ITCS 5102
Survey of Programming Languages

<u>Term Project Report</u>

Project Title: Expense Ease

Team Members:
Ayushi Mundra (801208050)
Harsh Raval (801257980)
Shivashish Naramdeo (801208044)
Urvashi Murari (801205124)
Vidit Sethi (801203308)

Introduction

Go is a programming language that is designed for systems programming, meaning that it is used to build software that interacts closely with computer hardware and systems. One of the key features of Go is its explicit support for concurrent programming, which allows multiple tasks to be executed simultaneously. This can help improve the performance and responsiveness of software applications.

Motivation for choosing Go Programming Language:

All of our team members possess expertise in various programming languages such as Python, Javascript, C, C#, C++, Java, etc. However, none of us actually ever worked on or built projects using Go Programming Language. As this would be a great opportunity to learn and understand new technologies, we decided to develop this project using Go Programming Language. Overall, learning and using Go for our project can be a rewarding experience for your team, and can lead to new opportunities for growth and development in your careers as software developers.

Language Specifications

Language Paradigm

GoLang is a highly versatile programming language that can cater to a wide range of user needs due to its multi-paradigm nature.

- Object Oriented
- Imperative
- Concurrent programming

History

The Go programming language was created by a team of three engineers at Google, led by Robert Griesemer, Rob Pike, and Ken Thompson. The development of Go began in 2007, and the first public release was made in November 2009.

Go was designed to be a modern programming language that would address the shortcomings of existing languages, such as C and C++. The goal was to create a language that was easy to use, fast, and efficient, with a focus on concurrency and scalability. Since its release, Go has become increasingly popular, particularly in the field of web development. It is now used by a wide range of companies and organizations, including Google, Dropbox, Uber, and Netflix.

Elements of Language

- **Reserved Words**

  Reserved words of any programming language plays an important role in the structure of any language. These keywords are reserved for their specific uses within the language and cannot be used as variable names or identifiers. Go have various reserved words, here are all reserved words of go language:

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

- **Data Types**
  There are numerous data types which go supports. They are as follows
  bool: a boolean type that represents either true or false.
  int, int8, int16, int32, int64: signed integer types with different sizes.
  uint, uint8 (or byte), uint16, uint32, uint64, uintptr: unsigned integer types with different sizes.
  float32, float64: floating-point types for representing decimal numbers.
  complex64, complex128: complex number types.
  string: a sequence of characters.

- **Structured Data Types**
  Array, Slice, Struct, Map, Channel

Basic Syntax

Basic syntax rules of Go programming language:

- Go programs are made up of packages. Each package contains one or more source files. The package declaration is always the first line of a Go source file and specifies the name of the package.
- The main package must contain a function called main, which is the entry point of the program. The main function does not take any arguments and does not return any values.
- Go uses curly braces {} to delimit blocks of code.
- Go uses semicolons ; to separate statements on the same line, but they are not necessary unless you want to put multiple statements on one line.
- Go variables are declared using the var keyword, followed by the variable name and type. Alternatively, variables can also be declared and initialized in a single line using the := operator.
- Go functions are defined using the func keyword, followed by the function name, any parameters, and the return type (if any).

Basic Control Abstractions

- **Conditional Control Statements**
  Go provides several control statements that allow us to divert the flow of execution in your code. The control statements in Go include:

  ➢ **if statement:** The if statement is used to execute a block of code if a condition is true.
  Syntax:
  ```
  if condition {
     // code to be executed
  ```

```
} else {
   // code to be executed if the condition is false
}
```

- **switch statement:** The switch statement allows you to execute different blocks of code based on the value of a variable or expression.
  Syntax:
  ```
  switch variable/expression {
  case value1:
     // code to be executed if the value matches value1
  case value2:
     // code to be executed if the value matches value2
  ...
  default:
     // code to be executed if no value matches
  }
  ```

- **for loop:** The for loop allows you to execute a block of code a certain number of times or iterate over a range of values.
  Syntax:
  ```
  for initialization; condition; increment {
     // code to be executed
  }
  ```

- **break statement:** The break statement is used to exit a loop prematurely.
  Syntax:
  ```
  for {
     // code to be executed
     if condition {
        break
     }
  }
  ```

- **continue statement:** The continue statement is used to skip the current iteration of a loop and continue with the next iteration.
  Syntax:
  ```
  for {
     // code to be executed
     if condition {
        continue
     }
     // code to be executed if the condition is false
  }
  ```

These are the main control statements available in Go that allow you to alter the flow of execution in your code.

