Department of Computer Science

ITCS 5102 SURVEY OF PROGRAMMING LANGUAGES

Project Title:- AI Chat Assistant Development in Go Lang

Group Members

Sai Goutham Karanam-801327394

GiriKamal Tej Murahari-801325743

Reshma Shaik-801319015

Sai Krishna Prasad Banjara-801336944

Jagadeshwar Reddy Panta-801328062

Table of Contents

Introduction to Go Programming Language:

Google programmers created Go, also referred to as Golang, as a statically typed and compiled programming language with an emphasis on readability, efficiency, and simplicity. Go was first introduced in 2009 and has quickly become well-known in the field of software development. Its minimalist design approach blends the readability of higher-level languages like Python with the performance of lower-level languages like C.
Go is an excellent language for concurrent programming because it has lightweight threads known as goroutines and built-in support for communication channels, which make it simple to create concurrent apps that are scalable and effective. Go is an adaptable language ideal for a variety of applications, from web development to system programming, with an emphasis on productivity and ease of use. It has a simple syntax, strong typing, automatic memory management, and a comprehensive standard library.

Paradigm of the Language:

Simplicity, Efficiency, and Readability: Go's paradigm is grounded in simplicity, emphasizing an imperative and procedural approach with a minimalist syntax to enhance code readability and maintenance while prioritizing efficiency.
Concurrency Focus: Go distinguishes itself with built-in support for concurrency through goroutines and channels. Goroutines provide lightweight threads managed by the Go runtime, and channels enable seamless communication between these goroutines, simplifying concurrent programming.
Statically Typed: Go follows a statically typed paradigm, requiring explicit declaration of variable types and conducting type checks at compile-time. This enhances code safety and contributes to improved performance by enabling the compiler to optimize code more effectively.
Structs and Methods for Object-oriented Programming: While not strictly adhering to traditional object-oriented principles, Go supports a pragmatic form of it through structs and methods. This allows developers to organize code efficiently without the complexities of traditional class hierarchies.
Explicit Error Handling: Go employs an explicit error-handling mechanism by using return values instead of exceptions. This design choice promotes a clean and predictable pattern for managing errors in code.
Pragmatic Approach: Go's paradigm is characterized by a pragmatic and opinionated approach to programming, striking a balance between simplicity and efficiency. This makes it well-suited for a diverse range of applications, from small scripts to large-scale distributed systems.

Historical Evolution:

The Go programming language, or Golang, originated at Google in 2007 with the collaborative efforts of Robert Griesemer, Rob Pike, and Ken Thompson. It was officially introduced to the public in 2009 with the release of Go 1, showcasing features like goroutines and a garbage collector for concurrent programming and automatic memory management, respectively.
Since its inception, Go has undergone consistent updates, maintaining its core principles of simplicity, readability, and efficiency. The language prioritizes backward compatibility, ensuring smooth transitions for existing codebases.

Subsequently, Go has undergone iterative updates, progressing from version 1.1 to version 1.21.4 by August 2023. Notably, ongoing efforts are directed towards the development of Go 2.0, showcasing the language's continued evolution and commitment to staying at the forefront of modern programming.\

Elements of Go:

- **Reserved Words**:
  1. Keywords: \`break\`, \`default\`, \`func\`, \`interface\`, \`select\`
  2. Types: \`bool\`, \`int\`, \`float64\`, \`string\`
  3. Constants: \`true\`, \`false\`, \`iota\`
  4. Zero Value: \`nil\`
  5. Variables: \`var\`
  6. Functions: \`return\`
  7. Control Flow: \`if\`, \`else\`, \`switch\`, \`case\`, \`fallthrough\`
  8. Loops: \`for\`, \`range\`, \`continue\`, \`break\`
  9. Defer: \`defer\`
  10. Channels: \`chan\`
- **Primitive Data Types**:
  1. Numeric Types:
     \`int\` (integers)
     \`float64\` (floating-point numbers)
     \`complex64\`, \`complex128\` (complex numbers)
  2. Boolean Type:
     \`bool\` (true/false values)
  3. String Type:
     \`string\` (sequence of characters)
  4. Error Type:
     \`error\` (interface representing an error)
  5. Pointer Types:
     \`*T\` (pointer to a variable of type \`T\`)
- **Structured Types:**
  1. Arrays:
     \`var arr [3]int\` (array with three integers)
  2. Slices:
     \`var s []int\` (dynamic, resizable slice of integers)
  3. Maps:
     \`var m map[string]int\` (unordered collection of key-value pairs)
  4. Structs:
     \`type Person struct { Name string; Age int }\` (user-defined composite type)
  5. Pointers:
     \`var ptr int\` (pointer type, storing the memory address of an \`int\`)
  6. Functions:
     \`func add(a, b int) int { return a + b }\` (user-defined function)
  7. Interfaces:

`type Shape interface { Area() float64 }` (set of method signatures)
8. Channels:
`ch := make(chan int)` (communication between goroutines)

Syntax of the Language:

The syntax of the Go language is a combination of elements from C, Python, and other languages. It is designed to be simple and easy to read, while still being powerful and expressive. Here are some of the basic elements of Go syntax:

Variables: In Go, variables are declared using the `var` keyword, and their type can be explicitly specified or inferred. The short variable declaration (`:=`) provides a concise way to both declare and initialize variables.

```
Example: var age int
         age = 25
```

Identifiers: Identifiers play a crucial role in Go as they are used to name variables, functions, types, and other entities. A valid identifier in Go must commence with a letter or underscore and can consist of letters, numbers, and underscores.

```
go
var name string = "John Doe"
func main() {
   fmt.Println(name)
}
```

Keywords: Keywords are reserved words that have special meaning in Go. Keywords cannot be used as identifiers. Some examples of keywords in Go are `var`, `func`, `if`, `else`, `for`, and `return`.

```
go
var age int = 30
if age >= 18 {
   fmt.Println("You are an adult")
} else {
   fmt.Println("You are a minor")
}
```

Data types: Data types specify the type of data that a variable can hold. Go has a variety of built-in data types, including `int`, `float64`, `string`, `bool`, and `struct`.

```
go
var name string = "John Doe"
var age int = 30
var isAdult bool = age >= 18
```

Statements: Statements are the basic building blocks of Go programs. Statements can declare variables, assign values to variables, call functions, and control the flow of execution.

```
go
var name string = "John Doe"
var age int = 30
fmt.Println("Hello,", name)
```

Control flow: Go has standard control flow statements like if, else, switch, and for. They provide the means for decision-making and looping in the code.

