

**Инструкция по установке,
ПО «HEALTH CHECK»**

г. Москва
2025 г.

Содержание

1.	ВВЕДЕНИЕ	3
2.	СИСТЕМНЫЕ ТРЕБОВАНИЯ	3
3.	УСТАНОВКА И НАСТРОЙКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ	3

1. Введение

В данном документе описаны процессы, установки ПО «HEALTH CHECK» (Далее - ПО).

2. Системные требования

Для использования программного обеспечения пользователь должен иметь постоянный доступ к сети Интернет.

Оборудование пользователя должно соответствовать рекомендуемым требованиям для функционирования браузера, через который пользователь использует программное обеспечение. Для использования программного обеспечения производитель рекомендует пользователю использовать следующие браузеры:

- Google Chrome 87 и выше;
- Mozilla Firefox 84 и выше;
- Safari 14.0;
- Opera 72 и выше.

3. Установка и настройка программного обеспечения

Программное обеспечение распространяется в виде Интернет-сервиса с элементами программного кода для интеграции с проектом пользователя для получения данных.

Программное обеспечение предоставляется пользователю в виде готового к работе Интернет-сервиса, пользователь производит самостоятельную установку и настройку программного обеспечения посредством использования браузера и сети Интернет проходит процедуру регистрации в уже настроенном и готовом к работе программном обеспечении, развернутом на оборудовании производителя.

Для получения данных с проекта необходимо интегрировать контроллер в проект пользователя. Интеграция производится самостоятельно или с помощью производителя.

В зависимости от стека (языка) проекта пользователя внедряется тот или иной контроллер.

Список языков, на которых можно реализовать контроллер:

PHP (Laravel)

```
namespace App\Http\Controllers\Api;

use Illuminate\Http\JsonResponse;
use Illuminate\Support\Facades\DB;
use App\Http\Controllers\Controller;

class HealthCheckController extends Controller
{
    public function index(): JsonResponse
    {
        return response()->json([
            'time_now' => now()->utc(),
            'disk' => $this->diskInfo(),
            'memory' => $this->memoryInfo(),
            'cpu' => $this->cpuInfo(),
            'db' => $this->dbInfo()
        ]);
    }

    private function diskInfo(): array
    {
        $output = shell_exec("df -BG | grep '/' -w");
        $parts = preg_split('/\s+/', trim($output));

        return [
            'size' => (int) rtrim($parts[1], 'G'),
            'used' => (int) rtrim($parts[2], 'G'),
            'used_percent' => (int) rtrim($parts[4], '%')
        ];
    }

    private function cpuInfo(): array
    {
        $stat1 = preg_split('/\s+/', file('/proc/stat')[0]);
        sleep(1);
        $stat2 = preg_split('/\s+/', file('/proc/stat')[0]);

        $total1 = array_sum(array_slice($stat1, 1, 3));
        $total2 = array_sum(array_slice($stat2, 1, 3));
        $idle1 = $stat1[4];
        $idle2 = $stat2[4];

        $totalDiff = ($total2 + $idle2) - ($total1 + $idle1);
    }
}
```

```

$activeDiff = $total2 - $total1;

return [
    'used_percent' => (int) (($activeDiff / $totalDiff) * 100)
];
}

private function memoryInfo(): array
{
    $meminfo = file('/proc/meminfo');
    $total = (int) filter_var($meminfo[0], FILTER_SANITIZE_NUMBER_INT);
    $free = (int) filter_var($meminfo[2], FILTER_SANITIZE_NUMBER_INT);
    $used = $total - $free;

    return [
        'size' => (int) ($total / 1024 / 1024),
        'used' => round($used / 1024 / 1024, 1),
        'used_percent' => (int) (($used / $total) * 100)
    ];
}

private function dbInfo(): array
{
    try {
        DB::connection()->getPdo();
        $connected = true;
    } catch (\Exception $e) {
        $connected = false;
    }

    return ['connection' => $connected];
}
}

```

Python (Flask)

```

from flask import Flask, jsonify
import time
import os

app = Flask(__name__)

@app.route('/api/health_check', methods=['GET'])
def health_check():

```

```

return jsonify({
    "time_now": time.strftime('%Y-%m-%dT%H:%M:%SZ', time.gmtime()),
    "disk": disk_info(),
    "memory": memory_info(),
    "cpu": cpu_info(),
    "db": db_info()
})

def disk_info():
    import subprocess
    output = subprocess.check_output("df -BG | grep '/' -w", shell=True).decode()
    parts = output.split()
    return {
        "size": int(parts[1][:-1]),
        "used": int(parts[2][:-1]),
        "used_percent": int(parts[4][:-1])
    }

def memory_info():
    with open('/proc/meminfo') as f:
        lines = f.readlines()
        total = int("".join(filter(str.isdigit, lines[0])))
        free = int("".join(filter(str.isdigit, lines[2])))
        used = total - free
        return {
            "size": int(total / 1024 / 1024),
            "used": round(used / 1024 / 1024, 1),
            "used_percent": int((used / total) * 100)
        }

def cpu_info():
    def read_cpu():
        with open('/proc/stat') as f:
            parts = f.readline().split()
            return list(map(int, parts[1:5]))

    cpu1 = read_cpu()
    time.sleep(1)
    cpu2 = read_cpu()

    total1 = sum(cpu1)
    total2 = sum(cpu2)
    idle1 = cpu1[3]
    idle2 = cpu2[3]

