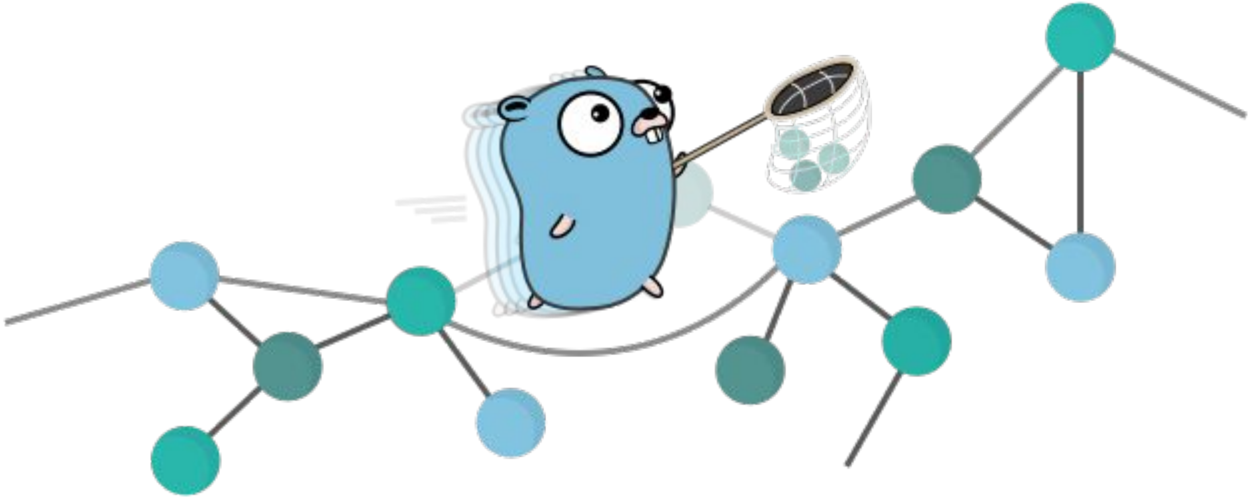
Переход с PHP на Go: архитектура Go приложений