- **Loops**
  Go provides us with three types of loops to use in our program:

  - **for loop:** The for loop is the most commonly used loop in Go. It allows you to iterate over a range of values, such as an array or a slice, or to execute a block of code a certain number of times.
    Syntax:
    ```
    for initialization; condition; increment {
       // code to be executed
    }
    ```
  - **while loop:** The while loop is used when you want to repeatedly execute a block of code as long as a condition is true.
    Syntax:
    ```
    for condition {
       // code to be executed
    }
    ```
  - **do-while loop:** Go doesn't have a built-in do-while loop, but you can simulate it using a for loop.
    Syntax:
    ```
    for {
       // code to be executed
       if condition {
          break
       }
    }
    ```

Abstraction

- **Interface**
  Go supports abstraction through interfaces, which are a way to define a set of methods that a type must implement. An interface specifies a behavior without providing any implementation details.
  In Go, interfaces are defined using the interface keyword, followed by a set of method signatures.
  Example:
  ```
  type Animal interface {
     Speak() string
  }

  type Dog struct {}
  ```

```
func (d Dog) Speak() string {
    return "Woof!"
}
```

- **Functions**

  In Go, functions are declared using the func keyword, followed by the function name, any parameters the function takes (in parentheses), and the return type (if any).

  Here's an example of a simple function that takes two integers as parameters and returns their sum:

  ```
  func add(x int, y int) int {
      return x + y
  }
  ```

  Go functions can have multiple return values. Here's an example of a function that returns both the quotient and remainder of dividing two integers:

  ```
  func divide(dividend int, divisor int) (int, int) {
      quotient := dividend / divisor
      remainder := dividend % divisor
      return quotient, remainder
  }
  ```

Evaluation of Language

- **Writability**

  Go language enables programmers to write programs that are clear, concise, and perform faster. Its ease of use and understandability make it a writable language.
- **Readability**

  The ease of writing, scalability, and simplicity of Go language make it highly readable.
- **Reliability**

  Reliability of a programming language is determined by various factors such as error management, impact of errors, and language efficiency. Go language is highly efficient and offers easy error handling. However, at times, managing errors can become challenging and it may lack a fluid interface compared to other programming languages.

Strengths

- **Simplicity:** Go is designed to be simple and easy to learn, with a concise syntax and a small set of keywords.
- **Concurrency:** Go has built-in support for concurrency, which makes it easy to write efficient and scalable code for multi-core and distributed systems.
- **Garbage collection:** Go has automatic memory management, which simplifies memory management for developers and reduces the risk of memory-related errors.

- **Fast compilation:** Go has a fast compilation time, which allows developers to iterate quickly and make changes to their code without waiting for long build times.
- **Cross-platform:** Go programs can be compiled to run on multiple platforms, including Windows, macOS, and Linux.

