Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

Swiss Confederation

Federal Department of Finance FDF
Federal Office of Information Technology, Systems, and Telecommunication FOITT
Staff Office of the Directorate

Generate att/inte-ractive Forms dynamically in Terminal

Vladica Stojic, Chadi Taieb

Zollikofen, 13.06.2024

TOC

- Motivation
- TUI frameworks
- The ELM Arch.
- The *Bubbletea* Family
- *Terminalform* Project
- Demo
- Unit Testing in terminal applications
- Debuggigng terminal applications

Motivation

- Survey not supported anymore!
  - *A Golang library for building interactive and accessible prompts with full support for windows and posix terminals (terminals supporting ANSI escape sequences)*
  - ⚠️ *This project is no longer maintained. For an alternative, please check out: https://github.com/charmbracelet/bubbletea* ⚠️
  - Since 2017, archived by the owner on Apr 19, 2024
  - 70+ releases
  - Used by 200
  - Contributors 69

TUI frameworks (1/2)

- termui - *Golang terminal dashboard:*
  - active 2016-2019
  - 5 releases
  - 49 contributors
  - 13k GH stars

- tui-go - *a UI library for terminal applications*:
  - repo archived by the owner on Oct 13, 2021
  - 4 releases
  - 24 contributors
  - 2.1k GH stars

TUI frameworks (2/2)

- go-prompt - *Building powerful interactive prompts in Go, inspired by python-prompt-toolkit:*
  - active 2017 - 2021
  - 9 releases
  - 23 contributors
  - 1.7k GH stars

- tview - *Terminal UI library with rich, interactive widgets — written in Go:*
  - active since 2017
  - 0 releases
  - 89 contributors
  - 10.2k GH stars
  - Used by many incl. GH (https://github.com/cli/cli)

The ELM architecture (1/4)

- https://guide.elm-lang.org/architecture/

- Elm is a functional language that compiles to JavaScript.
  It helps you make websites and web apps.
  It has a strong emphasis on simplicity and quality tooling.
  Functional programming language for building browser-based GUIs.

- Elm Playground (Online Editor): https://elm-lang.org/try

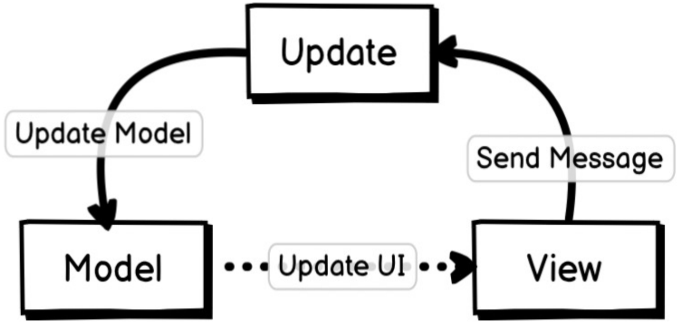
- The Elm Architecture is a pattern for architecting interactive programs, like webapps and games.

The ELM architecture (2/4)

- The basic pattern: “*The Elm program produces HTML to show on screen, and then the computer sends back messages of what is going on.*”

- The arch-pattern breaks into three parts (core concepts):

  - **Model** — the state of your application
  - **View** — the visual representation of the Model
  - **Update** — a way to update the Model based on messages

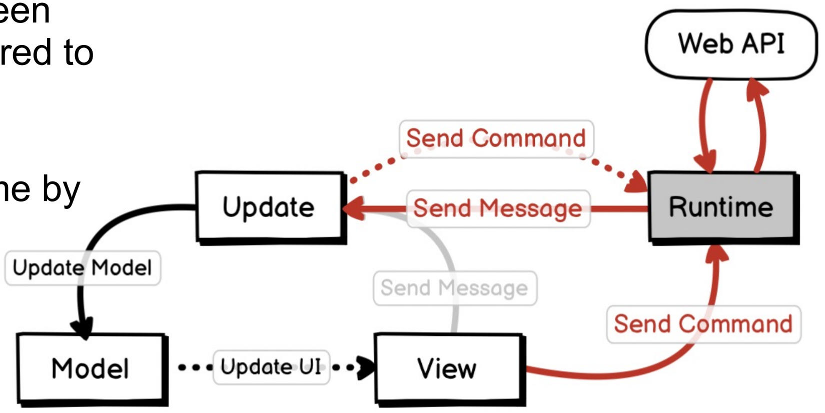


- Any user’s interaction on the View will trigger a **Message** that will be sent to the Update method, which will then modify the Model. Any changes to the Model will trigger updating the View.

