend driven
- Go + Templ
- sqlc



- https://haseebmajid.dev/slides/gophercon-uk-htmx-go-web-app/

Useful Links

- Banter Bus: https://gitlab.com/hmajid2301/banterbus
- Splitting Your Data & Application APIs: https://htmx.org/essays/splitting-your-apis/
- Nix Dev Shell: https://www.youtube.com/watch?v=bdGfn_ihHOk
- Playwright: https://www.youtube.com/watch?v=XdBhYt3-bbU