Yes, you can absolutely start testing this now, and yes, you can call the repo `typing-telemetry` while the brew formula installs as `typtel`. I'll show you both.

Below is a **minimal viable telemetry layer** that:

- counts keystrokes,
- has stubbed-out "stats",
- saves to a JSON file, and
- is unit-tested.

Then I'll show how to wire it into your existing Bubble Tea app with minimal changes.

---

1. Repo & naming (GitHub vs brew)

- Your GitHub repo can be **`github.com/you/typing-telemetry`**.
- You publish releases from that repo.
- Your **Homebrew formula** can be called `typtel.rb` in your tap repo, e.g.:
  - Tap repo: `github.com/you/homebrew-tap`
  - Formula file: `Formula/typtel.rb` pointing at the `typing-telemetry` releases.

When people run:

```
brew install you/tap/typtel
```

brew doesn't care that the GitHub repo is called `typing-telemetry`; the formula name (`typtel`) is what matters.

So: yes, repo name and brew package name can differ.

---

2. Minimal telemetry package (keystrokes → JSON)

Let's create a tiny package you can drop into `internal/telemetry`.

`internal/telemetry/telemetry.go`

```go
package telemetry

import (
    "encoding/json"
    "errors"
    "os"
    "path/filepath"
    "time"
)

const dateLayout = "2006-01-02"

type DailyStats struct {
    // ISO date string, e.g. "2025-12-09"
    Date       string `json:"date"`
    Keystrokes int    `json:"keystrokes"`

    // Stub fields for later
    EstimatedWords int `json:"estimated_words"`
}
```

```go
// NewToday returns a fresh stats object for "today".
func NewToday() *DailyStats {
    return &DailyStats{
        Date:          time.Now().Format(dateLayout),
        Keystrokes:    0,
        EstimatedWords: 0, // will compute later
    }
}

// AddKeystroke increments the keystroke count. You can later update
// EstimatedWords here or in a separate method.
func (ds *DailyStats) AddKeystroke() {
    ds.Keystrokes++
}

// IsForToday reports whether these stats belong to the current local date.
func (ds *DailyStats) IsForToday() bool {
    return ds.Date == time.Now().Format(dateLayout)
}

// EnsureDir ensures that the directory for the given file path exists.
func EnsureDir(path string) error {
    dir := filepath.Dir(path)
    if dir == "." {
        return nil
    }
    return os.MkdirAll(dir, 0o755)
}

// LoadOrNew loads a DailyStats from the given JSON file, or returns
// a new object for today if the file doesn't exist or is invalid/outdated.
func LoadOrNew(path string) (*DailyStats, error) {
    data, err := os.ReadFile(path)
    if err != nil {
        if errors.Is(err, os.ErrNotExist) {
            return NewToday(), nil
        }
        return nil, err
    }

    var ds DailyStats
    if err := json.Unmarshal(data, &ds); err != nil {
        // corrupt file or incompatible schema → start fresh
        return NewToday(), nil
    }

    if !ds.IsForToday() {
        // old stats; start a new day
        return NewToday(), nil
    }
    return &ds, nil
}

// Save writes the DailyStats to the given JSON file (overwriting).
func (ds *DailyStats) Save(path string) error {
    if err := EnsureDir(path); err != nil {
```

```
        return err
    }

    tmp := path + ".tmp"
    data, err := json.MarshalIndent(ds, "", "  ")
    if err != nil {
        return err
    }

    if err := os.WriteFile(tmp, data, 0o644); err != nil {
        return err
    }

    // Atomic-ish replace
    return os.Rename(tmp, path)
}
```

---

3. Minimal tests for telemetry

Create `internal/telemetry/telemetry_test.go`:

```
package telemetry

import (
    "os"
    "path/filepath"
    "testing"
    "time"
)

func TestNewTodayHasTodayDate(t *testing.T) {
    ds := NewToday()
    want := time.Now().Format(dateLayout)
    if ds.Date != want {
        t.Fatalf("NewToday.Date = %q, want %q", ds.Date, want)
    }
}

func TestAddKeystrokeIncrementsCount(t *testing.T) {
    ds := NewToday()
    ds.AddKeystroke()
    ds.AddKeystroke()

    if ds.Keystrokes != 2 {
        t.Fatalf("Keystrokes = %d, want 2", ds.Keystrokes)
    }
}

func TestSaveAndLoadRoundTrip(t *testing.T) {
    dir := t.TempDir()
    path := filepath.Join(dir, "stats.json")

    orig := NewToday()
    orig.Keystrokes = 42
    orig.EstimatedWords = 10
```

```
    if err := orig.Save(path); err != nil {
        t.Fatalf("Save() error = %v", err)
    }

    loaded, err := LoadOrNew(path)
    if err != nil {
        t.Fatalf("LoadOrNew() error = %v", err)
    }

    if loaded.Keystrokes != orig.Keystrokes {
        t.Fatalf("Keystrokes = %d, want %d", loaded.Keystrokes, orig.Keystrokes)
    }
    if loaded.EstimatedWords != orig.EstimatedWords {
        t.Fatalf("EstimatedWords = %d, want %d", loaded.EstimatedWords, orig.EstimatedWords)
    }
}

func TestLoadOrNewWhenFileMissingReturnsNewToday(t *testing.T) {
    dir := t.TempDir()
    path := filepath.Join(dir, "missing.json")

    ds, err := LoadOrNew(path)
    if err != nil {
        t.Fatalf("LoadOrNew() error = %v", err)
    }
    if ds.Keystrokes != 0 {
        t.Fatalf("Keystrokes = %d, want 0", ds.Keystrokes)
    }
    if _, err := os.Stat(path); err == nil {
        t.Fatalf("expected no file to be created by LoadOrNew, but found one")
    }
}
```

Run tests:

```
go test ./internal/telemetry
# eventually
go test ./...
```

This gives you a **tested core** before you even touch Bubble Tea or menu bars.