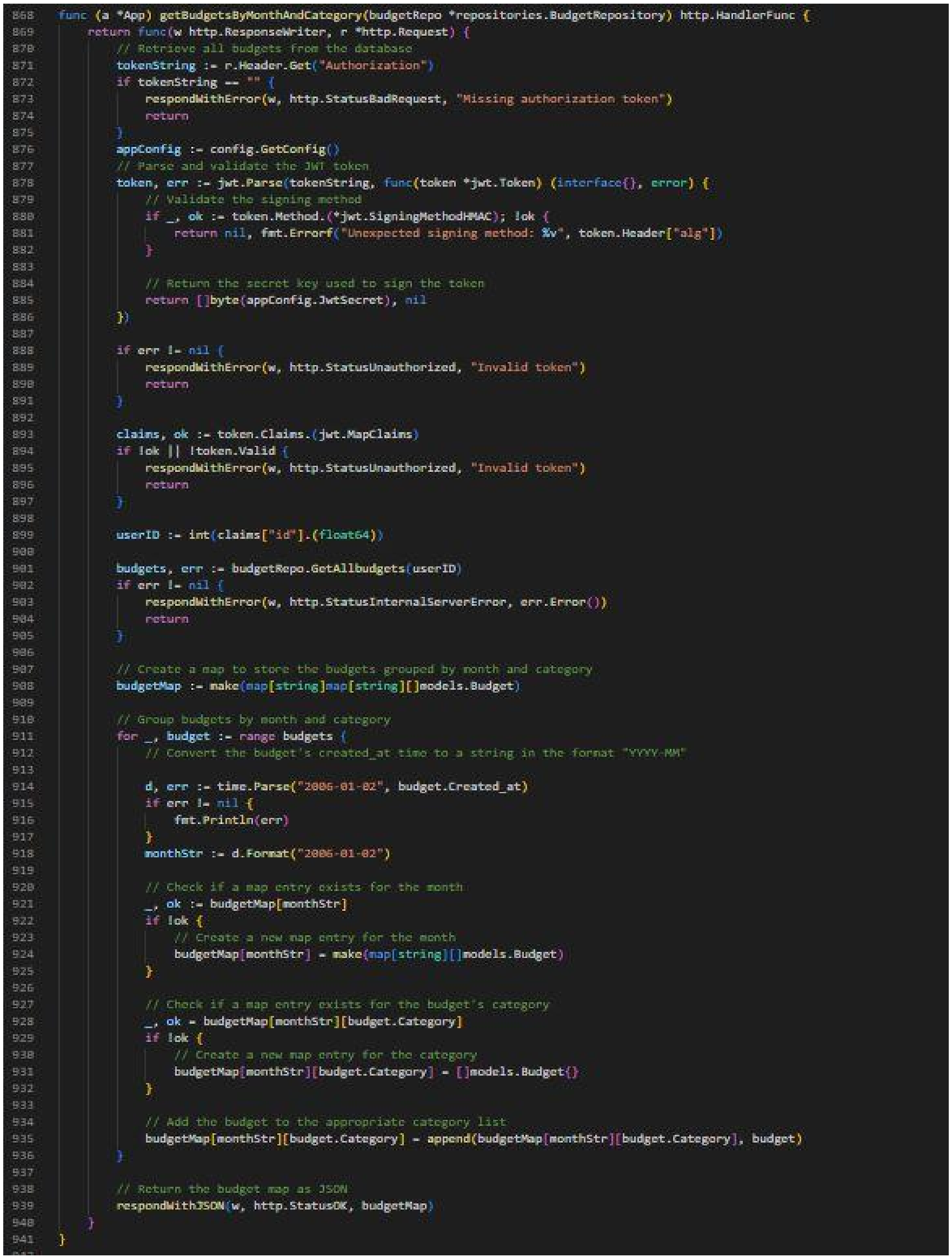
Weaknesses

- **Lack of generics:** Go does not have support for generics, which can make it harder to write generic code that can be reused across different types.
- **Error handling:** Go's error handling mechanism can be verbose and can lead to boilerplate code in some cases.
- **Immaturity of libraries:** As Go is a relatively new language, some libraries may be immature or incomplete, which can make it harder to find a library that meets specific needs.
- **Limited metaprogramming:** Go has limited support for metaprogramming, which can make it harder to write code that generates code or modifies code at runtime.