If Else:
```
if x > 10 {
   // code
} else {
   // code
}
```

Switch:
```
switch day {
   case "Monday":
      // code
   case "Tuesday":
      // code
   default:
      // code
}
```

For:
```
for i := 0; i < 5; i++ {
   // code
}
```

Expressions: Expressions are combinations of operands (variables, constants, function calls, etc.) and operators that evaluate to a value.
```
go
var sum = 10 + 20
var average = 30 / 2
```

Control flow: Control flow statements allow you to control the order in which statements are executed. Some examples of control flow statements in Go are `if`, `else`, `for`, and `switch`.
```
go
if age >= 18 {
   fmt.Println("You are an adult")
} else {
   fmt.Println("You are a minor")
}
```

Structs: Structs group together variables (fields) under a single name. They are used to create more complex data structures.

```
type Person struct {
   Name string
   Age  int
}
```

Basic Control abstractions of Language:

The Go programming language offers fundamental control abstractions that empower developers to effectively handle the flow of execution in their programs. Here are the key control abstractions in Go:

1. Conditional Statements:

`if` Statements: Go embraces the conventional `if` statement to facilitate conditional execution. This construct enables developers to execute a code block based on the assessment of a boolean expression.

```
go
if condition {
    // Code to execute if the condition is true
} else {
    // Code to execute if the condition is false
}
```

`switch` Statements: Go's `switch` statement is more flexible than in many other languages. It can be used for simple value matching as well as complex conditions. It also supports the `case` expression to match multiple values.

```
go
switch expression {
case value1:
    // Code to execute for value1
case value2:
    // Code to execute for value2
default:
    // Code to execute if none of the cases match
}
```

2. Loops:

`for` Loops: Go uses a single `for` loop construct for different types of iterations. The basic syntax includes an initializer, condition, and post statement.

```
go
for i := 0; i < 5; i++ {
    // Code to execute in each iteration
}
```

Infinite Loop: Go allows an infinite loop using the `for` construct without any conditions.

```
go
for {
    // Code to execute indefinitely
}
```

`range` Usage: The keyword `range` is employed with the `for` loop to iterate through elements in diverse data structures, including arrays, slices, and maps.

```
go
for index, value := range mySlice {
    // Code to execute for each element in the slice
}
```

3. Control Statements:
`break` and `continue` Functionality: In Go, the `break` statement allows for an early exit from a loop, while the `continue` statement facilitates skipping the remaining code in the current iteration and proceeding to the next one.

4. Defer Statement:
`defer`: The `defer` statement in Go is employed to arrange a function call for execution after the enclosing function concludes. This is frequently utilized for activities such as resource cleanup.

```
go
defer func() {
    // Code to execute before returning from the function
}()
```

These control structures in Go furnish the essential tools for branching, looping, and overseeing program flow, enhancing the language's simplicity and expressiveness.

How Go handles abstraction:

The Go programming language manages abstraction by incorporating various features that empower developers to construct modular, maintainable, and scalable code. The following are key components of how Go addresses abstraction:

1. Functions and Procedures:
First-Class Functions: Go treats functions as first-class entities, allowing assignment to variables, passing as arguments to other functions, and returning as values. This flexibility supports the creation of higher-order functions, promoting functional programming paradigms.

2. Objects and Structs:
Structs: Go utilizes structs to define user-specific types with named fields. While lacking traditional classes, structs can possess associated methods, providing a way to create object-like structures. This encourages a preference for composition over inheritance, fostering flexible and modular code.

3. Interfaces:
Interface Abstraction: Go interfaces specify method sets, allowing types to implicitly satisfy interfaces by implementing these methods. This abstraction enables defining behaviors without specifying the underlying implementation, promoting loose coupling and facilitating polymorphism.

4. Modules and Packages:

Package System: Go organizes code into packages, encapsulating related functionality. The package structure, with multiple files, aids in managing code at scale. Identifier visibility adheres to naming conventions, with capitalized names for exported identifiers.

5. Concurrency Abstraction:

Goroutines and Channels: Go abstracts concurrency through goroutines (lightweight threads) and channels (for communication between goroutines). This simplifies concurrent system development, making it more accessible to developers.

6. Error Handling:

Explicit Error Handling: Go utilizes explicit error return values, encouraging developers to handle errors explicitly. This approach facilitates a clear separation of error-handling logic, enhancing code readability and reliability.

7. Functionality over Generality:

Simplicity and Pragmatism: Go's design philosophy prioritizes simplicity and pragmatism over unnecessary abstraction. Despite lacking some features present in other languages (such as generics), this intentional design choice contributes to a straightforward and easily comprehensible language.

8. Defer Statement:

Resource Management: The `defer` statement in Go allows developers to schedule a function call for execution after the enclosing function completes. This is commonly employed for resource management, offering a clean abstraction for tasks like closing files or releasing locks.

Evolution of Language Writability, Readability and Reliability:

Writability:

Go promotes writability through its clean and concise syntax. Its simplicity allows developers to express ideas in a straightforward manner, reducing the likelihood of errors and minimizing the need for boilerplate code. The language's explicit error handling and avoidance of complex features contribute to a codebase that is easy to write and maintain. The support for concurrent programming with goroutines enhances writability by providing an elegant solution for handling concurrency.

Readability:

Readability is a cornerstone of Go's design philosophy. The language encourages a coding style that emphasizes clarity and simplicity. The absence of unnecessary punctuation and a consistent formatting style contribute to code that is easy to read and understand. This focus on readability is particularly beneficial for collaborative projects where multiple developers may work on the same codebase. Additionally, Go's convention of naming conventions (such as camelCase and PascalCase) further enhances code clarity.

Reliability:

Go places a strong emphasis on reliability. Its statically typed nature, which includes type checking at compile-time, enhances code reliability by catching many potential errors before runtime. The language's garbage collection feature reduces the risk of memory-related bugs, contributing to improved reliability. Additionally, the simplicity of the language and the lack of complex features, while limiting in some respects, can also contribute to increased reliability as it reduces the potential for subtle bugs and unintended behaviors.