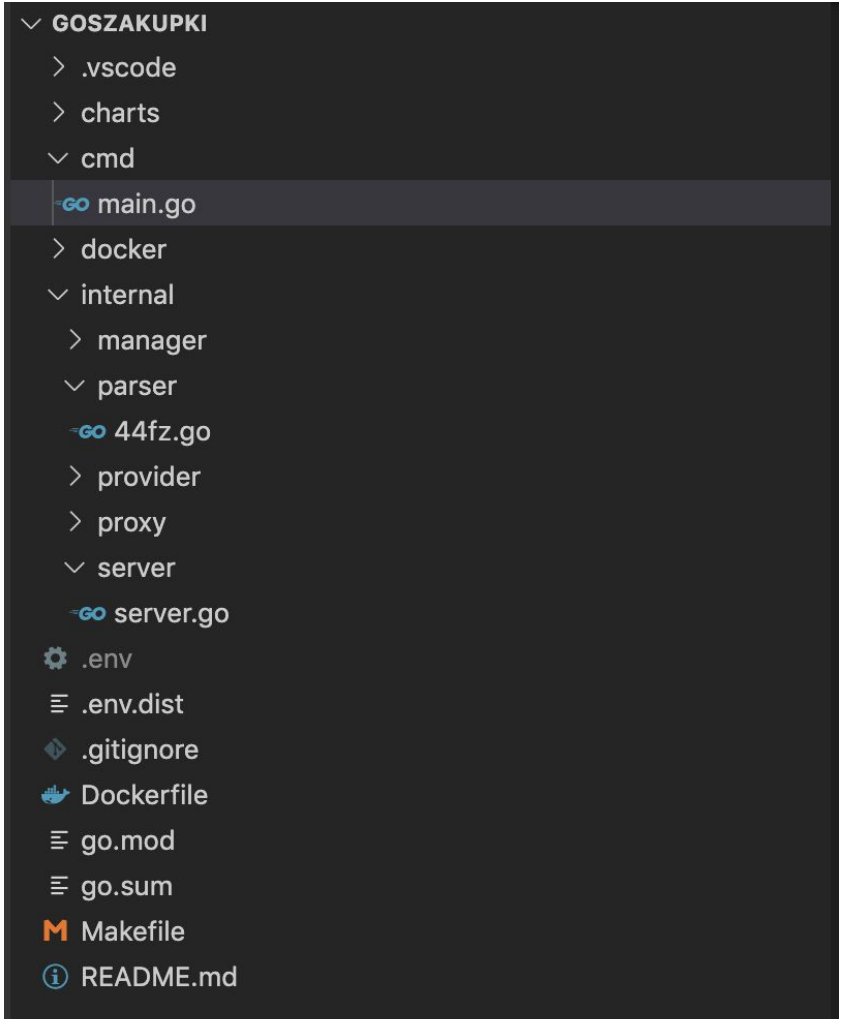
Автор: Денис Лимарев

Структура файлов

- Бизнес логика
- Провайдеры внешних систем
- Статические файлы

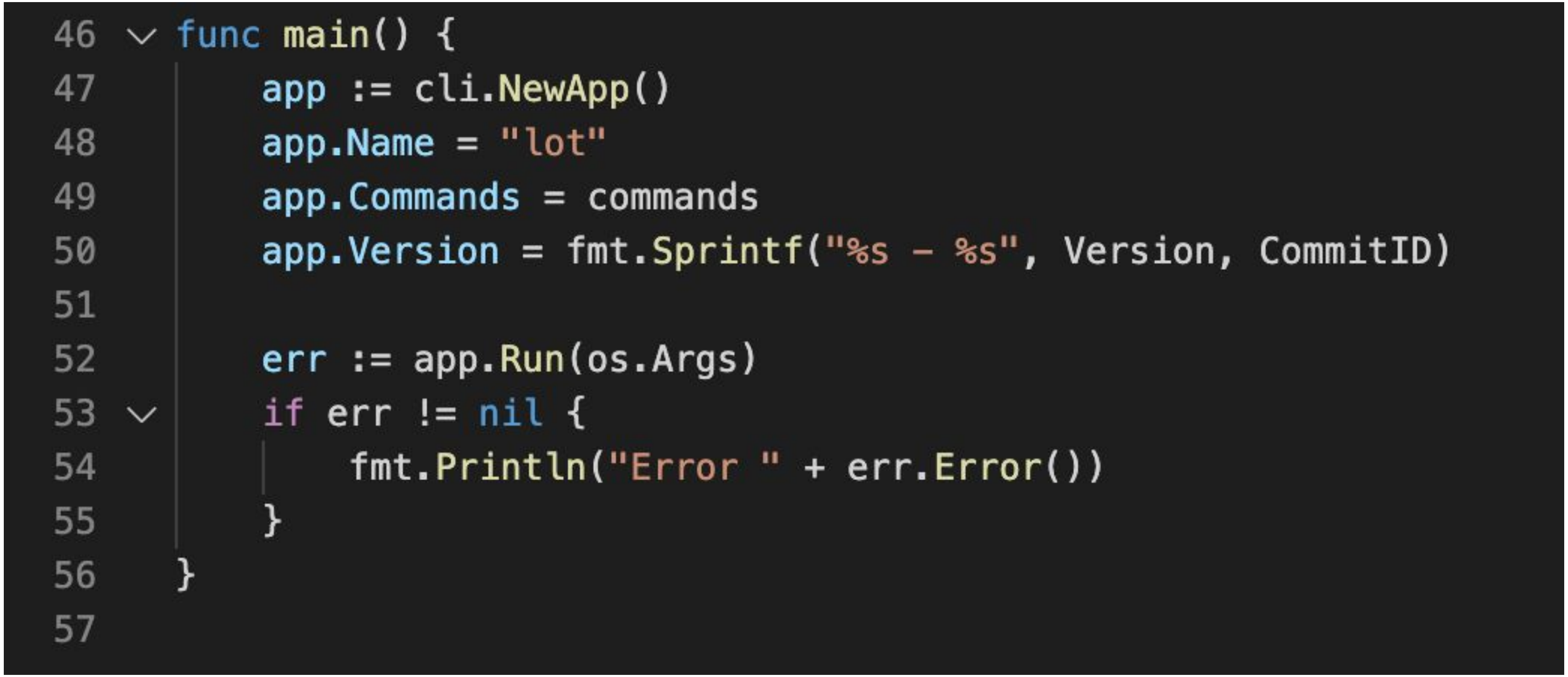
Примеры структуры проектов





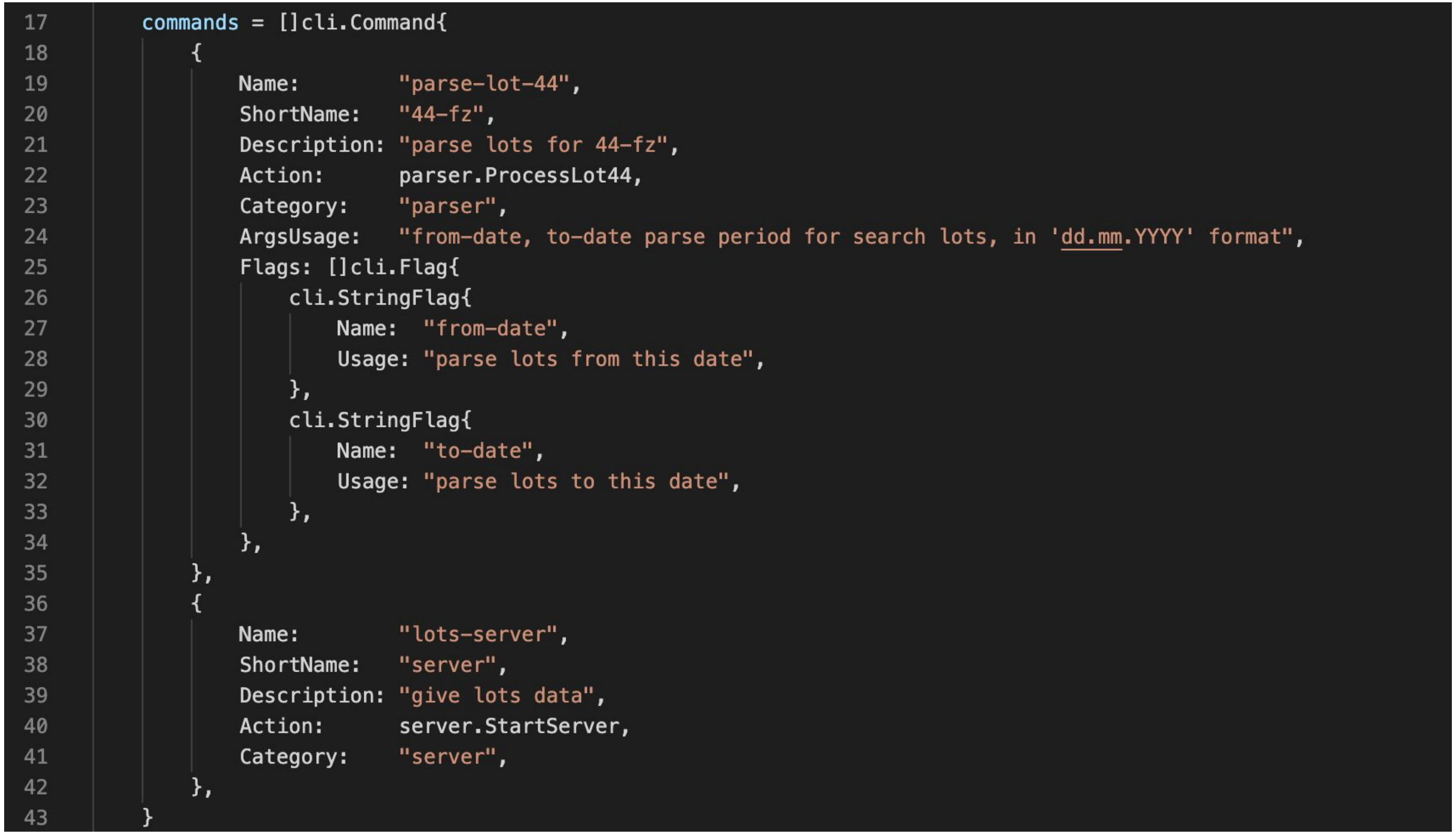
```
GOSZAKUPKI
  .vscode
  charts
  cmd
    main.go
  docker
  internal
    manager
    parser
      44fz.go
    provider
    proxy
    server
      server.go
  .env
  .env.dist
  .gitignore
  Dockerfile
  go.mod
  go.sum
  Makefile
  README.md
```

