**Московский государственный технический
университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования»
Отчет по Домашнему заданию
«Fun Wheel»

Выполнил:
студент группы ИУ5-36Б
Ордиянц Эрик

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2024 г.

Аннотация

Программа написана на языке GO, JavaScript и HTML и имеет слудующий функционал:
1. Создание колеса
2. Добавление на колесо собственных вариантов событий
3. Прокрутка колеса, чтобы определить случайный вариант
4. Удаление варианта, который победил последний
5. Очистка колеса

Код программы
bot/Dockerfile

```
FROM golang:1.20-alpine AS builder
WORKDIR /app
COPY . .
RUN go mod init fun-wheel-bot && go mod tidy && go build -o bot .

FROM alpine:latest
WORKDIR /app
COPY --from=builder /app/bot .
ENV TELEGRAM_BOT_TOKEN=${TELEGRAM_BOT_TOKEN}
CMD ["./bot"]
```

bot/main.go

```
package main

import (
    "log"
    "os"

    tgbotapi "github.com/go-telegram-bot-api/telegram-bot-api/v5"
    "github.com/joho/godotenv"
)

func main() {
    if err := godotenv.Load(); err != nil {
        log.Println("Не удалось загрузить файл .env")
    }

    botToken := os.Getenv("TELEGRAM_BOT_TOKEN")
    if botToken == "" {
        log.Println("Токен бота не найден. Убедитесь, что TELEGRAM_BOT_TOKEN указан в
.env")
    }

    bot, err := tgbotapi.NewBotAPI(botToken)
```

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

    bot.Debug = true
    log.Printf("Authorized on account %s", bot.Self.UserName)

    u := tgbotapi.NewUpdate(0)
    u.Timeout = 60

    updates := bot.GetUpdatesChan(u)

    for update := range updates {
        if update.Message != nil {
            chatID := update.Message.Chat.ID
            text := "Привет! Не можешь определиться с выбором? [Тогда жми
сюда](http://scrypze.ru)"
            msg := tgbotapi.NewMessage(chatID, text)
            msg.ParseMode = "Markdown"

            if _, err := bot.Send(msg); err != nil {
                log.Printf("Error sending message: %v", err)
            }
        }
    }
}
```

.env.example

```
export TELEGRAM_BOT_TOKEN=TOKEN_12345678
export SITE_HOST=example.com
export SITE_PORT=80
```

docker-compose.yaml

```
version: "3"

services:
  bot:
    build:
      context: ./bot
    environment:
      - TELEGRAM_BOT_TOKEN=${TELEGRAM_BOT_TOKEN}
    restart: unless-stopped

  site:
    build:
```

```
      context: ./site
    ports:
      - "8080:8080"
    environment:
      - HOST=${HOST:-0.0.0.0}
      - PORT=${PORT:-8080}
    networks:
      - fortune-wheel-network
    restart: unless-stopped

  nginx:
    image: nginx:alpine
    container_name: nginx
    volumes:
      - ./nginx/default.conf:/etc/nginx/conf.d/default.conf
      - /etc/letsencrypt:/etc/nginx/certs  # Путь к вашим сертификатам
    ports:
      - "80:80"
      - "443:443"
    networks:
      - fortune-wheel-network
    depends_on:
      - site
    restart: unless-stopped

networks:
  fortune-wheel-network:
    driver: bridge
```

Makefile

```
.PHONY: up down restart

up:
    docker compose up -d --build

down:
    docker compose down

restart: down up
    @echo "Контейнеры перезапущены"
```

site/static/index.html

```
<!DOCTYPE html>
<html lang="ru-RU, en-US">

<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
  <title>Колесо фортуны</title>
  <link rel="stylesheet" href="/static/style.css">
</head>

<body>
  <h1>Колесо фортуны</h1>
  <div class="wheel-container" style="position: relative;">
    <div class="arrow"></div>
    <canvas id="wheel" width="300" height="300"></canvas>
  </div>
  <div>
    <input type="text" id="item" placeholder="Введите элемент">
    <button onclick="addItem()">Add</button>
    <button onclick="resetItems()">Reset</button>
    <button onclick="spinWheel()">Spin</button>
    <button onclick="removeLastWinner()">Remove Last Winner</button>
  </div>
  <div id="result"></div>
  <script src="/static/script.js"></script>
</body>