The ELM architecture (3/4)

- Component that handles the comm. between app. and the external environment is referred to as the **Runtime**.

- Application communicates with the Runtime by sending **Commands**.

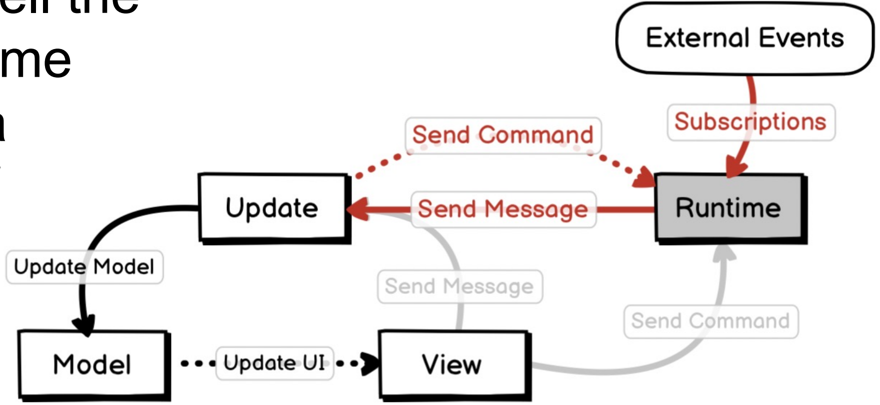


- When a Runtime receives a Command, it communicates with the external environment to get the work done and sends a Message back to the Update method of your application, so the UI can be updated with the result.

- If needed, the Update method can also send new Commands to the Runtime. The cycle continues.

The ELM architecture (4/4)

- Another important concept are **Subscriptions** - a way to tell the Runtime to subscribe to some external events and send a Message back to Update if needed.



- The rest of the flow is pretty much the same (in analogy to sending Commands).

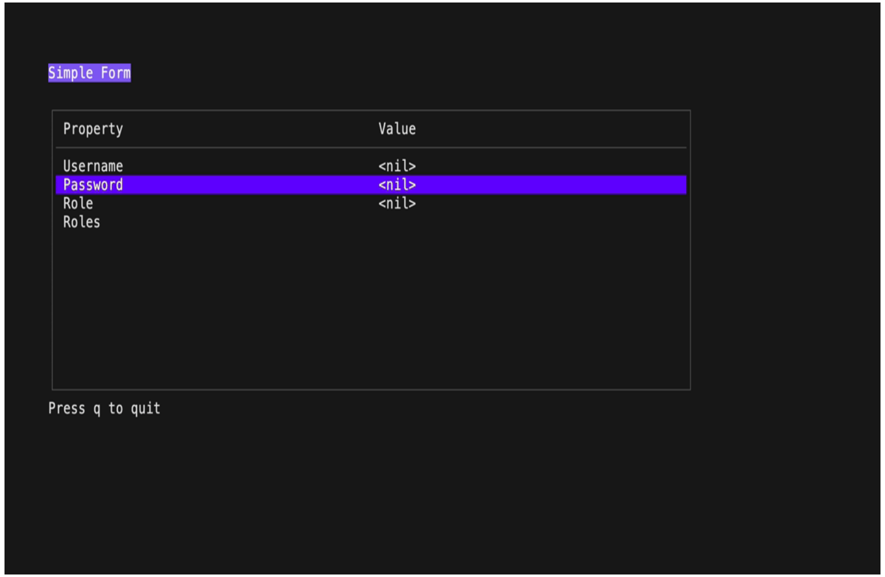
The *Bubbletea* Family (1/2) (*makes the command line glamorous*)

