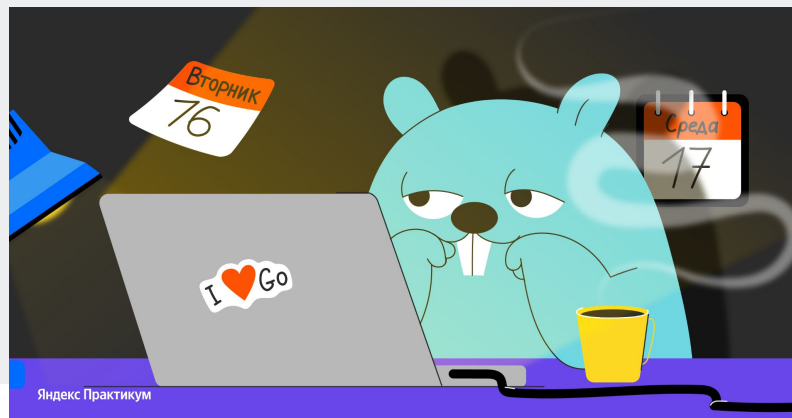


# Курсовой проект

«Накопительная система лояльности «Гофермарт»  
курс «Продвинутый Go-Разработчик»  
Алексей Якимчук

# Накопительная система лояльности «Гофермарт»

1. Задача
2. Этапы работы
3. Модели данных
4. Взаимодействие с сервисом начисления баллов (Как получить баллы?)
5. Проблемы или возможности
6. Результаты
7. Планы






# 1. Задача

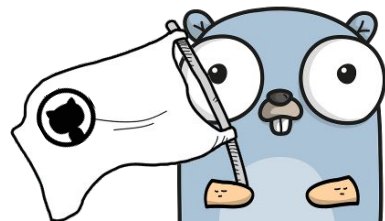
Разработать решение для начисления баллов за принятые заказы на счет пользователя и использование начисленных баллов на последующие покупки.



## 2. Этапы работы

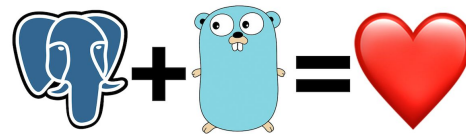
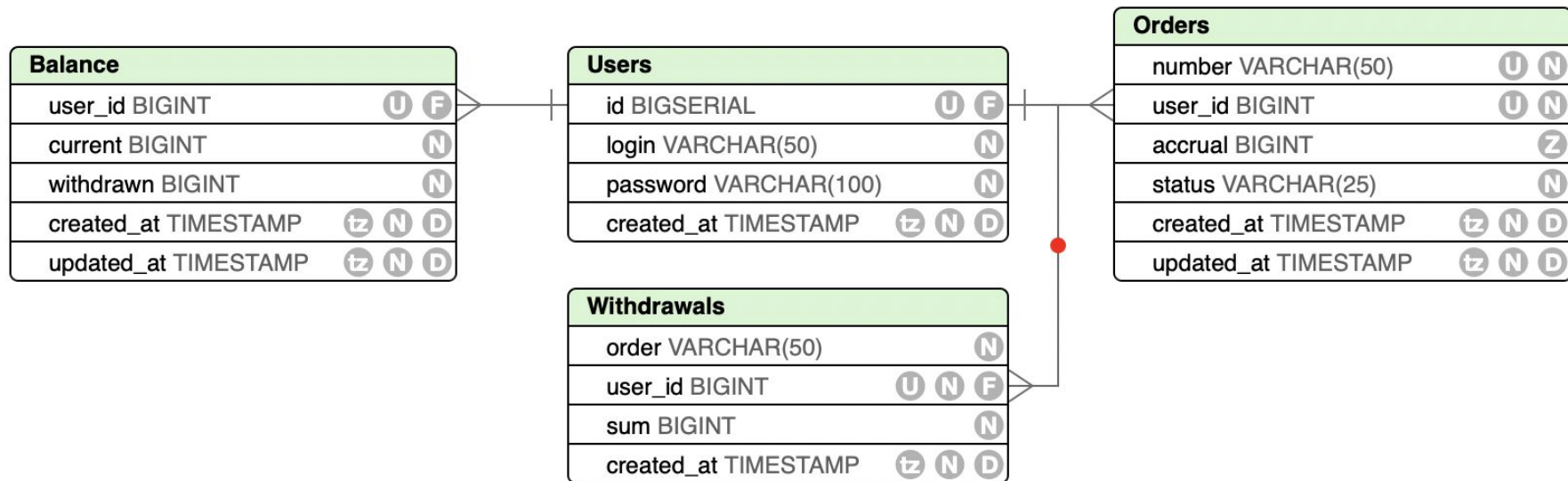
1. Подготовка структуры проекта, данных.  
+ Учет пользователей.
2. Регистрация заказов
3. Начисление баллов

