The Next Steps

Building something useful: GUIs and working some real data

Admin

- **At the end of this read chapter 13 in the book.**
  - Particularly beginning at encoding/json on page 327 up to Structured logging on page 344
- **Quizzes starting ??**
- **Project questions?**

Applications

- We have quite a bit of go basics now
- To build a useful application we need what in addition?

Applications

- **We have quite a bit of go basics now**
- **To build a useful application we need what in addition?**
  - Data/input
- **Where could we get our data?**

Applications

- **We have quite a bit of go basics now**
- **To build a useful application we need what in addition?**
  - Data/input
- **Where could we get our data?**
  - User
  - Files (like what?)
  - The internet (give me some common sources)
  - Databases (common examples?)

Applications

- **We have quite a bit of go basics now**
- **To build a useful application we need what in addition?**
  - Data/input
- **Where could we get our data?**
  - User
  - Files (like what?) Excel and text files
  - The internet : Rest APIs, Remote Procedure calls? Others?
  - Databases (common examples: postgres, sqlite, mariaDB)

Data

- **We'll look at other approaches later but for now we will get data from REST apis**
  - Which begs the question – what are REST APIs?
  - Lucky Volunteer?™
  -

Data

- **We'll look at other approaches later but for now we will get data from REST apis**
  - Which begs the question – what are REST APIs?
  - Lucky Volunteer?™
  - Mechanism for letting an application ask a website for data (usually json, sometimes xml)
    - Instead of "scraping" the webpage.

JSON

- **So what is JSON?**
  - Lucky Volunteer?™

JSON

- **So what is JSON?**
  - Text format to send data
    - As JSON objects
      - JSON objects are a lot like what that you have seen earlier in the program?
    - And lists of JSON objects

JSON

- **So what is JSON?**
  - Text format to send data
    - As JSON objects
      - JSON objects are a lot like what that you have seen earlier in the program?
        - Python dictionaries
        - Or
        - Java record classes
        - Which is closer to the way GO uses do you suppose?
    - And lists of JSON objects

JSON

- **So what is JSON?**
  - Text format to send data
    - As JSON objects
      - JSON objects are a lot like what that you have seen earlier in the program?
        - Python dictionaries
        - Or
        - Java record classes
        - Which is closer to the way GO uses do you suppose: Closer to the Java approach
    - And lists of JSON objects

JSON

Sample JSON

From https://openlibrary.org/dev/docs/api/lists


```
{
   "links": {
      "self": "/people/george08/lists.json",
      "next": "/people/george08/lists.json?limit=5&offset=5"
   },
   "size": 12,
   "entries": [
      {
         "url": "/people/george08/lists/OL13L",
         "full_url": "/people/george08/lists/OL13L/Various_Seeds_for_Testing",
         "name": "Various Seeds for Testing",
         "last_update": "2010-12-21T00:46:17.712513",
         "seed_count": 13,
         "edition_count": 13181
      }
```


Sample JSON part II


```
{
      "url": "/people/george08/lists/OL97L",
      "full_url":
```


- "/people/george08/lists/OL97L/Time_Travel",


```
      "name": "Time Travel",
      "last_update": "2010-12-17T18:27:14.781336",
      "seed_count": 5,
      "edition_count": 838
   },
   ...
]}
```

Getting Data from the web

- **What protocol do we use to get data from the web?**
  - Students who have taken the networks class

Getting Data from the web

- **What protocol do we use to get data from the web?**
  - Students who have taken the networks class
- **http(s) right?**
- **Go go developed in late 2000s so *good* support in the standard library**
  - net/http package
  - Updated fairly recently
  - Returns a response struct and an error
    - https://golang.org/src/net/http/response.go
    - Mostly we care about StatusCode
    - And Body

Http package structs and interfaces

- **Here are the bits of the http package we need for this kind of task**
  - this is the newer way – many LLMs are still suggesting the 2010s way

```
http.Client   struct
```

  - will connect to the server, for now use zero value for all but Timeout

```
http.NewRequestWithContext
```

  - The 2020s way to create a request struct. Several params needed

```
apiClient.Do(request)
```

  - You call the client.Do function passing the request, returns response and error

Painless JSON in go

- **OK well ‘painless’ is optimistic, but far less painful than it was just 2 years ago.**
- **First create the structs that you need for the JSON**
  - This is something that LLMs are really good at
- **Then Use the encoding/json package.**

Let’s Try it.