Overview:

One of the advantages of using Golang for creating microservices is its efficiency and performance. Golang is designed to be fast and lightweight, with a small memory footprint and a low overhead. This makes it an ideal choice for creating microservices that need to be highly responsive and efficient. In our application, Golang is used as a backend service for building REST APIs. These Go REST APIs are responsible for connecting to the Postgres database and performing CRUD operations as and when required. With the use of React, our application provides an interactive interface to the client machine by consuming relevant data from Rest APIs.
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.
Finally, all the requests coming from our frontend components are stored in Postgres Database.

Sample Program



```go
func (a *App) getBudgetsByMonthAndCategory(budgetRepo *repositories.BudgetRepository) http.HandlerFunc {
    return func(w http.ResponseWriter, r *http.Request) {
        // Retrieve all budgets from the database
        tokenString := r.Header.Get("Authorization")
        if tokenString == "" {
            respondWithError(w, http.StatusBadRequest, "Missing authorization token")
            return
        }
        appConfig := config.GetConfig()
        // Parse and validate the JWT token
        token, err := jwt.Parse(tokenString, func(token *jwt.Token) (interface{}, error) {
            // Validate the signing method
            if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
                return nil, fmt.Errorf("Unexpected signing method: %v", token.Header["alg"])
            }

            // Return the secret key used to sign the token
            return []byte(appConfig.JwtSecret), nil
        })

        if err != nil {
            respondWithError(w, http.StatusUnauthorized, "Invalid token")
            return
        }

        claims, ok := token.Claims.(jwt.MapClaims)
        if !ok || !token.Valid {
            respondWithError(w, http.StatusUnauthorized, "Invalid token")
            return
        }

        userID := int(claims["id"].(float64))

        budgets, err := budgetRepo.GetAllbudgets(userID)
        if err != nil {
            respondWithError(w, http.StatusInternalServerError, err.Error())
            return
        }

        // Create a map to store the budgets grouped by month and category
        budgetMap := make(map[string]map[string][]models.Budget)

        // Group budgets by month and category
        for _, budget := range budgets {
            // Convert the budget's created_at time to a string in the format "YYYY-MM"

            d, err := time.Parse("2006-01-02", budget.Created_at)
            if err != nil {
                fmt.Println(err)
            }
            monthStr := d.Format("2006-01-02")

            // Check if a map entry exists for the month
            _, ok := budgetMap[monthStr]
            if !ok {
                // Create a new map entry for the month
                budgetMap[monthStr] = make(map[string][]models.Budget)
            }

            // Check if a map entry exists for the budget's category
            _, ok = budgetMap[monthStr][budget.Category]
            if !ok {
                // Create a new map entry for the category
                budgetMap[monthStr][budget.Category] = []models.Budget{}
            }

            // Add the budget to the appropriate category list
            budgetMap[monthStr][budget.Category] = append(budgetMap[monthStr][budget.Category], budget)
        }

        // Return the budget map as JSON
        respondWithJSON(w, http.StatusOK, budgetMap)
    }
}
```

Problem Definition

Problem statement

Managing personal finances can be a challenging task, especially in today's fast-paced world where expenses can quickly pile up. It is all too easy to lose track of where your money is going and end up spending more than you intended. This is where expense tracking applications can be incredibly helpful. One of the key benefits of using an expense tracking application is that it can help users identify areas where they are overspending.

What does our project do?

ExpenseEase is a personal financial tracking application that solves all expenses related issues at a single place. The system lets the user add categorized monthly expenses, budget and income. The system then uses this data to analyze the monthly expenses to determine how much the user should spend on the listed categories for the rest of the timeframe.

How does it solve the problem?

By tracking expenses in real-time, users can see where their money is going and make adjustments to their spending habits leading to more. It allows users to stay organized by providing a centralized location for all of their financial information. Instead of relying on multiple spreadsheets or paper receipts, users can store all of their expense data in one place. This can make it easier to manage and analyze financial data, as well as prepare for tax season.