 <b>iter3 - feature/balance-and-accrual</b> ✓
#3 opened last week by webkimru
 <b>iter2 - feature/orders</b> ✗
#2 opened 2 weeks ago by webkimru
 <b>Iter1 - feature/structure-and-users</b> ✗
#1 opened 2 weeks ago by webkimru

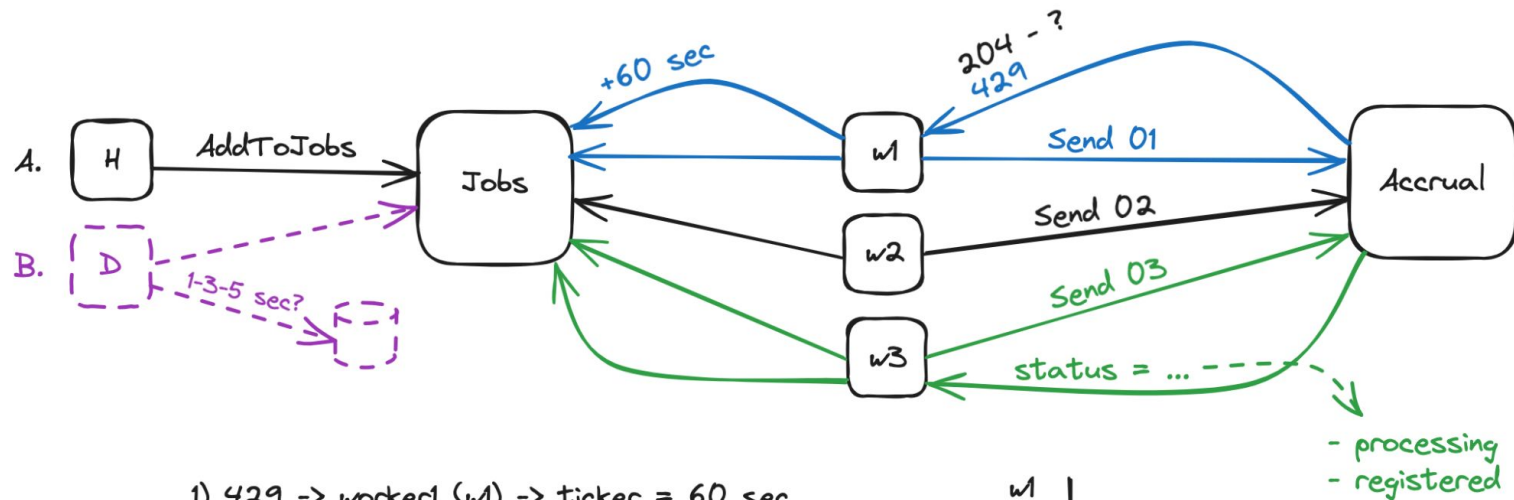


#программирование

# 3. Модели данных



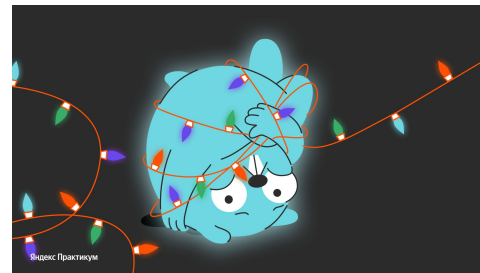
## 4. Как получить баллы? «Лебедь, рак и щука»



1) 429 → worker1 (w1) → ticker = 60 sec  
429 → worker2 (w2) → ticker = 60 sec + ?  
429 → worker3 (w3) → ticker = 60 sec + ? OR

w1  
...  
w3  
↓  
= 60 sec?

2) sync.Cond  
429 → worker1  
|  
sync +60 sec  
↙      ↘  
worker2   worker3



## 5. Возможности (не проблемы)

```
var storePriority config.Store
var repo *handlers.Repository
switch {
case app.DatabaseDSN != "": // DB
    storePriority = config.Database
    conn, err := pg.ConnectToDB(app.DatabaseDSN)
    if err != nil {
        log.Fatal(err)
    }
    if err := pg.Bootstrap(ctx, conn); err != nil {
        log.Fatal(err)
    }
    db := pg.NewStore(conn)
    pg.DB = db
    storage := db
    repo = handlers.NewRepo(storage)

default: // in memory
    storePriority = config.Memory
    storage := store.NewMemStorage()
    // загружать ранее сохранённые значения из указ
    if app.FileStore.Restore {
        res, err := file.Reader()
        if err != nil : nil, err ↗
        // если не пустой файл
        if res != nil {
            storage.Counter = res.Counter
            storage.Gauge = res.Gauge
        }
    }
    // инициализируем репозиторий хендлеров с указан
    repo = handlers.NewRepo(storage)
}
```

```
// init store:
var db store.Repositories
switch app.StoreDriver {
case "postgresql", "postgres":
    db = &pg.Store{}
default:
    logger.Log.Fatalf("Unknown storage app.StoreDriver=#{app.S
}
if err := db.Initialize(ctx, app); err != nil : nil, err ↗

// init app:
repo := api.NewRepo(db)
api.NewHandlers(repo, &app)
```

```

var order models.Order
order.Number = strconv.Itoa(int(orderNumber))
if !order.IsValid() {
    // `422` — неверный формат номера заказа;
    w.WriteHeader(http.StatusUnprocessableEntity)
    return
}