- **We’ll start with a fairly straightforward API**
  - The open library API
  - It is free – not even an API key needed (as long as you don’t make too many requests per second)
  - Sample query
    - https://openlibrary.org/search.json?author=zeihan
  - I’ve picked an Author that has a limited number of records, but enough interesting going on that we can get the right structs

LLMs

- **Tell me about LLMs – the current hotness in AI**

LLMs

- **Tell me about LLMs – the current hotness in AI**
  - Good:
    - Can often help someone who is really bad at a task look somewhat competent
      - I'm not good at creating images – so I can use AI and manage
    - They are really good doing the small (even if they seem large to students) projects that we do in academia
  - Bad:
    - Often very bad at creating something that has to work in a larger context
    - Often very bad at doing something that requires specialized knowledge.

LLMs and JSON

- **LLMs are usually really good at limited scope, self contained, well known problems.**
  - And that is exactly the kind of problem that building the go structs that will represent the json data is
  - So run the API link that I had in your favorite browser.
  - And copy the resulting json
  - Feed it into your favorite LLM harness and ask it to give you the go structs

LLMs and JSON

- **Normally one would want to double check the code generated by LLMs (if one wanted maintainable software)**
  - Despite what AI company marketing people might say
  - But for this task it almost always gets it right.
- **Big Caveat.**
  - Some JSON APIs have fields that are optional: they exist in some records but not in others.
  - You need to find a json response that has both to get the LLM to generate the right structs

Getting the JSON

- In the examples to follow, know that my LLM called the two structs needed

OpenLibrarySearchResponse

And

Book

- So those are what you see in this exemplar code

What do we need to check for?

- **Whenever we do any of the following, what do we need to check for?**
  - Open a file
  - Read a file
  - Connect to the internet
  - Get data from that internet connection?
- **How are they handled in python and java?**

What do we need to check for?

- **Whenever we do any of the following, what do we need to check for? Errors that aren't programming errors**
  - Open a file
  - Read a file
  - Connect to the internet
  - Get data from that internet connection?
- **How are they handled in python and java?**
  - Exceptions!
  - Go doesn't use exceptions – errors that have to be handled

Example function: let's look and ask questions

```
func getBookData(author string) ([]Book, error) {
	var result []Book
	apiClient := &http.Client{
		Timeout: 30 * time.Second,
	}
	fullAddress := fmt.Sprintf("%s%s", ServerAddress, author)
	request, err := http.NewRequestWithContext(
		  context.Background(),
		http.MethodGet, fullAddress, nil)
	if err != nil {
		return nil, err
	}
	response, err := apiClient.Do(request)
	if err != nil {
		return nil, err
	}
	if response.StatusCode != http.StatusOK {
		return nil, fmt.Errorf("unexpected status code: %d",
response.StatusCode)
	}
	defer response.Body.Close()

	var jsonResponse = OpenLibrarySearchResponse{}
	err =
json.NewDecoder(response.Body).Decode(&jsonResponse)
	if err != nil && err != io.EOF {
		print("Error the actual response body was: ",
response.Body, err)
		return nil, err
	}

	result = jsonResponse.Docs
	return result, nil
}
```

```
const ServerAddress = "https://openlibrary.org/search.json?author="
```

In class exercise

- **Write a main function that will call the function from previous slide**
  - Ask the user for an author
  - Call the getBookData function, passing the author from user
  - If there was an error, report the error.
  - If there was not, then print each book struct on its own line.

In class exercise

- **Write a main function that will call the function from previous slide**
  - Ask the user for an author
  - Call the getBookData function, passing the author from user
  - If there was an error, report the error.
  - If there was not, then print each book struct on its own line.

GUIs

- **What kind of GUI programming have you done?**
  - Quick survey

GUIs

- **What kind of GUI programming have you done?**
- **My guess before talking with you all**
  - Java: JavaFX, maybe android and swing,
  - Kotlin: android?
  - Swift: iOS native?
  - Python: pyQT or similar?
  - javascript: react, angular etc
  - Anything else?