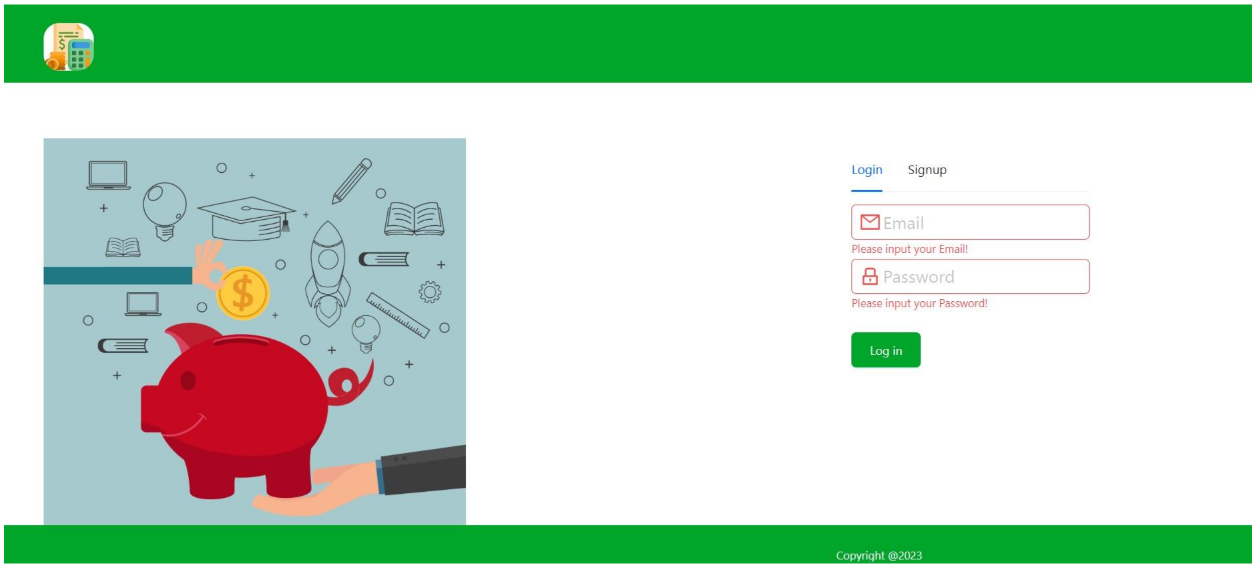
Use cases need to be specified.

- **Tax preparation:** By tracking expenses throughout the year, individuals and businesses can make tax preparation easier and more accurate.
- **Personal budgeting:** An expense tracking application can help individuals set and track personal budgets by categorizing expenses and highlighting areas where they are overspending.
- **Family budgeting:** An expense tracking application can help families manage their expenses by allowing them to track spending across multiple accounts and identify areas where they can save money.
- **Non-profit organizations:** ExpenseEase can help non-profit organizations track their expenses and ensure that they are using their funds appropriately and transparently.