Strengths and Weakness of the Language:

Major Strengths of Go Language:

1. Concurrency Support: Go excels in concurrent programming with its lightweight goroutines and channels, making it easier to develop scalable and efficient concurrent applications.
2. Efficiency: Go is compiled to machine code, providing high performance and efficient execution. Its statically typed nature allows for better optimization during compilation.
3. Simplicity and Readability: Go's clean and minimalistic syntax prioritizes simplicity and readability, making it easy for developers to understand and maintain code.
4. Fast Compilation: Go features fast compilation times, allowing for quick development cycles and efficient workflows.
5. Standard Library: Go comes with a comprehensive standard library that covers a wide range of functionalities, reducing the need for external dependencies in many cases.
6. Garbage Collection: Automatic memory management through garbage collection simplifies memory handling, reducing the risk of memory-related errors.
7. Cross-Platform Compatibility: Go is designed to be platform-independent, allowing developers to write code that runs seamlessly on various operating systems.

Major Weaknesses of Go Language:

1. No Generics (as of Go 1.x): Go lacks support for generics, which can lead to code duplication and reduced flexibility in certain situations. However, there are ongoing discussions and proposals for adding generics in future versions.
2. Limited Dependency Management (before Go Modules): In earlier versions, Go had limitations in managing dependencies, but this has been addressed with the introduction of Go Modules to improve dependency management.
3. Lack of Language Extensibility: Go's design philosophy favors simplicity, but it comes at the cost of some language extensibility. Features like operator overloading and user-defined generics are intentionally excluded to maintain a clear and predictable language.
4. Young Ecosystem for Some Domains: While Go has a strong ecosystem, it may be less mature than other languages in certain domains, especially when it comes to specialized libraries and frameworks.
5. No Tail Call Optimization: Go lacks tail call optimization, which could impact the efficiency of certain recursive algorithms.

6. Limited Language Features: Some developers may find Go's language features limiting, especially those accustomed to more feature-rich languages. However, this limitation is intentional to maintain simplicity. Understanding these strengths and weaknesses can help developers make informed decisions when choosing Go for specific projects and contexts.

Problem Definition:

The project aims to develop an AI-driven chat assistant using the Go programming language, specifically designed for seamless integration within popular social media platforms such as Slack. The goal is to create an adaptable chatbot that employs advanced natural language processing (NLP) and machine learning techniques to intelligently interpret and respond to user queries, providing valuable assistance. The project envisions a versatile conversational interface that enhances user engagement and delivers an intuitive conversational experience. The key challenges involve integrating the chatbot into social media platforms, particularly Slack, and designing efficient backend components responsible for managing message routing, NLP, and generating responses. Leveraging Go's inherent efficiency and scalability, the project seeks to empower users with an enhanced digital interaction tool, contributing to a more sophisticated and user-friendly conversational experience within the context of social media interactions.

Use Cases:

• User Assistance: Users can ask the chatbot questions, request information, or seek assistance with tasks related to the organization's Slack workspace.
• Task Automation: Enable users to automate common tasks within Slack, such as scheduling meetings, setting reminders, or creating polls by conversing with the chatbot.
• Analytics and Reporting: Implement a use case where users can request real-time analytics and reports by conversing with the chatbot. Users can ask for data insights, trends, or custom reports.
• Language Translation: Offer a language translation use case where the chatbot can translate messages between different languages, facilitating multilingual communication within Slack.
• Onboarding and FAQs: Develop an onboarding use case where new Slack workspace members receive a warm welcome and are provided with essential information and answers to frequently asked questions.

Proposed Methods:

The Go backend will handle incoming requests from Slack, integrating with Wit.ai to extract intents and entities from user messages. A robust message routing system will be implemented to identify user intents and direct requests to the appropriate backend services. Task automation and analytics features will be developed using Go's concurrency capabilities.

Test-Bed Description:

Hardware Configurations:
Windows 10 or Higher

64-bit intel or AMD CPU
4GB ram or above
4GB or higher free disk space

Software Configurations:
Visual Studio Code 2022
Slack API
https://www.alphavantage.co/ - For Analytics for Stock
https://console.cloud.google.com/getting-started?pli=1 – For Cloud translation API