Точка входа



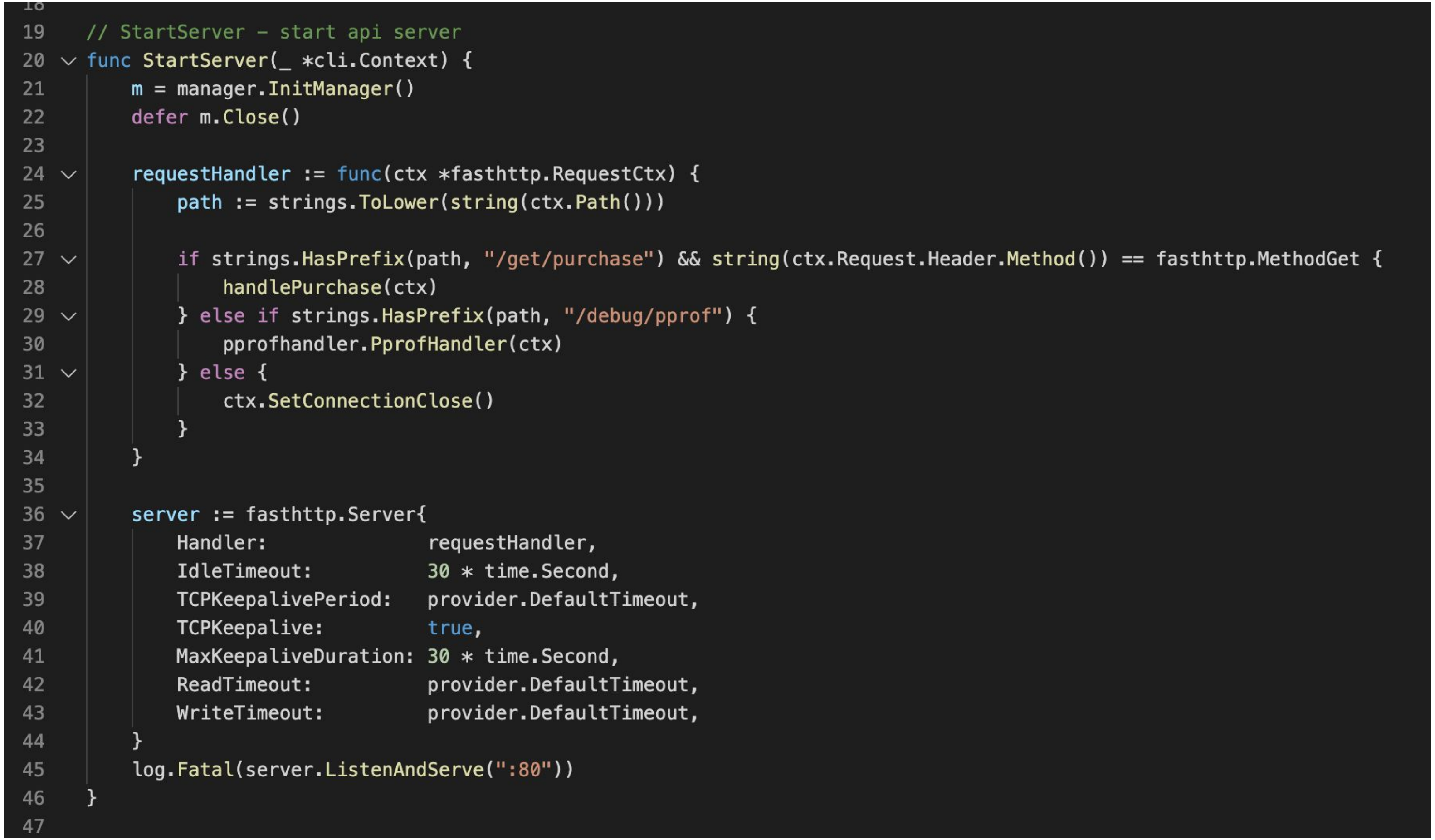
```go
func main() {
	app := cli.NewApp()
	app.Name = "lot"
	app.Commands = commands
	app.Version = fmt.Sprintf("%s - %s", Version, CommitID)

	err := app.Run(os.Args)
	if err != nil {
		fmt.Println("Error " + err.Error())
	}
}