</html>
```

site/static/script.js

```
const canvas = document.getElementById('wheel');
const ctx = canvas.getContext('2d');
let items = [];
let spinning = false;

function drawWheel() {
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    if (items.length === 0) {
        return;
    }

    const radius = canvas.width / 2;
    const center = { x: radius, y: radius };
    const sliceAngle = (2 * Math.PI) / items.length;

    ctx.beginPath();
    ctx.arc(center.x, center.y, radius - 2, 0, 2 * Math.PI);
    ctx.strokeStyle = '#333';
    ctx.lineWidth = 4;
    ctx.stroke();

    ctx.font = "14px Arial";

    items.forEach((item, i) => {
        const startAngle = i * sliceAngle;
```

```
        const endAngle = startAngle + sliceAngle;

        ctx.beginPath();
        ctx.moveTo(center.x, center.y);
        ctx.arc(center.x, center.y, radius - 4, startAngle, endAngle);
        ctx.closePath();

        ctx.fillStyle = i % 2 === 0 ? '#FFDD57' : '#FFAB57';
        ctx.fill();
        ctx.strokeStyle = '#333';
        ctx.lineWidth = 1;
        ctx.stroke();

        ctx.save();
        ctx.translate(center.x, center.y);
        ctx.rotate(startAngle + sliceAngle / 2);
        ctx.textAlign = "right";
        ctx.fillStyle = "#000";
        ctx.fillText(item, radius - 20, 5);
        ctx.restore();
    });

    ctx.beginPath();
    ctx.arc(center.x, center.y, 10, 0, 2 * Math.PI);
    ctx.fillStyle = '#333';
    ctx.fill();
}

async function loadItems() {
    try {
        const response = await fetch('/items');
        if (!response.ok) {
            throw new Error('Ошибка загрузки элементов');
        }
        const data = await response.json();
        items = data.items;
        drawWheel();
    } catch (error) {
        console.error('Ошибка при загрузке элементов:', error);
    }
}

async function addItem() {
    const input = document.getElementById('item');
    const item = input.value.trim();
    if (item) {
        try {
            const response = await fetch('/add', {
                method: 'POST',
                headers: { 'Content-Type': 'application/json' },
                body: JSON.stringify({ text: item })
            });
            if (!response.ok) {
```

```
                throw new Error('Ошибка добавления элемента');
            }
            input.value = '';
            await loadItems();
            document.getElementById('result').innerText = '';
        } catch (error) {
            console.error('Ошибка при добавлении элемента:', error);
            alert('Не удалось добавить элемент');
        }
    }
}

async function resetItems() {
    if (spinning) return;

    try {
        const response = await fetch('/reset', { method: 'POST' });
        if (!response.ok) {
            throw new Error('Ошибка сброса элементов');
        }
        items = [];
        drawWheel();
        document.getElementById('result').innerText = '';
    } catch (error) {
        console.error('Ошибка при сбросе элементов:', error);
        alert('Не удалось сбросить элементы');
    }
}

let currentSpinInterval = null;

async function spinWheel() {
    if (spinning || items.length === 0) {
        return;
    }

    try {
        spinning = true;
        const response = await fetch('/spin');
        if (!response.ok) {
            spinning = false;
            throw new Error('Ошибка вращения колеса');
        }

        const data = await response.json();
        items = data.items;
        const winnerIndex = items.indexOf(data.winner);
        let currentAngle = 0;
        const targetAngle = (2 * Math.PI) * 5 + winnerIndex * (2 * Math.PI /
items.length);

        if (currentSpinInterval) {
            clearInterval(currentSpinInterval);
```

```
        }

        currentSpinInterval = setInterval(() => {
            currentAngle += 0.1;
            if (currentAngle >= targetAngle) {
                clearInterval(currentSpinInterval);
                currentSpinInterval = null;
                spinning = false;
                document.getElementById('result').innerText = `Победитель:
${data.winner}`;
            }
            ctx.save();
            ctx.translate(canvas.width / 2, canvas.height / 2);
            ctx.rotate(currentAngle);
            ctx.translate(-canvas.width / 2, -canvas.height / 2);
            drawWheel();
            ctx.restore();
        }, 16);
    } catch (error) {
        console.error('Ошибка при вращении колеса:', error);
        alert('Не удалось провести вращение');
        spinning = false;
        if (currentSpinInterval) {
            clearInterval(currentSpinInterval);
            currentSpinInterval = null;
        }
    }
}