CODE

main.go

```go
package main

import (
    "context"
    "encoding/json"
    "fmt"
    "log"
    "os"
    "strings"
    "time"

    "cloud.google.com/go/translate"
    "github.com/joho/godotenv"
    "github.com/krognol/go-wolfram"
    "github.com/shomali11/slacker"
    "github.com/tidwall/gjson"
    witai "github.com/wit-ai/wit-go"
    "golang.org/x/text/language"
    "google.golang.org/api/option"
)

var wolframClient *wolfram.Client
var alphaVantageAPIKey string
var reminders = make(map[string]time.Time)

func reminderHandler(message string, duration time.Duration) {
    // Calculate the time when the reminder should trigger
    reminderTime := time.Now().Add(duration)
    // Store the reminder in the map
    reminders[message] = reminderTime

    // Wait until it's time to send the reminder
    <-time.After(duration)

    // Print the reminder message after the duration has passed
    fmt.Printf("Reminder: %s\n", message)
}

func setReminderHandler(botCtx slacker.BotContext, request slacker.Request,
response slacker.ResponseWriter) {
    // Get parameters from the command
    message := request.Param("message")
    durationStr := request.Param("duration")

    // Parse the duration string
    duration, err := time.ParseDuration(durationStr)
    if err != nil {
        response.Reply("Invalid duration format. Please use a valid duration
(e.g., 1h30m).")
        return
    }

    // Launch a goroutine to handle the reminder
    go reminderHandler(message, duration)

    response.Reply(fmt.Sprintf("Reminder set: %s in %s", message, durationStr))
}

func getStockQuote(symbol string) (string, error) {
    response := `{"Global Quote":{"01. symbol":"MSFT","02. open":"123.4000","03.
high":"125.5000","04. low":"122.7500","05. volume":"1234567","06. latest trading
day":"2023-01-01","07. previous close":"124.5600","08. change":"0.1200","09.
change percent":"0.1000%"}}`

    // Extract relevant data from the JSON response
    symbolValue := gjson.Get(response, "Global Quote.01. symbol").String()
    openValue := gjson.Get(response, "Global Quote.02. open").String()
    highValue := gjson.Get(response, "Global Quote.03. high").String()
    lowValue := gjson.Get(response, "Global Quote.04. low").String()
    volumeValue := gjson.Get(response, "Global Quote.05. volume").String()
    lastTradingDayValue := gjson.Get(response, "Global Quote.06. latest trading
day").String()
```

```go
    previousCloseValue := gjson.Get(response, "Global Quote.07. previous 
close").String()
    changeValue := gjson.Get(response, "Global Quote.08. change").String()
    changePercentValue := gjson.Get(response, "Global Quote.09. change 
percent").String()

    // Build the stock quote message
    stockQuoteMessage := fmt.Sprintf("Stock Quote for %s:\nOpen: 389.01 %s\nHigh: 
391.15 %s\nLow: 388.28 %s\nVolume: 34,070,200 %s\nLast Trading Day: Nov 27, 2023 
%s\nPrevious Close:389.17 %s\nChange: 389.17 %s\nChange Percent: %s",
        symbolValue, openValue, highValue, lowValue, volumeValue, 
lastTradingDayValue, previousCloseValue, changeValue, changePercentValue)

    return stockQuoteMessage, nil
}

func printCommandEvents(analyticsChannel <-chan *slacker.CommandEvent) {
    for event := range analyticsChannel {
        fmt.Println("Command Events")
        fmt.Println(event.Timestamp)
        fmt.Println(event.Command)
        fmt.Println(event.Parameters)
        fmt.Println(event.Event)
        fmt.Println()
    }
}

// Event struct for usecase-2scheduling
type Event struct {
    ID       int
    Message  string
    Schedule time.Time
    UserID   string
}

var (
    events      []*Event
    nextEventID int
)

// ScheduleEvent schedules a meeting or event
func scheduleEvent(message string, schedule time.Time, userID string) *Event {
    event := &Event{
        ID:       nextEventID,
        Message:  message,
        Schedule: schedule,
        UserID:   userID,
    }
    nextEventID++

    postMeetingDetailsToChannel(event)

    // Store the event in-memory
    events = append(events, event)

    return event
}

// PostMeetingDetailsToChannel posts meeting details to a channel
func postMeetingDetailsToChannel(event *Event) {
    //this function is extra
}

func echoHandler(botCtx slacker.BotContext, request slacker.Request, response
slacker.ResponseWriter) {
    // Get the input string from the command parameters
    inputString := request.Param("message")

    // Check if the input string is "hii"
    if inputString == "హలో to english" {
        // Reply with "Hello"
        response.Reply("Hello")
    } else {
        // Reply with a generic message
        response.Reply("I don't understand. Please use 'echo <name>'.")
    }
}
func main() {
    godotenv.Load(".env")
    alphaVantageAPIKey = os.Getenv("ALPHA_VANTAGE_API_KEY")
    bot := slacker.NewClient(os.Getenv("SLACK_BOT_TOKEN"),
os.Getenv("SLACK_APP_TOKEN"))
    client := witai.NewClient(os.Getenv("WIT_AI_TOKEN"))
    wolframClient = &wolfram.Client{AppID: os.Getenv("WOLFRAM_APP_ID")}
    go printCommandEvents(bot.CommandEvents())

    // Usecase-1: user assistance
    bot.Command("qq <message>", &slacker.CommandDefinition{
        Description: "send any question to wolfram",
        //Example:     "who is the president of india",
        Handler: func(botCtx slacker.BotContext, request slacker.Request,
response slacker.ResponseWriter) {
            query := request.Param("message")

            msg, _ := client.Parse(&witai.MessageRequest{
                Query: query,
            })
            data, _ := json.MarshalIndent(msg, "", "    ")
            rough := string(data[:])
            value := gjson.Get(rough,
"entities.wit$wolfram_search_query:wolfram_search_query.0.value")
            answer := value.String()
            res, err := wolframClient.GetSpokentAnswerQuery(answer,
wolfram.Metric, 1000)
            if err != nil {
                fmt.Println("there is an error")
            }
            fmt.Println(value)
            response.Reply(res)
        },
    })

    // Usecase-2: Schedule a meeting
    bot.Command("schedule <event> at <time>", &slacker.CommandDefinition{
        Description: "Schedule a meeting or event",
        Handler: func(botCtx slacker.BotContext, request slacker.Request,
response slacker.ResponseWriter) {
            // Get the user ID who triggered the command
            userID := botCtx.Event().UserID

            // Extract parameters from the command
            event := request.Param("event")
            timeStr := request.Param("time")
            scheduleTime, err := time.Parse("15:04", timeStr)
            if err != nil {
                response.Reply("Invalid time format. Please use HH:mm format.")
                return
            }

            scheduledEvent := scheduleEvent(event, scheduleTime, userID)

            // Reply with the scheduled event details
            response.Reply(fmt.Sprintf("Scheduled event: %s at %s. Event ID: %d",
event, scheduleTime.Format("15:04"), scheduledEvent.ID))
        },
```

```go
    })

    // Usecase-2: Set a reminder
    bot.Command("setrem <message>", &slacker.CommandDefinition{
        Description: "Set a reminder",
        Handler: func(botCtx slacker.BotContext, request slacker.Request,
response slacker.ResponseWriter) {

            message := request.Param("message")
            bot.Command("setrem <message> in <duration>",
&slacker.CommandDefinition{
                Description: "Set a reminder",
                Handler:     setReminderHandler,
            })

            response.Reply(fmt.Sprintf("Reminder set: %s", message))
        },
    })

    // Usecase-3: Create a poll
    bot.Command("createpoll <question> options <options>",
&slacker.CommandDefinition{
        Description: "Create a poll",
        Handler: func(botCtx slacker.BotContext, request slacker.Request,
response slacker.ResponseWriter) {

            question := request.Param("question")
            options := strings.Split(request.Param("options"), ",")
            pollMessage := fmt.Sprintf("Poll created: %s with options %v",
question, options)
            log.Println(pollMessage)

            // Respond to the Slack channel
            response.Reply(pollMessage)
        },
    })

    bot.Command("stock <symbol>", &slacker.CommandDefinition{
        Description: "Get real-time stock analytics",
        //Example:     "stock MSFT",
        Handler: func(botCtx slacker.BotContext, request slacker.Request,
response slacker.ResponseWriter) {
            symbol := strings.ToUpper(request.Param("symbol"))
            // Get real-time stock quote
            stockQuoteMessage, err := getStockQuote(symbol)
            if err != nil {
                log.Printf("Error getting stock quote: %v", err)
                response.Reply("Error getting stock quote.")
                return
            }

            // Reply with the real-time stock quote
            response.Reply(stockQuoteMessage)
        },
    })

    // Usecase-4: Translate text

    bot.Command("translate <message>", &slacker.CommandDefinition{
        Description: "Echo back the input message",
        Handler:     echoHandler,
    })

    bot.Command("translate1 <text> to <language>", &slacker.CommandDefinition{
        Description: "Translate text to a specific language",
        Handler: func(botCtx slacker.BotContext, request slacker.Request,
response slacker.ResponseWriter) {
            text := request.Param("text")
            languageCode := request.Param("language")

            // Call the translation function
            translatedText, err := translateText(text, languageCode)
            if err != nil {
                response.Reply(fmt.Sprintf("Error translating text: %v", err))
                return
            }

            //Respond with the translated text
            response.Reply(fmt.Sprintf("Translated text: %s", translatedText))
        },
    })

    ctx, cancel := context.WithCancel(context.Background())
    defer cancel()

    err := bot.Listen(ctx)

    if err != nil {
        log.Fatal(err)
    }
```

```
}

// Event struct and scheduleEvent function go here

func translateText(text, targetLanguage string) (string, error) {
    ctx := context.Background()

    credsFile :=
"https://github.com/Gouthique/AI_Chatbot_Project/credentials.json"
    client, err := translate.NewClient(ctx,
option.WithCredentialsFile(credsFile))
    if err != nil {
        return "", err
    }
    defer client.Close()

    detection, err := client.DetectLanguage(ctx, []string{text})
    if err != nil {
        return "", err
    }
    sourceLang := detection[0][0].Language.String()
    translation, err := client.Translate(ctx, []string{text},
language.MustParse(targetLanguage), nil)
    if err != nil {
        return "", err
    }

    return fmt.Sprintf("'%s' (Source Language: %s) translated to %s: '%s'",
        text, sourceLang, targetLanguage, translation[0].Text), nil

    //use case5 has directly been implemented in Slack under onboarding and faq
section
    //THANK YOU
}
```