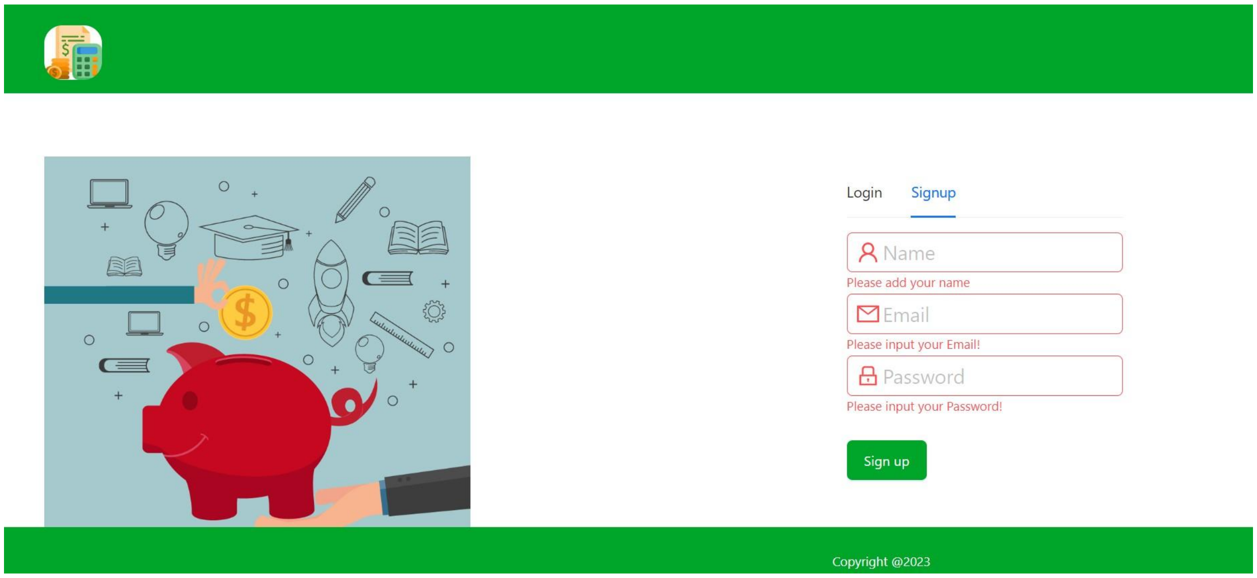
Experiments

Screenshots of the running project:

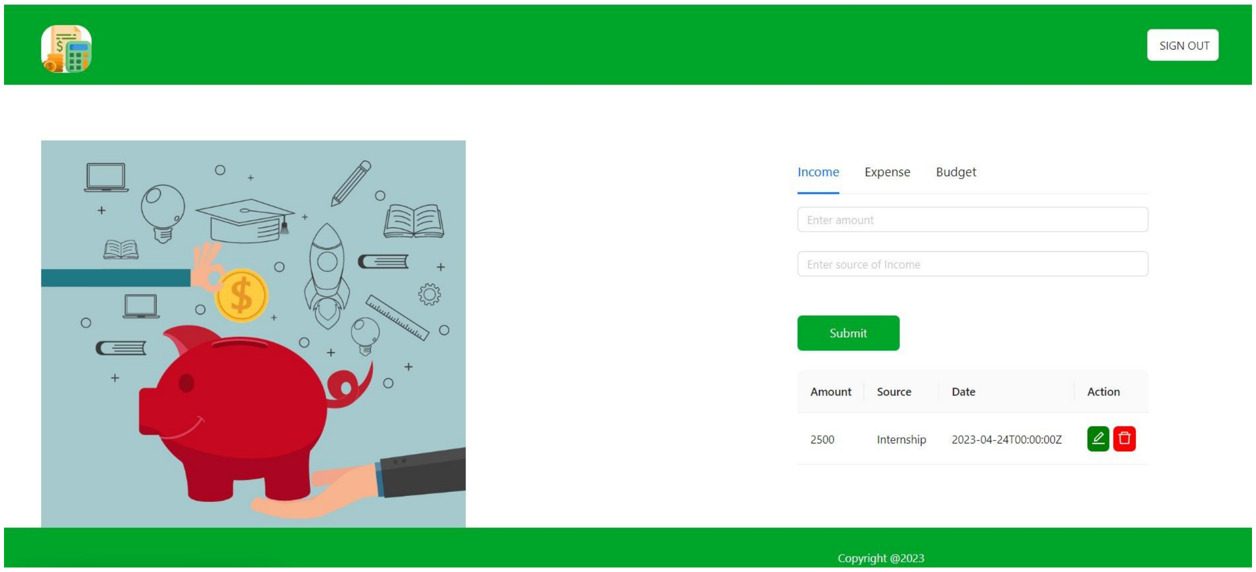
Home and Login Page



Sign up Page

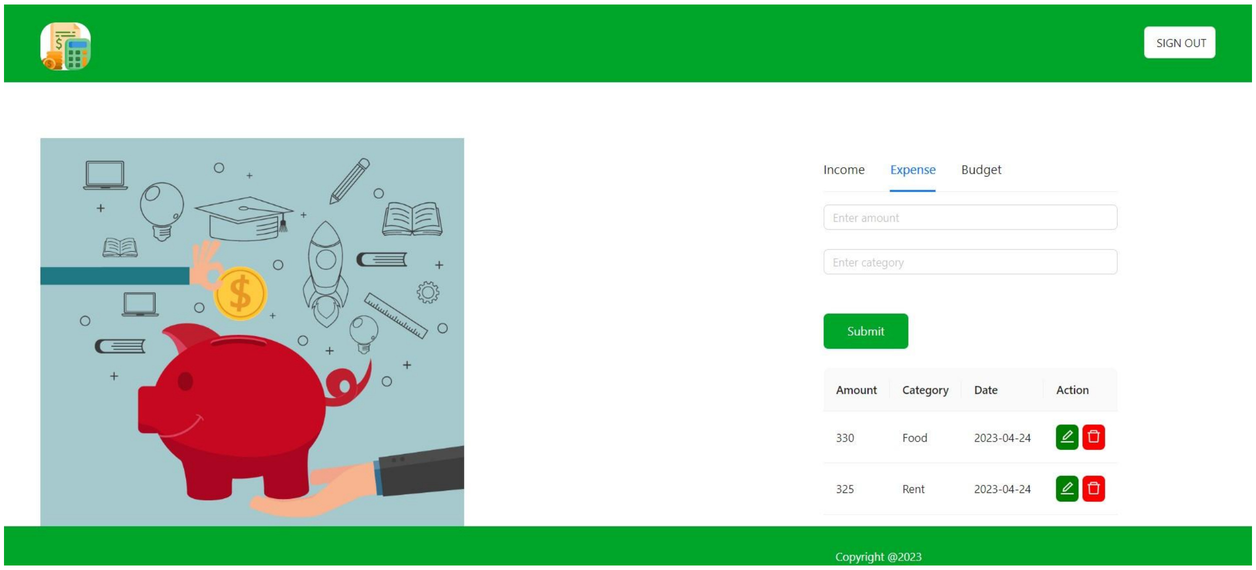


Income Page



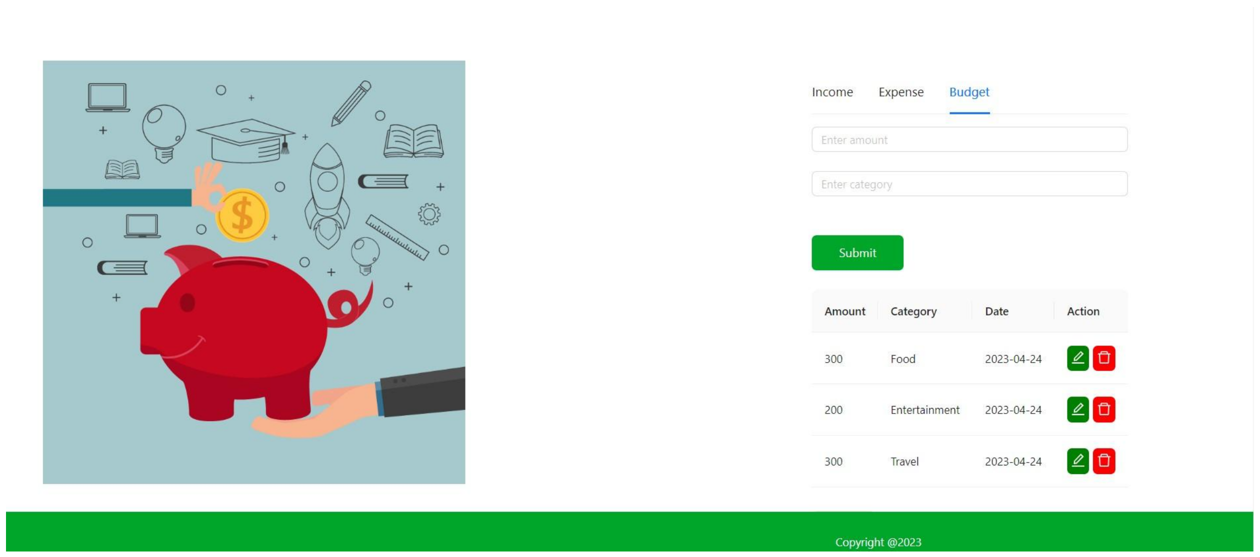
Amount	Source	Date	Action
2500	Internship	2023-04-24T00:00:00Z	