async function removeLastWinner() {
    if (spinning) return;

    try {
        const response = await fetch('/remove-winner', { method: 'POST' });
        if (!response.ok) {
            throw new Error('Ошибка удаления победителя');
        }
        await loadItems();
        document.getElementById('result').innerText = '';
    } catch (error) {
        console.error('Ошибка при удалении победителя:', error);
        alert('Не удалось у??алить победителя');
    }
}

loadItems();
```

site/static/style.css

```css
body {
    font-family: Arial, sans-serif;
    text-align: center;
}

canvas {
    margin: 20px auto;
    display: block;
}

button {
    margin: 5px;
    padding: 10px 20px;
    font-size: 16px;
}

.arrow {
    position: absolute;
    width: 0;
    height: 0;
    border-left: 20px solid transparent;
    border-right: 20px solid transparent;
    border-top: 40px solid #ff0000;
    left: 50%;
    transform: translateX(-50%);
    z-index: 1;
}
```

site/Dockerfile

```
FROM golang:1.20-alpine AS builder
WORKDIR /app
COPY . .
RUN go mod init fun-wheel-bot-site && go mod tidy && go build -o site .

FROM alpine:latest
WORKDIR /app
COPY --from=builder /app/site .
COPY static ./static
CMD ["./site"]
```

Site/main.go

```
package main

import (
    "encoding/json"
    "fmt"
    "log"
    "math/rand"
    "net/http"
    "os"
    "path/filepath"
    "sync"
    "time"

    "github.com/google/uuid"
    "github.com/gorilla/sessions"
    "github.com/joho/godotenv"
)

type Session struct {
    Items      []string
    LastWinner string
}

type WheelService struct {
    sessions map[string]*Session
    store    *sessions.CookieStore
    mux      sync.RWMutex
}

func NewWheelService() *WheelService {
    return &WheelService{
        sessions: make(map[string]*Session),
        store:    sessions.NewCookieStore([]byte("secret-key")),
    }
}

func (ws *WheelService) getOrCreateSession(w http.ResponseWriter, r *http.Request)
(*Session, string) {
    session, _ := ws.store.Get(r, "wheel-session")

    sessionID, ok := session.Values["id"].(string)
    if !ok {
        sessionID = uuid.New().String()
        session.Values["id"] = sessionID
    }

    ws.mux.Lock()
    defer ws.mux.Unlock()