```

**А не:**  
 orderIsValid := checkOrderNumber(order.Number);

```

func (o Order) IsValid() bool {
    // алгоритм Луна - https://ru.wikipedia.org/wiki/Алгоритм\_Луны
    sum := 0
    parity := len(o.Number) % 2

    for i, value := range o.Number {
        digit := int(value - '0')

        if i%2 == parity {
            digit *= 2
            if digit > 9 {
                digit -= 9
            }
        }
        sum += digit
    }

    return sum%10 == 0
}

```



```
func (m *Repository) GetOrders(w http.ResponseWriter, r *http.Request) {
    //- `200` - успешная обработка запроса.
    //- `204` - нет данных для ответа.
    //- `401` - пользователь не авторизован.
    //- `500` - внутренняя ошибка сервера.
    authUserID := m.GetUserID(r)
    orders, err := m.Store.GetOrders(r.Context(), authUserID)
    if err != nil {
        logger.Log.Errorln( args...: "failed GetOrders()= ", err)
        w.WriteHeader(http.StatusInternalServerError)
        return
    }
}
```

```
func (m *Repository) GetUserID(r *http.Request) int64 {
    authorization := r.Header.Get( key: "Authorization")
    token := authorization[len(bearerSchema):]
    userID := GetUserID(token)

    return userID
}
```

```
func CheckAuth(next http.Handler) http.Handler {
    return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
        authorization := r.Header.Get( key: "Authorization")
        if authorization == "" || !strings.HasPrefix(authorization, prefix: "Bearer ") {
            w.WriteHeader(http.StatusUnauthorized)
            return
        }
        token := authorization[len(bearerSchema):]
        userID := api.GetUserID(token)
        if userID < 0 {
            w.WriteHeader(http.StatusUnauthorized)
            return
        }

        next.ServeHTTP(w, r)
    })
}
```

123 id	ABC login	ABC password	🕒 created_at
1	test1	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08	2024-03-03 20:45:03.000 +0300
3	test2	9f86d081884c7d659a2feaa0c55ad015a3bf4f1b2b0b822cd15d6c15b0f00a08	2024-03-03 20:49:19.000 +0300

Одинаковый хеш

**Вопрос:** Быть может, каждый раз сохранять в качестве пароля случайную последовательность одинаковой длины? Как?

```

func (s *Store) SetWithdrawal(ctx context.Context, withdrawal models.Withdrawal) error {
    tx, err := s.Conn.BeginTx(ctx, opts: nil)
    if err != nil : err ↗

    defer tx.Rollback()

    row, err := tx.ExecContext(ctx, `
        WITH cte AS (
            SELECT user_id FROM gophermart.balance
            WHERE current - $1 >= 0
        )
        UPDATE gophermart.balance
        SET current = current - $1, withdrawn = withdrawn + $1
        FROM cte
        WHERE balance.user_id = cte.user_id AND balance.user_id = $2
        RETURNING current
    `, withdrawal.Sum, withdrawal.UserID)
    if err != nil : err ↗
    affected, err := row.RowsAffected()
    if err != nil : err ↗
    if affected == 0 : api.ErrNotEnoughMoney ↗

    _, err = tx.ExecContext(ctx, `
        INSERT INTO gophermart.withdrawals ("order", user_id, sum) VALUES($1, $2, $3)
        ON CONFLICT ("order") DO NOTHING
    `, withdrawal.Order, withdrawal.UserID, withdrawal.Sum)
    if err != nil : err ↗

    return tx.Commit()
}

```

## Выполняем одной транзакцией:

- списание с баланса
- добавление в журнал списаний

```

46
47 type Balance struct {
48     UserID    int64 `json:"-`
49     Current   Money `json:"current"`
50     Withdrawn Money `json:"withdrawn"`
51     CreatedAt string `json:"-`
52 }
53
54 type Money float32
55
56 func (m Money) Set() int64 {
57     return int64(m * 100)
58 }
59
60 func (m Money) Get() float32 {
61     return float32(m) / 100
62 }

```

## GetBalance

```

balance := models.Balance{
    UserID:    userDB,
    Current:   models.Money(current.Get()),
    Withdrawn: models.Money(withdrawn.Get()),
}