Threads and event handling

- What are threads and events?
- How do most GUI libraries handle threads and events?

Threads and event handling

- **How do most GUI libraries handle threads and events?**
  - Main thread spawns new thread which will run the GUI, your main thread often becomes vestigial
    - java's main 'method' or
    - pythons first method called in the `if __name__ == '__main__'` block
  - How do events get handled?

Threads and event handling

- **How do most GUI libraries handle threads and events?**
  - Main thread spawns new thread which will run the GUI, your main thread often becomes vestigial
    - java's main 'method' or
    - pythons first method called in the `if __name__ == '__main__'` block
  - How do events get handled?
    - The GUI library has an event loop, you provide callback functions in your code.
    - Event loop? Callbacks? (let the students answer)

Look, feel and layout?

- **How was the arraignment and look and feel done for GUIs in the ‘classical era’ of programming?**
  - 1980ish-2010ish?

Look, feel and layout?

- **How was the arraignment and look and feel done for GUIs in the ‘classical era’ of programming?**
  - 1980ish-2010ish?
    - Put controls on window, use some sort of layout to arrange them. Use static properties to paint them the right way.
    - Best in an era of laptops and desktops – aspect & size doesn’t change during run
  - And in current era 2010ish-now ?

Look, feel and layout?

- **How was the arraignment and look and feel done for GUIs in the ‘classical era’ of programming?**
  - 1980ish-2010ish?
    - Put controls on window, use some sort of layout to arrange them. Use static properties to paint them the right way.
    - Best in an era of laptops and desktops – aspect & size doesn’t change during run
  - And in current era 2010ish-now ?
    - Use some sort of layout to arrange them, use CSS to make them look nice.
    - Useful for mobile and multi-monitor era size and aspect ratio might change

GUIs and Go

- **If we look at the awesome go list**
  - Lots of GUI choices
  - https://github.com/avelino/awesome-go#gui
  - Three important categories
    - Go wrappers around c/c++ libraries (wxwindows qt etc)
    - Electron wrappers or clones (someone tell me about electron) (wails etc)
    - Pure go implementations (fyne, gio etc)

Fyne

- **I'll use Fyne in this class for GUI**
  - 'materials design' GUI
    - Originally a google standard
    - Designed to have the same application look the same across platforms.
  - For last couple of years, can now compile to webassembly
    - And deploy applications to web.
  - Otherwise classic GUI development feel
  - Official docs https://fyne.io/

Hello Fyne

- Here is the official hello world for fyne

```
package main
import (
"fyne.io/fyne/v2/app"
"fyne.io/fyne/v2/widget"
)
func main() {
	a := app.New()
	w := a.NewWindow("Hello")

	w.SetContent(widget.NewButton(
		"Click me",
		func() { println("Hello!") },
	))

	w.ShowAndRun()
}
```

- Create a fyne app
- Create one more more windows
- Add come widgets to the window
- Finally call one of the run or show functions on the main window

Widgets

- **GUI components**
  - Called widgets, controls, or components depending on the library
- **Widgets divided into two groups in docs**
  - Widgets for one piece of data
    - Eg: https://docs.fyne.io/widget/label/
  - Widgets for holding aggregate data
    - Eg https://docs.fyne.io/collection/list/
  - Let’s take a quick look at what is available there

Layouts

- **Use layouts as a way to arrange components on the window**
  - BorderLayout
    - Use this popular layout if you need something to be big (center) and other smaller stuff around it
  - Box layout: (vbox or hbox)
    - Use to build a column or row of components
  - Grid
    - If you are building something like a calculator, a grid of equally sized components
  - AppTabs
    - Tabs at the top (or side) with a large area that will hold its own layouts

Update to main for GUI

- Update to main and new package wide variables

```
func main() {
	reader := bufio.NewReader(os.Stdin)
	fmt.Print("What author would you like to search for? ")
	author, err := reader.ReadString('\n')
	author = strings.TrimSpace(author)
	data, err := getBookData(author)
	if err != nil {
		fmt.Println("Error:", err)
		return
	}
	gui := *buildBookGUI(data)
	gui.ShowAndRun()
}
```

```
var (
	books          []Book
	authorsDisplay  *widget.Entry
	languageDisplay *widget.Entry
	publishDisplay  *widget.Entry
)
```