- bubbletea - *A powerful little TUI framework* 🏗
  - 36 releases
  - 25k GH stars
  - 7.4k users, also used in nearly 100 other GH repos
  - 124 contributors
- bubbles - *TUI components for Bubble Tea* 🫧
  - 24 releases
  - 5k GH stars
  - 6.2k users
  - 66 contributors
- Huh? - *Build terminal forms and prompts* 🤷🏻‍♀️ *(A simple, powerful library for building interactive forms and prompts in the terminal)*
  - 3.6k GH stars
  - 7 releases
  - 426 users
  - 27 contributors

The *Bubbletea* Family (2/2) (*makes the command line glamorous*)

- Bubble Tea programs are comprised of a model that describes the application state and three simple methods on that model:
  - **Init**, a function that returns an initial command for the application to run.
  - **Update**, a function that handles incoming events and updates the model accordingly.
  - **View**, a function that renders the UI based on the data in the model.

Terminalform – Building forms in terminal easily (from Devs to Devs)



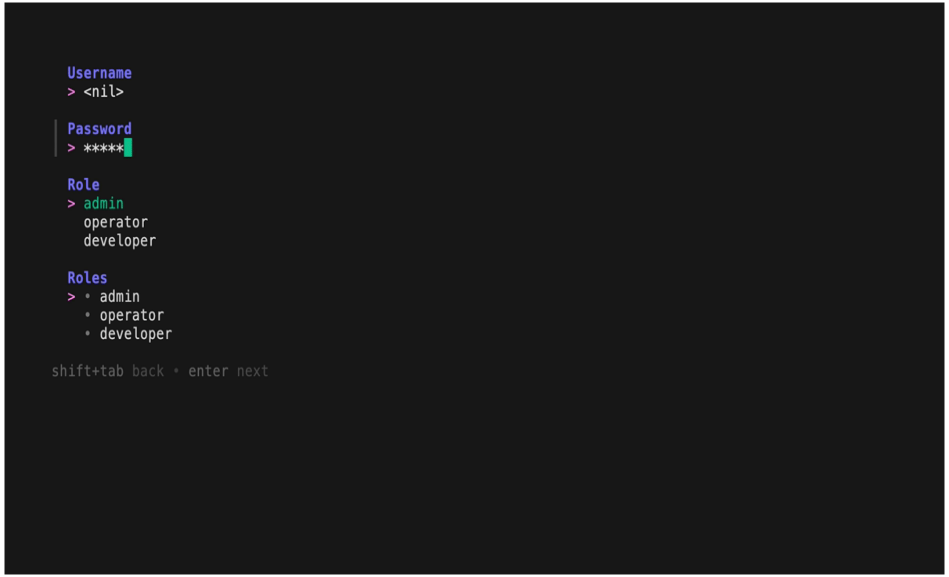
```
Simple Form

Property                                    Value

Username                                    <nil>
Password                                    <nil>
Role                                        <nil>
Roles

Press q to quit
```

Table view



```
Username
> <nil>

Password
> *****

Role
> admin
  operator
  developer

Roles
> • admin
  • operator
  • developer

shift+tab back • enter next
```

Form view

What *Terminalform* brings ?