    if _, exists := ws.sessions[sessionID]; !exists {
        ws.sessions[sessionID] = &Session{
```

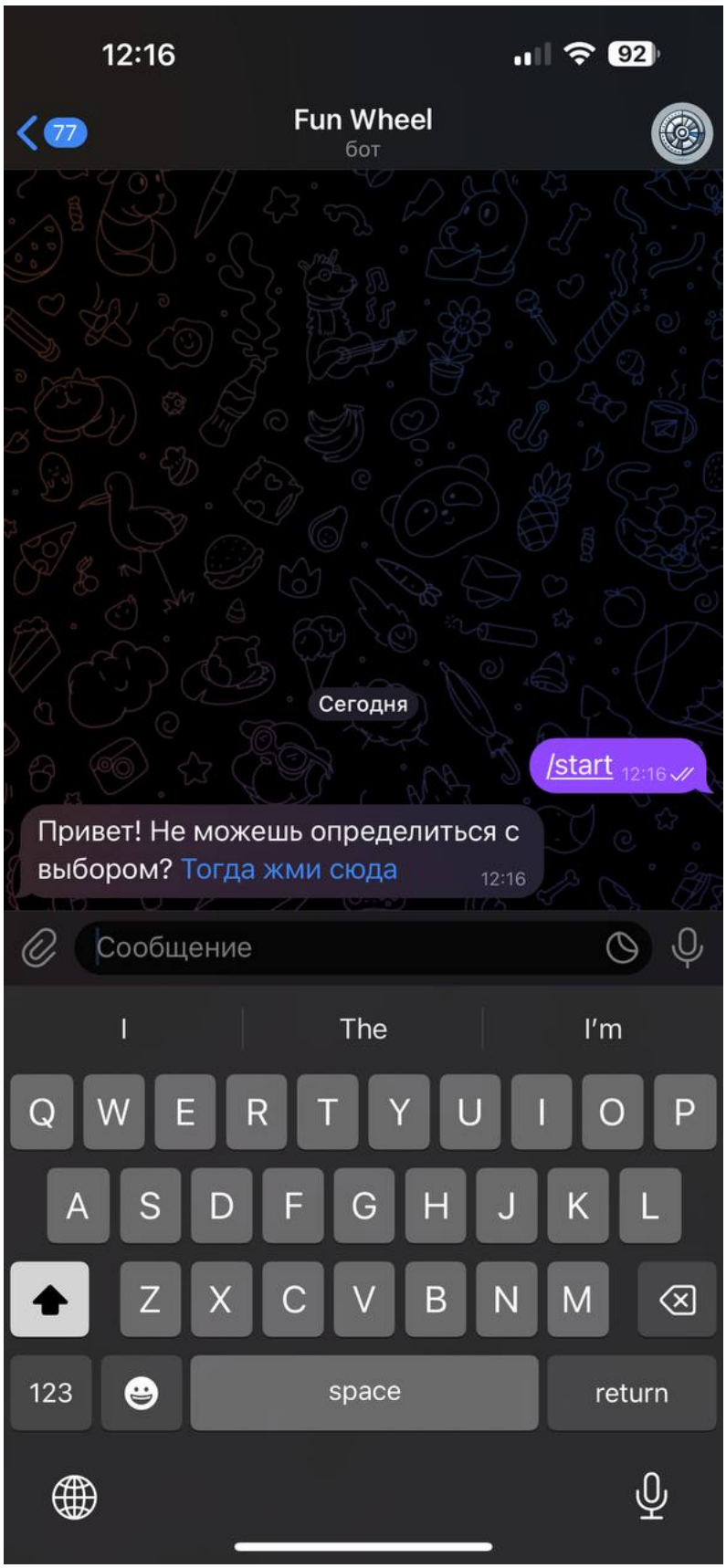
```
            Items: []string{},
        }
    }

    session.Save(r, w)

    return ws.sessions[sessionID], sessionID
}

func (ws *WheelService) AddItem(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    session, _ := ws.getOrCreateSession(w, r)

    var item struct {
        Text string `json:"text"`
    }
    if err := json.NewDecoder(r.Body).Decode(&item); err != nil {
        http.Error(w, "Invalid input", http.StatusBadRequest)
        return
    }

    ws.mux.Lock()
    defer ws.mux.Unlock()
    session.Items = append(session.Items, item.Text)
    log.Printf("Added item: %s", item.Text)
    w.WriteHeader(http.StatusOK)
}

func (ws *WheelService) ResetItems(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    session, _ := ws.getOrCreateSession(w, r)

    ws.mux.Lock()
    defer ws.mux.Unlock()
    session.Items = []string{}
    session.LastWinner = ""
    log.Println("Reset all items")
    w.WriteHeader(http.StatusOK)
}

