FINAL PROJECT

REPORT

Survey of Programming Languages

ITCS 4104/5102

Fall 2021

Project Title: Handyman Website

Using Go language

Students:	**Professor:**
Varad Deshpande 801243927	Dr. Ali Sever
Keerthi Avittan Elamparithi 801166818	
Surya Teja Adusumalli. 801218602	
Satyadev Tummala 801202003	
Sowri Madduri 801196202	
Manissha Malapati 801209658	
Kundana Siri Chukka 801200452	

Introduction:

Go is a general-purpose programming language with a focus on systems programming. It offers explicit support for concurrent programming and is strongly typed and refused. Packages are the building blocks of programs, and their attributes allow for efficient dependency management.

Problem Definition:

Help for simple household work like fixing floors, cupboards, etc is very hard to find, and for handymen finding work is also very difficult as they have to search through various agencies. The solution we came up with is to create a website that lets people and handymen connect thereby eliminating the need for a middleman.

Our website would register the handyman in the system and make them available for work to the users of the website. Whenever the user needs some domestic help he can just log in to the website and search for the type of help he wants and the website would display all the handymen in that particular area and type of work.

Paradigm of Go Language:

Go is a multi-paradigm programming language that is geared on creating safe, high-quality agent-based applications. It has several threads, is tightly typed, and is of higher order (in the functional programming sense). It includes definitions for relationships, functions, and action procedures. As needed, threads run action procedures, invoke functions, and query relations. Asynchronous messages are used by threads in various agents to collaborate closely. Shared dynamic relations can also be used by threads inside the same agent to operate as Linda-style tuple stores.

The paradigms of Go-

Although Go supports types and functions, as well as an object-oriented programming approach, it lacks a class hierarchy in favour of interfaces. Methods for all kinds, including primitive types like int, can be developed. The struct object in Go is so light-weight that it resembles C rather than C++ or Java.

Imperative: Go is a language that is written in the imperative mode. According to the Go requirements, it has loops, statements, and selections.

Concurrent programming: Go comes with built-in support for multi-threaded operations. Multi-threading, multi-processing, and asynchrony are all examples of concurrency in Go.

Origin and History:

On September 21, 2007, Robert Griesemer, Rob Pike, and Ken Thompson began sketching the aims for a new language on a white board. Within a few days, the objectives had solidified into a plan to accomplish something and a good notion of what that something would be. Parallel to unrelated tasks, design continued part-time. By January 2008, Ken had begun work on a compiler that would allow him to experiment with new ideas and would output C code. By the middle of the year, the language had grown into a full-time effort, and it had settled down enough to try a production compiler. Ian Taylor began working on a GCC front end for Go using the draft specification in May 2008. Russ Cox arrived in late 2008 and was instrumental in bringing the language and libraries from concept to reality.

On November 10, 2009, Go became a public open-source project. Many members of the community have contributed their thoughts, comments, and code.

There are currently millions of Go programmers (also known as gophers) all around the world, and the number is growing every day. Our hopes for Go's success were considerably exceeded.

Go is largely in the C family (basic syntax), with some contributions from Pascal/Modula/Oberon (declarations, packages), as well as certain concepts from languages inspired by Tony Hoare's CSP, such as Newsqueak and Limbo (concurrency). It is, however, a new language in general. In every way, the language was created with programmers in mind, with the goal of making programming, at least the kind we do, more efficient.

Evolution of Go:

For the work we were doing at Google, we were frustrated with existing languages and environments. The choice of languages was largely to blame for the difficulty of programming. Efficient compilation, execution, or programming ease had to be chosen; all three were not available in the same popular language. Programmers who could did so, preferring dynamically typed languages like Python and JavaScript over C++ and, to a lesser extent, Java, chose convenience over safety and performance.

We weren't the only ones who were worried. After a long period of relative silence in the programming language world, Go was one of the first of a slew of new languages—Rust, Elixir, Swift, and others—that has resurrected programming language development as an active, virtually mainstream field.

Go attempted to combine the simplicity of programming of an interpreted, dynamically typed language with the efficiency and safety of a statically typed, compiled language to address these challenges. It also aimed to be cutting-edge, with networked and multicore computing capabilities. Finally, Go is designed to be quick: building a big executable on a single computer should take no more than a few seconds. To achieve these objectives, a variety of linguistic challenges had to be addressed, including an expressive yet lightweight type system, concurrency and garbage collection, tight dependency specification, and so on. Libraries and tools could not adequately address these issues; a new language was required.