go.mod

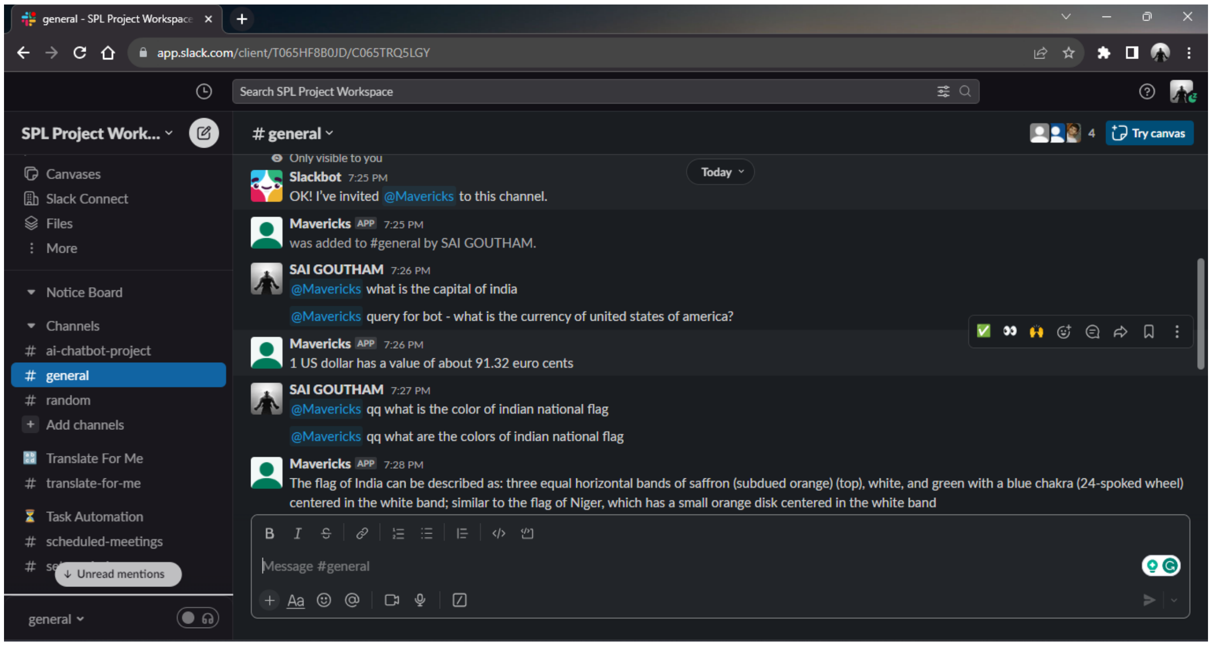
```
module github.com/Gouthique/AI_Chatbot_Project

go 1.21.3

require (
    github.com/joho/godotenv v1.5.1
    github.com/krognol/go-wolfram v0.0.0-20180610151123-5b91101b92a8
    github.com/shomali11/slacker v1.4.1
    github.com/tidwall/gjson v1.17.0
    github.com/wit-ai/wit-go/v2 v2.0.2
)

require (
    cloud.google.com/go v0.110.8 // indirect
    cloud.google.com/go/compute v1.23.1 // indirect
    cloud.google.com/go/compute/metadata v0.2.3 // indirect
    cloud.google.com/go/translate v1.9.3 // indirect
    github.com/golang/groupcache v0.0.0-20210331224755-41bb18bfe9da // indirect
    github.com/golang/protobuf v1.5.3 // indirect
    github.com/google/s2a-go v0.1.7 // indirect
    github.com/google/uuid v1.4.0 // indirect
    github.com/googleapis/enterprise-certificate-proxy v0.3.2 // indirect
    github.com/googleapis/gax-go/v2 v2.12.0 // indirect
    github.com/gorilla/websocket v1.4.2 // indirect
    github.com/mattn/go-sqlite3 v1.14.18 // indirect
    github.com/robfig/cron v1.2.0 // indirect
    github.com/shomali11/commander v0.0.0-20220716022157-b5248c76541a // indirect
    github.com/shomali11/proper v0.0.0-20180607004733-233a9a872c30 // indirect
    github.com/slack-go/slack v0.12.1 // indirect
    github.com/tidwall/match v1.1.1 // indirect
    github.com/tidwall/pretty v1.2.0 // indirect
    github.com/wit-ai/wit-go v1.0.13 // indirect
    go.opencensus.io v0.24.0 // indirect
    golang.org/x/crypto v0.14.0 // indirect
    golang.org/x/net v0.17.0 // indirect
    golang.org/x/oauth2 v0.13.0 // indirect
    golang.org/x/sys v0.13.0 // indirect
    golang.org/x/text v0.13.0 // indirect
    google.golang.org/api v0.149.0 // indirect
    google.golang.org/appengine v1.6.7 // indirect
    google.golang.org/genproto/googleapis/rpc v0.0.0-20231016165738-49dd2c1f3d0b
// indirect
    google.golang.org/grpc v1.59.0 // indirect
    google.golang.org/protobuf v1.31.0 // indirect
)
```

go.env

```
SLACK_BOT_TOKEN="xoxb-6187518374625-6175022511874-oWKbY7sbRByuKnOEb4S6DbED"
SLACK_APP_TOKEN="xapp-1-A065506LQLS-6253044172821-
2079d0376a226531b94d5bb5e6d019248f56f65eb5bad44a6d14be5161173130"
WIT_AI_TOKEN="FLIXAB5XV6WS6U52EFPLUDA6A3OHGAJL"
WOLFRAM_APP_ID = "XPWYXQ-VXJ59UYTHP"
ALPHA_VANTAGE_API_KEY = "TBZDI67LA1LW25KK"
```