The BuildBookGUI function

```
func buildBookGUI(bookData []Book) *fyne.Window {
	guiApp := app.New()
	books = bookData
	mainWindow := guiApp.NewWindow("Book GUI")
	directions := widget.NewLabel("Select a book")
	directions.Alignment = fyne.TextAlignCenter
	directions.TextStyle = fyne.TextStyle{Bold: true}
	titleList := widget.NewList(func() int { return len(books) },
		func() fyne.CanvasObject {
			return widget.NewLabel("Placeholder")
		},
		func(i widget.ListItemID, selected fyne.CanvasObject) {
			selected.(*widget.Label).SetText(books[i].Title)
		})
	titleList.OnSelected = bookSelected
	detailsPanel := makeDetailsPanel()
	mainLayout := container.NewBorder(directions, nil, nil, detailsPanel, titleList)
	mainWindow.SetContent(mainLayout)
	mainWindow.Resize(fyne.NewSize(800, 800))
	return &mainWindow
}
```

- **Some of this you've seen**
  - List gets created with three functions as params
    - First two usually always same,
      - return len of data
      - and create lable with default text
    - Then function to actually put correct text
    - After that add OnSelected function
  - Border panel param order
    - (top,bottom, left, right, center)

AddDetailsPanel

```
func makeDetailsPanel() *fyne.Container {

<snip>
        detailsPanel := container.New(layout.NewVBoxLayout())
        a := widget.NewLabel("Authors displayed below:")
        authorsPanel.Add(a)
        authorsDisplay = widget.NewEntry()
        authorsDisplay.MultiLine = true
        authorsDisplay.PlaceHolder = "Authors will appear here"
        authorsDisplay.Resize(fyne.NewSize(200, 100))
        authorsPanel.Add(authorsDisplay)
        detailsPanel.Add(authorsPanel)
        a = widget.NewLabel("Available in these Languages:")
        languagePanel.Add(a)
        languageDisplay = widget.NewEntry()
        languageDisplay.MultiLine = true
        languageDisplay.PlaceHolder = "Languages will appear here"
        languageDisplay.Resize(fyne.NewSize(200, 100))
        languagePanel.Add(languageDisplay)
        detailsPanel.Add(languagePanel)
        <snip>
        return detailsPanel
}
```

- **I cut a third of this, but you can see the general Idea**
  - Questions?

OnSelected

```
func bookSelected(i widget.ListItemID) {
	bookAuthors := books[i].AuthorName //AuthorName is a misnomer, it is a possibly empty list of authors
	if len(bookAuthors) == 0 {
		authorsDisplay.SetText("")
		return
	}
	var authorsNames = strings.Builder{}
	for _, author := range bookAuthors {
		authorsNames.WriteString(author + "\n")
	}
	authorsDisplay.SetText(authorsNames.String())
	pubDate := books[i].FirstPublishYear
	publishDisplay.SetText(strconv.Itoa(pubDate))
	<snip>
}
```

- Once again I cut some of it to get something on the slide
- But setting the languages works a lot like the authors.
- Let's look at the full thing which will be on github
- And ask questions
- And try stuff out.

Display and image

- **Fyne treats images as a widget**
  - Can put them in a container like any other widget – then add container to window

```
flagPict := canvas.NewImageFromFile("flag.png")
flagPict.FillMode = canvas.ImageFillOriginal
flagPanel := container.NewHBox(flagPict)
```

  - There are several NewImageFromXXX methods in the fyne canvas package.
  - The fill mode shows display properties
    - Fill original will keep original size and aspect
    - Which could be quite large for some images

Display an image

- **You can also use**

```
flagPict := canvas.NewImageFromFile("flag.png")
flagPict.SetMinSize(fyne.NewSize(100, 60))
flagPict.FillMode = canvas.ImageFillContain
flagPanel := container.NewHBox(flagPict)
```

- **ImageFillContain**
  - Will keep aspect ratio
  - MinSize defaults to (0,0)
  - So in side of boderlayout won’t be drawn without a new MinSize

Let’s Look at Grad Project 2
ugrad project3