```

## SetBalance

```

authUserID := m.GetUserID(r)
withdrawal.UserID = authUserID
withdrawal.Sum = models.Money(withdrawal.Sum.Set())
err := m.Store.SetWithdrawal(r.Context(), withdrawal)

```

# Ошибки

```
139
140 === RUN   TestGophermart/TestUserOrders/orders_list
141     gophermart_orders_test.go:172:
142         Error Trace:      /__w/go-shop-loyalty/go-shop-loyalty/gophermart_orders_test.go:172
143                             /__w/go-shop-loyalty/go-shop-
144                             loyalty/suite.go:91
145         Error:              "[{3035227 NEW %!s(float32=0) 2024-03-02 12:05:21 +0000 UTC}
146                             {51524766113713 NEW %!s(float32=0) 2024-03-02 12:05:01 +0000 UTC}]" should have 1 item(s), but has 2
147         Test:               TestGophermart/TestUserOrders/orders_list
148         Messages:           Ожидаем отличное от полученного кол-во заказов
149         gophermart_orders_test.go:181: Оригинальный запрос:
150
151         GET /api/user/orders HTTP/1.1
152         Host: localhost:8080
153         Authorization: ***
154         User-Agent: go-resty/2.7.0 (https://github.com/go-resty/resty)
155
```

```

19  === RUN   TestGophermart/TestEndToEnd/await_order_processed
20      gophermart_e2e_test.go:180:
21          Error Trace:   /__w/go-shop-loyalty/go-shop-loyalty/gophermart_e2e_test.go:180
22                          /__w/go-shop-loyalty/go-shop-
23                          loyalty/suite.go:91
24          Error:          Received unexpected error:
25                          json: cannot unmarshal number into Go struct field order.number of
26                          type string
27          Test:           TestGophermart/TestEndToEnd/await_order_processed
28          Messages:      Ошибка при попытке сделать запрос на получение статуса расчета
29                          начисления в системе лояльности
30          gophermart_e2e_test.go:190: Оригинальный запрос:
31
32          GET /api/user/orders HTTP/1.1
33          Host: localhost:8080
34          Authorization: ***
35          User-Agent: go-resty/2.7.0 (https://github.com/go-resty/resty)
36

```

## 6. Результаты

```
79 --- PASS: TestGophermart (3.23s)
80 --- PASS: TestGophermart/TestEndToEnd (2.01s)
81 --- PASS: TestGophermart/TestEndToEnd/register_accrual_mechanic (0.00s)
82 --- PASS: TestGophermart/TestEndToEnd/register_order_for_accrual (0.00s)
83 --- PASS: TestGophermart/TestEndToEnd/register_user (0.00s)
84 --- PASS: TestGophermart/TestEndToEnd/order_upload (0.00s)
85 --- PASS: TestGophermart/TestEndToEnd/await_order_processed (2.00s)
86 --- PASS: TestGophermart/TestEndToEnd/check_balance (0.00s)
87 --- PASS: TestGophermart/TestEndToEnd/withdraw_balance (0.00s)
88 --- PASS: TestGophermart/TestEndToEnd/recheck_balance (0.00s)
89 --- PASS: TestGophermart/TestEndToEnd/check_withdrawals (0.00s)
90 --- PASS: TestGophermart/TestUserAuth (0.00s)
91 --- PASS: TestGophermart/TestUserAuth/register_user (0.00s)
92 --- PASS: TestGophermart/TestUserAuth/login_user (0.00s)
93 --- PASS: TestGophermart/TestUserOrders (0.01s)
94 --- PASS: TestGophermart/TestUserOrders/unauthorized_order_upload (0.00s)
95 --- PASS: TestGophermart/TestUserOrders/unauthorized_orders_list (0.00s)
96 --- PASS: TestGophermart/TestUserOrders/register_user (0.00s)
97 --- PASS: TestGophermart/TestUserOrders/bad_order_upload (0.00s)
98 --- PASS: TestGophermart/TestUserOrders/order_upload (0.00s)
99 --- PASS: TestGophermart/TestUserOrders/duplicate_order_upload_same_user (0.00s)
100 --- PASS: TestGophermart/TestUserOrders/orders_list (0.00s)
101 --- PASS: TestGophermart/TestUserOrders/duplicate_order_upload_other_user (0.00s)
102 PASS
```

НО...

Гораздо важнее наличие  
персонального прогресса!



## 7. Планы до конца курса

- Научиться писать тесты (не менее 80% кода), включая моки.
- Сделать пример с Sync.Cond.
- Разобрать и сделать вариант решения с чистой архитектурой (сервисный слой) + покрытие тестами.
- Разобрать и сделать еще один вариант приложения работы с каналами, вейтгруппами, логгером zap, где они передаются полями структуры.



Спасибо  
за внимание!

Вопросы?