The elements of Go language:

Reserved words -- break, default, func, interface, select, case, defer, go, map, struct,chan, else, goto, package, switch, const, fallthrough, if, range, type, continue, for, import, return, var.

Primitive data types -- Go language has int, float, byte, string & boolean as primitive data types.

Structured data types -- Array, Slice, Struct, Map, Channel

Syntax of the language:

 The syntax will be familiar for programmers used to languages from the C-syntax family but it is clean and resembles how easy it is to write Python code.

Here is a simple “Hello World” program.

```
        package main


        import (
                “Fmt”
        )
        function main() {
                fmt.println(“Hello World”)
}
```

Go has the types a programmer would expect: integers, floating point numbers, strings, characters.

```
type funcCollection [1]( func (* int) string )

var str string = " golang "

func f0 ( i *int) * string {

 return & str

}


func main () {

 fnColl := funcCollection { f0 }

 input := 32

 fmt . Printf ("%s\n", * fnColl [0](& input ) )

}
```

Golang Conditional Statements

Golang has several of its control flow syntax from the C-family of languages. In Golang we have the following conditional statements:

- The if statement - executes some code if one condition is true
- The if...else statement - executes some code if a condition is true and another code if that condition is false
- The if...else if....else statement - executes different codes for more than two conditions
- The switch...case statement - selects one of many blocks of code to be executed

The if statement supports a composite syntax where the tested expression is preceded by an initialization statement.

IF statement:

```
package main

import (

	"fmt"

)


func main() {

	var s = "Japan"

	x := true

	if x {

			fmt.Println(s)

	}

}
```

If….else statement:

The if....else statement allows you to execute one block of code if the specified condition is evaluates to true and another block of code if it is evaluates to false.

```
package main
import (
	"fmt"
)
func main() {
	x := 100
	if x == 100 {
			fmt.Println("Japan")
	} else {
			fmt.Println("Canada")
	}
}
```

If….else if…else statement:

The if...else if...else statement allows to combine multiple if...else statements.

```
package main
import (
	"fmt"
)
func main() {
	x := 100
	if x == 50 {
		fmt.Println("Germany")
	} else if x == 100 {
		fmt.Println("Japan")
	} else {
		fmt.Println("Canada")
	}
}
```

The switch statement:

The switch statement is used to select one of many blocks of code to be executed. Consider the following example, which display a different message for particular day.

```
package main
import (
	"fmt"
	"time"
)
func main() {
	today := time.Now()
	var t int = today.Day()
```

```
    switch t {
    case 5, 10, 15:
            fmt.Println("Clean your house.")
    case 25, 26, 27:
            fmt.Println("Buy some food.")
    case 31:
            fmt.Println("Party tonight.")
    default:
            fmt.Println("No information available for that day.")
    }
}
```

For loop:

A for loop is used for iterating over a sequence (that is either a slice, an array, a map, or a string.

```
package main
import "fmt"
func main() {
    i := 5
    for {
            fmt.Println("Hello")
        if i == 10 {
                break
        }
            i++
    }
}
```

Functions in Go language:

A function is a collection of statements in a program that are used to complete a certain activity. A function, at its most basic level, takes an input and returns an output.
Functions allow you to condense frequently used code into a single component.
The main() function is the most used Go function, and it is used in every independent Go application.
A function declaration starts with the func keyword, then the name you wish for the function, a pair of parentheses (), and the code for the function.

SimpleFunction is the name of a function in the following example. It doesn't take any parameters and doesn't return any results.

```
package main

import "fmt"

// SimpleFunction prints a message
func SimpleFunction() {
	fmt.Println("Hello World")
}

func main() {
	SimpleFunction()
}
```

Arguments can be used to pass information to functions. A variable is the same as an argument.
Arguments are supplied inside parentheses after the function name. You can use as many arguments as you like; just use a comma to separate them.
A function with two int type arguments is shown in the following example. We supply two integer values to the add() method when it is invoked.

```
package main

import "fmt"

// Function accepting arguments
func add(x int, y int) {
	total := 0
	total = x + y
	fmt.Println(total)
}

func main() {
	// Passing arguments
	add(20, 30)
}
```