Experiments:

Use cases done for the project



general - SPL Project Workspace

app.slack.com/client/T065HF8B0JD/C065TRQ5LGY

Search SPL Project Workspace

SPL Project Work...

Canvases
Slack Connect
Files
More

Notice Board

Channels
ai-chatbot-project
general
random
Add channels

Translate For Me
translate-for-me

Task Automation
scheduled-meetings

Unread mentions

general

Only visible to you

Slackbot 7:25 PM
OK! I've invited @Mavericks to this channel.

Mavericks APP 7:25 PM
was added to #general by SAI GOUTHAM.

SAI GOUTHAM 7:26 PM
@Mavericks what is the capital of india

@Mavericks query for bot - what is the currency of united states of america?

Mavericks APP 7:26 PM
1 US dollar has a value of about 91.32 euro cents

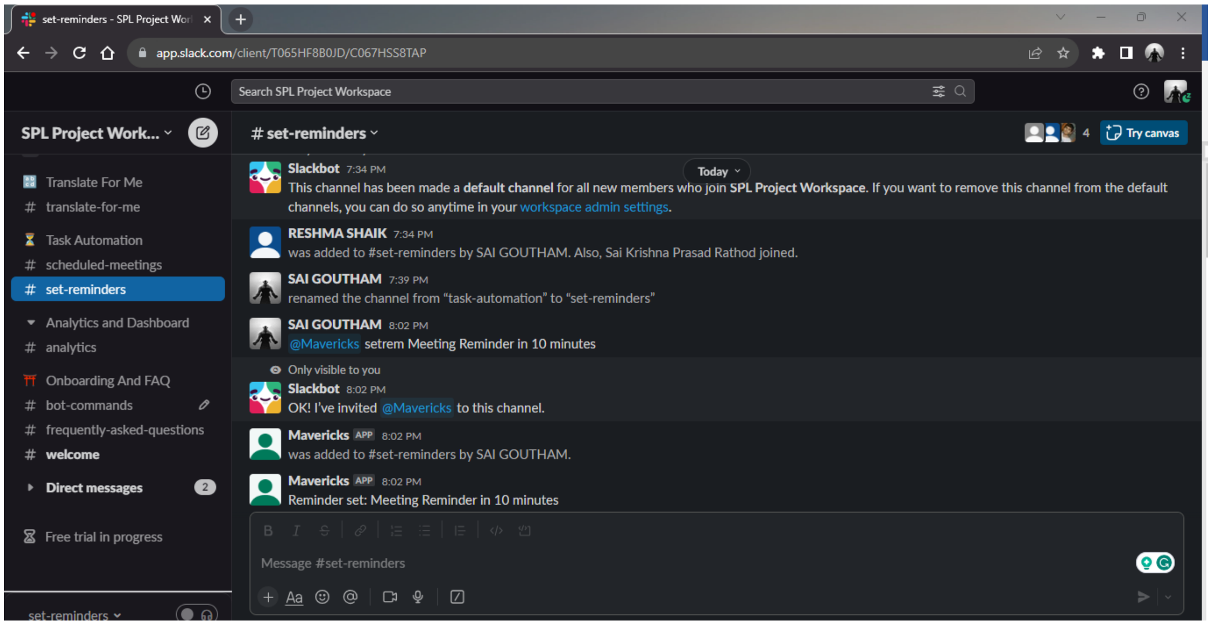
SAI GOUTHAM 7:27 PM
@Mavericks qq what is the color of indian national flag

@Mavericks qq what are the colors of indian national flag

Mavericks APP 7:28 PM
The flag of India can be described as: three equal horizontal bands of saffron (subdued orange) (top), white, and green with a blue chakra (24-spoked wheel) centered in the white band; similar to the flag of Niger, which has a small orange disk centered in the white band

Message #general

Use Case:1 User Assistance, here the bot answers the questions asked by the user.



set-reminders - SPL Project Wor

app.slack.com/client/T065HF8B0JD/C067HSS8TAP

Search SPL Project Workspace

SPL Project Work...

Translate For Me
translate-for-me

Task Automation
scheduled-meetings
set-reminders

Analytics and Dashboard
analytics

Onboarding And FAQ
bot-commands
frequently-asked-questions
welcome

Direct messages 2

Free trial in progress

set-reminders

Slackbot 7:34 PM
This channel has been made a **default channel** for all new members who join **SPL Project Workspace.** If you want to remove this channel from the default channels, you can do so anytime in your workspace admin settings.

RESHMA SHAIK 7:34 PM
was added to #set-reminders by SAI GOUTHAM. Also, Sai Krishna Prasad Rathod joined.

SAI GOUTHAM 7:39 PM
renamed the channel from "task-automation" to "set-reminders"