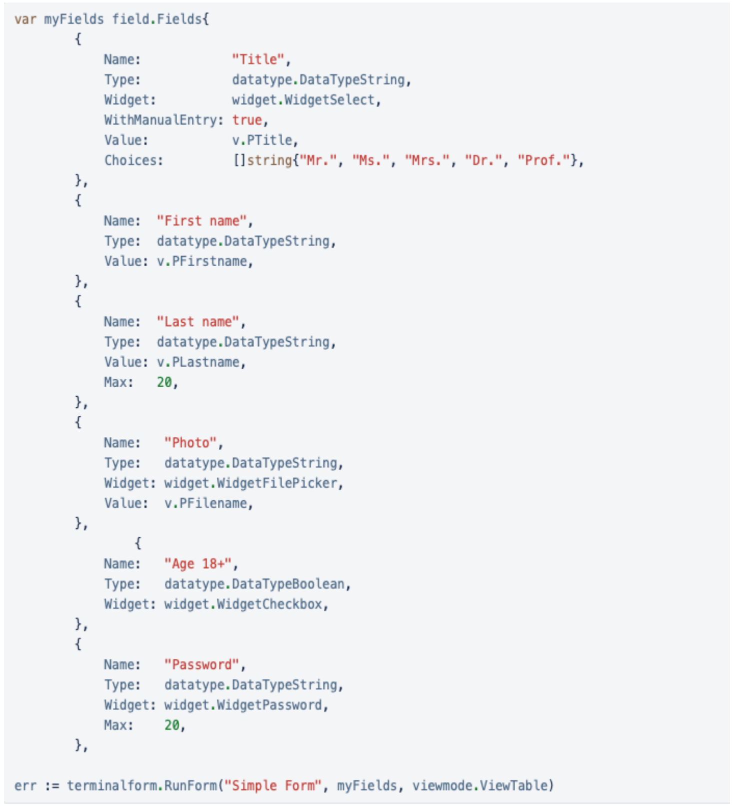
- Customizable widgets.

- You write Golang objects - you get terminal forms.

- Not in the mood to write Go code?
  No problem, just provide a proper YAML file (declarative approach).

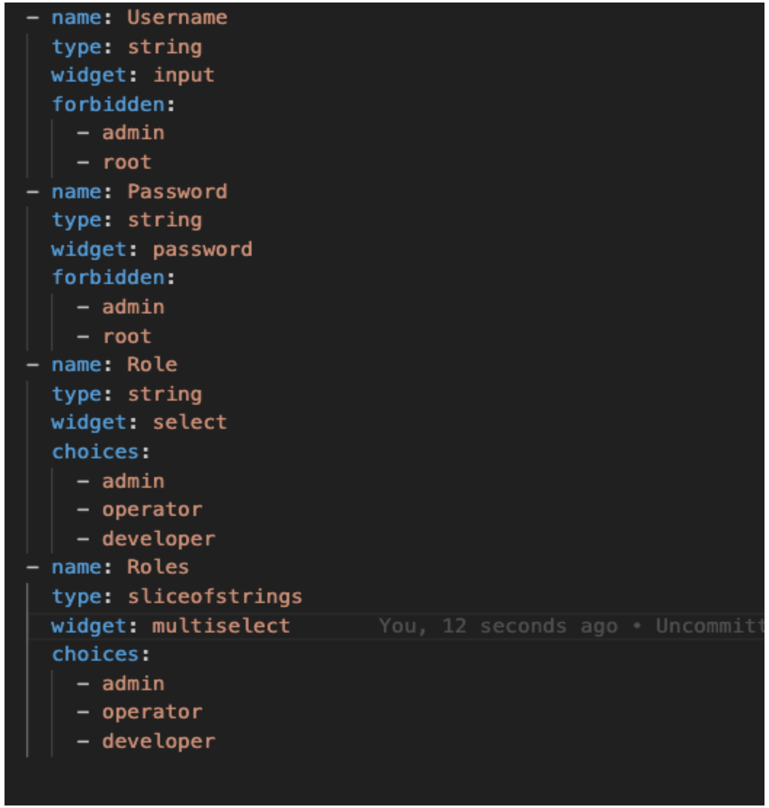
- Too lazy to write YAML files?
  No problem, just use *Straw* (form designer), to create *terminalform* specific YAML file.

Hands On? Well, barely !