The return values of a function in Golang can be named. We can also designate the return value by defining variables; for example, in the function declaration, a variable total of integer type is established for the value that the function returns.

```
package main

import "fmt"

func rectangle(l int, b int) (area int) {
	var parameter int
	parameter = 2 * (l + b)
	fmt.Println("Parameter: ", parameter)

	area = l * b
	return // Return statement without specify variable name
}

func main() {
	fmt.Println("Area: ", rectangle(20, 30))
}
```

Go language methods:

A method is defined in the same way that any other Go function is. A method is a Go function that is defined with a specific scope or connected to a certain type. Methods are a mechanism to give user-defined types behaviour. Methods are actually functions with an additional parameter provided between the keywords func and the function name.

In Go, a method is a special type of function that acts on variables of a specific type. The receiver, which is an extra parameter placed before the method's name and used to specify the moderator type to which the method is attached, is an extra parameter placed before the method's name and used to specify the moderator type to which the method is attached. The receiver type does not have to be a struct type: any type, including function types and alias types for int, bool, string, and array, can contain methods.

```
package main
import "fmt"

type multiply int

func (m multiply) tentimes() int {
        return int(m * 10)
}

func main() {
        var num int
        fmt.Print("Enter any positive integer: ")
        fmt.Scanln(&num)
        mul:= multiply(num)
        fmt.Println("Ten times of a given number is: ",mul.tentimes())
}
```

Objects in Go language:

In Go, there is no such thing as a class type that serves as the foundation for objects. In Go, any data type can be utilized to create an object. In Go, struct types can be given a set of methods to specify their behaviour. In GO, there is no such thing as a class or an object. The struct type in Go is the closest thing to an object in other programming languages. The majority of principles associated with object-oriented programming are supported by Go.

Go supports physical and logical modularity at its core, thanks to notions like packages and an extensible type system; as a result, we were able to achieve modularity and encapsulation in Go.

A newly declared name type does not inherit all attributes of its underlying type and are treated variously by the type system. Hence GO doesn't support polymorphism through inheritance. But it is possible to create objects and express their polymorphic relationships through composition using a type such as a struct or an interface.

Evaluation of the language

Writability: Using Go language makes the program clear, concise and faster. It is easy and understandable so it makes the Golang writable.

Readability: Go language is easy to write, scalable and a non-complex language and makes it readable.

Reliability: Reliability depends on the errors, impact of errors and error handling. It also depends on the efficiency of language. Go is an efficient language and is easy to handle the errors. However, sometimes it gets difficult to handle the errors and it lacks the fluid interface like other programming languages.

Strengths and weaknesses of Golang:

The Strengths:

- It's relatively simple to learn and be effective with. Easy to write and scale applications.
- Fewer features mean fewer things to worry about: no OOP means no classes, objects, inheritance, polymorphism, or the complexities that come with them.
- Errors are treated as return values rather than thrown exceptions. This is a new method of looking at a mistake or exception.
- The world is returning to statically typed languages, and Go is one of them. Easy to implement security features.
- Goroutines make concurrency programming easier.
- It is possible that it uses fewer resources (memory, CPU, etc.) than other programming languages.
- Has the benefit of Garbage Collection in the JVM/DotNet world, which relieves the programmer of the stress of freeing crucial resources.
- Pointers are only used in select situations (such as Pass-By-Reference to alter state or gain a reference to a specific object) where there are no other options.

The weakness:

- If you're used to OO programming, you'll be confused with OOP. As an alternative, you can use methods linked to Struct constructions as a replacement.
- The lack of abstractions provided in high-level languages may be felt in Go, indicating that it was created as a viable alternative to C programming language. Consider the 'slice' data-structure in Go, which is a wrapper around the 'array' data-structure for size flexibility. The slice data structure, on the other hand, would not offer you with any easy way to remove an element from it.
- Because Go lacks the fluid interfaces that languages like Java, Python, and others offer, you must frequently switch contexts between the business problem you're solving and the nitty-gritty of the programming language. Instead of employing high-level constructs like 'for each' to iterate a collection, you use your old methods of inserting local variables in the for loop construct. And I don't think you can have fluent interfaces without going through the Generics process first.
- As things stand now, there isn't much in the way of mature tooling support in terms of IDEs or libraries.