Expense Page



Amount	Category	Date	Action
330	Food	2023-04-24	
325	Rent	2023-04-24	

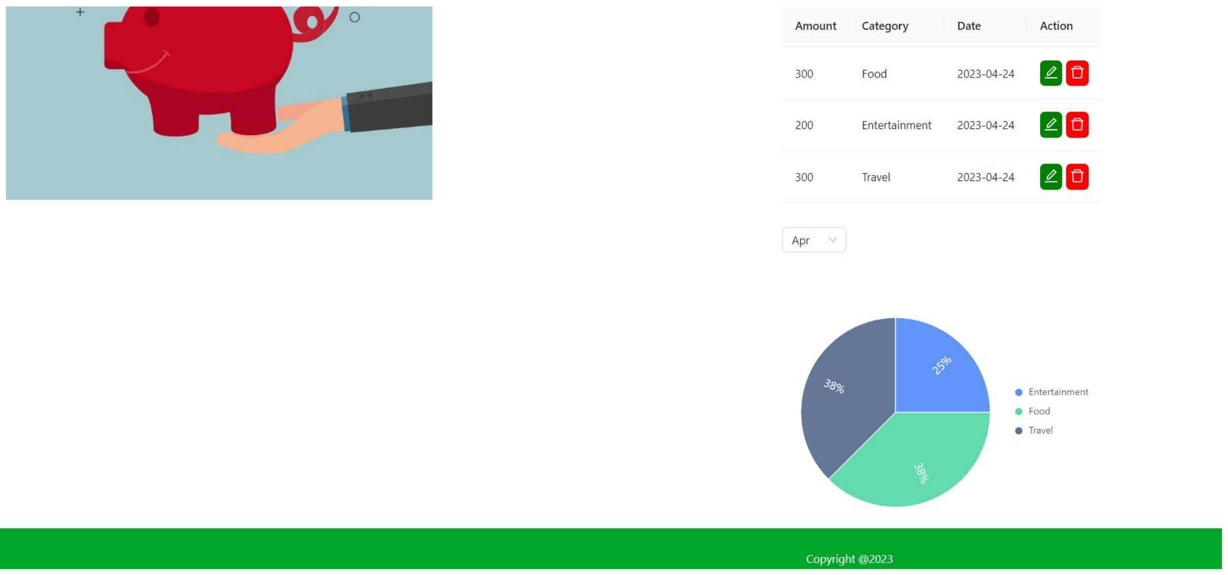
Budget page



Amount	Category	Date	Action
300	Food	2023-04-24	
200	Entertainment	2023-04-24	
300	Travel	2023-04-24	

Copyright @2023

Graph showing budget analysis



Amount	Category	Date	Action
300	Food	2023-04-24	
200	Entertainment	2023-04-24	
300	Travel	2023-04-24	

Copyright @2023

Description of Test bed

List of questions your experiments are designed to answer

a) Can a user create his/her account successfully?
b) Can they login successfully?
c) How can they add their expenses?
d) Is there a way to specify the budget?
e) Is there a way to track my income history?

f) Is it possible to analyze my monthly expenses at a single place?

Conclusion:

We successfully completed creating a project using Go programming language. This was a great opportunity for all our team members to learn and understand a new programming language that we always wanted to learn. Our application can include features such as transaction tracking, budget setting, and expense categorization. Users can input their expenses, and the application can automatically categorize them into different spending categories, such as food, transportation, or entertainment. Users can also set budgets for each category and receive notifications when they are approaching or exceeding their budget limits. Lastly, we really enjoyed working on this project and thereby adding a new programming language to our skillset.

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/