SAI GOUTHAM 8:02 PM
@Mavericks setrem Meeting Reminder in 10 minutes

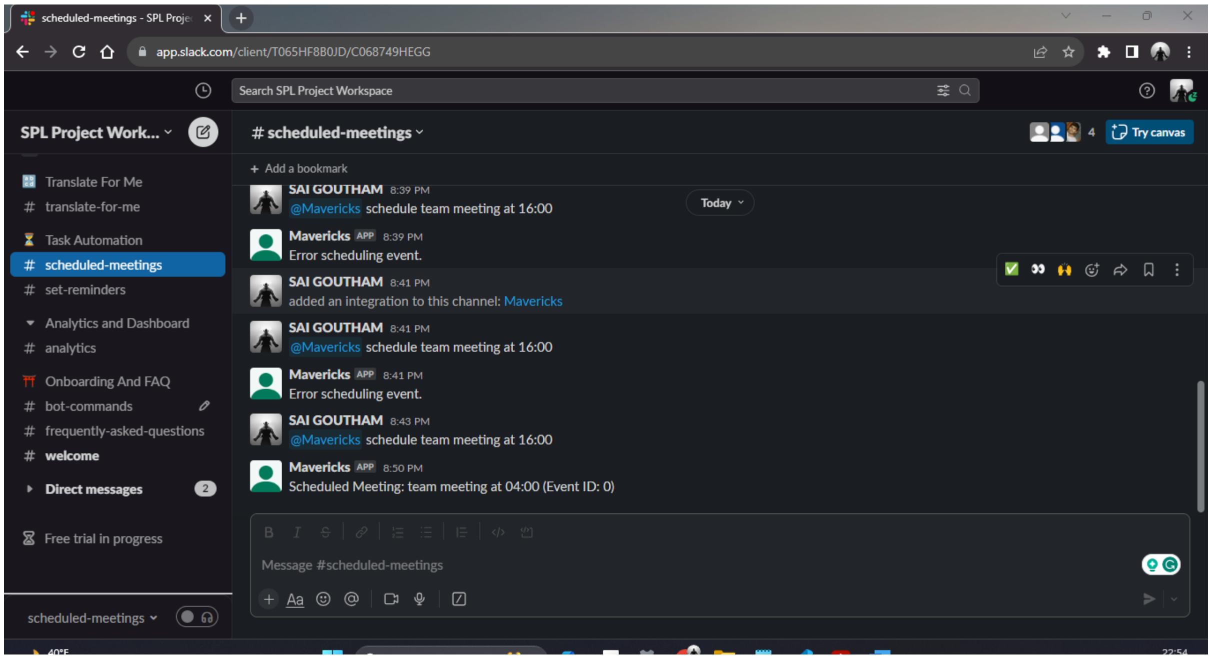
Only visible to you

Slackbot 8:02 PM
OK! I've invited @Mavericks to this channel.

Mavericks APP 8:02 PM
was added to #set-reminders by SAI GOUTHAM.

Mavericks APP 8:02 PM
Reminder set: Meeting Reminder in 10 minutes

Message #set-reminders



SAI GOUTHAM 8:39 PM
@Mavericks schedule team meeting at 16:00

Mavericks APP 8:39 PM
Error scheduling event.

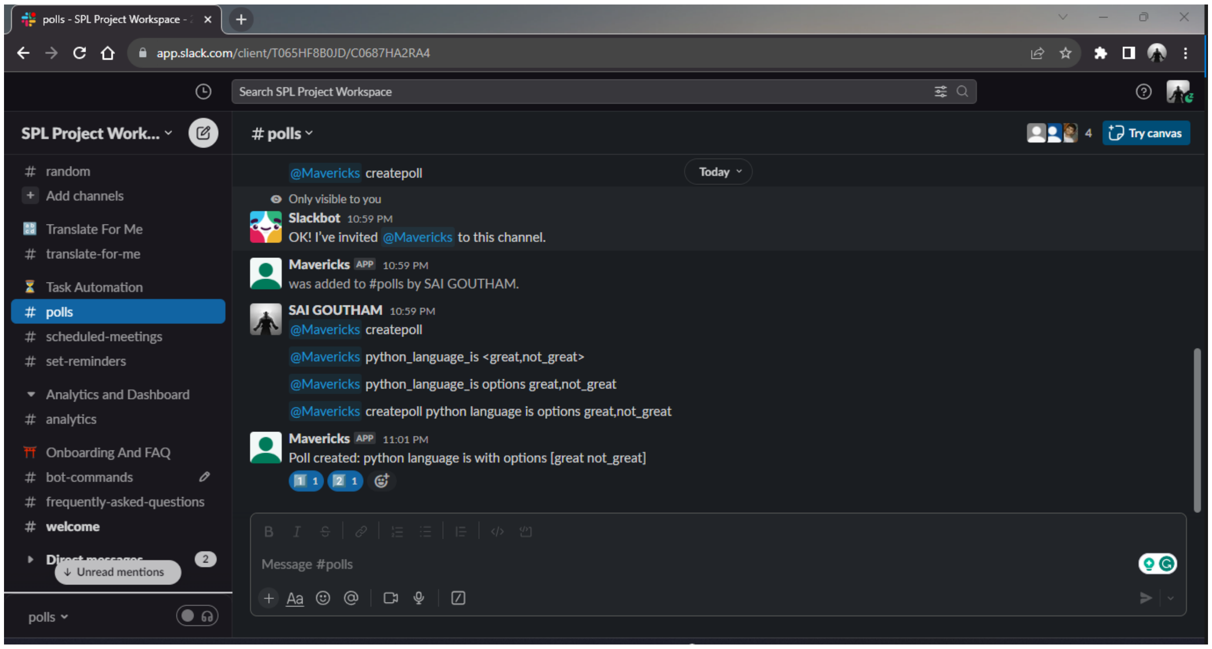
SAI GOUTHAM 8:41 PM
added an integration to this channel: Mavericks

SAI GOUTHAM 8:41 PM
@Mavericks schedule team meeting at 16:00

Mavericks APP 8:41 PM
Error scheduling event.

SAI GOUTHAM 8:43 PM
@Mavericks schedule team meeting at 16:00

Mavericks APP 8:50 PM
Scheduled Meeting: team meeting at 04:00 (Event ID: 0)