---

4. Wire telemetry into your existing TUI (MVP)

Now we minimally modify your `main.go` to:

- create/load a `DailyStats`,
- increment it on each keystroke,
- save to JSON on each change (MVP; you can optimise later).

Example: adapted `main.go` (only key changes)

Add imports:

```
import (
    // ...
    "path/filepath"
```

```
    "github.com/you/typing-telemetry/internal/telemetry"
)
```

Extend your `model`:

```
type model struct {
    target []rune // the passage to type
    typed  []rune // what the user has typed

    passageName string

    startedAt  time.Time
    finishedAt time.Time
    done       bool

    stats      *telemetry.DailyStats
    statsPath  string
}
```

Update `newRandomModel` to accept stats:

```
func newRandomModel(stats *telemetry.DailyStats, statsPath string) model {
    if len(passages) == 0 {
        passages = defaultPassages()
    }
    rand.Seed(time.Now().UnixNano())
    p := passages[rand.Intn(len(passages))]

    return model{
        target:      []rune(strings.TrimSpace(p.Text)),
        passageName: p.Name,
        stats:       stats,
        statsPath:   statsPath,
    }
}
```

Add a helper to record a keystroke and save:

```
func (m *model) recordKeystroke() {
    if m.stats == nil {
        return
    }
    m.stats.AddKeystroke()
    // MVP: save on every keystroke. Later you can batch.
    _ = m.stats.Save(m.statsPath)
}
```

Modify your `Update` where you handle keypresses (inside `case tea.KeyMsg:`):

Right after you decide that a keypress should count (i.e., after ignoring non-character keys and before appending to `m.typed`), call `recordKeystroke()`.

For example:

```
case tea.KeyMsg:
    switch msg.Type {
    case tea.KeyCtrlC, tea.KeyEsc:
        return m, tea.Quit

    case tea.KeyBackspace:
```

```
        if !m.done && len(m.typed) > 0 {
            m.typed = m.typed[:len(m.typed)-1]
            // You might or might not want to count backspaces.
            // For now, we don't count them as keystrokes; if you do, call m.recordKeystroke() here.
        }
        return m, nil
    }

    if m.done {
        if msg.String() == "r" {
            // reuse same stats + path so keystrokes are cumulative
            return newRandomModel(m.stats, m.statsPath), nil
        }
    }

    // convert keypress into a rune:
    var r rune
    switch msg.Type {
    case tea.KeyEnter:
        r = '\n'
    case tea.KeySpace:
        r = ' '
    default:
        if len(msg.Runes) == 0 {
            return m, nil
        }
        r = msg.Runes[0]
    }

    // count this keystroke:
    m.recordKeystroke()

    if m.startedAt.IsZero() {
        m.startedAt = time.Now()
    }
    m.typed = append(m.typed, r)
    if len(m.typed) >= len(m.target) {
        m.done = true
        m.finishedAt = time.Now()
    }
```

Finally, in `main()` initialise stats and pass them in:

```
func main() {
    filePath := flag.String("file", "", "path to a text file to use as the passage")
    rawText := flag.String("text", "", "inline text to use as the passage (overrides -file)")
    statsFile := flag.String("stats-file", "", "path to stats JSON file (default: ~/.local/share/typtel/stats.js
    flag.Parse()

    if *statsFile == "" {
        home, err := os.UserHomeDir()
        if err != nil {
            fmt.Fprintf(os.Stderr, "could not find home dir: %v\n", err)
            os.Exit(1)
        }
        *statsFile = filepath.Join(home, ".local", "share", "typtel", "stats.json")
    }
```

```
    stats, err := telemetry.LoadOrNew(*statsFile)
    if err != nil {
        fmt.Fprintf(os.Stderr, "could not load stats: %v\n", err)
        stats = telemetry.NewToday()
    }

    // decide passages like before...
    if *rawText != "" {
        passages = []passage{
            {Name: "Custom text", Text: *rawText},
        }
    } else if *filePath != "" {
        data, err := os.ReadFile(*filePath)
        if err != nil {
            fmt.Fprintf(os.Stderr, "could not read file %q: %v\n", *filePath, err)
            passages = defaultPassages()
        } else {
            passages = []passage{
                {Name: *filePath, Text: string(data)},
            }
        }
    } else {
        passages = defaultPassages()
    }

    p := tea.NewProgram(newRandomModel(stats, *statsFile), tea.WithAltScreen())
    if err := p.Start(); err != nil {
        fmt.Println("Error running program:", err)
        os.Exit(1)
    }
}
```

Now, every time you type in the TUI:

- `Keystrokes` increments,
- `~/.local/share/typtel/stats.json` gets updated with something like:

```
{
  "date": "2025-12-09",
  "keystrokes": 1234,
  "estimated_words": 0
}
```

You can inspect that file any time, and you already have tests guarding the core behaviour.

---

5. Next steps

Once this is working:

- Add a `typtel stats` subcommand that:
  - loads the JSON,
  - prints "Keystrokes today: N (stub words: 0)".
- Later, modify `DailyStats` to compute `EstimatedWords` (e.g. `Keystrokes / 5`).
- Then build the daemon that uses the same `telemetry.DailyStats` and `Save` method.

If you'd like, next I can:

- Give you a `cmd/typtel/main.go` that uses subcommands (`practice`, `stats`) and reuses this telemetry package cleanly.