func (ws *WheelService) SpinWheel(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodGet {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }
```

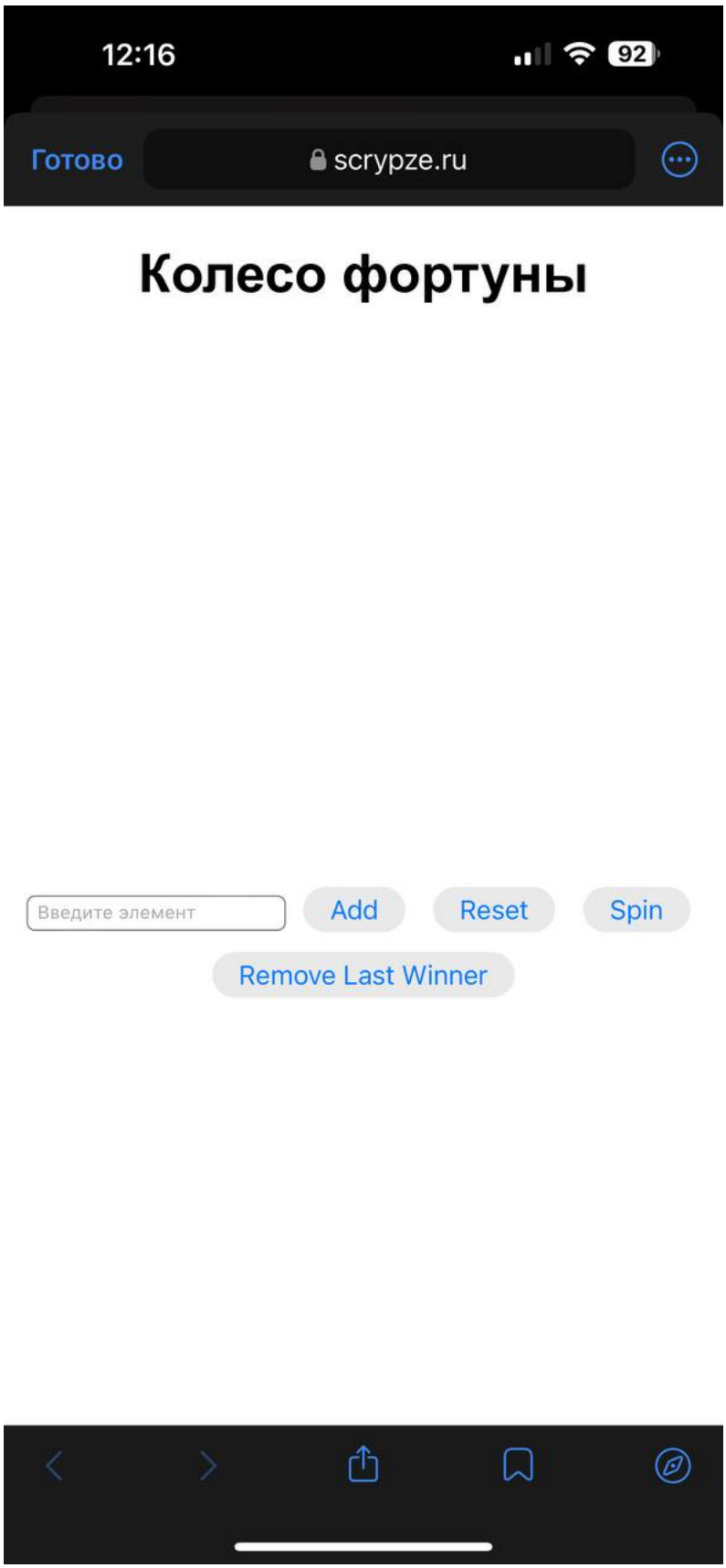
```
    session, _ := ws.getOrCreateSession(w, r)

    ws.mux.Lock()
    defer ws.mux.Unlock()

    if len(session.Items) == 0 {
        http.Error(w, "No items in the wheel", http.StatusBadRequest)
        return
    }

    rng := rand.New(rand.NewSource(time.Now().UnixNano()))
    index := rng.Intn(len(session.Items))
    session.LastWinner = session.Items[index]
    log.Printf("Winner: %s", session.LastWinner)

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(map[string]interface{}{
        "winner": session.LastWinner,
        "index":  index,
        "items":  session.Items,
    })
}

func (ws *WheelService) RemoveLastWinner(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodPost {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    session, _ := ws.getOrCreateSession(w, r)

    ws.mux.Lock()
    defer ws.mux.Unlock()

    if session.LastWinner == "" {
        http.Error(w, "No winner to remove", http.StatusBadRequest)
        return
    }

    newItems := []string{}
    for _, item := range session.Items {
        if item != session.LastWinner {
            newItems = append(newItems, item)
        }
    }

    session.Items = newItems
    log.Printf("Removed last winner: %s", session.LastWinner)
    session.LastWinner = ""
    w.WriteHeader(http.StatusOK)
}
```

```go
func (ws *WheelService) GetItems(w http.ResponseWriter, r *http.Request) {
    if r.Method != http.MethodGet {
        http.Error(w, "Invalid request method", http.StatusMethodNotAllowed)
        return
    }

    session, _ := ws.getOrCreateSession(w, r)

    ws.mux.RLock()
    defer ws.mux.RUnlock()