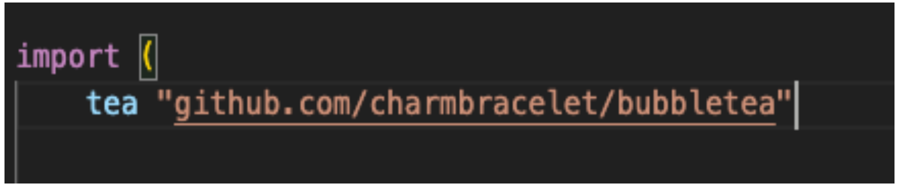
```go
var myFields field.Fields{
        {
            Name:           "Title",
            Type:           datatype.DataTypeString,
            Widget:         widget.WidgetSelect,
            WithManualEntry: true,
            Value:          v.PTitle,
            Choices:        []string{"Mr.", "Ms.", "Mrs.", "Dr.", "Prof."},
        },
        {
            Name:  "First name",
            Type:  datatype.DataTypeString,
            Value: v.PFirstname,
        },
        {
            Name:  "Last name",
            Type:  datatype.DataTypeString,
            Value: v.PLastname,
            Max:   20,
        },
        {
            Name:   "Photo",
            Type:   datatype.DataTypeString,
            Widget: widget.WidgetFilePicker,
            Value:  v.PFilename,
        },
                {
            Name:   "Age 18+",
            Type:   datatype.DataTypeBoolean,
            Widget: widget.WidgetCheckbox,
        },
        {
            Name:   "Password",
            Type:   datatype.DataTypeString,
            Widget: widget.WidgetPassword,
            Max:    20,
        },

err := terminalform.RunForm("Simple Form", myFields, viewmode.ViewTable)
```

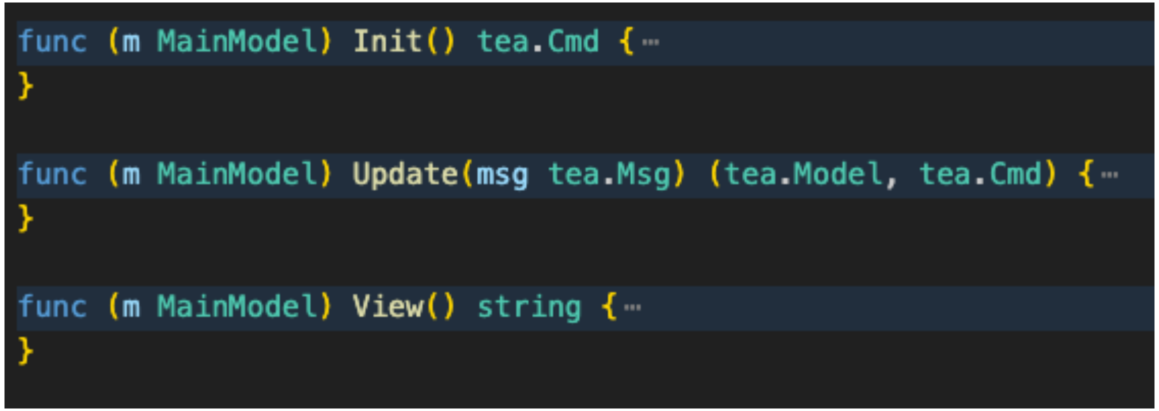


```yaml
- name: Username
  type: string
  widget: input
  forbidden:
    - admin
    - root
- name: Password
  type: string
  widget: password
  forbidden:
    - admin
    - root
- name: Role
  type: string
  widget: select
  choices:
    - admin
    - operator
    - developer
- name: Roles
  type: sliceofstrings
  widget: multiselect       You, 12 seconds ago • Uncommitt
  choices:
    - admin
    - operator
    - developer
```

Ultimately, what is a *widget* ?



```go
import (
    tea "github.com/charmbracelet/bubbletea"
```



```go
func (m MainModel) Init() tea.Cmd {…
}

func (m MainModel) Update(msg tea.Msg) (tea.Model, tea.Cmd) {…
}

func (m MainModel) View() string {…
}
```