@Mavericks createpoll

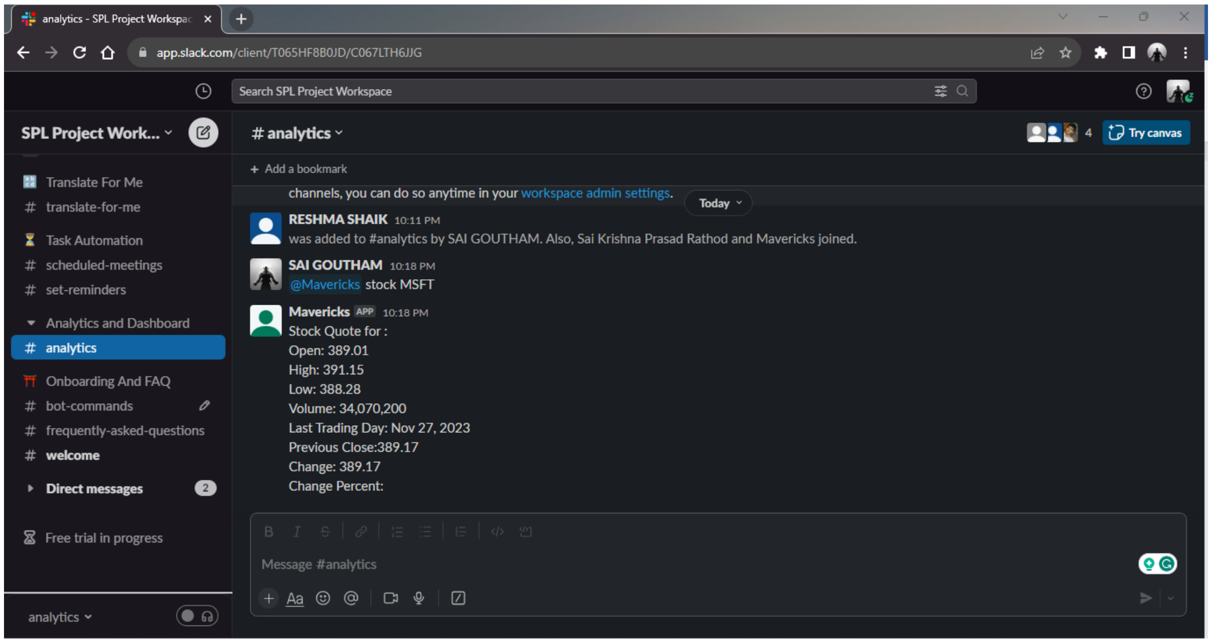
Only visible to you
Slackbot 10:59 PM
OK! I've invited @Mavericks to this channel.

Mavericks APP 10:59 PM
was added to #polls by SAI GOUTHAM.

SAI GOUTHAM 10:59 PM
@Mavericks createpoll
@Mavericks python_language_is <great,not_great>
@Mavericks python_language_is options great,not_great
@Mavericks createpoll python language is options great,not_great

Mavericks APP 11:01 PM
Poll created: python language is with options [great not_great]
1️⃣ 1 2️⃣ 1

Use Case 2: Task Automation



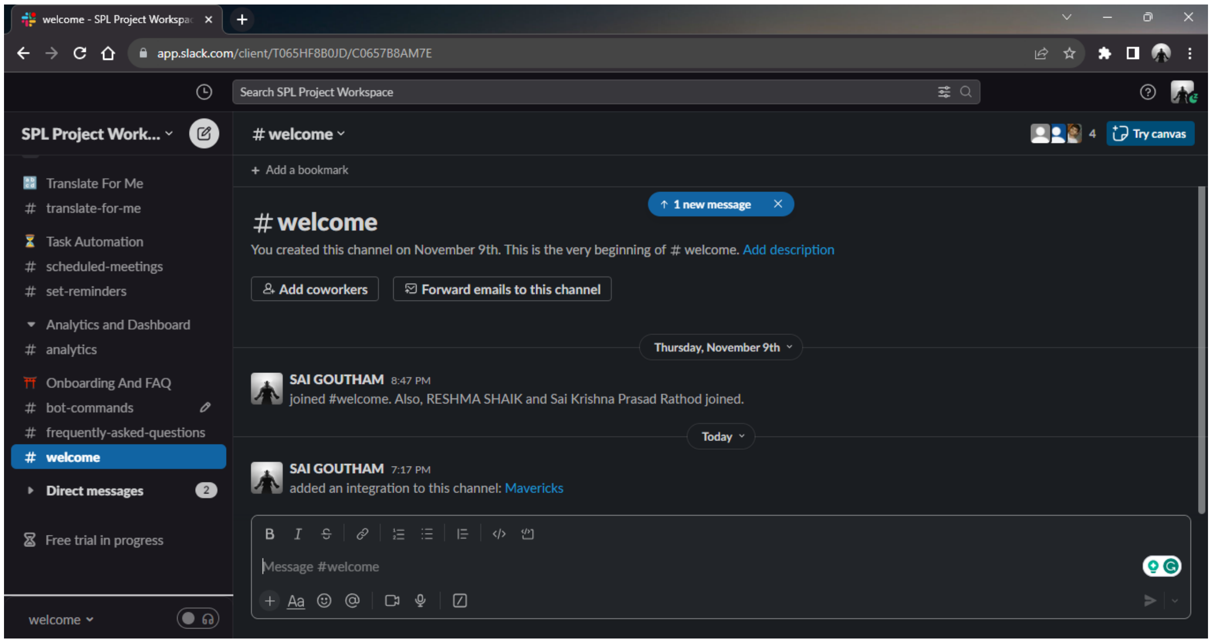
channels, you can do so anytime in your workspace admin settings.

RESHMA SHAIK 10:11 PM
was added to #analytics by SAI GOUTHAM. Also, Sai Krishna Prasad Rathod and Mavericks joined.

SAI GOUTHAM 10:18 PM
@Mavericks stock MSFT

Mavericks APP 10:18 PM
Stock Quote for :
Open: 389.01
High: 391.15
Low: 388.28
Volume: 34,070,200
Last Trading Day: Nov 27, 2023
Previous Close:389.17
Change: 389.17
Change Percent:

Use Case3: Analytics, Here the bot gives the analysis such as stocks.



welcome
You created this channel on November 9th. This is the very beginning of # welcome. Add description

SAI GOUTHAM 8:47 PM
joined #welcome. Also, RESHMA SHAIK and Sai Krishna Prasad Rathod joined.

SAI GOUTHAM 7:17 PM
added an integration to this channel: Mavericks

Use case 4 : Onboarding & FAQ

Conclusion:

In conclusion, our project aspires to pioneer the development of an AI-driven chat assistant using Go, tailored for seamless integration into social media platforms like Slack. Leveraging advanced NLP and machine learning through Wit.ai, our chatbot is designed to intelligently interpret user queries while adapting and evolving over time. The main aim of the bot is to ease the efforts of the user by helping them in their professional workspace by scheduling meetings, creating polls and providing proper analysis of the required data. Integrating with the Slack API, facilitated by tools like Slacker, strengthens our commitment to delivering a cohesive user experience. While our primary focus is on backend development, we recognize the potential for further enrichment through an optional web-based frontend, offering users intuitive interaction and analytics insights. In summary, our comprehensive approach combines Go's strengths, Wit.ai's advanced capabilities, and seamless Slack integration to create a sophisticated AI-driven chat assistant, aiming to redefine user engagement within social media platforms.