```

Описание команд

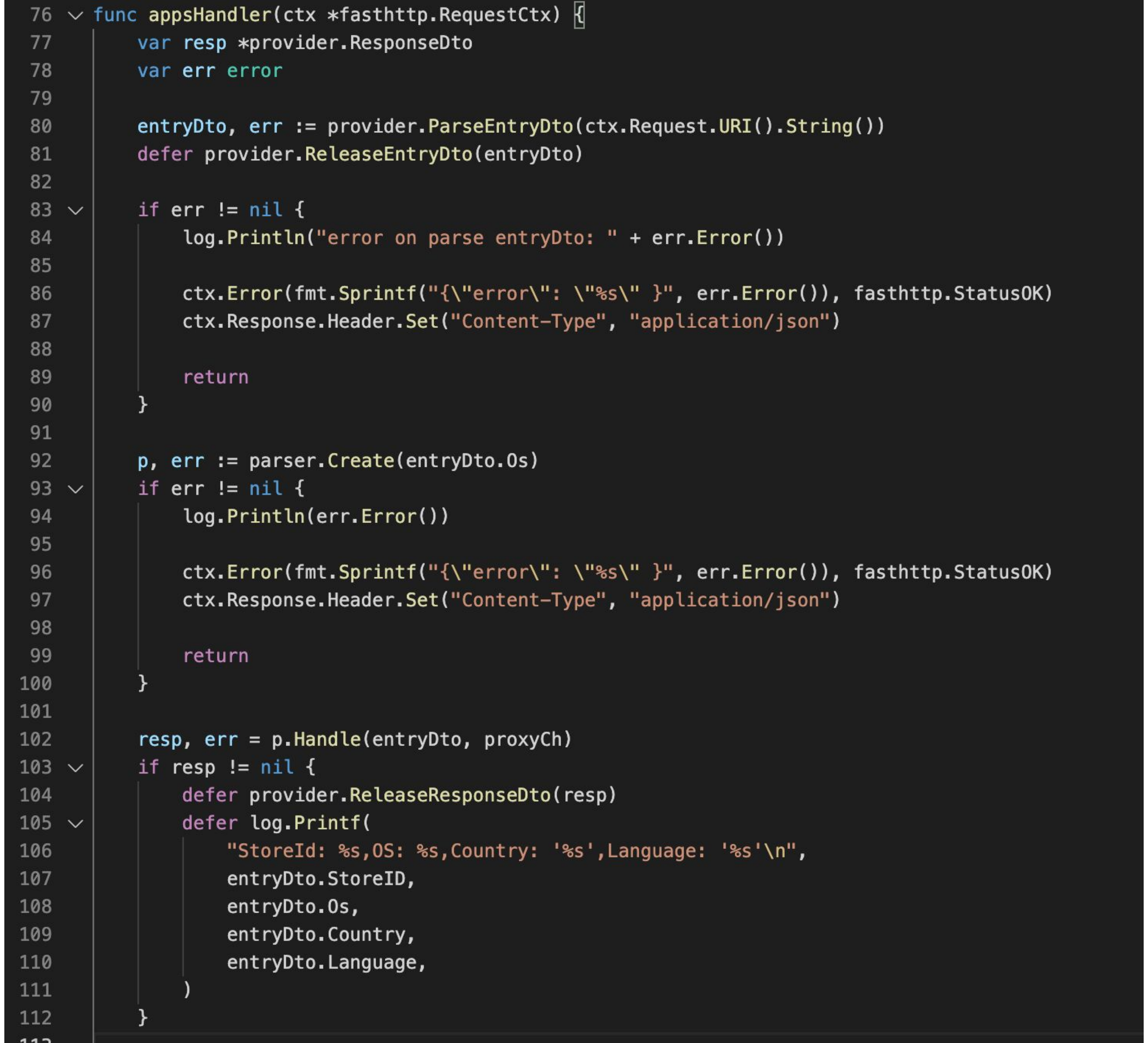


```go
	commands = []cli.Command{
		{
			Name:        "parse-lot-44",
			ShortName:   "44-fz",
			Description: "parse lots for 44-fz",
			Action:      parser.ProcessLot44,
			Category:    "parser",
			ArgsUsage:   "from-date, to-date parse period for search lots, in 'dd.mm.YYYY' format",
			Flags: []cli.Flag{
				cli.StringFlag{
					Name:  "from-date",
					Usage: "parse lots from this date",
				},
				cli.StringFlag{
					Name:  "to-date",
					Usage: "parse lots to this date",
				},
			},
		},
		{
			Name:        "lots-server",
			ShortName:   "server",
			Description: "give lots data",
			Action:      server.StartServer,
			Category:    "server",
		},
	}
```