Overview:

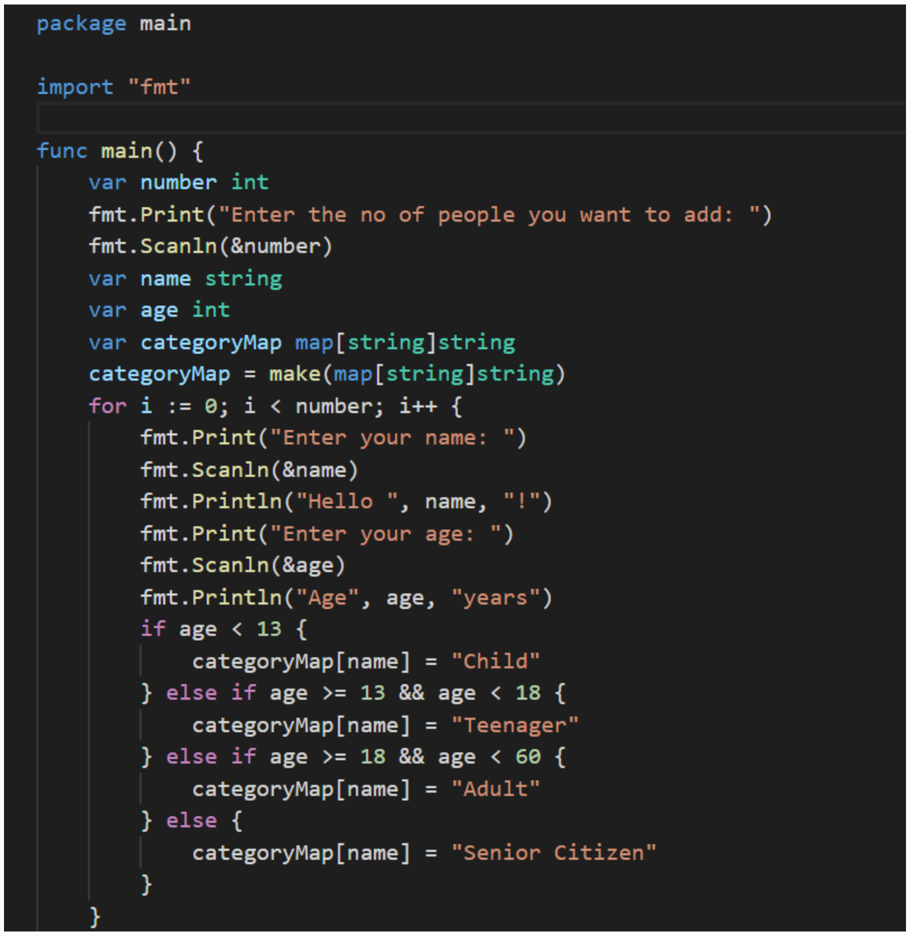
Golang is an excellent language for creating lightweight microservices. It's presently being used to create APIs that interface with our front-end apps. Golang is an excellent tool to utilize if you want to quickly construct a small working microservice.

One of the most noticeable drawbacks of coding in Go is that it lacks several of the capabilities that I find handy in other programming languages like C#. When working with collections of data in C#, the LINQ functions are the most important functionality lacking.

We wanted to move away from JavaScript for back-end development and needed a language that would allow us to cross-compile software across OS platforms. Because we were also aiming for a service-oriented architecture, Golang felt like a good fit. Many of our new microservices are now written in the Go programming language.

The language's simplicity facilitates code review and debugging, and the notion of leanness in libraries also aids debugging. Meanwhile, because all dependencies are included in the built binary, deployments are simple.