```

```

total_diff = total2 - total1
idle_diff = idle2 - idle1
used_percent = int(((total_diff - idle_diff) / total_diff) * 100)

return {"used_percent": used_percent}

def db_info():
    try:
        import psycopg2
        conn = psycopg2.connect("dbname=health_check user=postgres password=postgres
host=db")
        conn.close()
        return {"connection": True}
    except:
        return {"connection": False}

if __name__ == '__main__':
    app.run(debug=True)

```

Go (Golang, net/http)

```

package main

import (
    "encoding/json"
    "net/http"
    "os/exec"
    "strconv"
    "strings"
    "time"
    "io/ioutil"
)

func main() {
    http.HandleFunc("/api/health_check", healthCheck)
    http.ListenAndServe(":8080", nil)
}

func healthCheck(w http.ResponseWriter, r *http.Request) {
    data := map[string]interface{}{
        "time_now": time.Now().UTC().Format(time.RFC3339),
        "disk":    diskInfo(),
        "memory":  memoryInfo(),
        "cpu":     cpuInfo(),
    }
}

```

```

        "db":    dbInfo(),
    }

    w.Header().Set("Content-Type", "application/json")
    json.NewEncoder(w).Encode(data)
}

func diskInfo() map[string]int {
    out, _ := exec.Command("bash", "-c", "df -BG | grep '/' -w").Output()
    fields := strings.Fields(string(out))
    size, _ := strconv.Atoi(strings.TrimSuffix(fields[1], "G"))
    used, _ := strconv.Atoi(strings.TrimSuffix(fields[2], "G"))
    percent, _ := strconv.Atoi(strings.TrimSuffix(fields[4], "%"))
    return map[string]int{"size": size, "used": used, "used_percent": percent}
}

func memoryInfo() map[string]interface{} {
    data, _ := ioutil.ReadFile("/proc/meminfo")
    lines := strings.Split(string(data), "\n")
    memTotal := parseMem(lines[0])
    memFree := parseMem(lines[2])
    memUsed := memTotal - memFree
    return map[string]interface{}{
        "size":    memTotal / 1024 / 1024,
        "used":    float64(memUsed) / 1024.0 / 1024.0,
        "used_percent": (memUsed * 100) / memTotal,
    }
}

func parseMem(line string) int {
    fields := strings.Fields(line)
    val, _ := strconv.Atoi(fields[1])
    return val
}

func cpuInfo() map[string]int {
    readCPU := func() []int {
        data, _ := ioutil.ReadFile("/proc/stat")
        fields := strings.Fields(strings.Split(string(data), "\n")[0])
        var res []int
        for _, f := range fields[1:5] {
            val, _ := strconv.Atoi(f)
            res = append(res, val)
        }
    }
    return res
}

```

```

    }
    c1 := readCPU()
    time.Sleep(1 * time.Second)
    c2 := readCPU()
    total1 := c1[0] + c1[1] + c1[2] + c1[3]
    total2 := c2[0] + c2[1] + c2[2] + c2[3]
    idleDiff := c2[3] - c1[3]
    totalDiff := total2 - total1
    used := 100 - (idleDiff * 100 / totalDiff)
    return map[string]int{"used_percent": used}
}

func dbInfo() map[string]bool {
    // Здесь должен быть код подключения к PostgreSQL (omitted for brevity)
    return map[string]bool{"connection": true}
}

```

Ruby

```

module Api
  class HealthCheckController < Api::ApiController
    def index
      render json: { time_now: Time.zone.now.utc, disk: disk_info, memory: memory_info, cpu:
cpu_info, db: db_info }
    end

    private

    def disk_info
      storage_info = df -BG | grep "/" -w
      {
        size: storage_info.split(' ')[1].to_i,
        used: storage_info.split(' ')[2].to_i,
        used_percent: storage_info.split(' ')[4].to_i
      }
    end

    def cpu_info
      proc0 = File.readlines('/proc/stat').grep(/^cpu /).first.split(' ')
      sleep 1
      proc1 = File.readlines('/proc/stat').grep(/^cpu /).first.split(' ')

      proc0_sum = proc0[1].to_i + proc0[2].to_i + proc0[3].to_i
      proc1_sum = proc1[1].to_i + proc1[2].to_i + proc1[3].to_i
    end
  end
end

```

```

proc_active = Float(proc1_sum - proc0_sum)

proc0_total = Float(proc0_sum + proc0[4].to_i)
proc1_total = Float(proc1_sum + proc1[4].to_i)
proc_total = proc1_total - proc0_total

{
  used_percent: ((proc_active / proc_total) * 100).to_i
}
end

def memory_info
  result = File.readlines('/proc/meminfo')
  mem_total = result[0].gsub(/^[^0-9]/, "").to_i
  mem_free = result[2].gsub(/^[^0-9]/, "").to_i
  mem_active = mem_total - mem_free
  {
    size: (mem_total * 0.000001).to_i,
    used: (mem_active * 0.000001).round(1),
    used_percent: (mem_active * 100) / mem_total
  }
end

def db_info
  {
    connection: ActiveRecord::Base.connected?
  }
end
end
end

```

Если пользователь не уверен, что может провести интеграцию самостоятельно, он может запросить готовую библиотеку под его стек или согласовать внедрение контроллера силами производителя.

После завершения процедуры регистрации и интеграции контроллера пользователь получает возможность использовать программное обеспечение в соответствии с его функциональным назначением.