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(map[string]interface{}{
        "items": session.Items,
    })
}

func (ws *WheelService) ServeHTML(w http.ResponseWriter, r *http.Request) {
    currentDir, err := os.Getwd()
    if err != nil {
        http.Error(w, "Internal Server Error", http.StatusInternalServerError)
        return
    }
    htmlPath := filepath.Join(currentDir, "static", "index.html")
    http.ServeFile(w, r, htmlPath)
}

func main() {
    if err := godotenv.Load(); err != nil {
        log.Println("Не удалось загрузить файл .env, будут использованы стандартные
значения")
    }

    host := os.Getenv("HOST")
    if host == "" {
        host = "localhost"
    }

    port := os.Getenv("PORT")
    if port == "" {
        port = "8080"
    }


    address := host + ":" + port

    fmt.Printf(address)

    service := NewWheelService()

    http.HandleFunc("/", service.ServeHTML)
    http.HandleFunc("/add", service.AddItem)
    http.HandleFunc("/reset", service.ResetItems)
```

```
    http.HandleFunc("/spin", service.SpinWheel)
    http.HandleFunc("/remove-winner", service.RemoveLastWinner)
    http.HandleFunc("/items", service.GetItems)

    staticDir := http.Dir("static")
    http.Handle("/static/", http.StripPrefix("/static/", http.FileServer(staticDir)))

    log.Printf("Server started at http://%s", address)

    if err := http.ListenAndServe(address, nil); err != nil {
        log.Fatalf("Server failed: %v", err)
    }
}
```

Пример работы:


12:16
92
77
Fun Wheel
бот
Сегодня
/start 12:16
Привет! Не можешь определиться с выбором? Тогда жми сюда 12:16
Сообщение
I
The
I'm
Q W E R T Y U I O P
A S D F G H J K L
Z X C V B N M
123
space
return


12:16
92
Готово
scrypze.ru
Колесо фортуны
Введите элемент
Add
Reset
Spin
Remove Last Winner

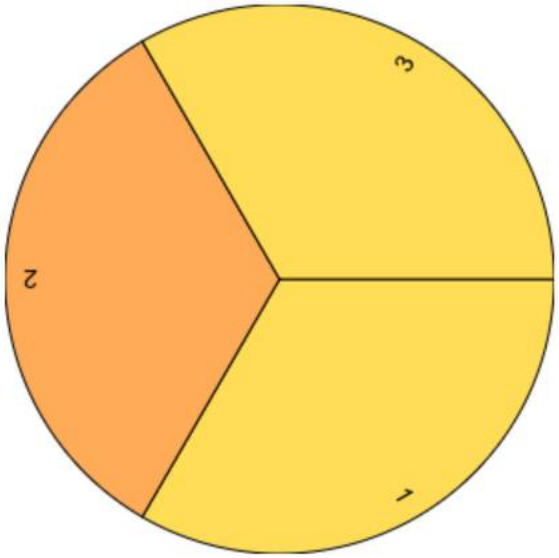
12:17

92

Готово

scrypze.ru

Колесо фортуны



Введите элемент

Add

Reset

Spin

Remove Last Winner

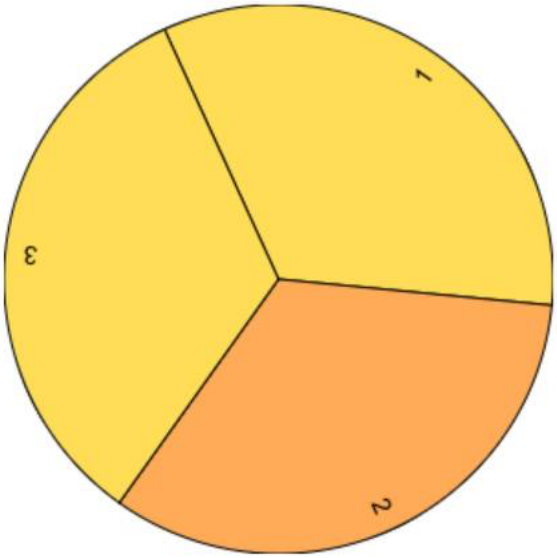
12:17

91

Готово

scrypze.ru

Колесо фортуны



Введите элемент

Add

Reset

Spin

Remove Last Winner

Победитель: 3