Sample run of programs



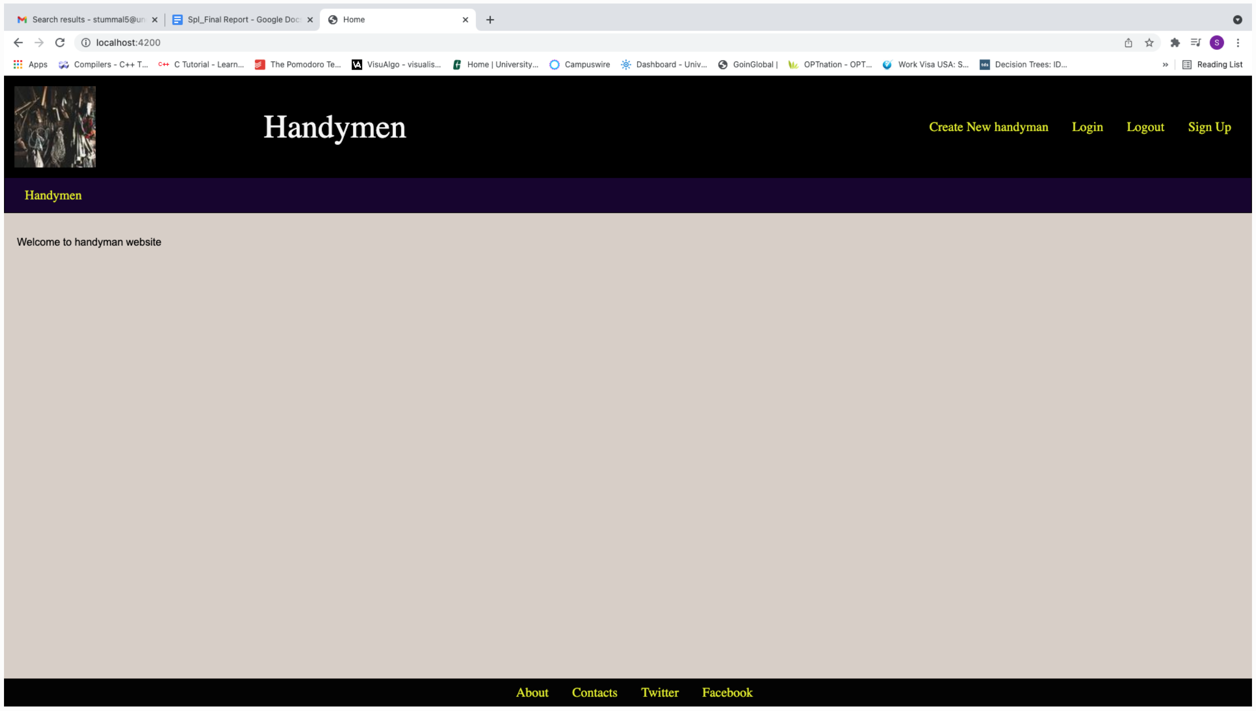
```go
package main

import "fmt"

func main() {
    var number int
    fmt.Print("Enter the no of people you want to add: ")
    fmt.Scanln(&number)
    var name string
    var age int
    var categoryMap map[string]string
    categoryMap = make(map[string]string)
    for i := 0; i < number; i++ {
        fmt.Print("Enter your name: ")
        fmt.Scanln(&name)
        fmt.Println("Hello ", name, "!")
        fmt.Print("Enter your age: ")
        fmt.Scanln(&age)
        fmt.Println("Age", age, "years")
        if age < 13 {
            categoryMap[name] = "Child"
        } else if age >= 13 && age < 18 {
            categoryMap[name] = "Teenager"
        } else if age >= 18 && age < 60 {
            categoryMap[name] = "Adult"
        } else {
            categoryMap[name] = "Senior Citizen"
        }
    }
```

Output:

```
Enter your name: Carrol
Hello Carrol !
Enter your age: 64
Age 64 years
```