Контроллер

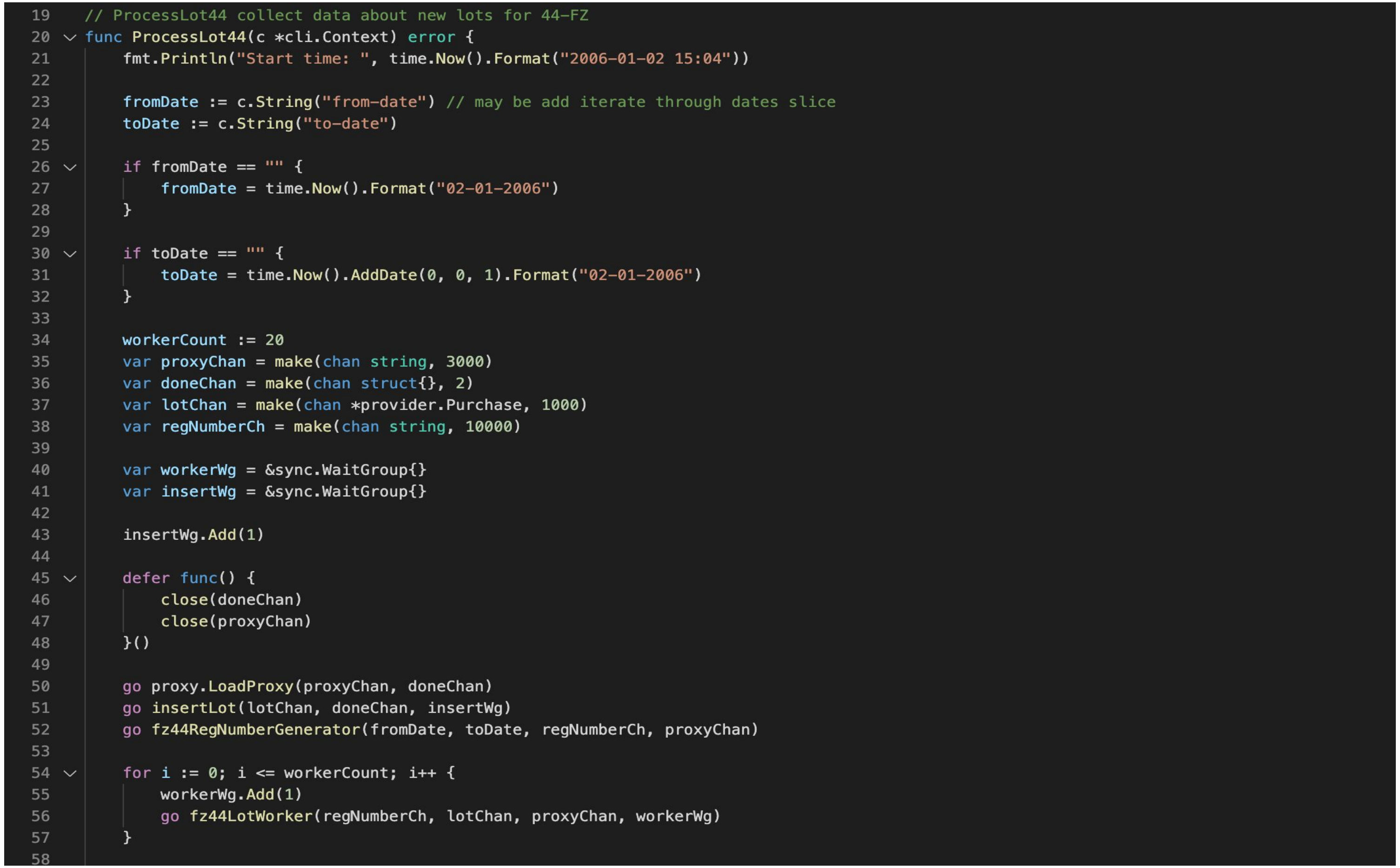


```go
// StartServer - start api server
func StartServer(_ *cli.Context) {
    m = manager.InitManager()
    defer m.Close()

    requestHandler := func(ctx *fasthttp.RequestCtx) {
        path := strings.ToLower(string(ctx.Path()))

        if strings.HasPrefix(path, "/get/purchase") && string(ctx.Request.Header.Method()) == fasthttp.MethodGet {
            handlePurchase(ctx)
        } else if strings.HasPrefix(path, "/debug/pprof") {
            pprofhandler.PprofHandler(ctx)
        } else {
            ctx.SetConnectionClose()
        }
    }

    server := fasthttp.Server{
        Handler:              requestHandler,
        IdleTimeout:          30 * time.Second,
        TCPKeepalivePeriod:   provider.DefaultTimeout,
        TCPKeepalive:         true,
        MaxKeepaliveDuration: 30 * time.Second,
        ReadTimeout:          provider.DefaultTimeout,
        WriteTimeout:         provider.DefaultTimeout,
    }
    log.Fatal(server.ListenAndServe(":80"))
}
```



```go
func appsHandler(ctx *fasthttp.RequestCtx) {
	var resp *provider.ResponseDto
	var err error

	entryDto, err := provider.ParseEntryDto(ctx.Request.URI().String())
	defer provider.ReleaseEntryDto(entryDto)

	if err != nil {
		log.Println("error on parse entryDto: " + err.Error())

		ctx.Error(fmt.Sprintf("{\"error\": \"%s\" }", err.Error()), fasthttp.StatusOK)
		ctx.Response.Header.Set("Content-Type", "application/json")

		return
	}

	p, err := parser.Create(entryDto.Os)
	if err != nil {
		log.Println(err.Error())

		ctx.Error(fmt.Sprintf("{\"error\": \"%s\" }", err.Error()), fasthttp.StatusOK)
		ctx.Response.Header.Set("Content-Type", "application/json")

		return
	}

	resp, err = p.Handle(entryDto, proxyCh)
	if resp != nil {
		defer provider.ReleaseResponseDto(resp)
		defer log.Printf(
			"StoreId: %s,OS: %s,Country: '%s',Language: '%s'\n",
			entryDto.StoreID,
			entryDto.Os,
			entryDto.Country,
			entryDto.Language,
		)
	}
```

Команда



```go
// ProcessLot44 collect data about new lots for 44-FZ
func ProcessLot44(c *cli.Context) error {
    fmt.Println("Start time: ", time.Now().Format("2006-01-02 15:04"))

    fromDate := c.String("from-date") // may be add iterate through dates slice
    toDate := c.String("to-date")

    if fromDate == "" {
        fromDate = time.Now().Format("02-01-2006")
    }

    if toDate == "" {
        toDate = time.Now().AddDate(0, 0, 1).Format("02-01-2006")
    }

    workerCount := 20
    var proxyChan = make(chan string, 3000)
    var doneChan = make(chan struct{}, 2)
    var lotChan = make(chan *provider.Purchase, 1000)
    var regNumberCh = make(chan string, 10000)

    var workerWg = &sync.WaitGroup{}
    var insertWg = &sync.WaitGroup{}

    insertWg.Add(1)

    defer func() {
        close(doneChan)
        close(proxyChan)
    }()

    go proxy.LoadProxy(proxyChan, doneChan)
    go insertLot(lotChan, doneChan, insertWg)
    go fz44RegNumberGenerator(fromDate, toDate, regNumberCh, proxyChan)

    for i := 0; i <= workerCount; i++ {
        workerWg.Add(1)
        go fz44LotWorker(regNumberCh, lotChan, proxyChan, workerWg)
    }

```

Полезные ссылки

По структуре:

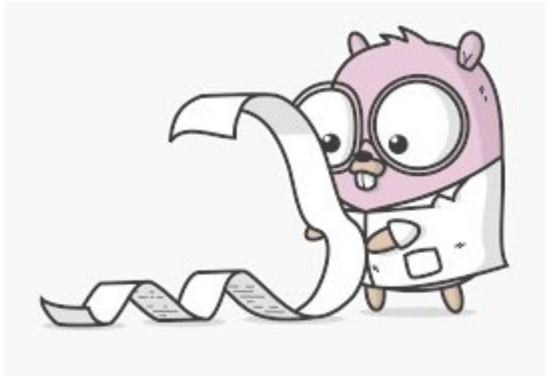
- https://github.com/golang-standards/project-layout (пример структуры с описаниями)
- https://github.com/urfave/cli (плагин для описания команд)
- https://github.com/peakle/goszakupki-parser (пример стуктуры)