Demo

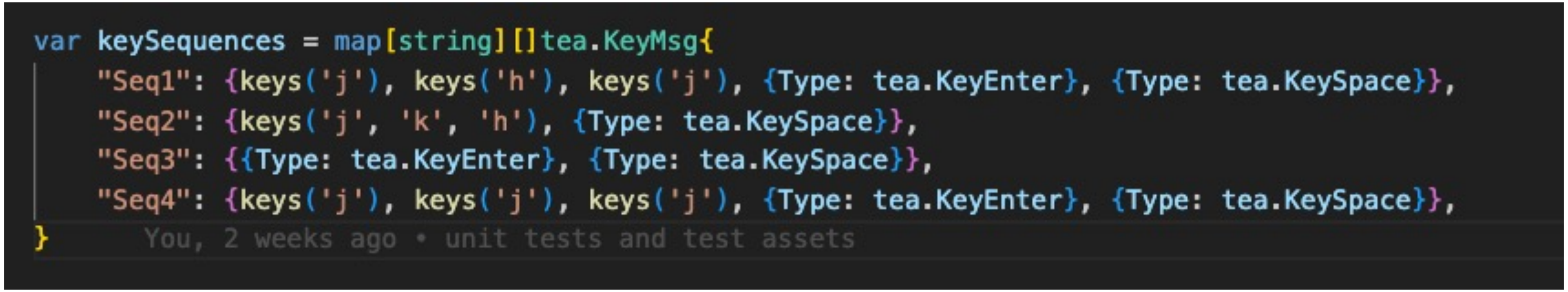
Challenges

- ➢Unit testing
- ➢Debugging

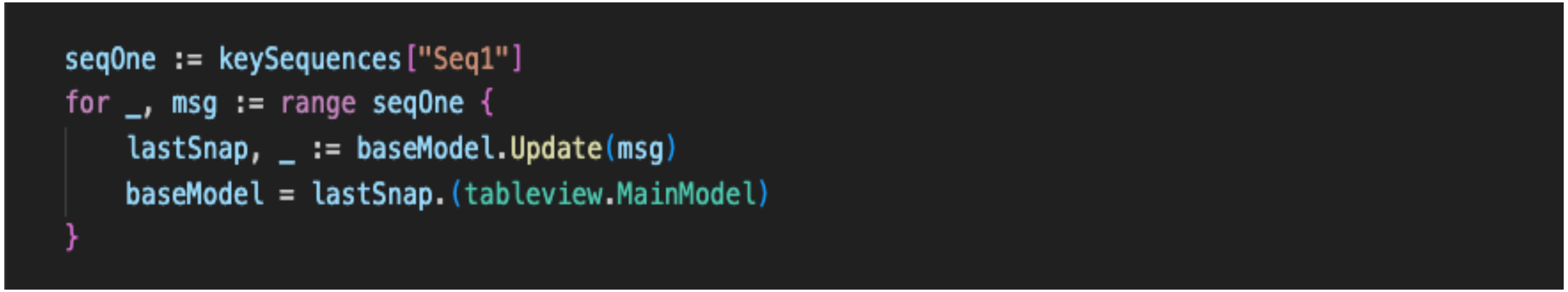
Unit Testing in Terminalform (1/2)

- Provide Keyboard as objects Keys to the model
- Generate View (ultimately, a string)
- Compare with the expected View in predefined asset

Unit Testing in Terminalform (2/2)



```go
var keySequences = map[string][]tea.KeyMsg{
    "Seq1": {keys('j'), keys('h'), keys('j'), {Type: tea.KeyEnter}, {Type: tea.KeySpace}},
    "Seq2": {keys('j', 'k', 'h'), {Type: tea.KeySpace}},
    "Seq3": {{Type: tea.KeyEnter}, {Type: tea.KeySpace}},
    "Seq4": {keys('j'), keys('j'), keys('j'), {Type: tea.KeyEnter}, {Type: tea.KeySpace}},
}        You, 2 weeks ago • unit tests and test assets
```



```go
seqOne := keySequences["Seq1"]
for _, msg := range seqOne {
    lastSnap, _ := baseModel.Update(msg)
    baseModel = lastSnap.(tableview.MainModel)
}
```

Debugging TUI apps TL;DR

-
  ```
  dlv debug \
   -headless \
   -api-version=2 \
   listen=127.0.0.1:{port_of_choice} .
  ```

- In another terminal, just run:
  ```
  dlv connect 127.0.0.1:{same_port}
  ```

- In VS Code, just attach to a running debug session (remotely).

- And last, but not least, ensure you deactivated mouse events.

Questions

- ➢Thank you for listening, any questions ?