Project Screenshosts

Home Page

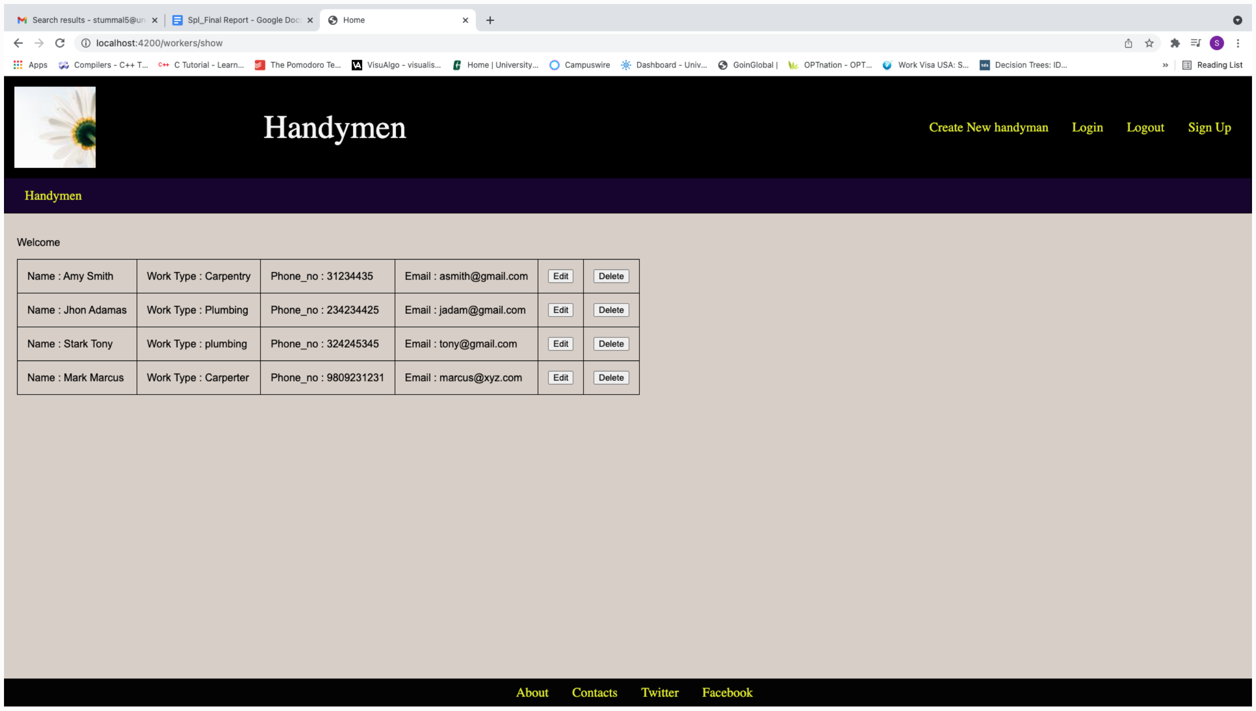


Handymen

Handymen

Welcome to handyman website

Show all handymens



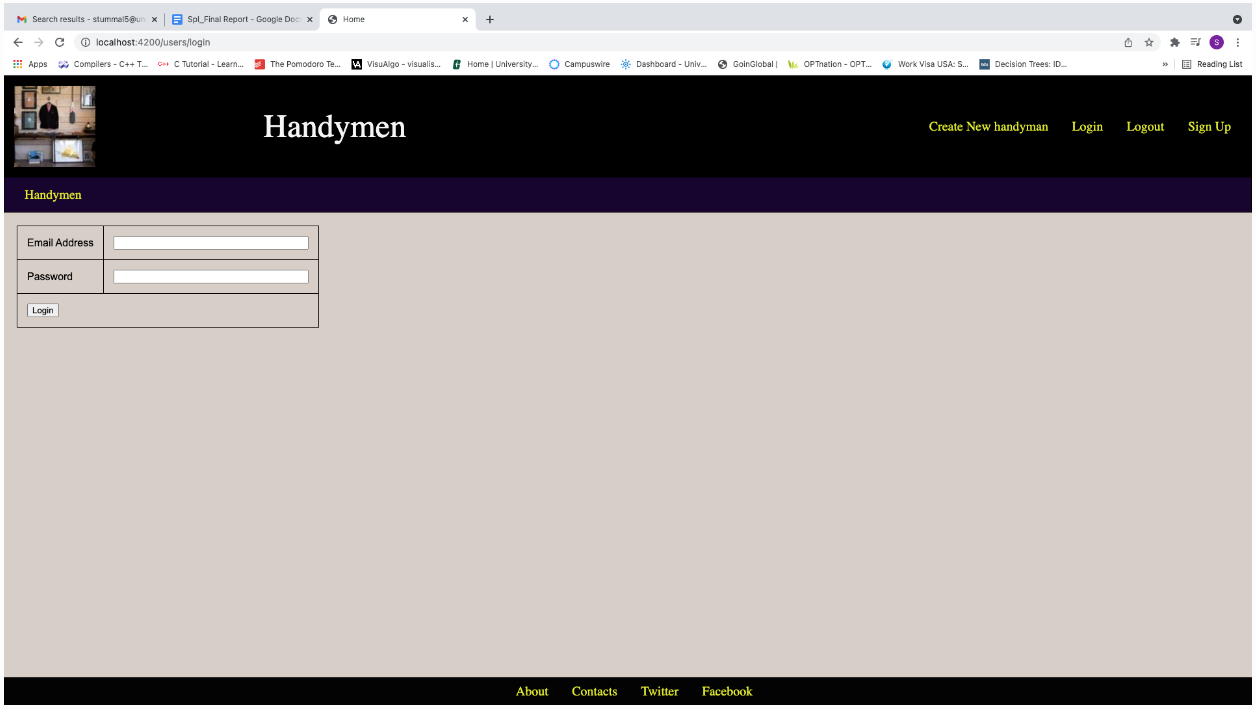
Handymen

Handymen

Welcome

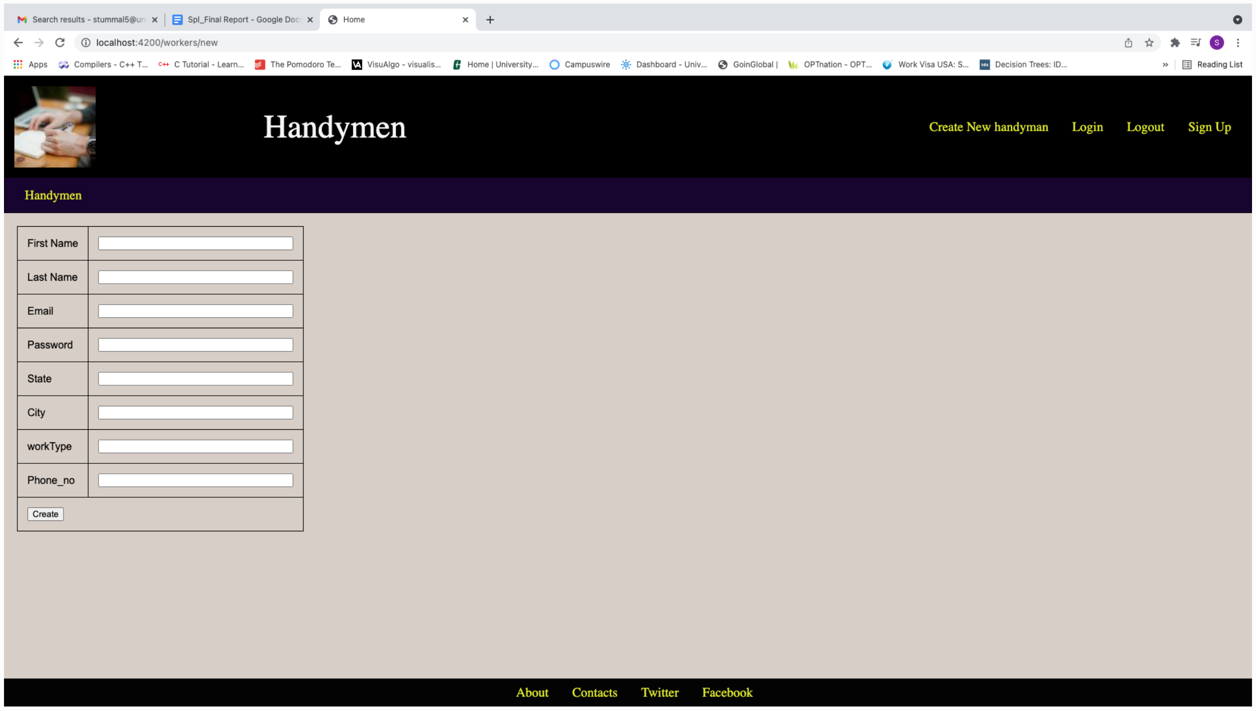
Name : Amy Smith	Work Type : Carpentry	Phone_no : 31234435	Email : asmith@gmail.com	Edit	Delete
Name : Jhon Adamas	Work Type : Plumbing	Phone_no : 234234425	Email : jadam@gmail.com	Edit	Delete
Name : Stark Tony	Work Type : plumbing	Phone_no : 324245345	Email : tony@gmail.com	Edit	Delete
Name : Mark Marcus	Work Type : Carperter	Phone_no : 9809231231	Email : marcus@xyz.com	Edit	Delete

Login Page



Email Address	
Password	

Create new Handymen



First Name	
Last Name	
Email	
Password	
State	
City	
workType	
Phone_no	

References:

https://go.dev/doc/faq#Is_Go_an_object-oriented_language

https://go.dev/ref/spec#Statements

https://go.dev/ref/spec

https://go.dev/doc/