Общие советы:

- https://github.com/avelino/awesome-go (сборник библиотек)
- https://github.com/cristaloleg/go-advice (список советов и трюков с Go)

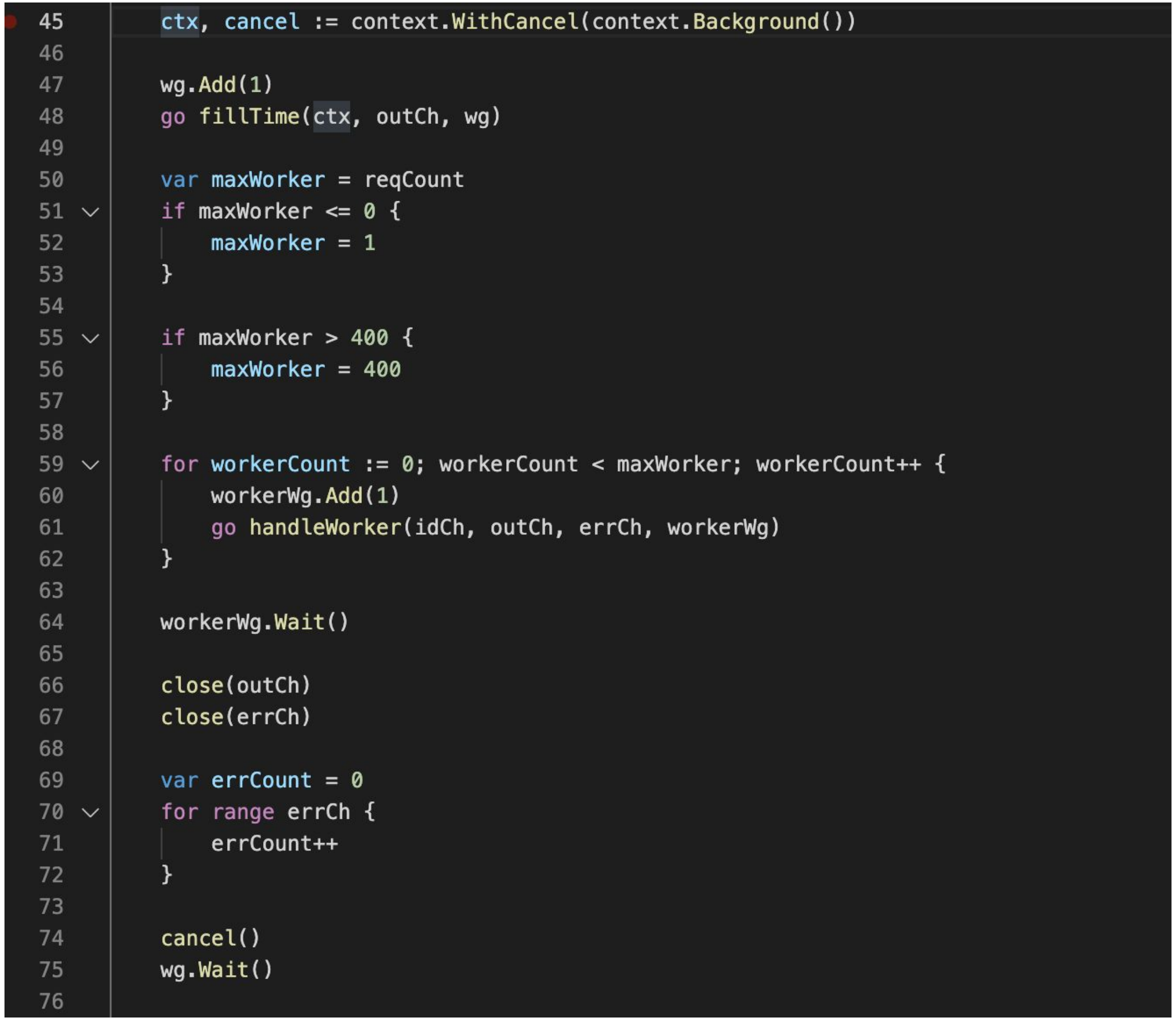
Спасибо за внимание

Ответы на вопросы

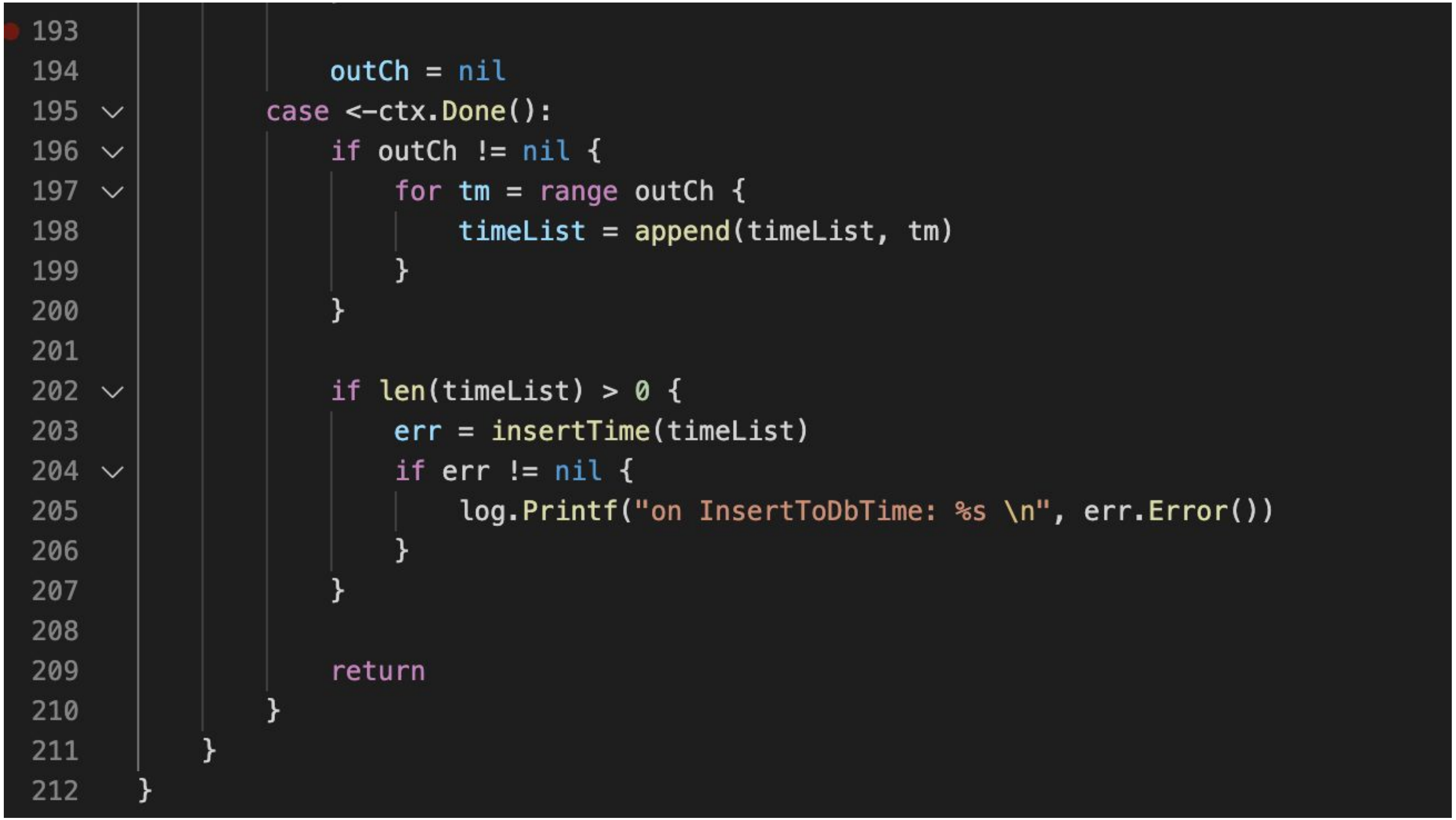


Как устроена многопоточность в go?

Как работает пакет Context?

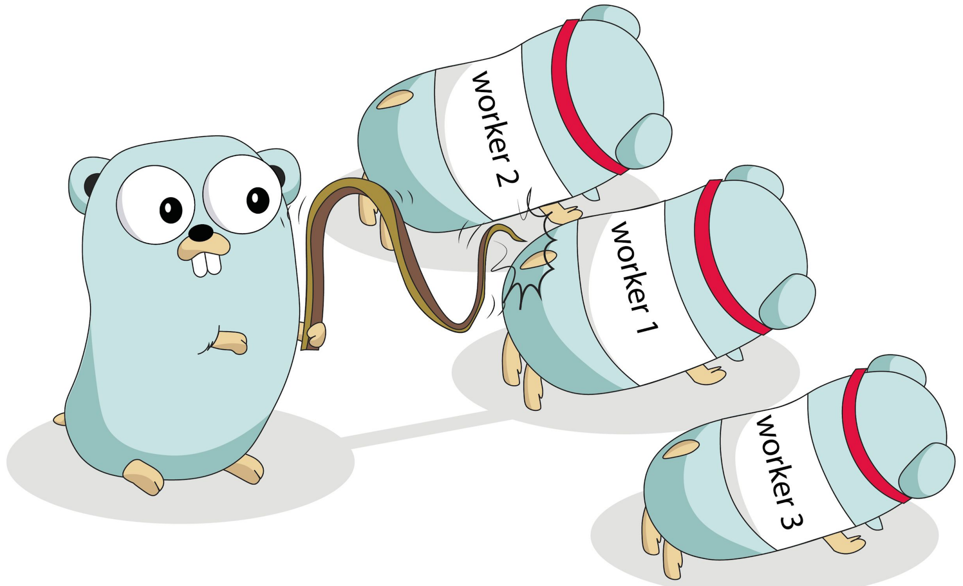


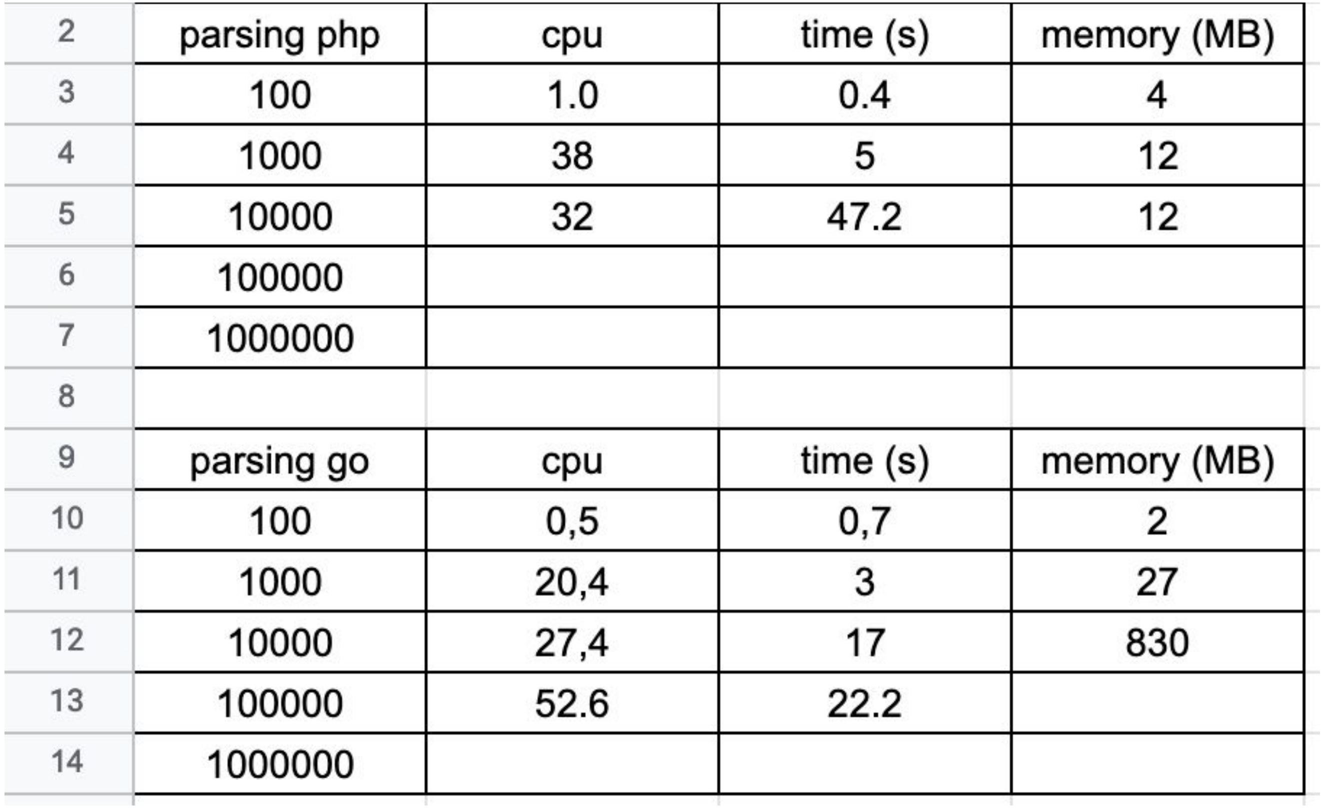
```go
	ctx, cancel := context.WithCancel(context.Background())

	wg.Add(1)
	go fillTime(ctx, outCh, wg)

	var maxWorker = reqCount
	if maxWorker <= 0 {
		maxWorker = 1
	}

	if maxWorker > 400 {
		maxWorker = 400
	}

	for workerCount := 0; workerCount < maxWorker; workerCount++ {
		workerWg.Add(1)
		go handleWorker(idCh, outCh, errCh, workerWg)
	}

	workerWg.Wait()

	close(outCh)
	close(errCh)

	var errCount = 0
	for range errCh {
		errCount++
	}

	cancel()
	wg.Wait()

```



```go

            outCh = nil
        case <-ctx.Done():
            if outCh != nil {
                for tm = range outCh {
                    timeList = append(timeList, tm)
                }
            }

            if len(timeList) > 0 {
                err = insertTime(timeList)
                if err != nil {
                    log.Printf("on InsertToDbTime: %s \n", err.Error())
                }
            }

            return
        }
    }
}
```

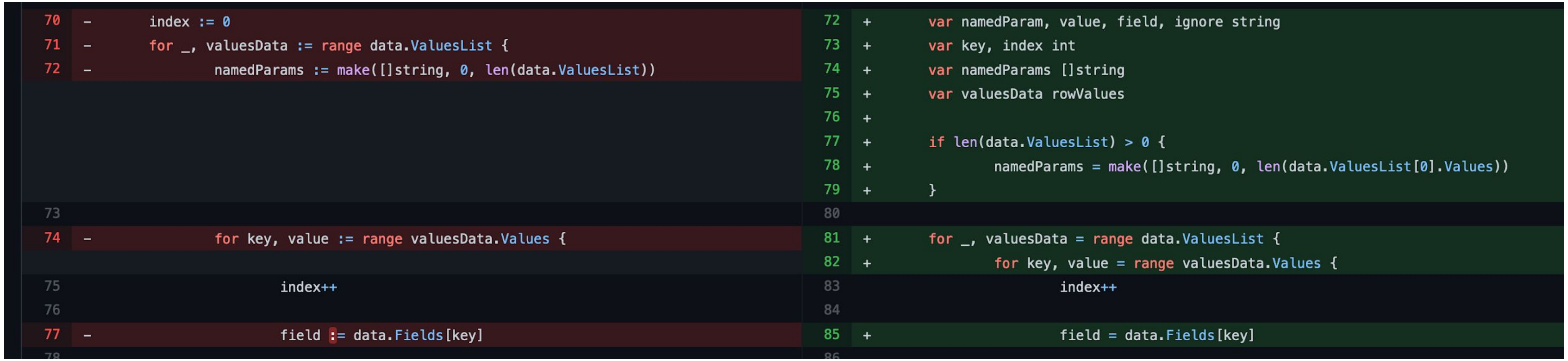
Ошибки недели





parsing php	cpu	time (s)	memory (MB)
100	1.0	0.4	4
1000	38	5	12
10000	32	47.2	12
100000			
1000000			

parsing go	cpu	time (s)	memory (MB)
100	0,5	0,7	2
1000	20,4	3	27
10000	27,4	17	830
100000	52.6	22.2	
1000000			



```
-    index := 0
-    for _, valuesData := range data.ValuesList {
-        namedParams := make([]string, 0, len(data.ValuesList))

-        for key, value := range valuesData.Values {
            index++

-            field := data.Fields[key]
```

```
+    var namedParam, value, field, ignore string
+    var key, index int
+    var namedParams []string
+    var valuesData rowValues
+
+    if len(data.ValuesList) > 0 {
+        namedParams = make([]string, 0, len(data.ValuesList[0].Values))
+    }

+    for _, valuesData = range data.ValuesList {
+        for key, value = range valuesData.Values {
            index++

+            field = data.Fields[key]
```

https://github.com/wakeapp/go-sql-generator/pull/1

parsing php	time (s)	memory (MB)
100	0.17	4
1000	5	12
10000	47.2	12
100000		
1000000		

parsing go	time (s)	memory (MB)
100	0,2	2
1000	0,4	10
10 000	7,3	20
100000	24,5	20
1000000	222,7	20