|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| Федеральное государственное бюджетное образовательное учреждение  высшего образования  **«МИРЭА − Российский технологический университет»**  **РТУ МИРЭА** |
| Институт кибербезопасности и цифровых технологий |
| Кафедра КБ-14 «Цифровые технологии обработки данных» |

**ОТЧЕТ**

**по практической работе**

**«№4. ВЫПОЛНЕНИЕ ЗАДАЧ ПО РАСПИСАНИЮ»**

**по дисциплине «Обработка данных с веб-ресурсов и поисковых машин»**

|  |  |  |  |
| --- | --- | --- | --- |
| Выполнил |  |  | Парамонов П.К.  *фамилия, имя, отчество* |
| шифр | 22Б0723 | группа | БСБО-05-22 |
|  |  |  |  |
| Проверил |  | к.т.н., доцент  *ученая степень, должность* | Ильин Д.Ю.  *фамилия, имя, отчество* |

**Москва 2024г.**

## **Цель практической работы**

Цель настоящей практической работы — научиться конфигурировать исполнение задач по расписанию для сервисов сбора данных.

## **Задачи практической работы**

Для достижения поставленной цели необходимо решить следующие задачи:

1. Дополнить базу данных таблицей(ами) для фиксирования статусов исполнения задач.
2. Доработать проект на Node.js на основе результатов работы №3.
3. Установить программные библиотеки, реализующие исполнение задач по расписанию (либо удостовериться в функционировании cron-задач на уровне операционной системы).
4. Осуществить настройку исполнения задачи по расписанию.
5. Выполнить сбор данных в фоновом режиме.
6. Удостовериться в корректном исполнении задач и сохранении результатов в БД.
7. Подготовить ответы на контрольные вопросы.
8. Составить отчет о проведенной работе

**Ход выполнения практической работы**

Данная работа будет выполнена при помощи библиотеки node-schelude, а так же библиотеки seqeulize для создания модели логирования выполнения заданий.

С помощью node-schelude можно выполнить как и единоразовое задание, так и запрограммировать задание на цикличное выполнение.

Работа представлена 2 скриптами: taskLogFunc.js иt taskLogModel.js. В файле TaskLogModel.js представлена модель для записи в БД, в taskLogFunc.js прописан весь функционал необходимый для данной практической. Код данных скриптов будет в Приложении А.

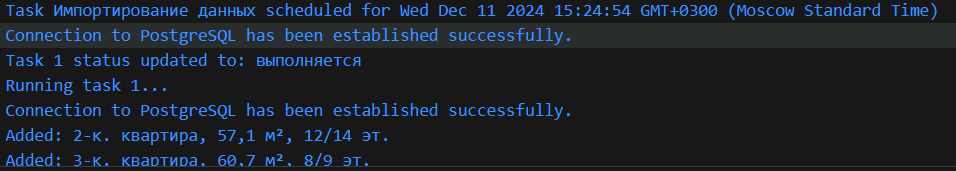


Рис 1 – пример выполнения задачи

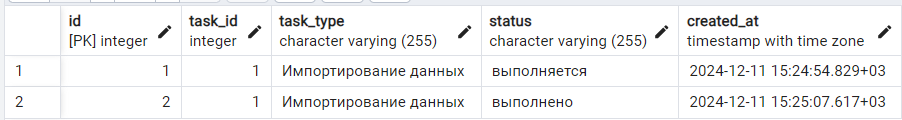


Рис 2 – пример записи задачи в БД

## **Трудности в ходе выполнения практической работы**

Никаких трудностей не возникло.

## **Ответы на контрольные вопросы**

**1. В чем отличия подходов с использованием Cron и программных библиотек исполнения задач по расписанию?**

* **Cron**: Работает на уровне операционной системы, не зависит от приложения, требует ручной настройки через файл Crontab.
* **Программные библиотеки**: Работают в рамках приложения, позволяют управлять задачами из кода, могут учитывать внутреннюю логику и состояния приложения.

**2. Какие параметры расписания могут быть заданы в файле Crontab?**

* Минуты (0-59)
* Часы (0-23)
* Дни месяца (1-31)
* Месяцы (1-12)
* Дни недели (0-7, где 0 и 7 — воскресенье)
* Специальные символы:
  + \* — любое значение.
  + , — перечисление значений.
  + - — диапазон значений.
  + / — шаг выполнения.

**3. Какими редакторами может изменяться файл Crontab?**

* Стандартный редактор по умолчанию (nano, vim, или другой, настроенный в $EDITOR).
* Можно указать редактор вручную перед запуском:

EDITOR=nano crontab -e

**4. В чем отличия пользовательских файлов Crontab от системных?**

* **Пользовательские**:
  + Создаются для конкретного пользователя.
  + Управляются командой crontab.
  + Выполняются от имени пользователя.
* **Системные**:
  + Находятся в /etc/crontab или /etc/cron.d/.
  + Требуют указания имени пользователя в строке задачи.
  + Могут выполняться от имени системных учётных записей.

**5. Как записать команду в файле Crontab, если она содержит переносы строк?**

\* \* \* \* \* echo "Начало задачи" && \

echo "Продолжение задачи" && \

echo "Конец задачи"

**Вывод**

В ходе выполнения практической работы «Выполнение задач по расписанию**»** была рассмотрена библиотека node-schelude для автоматизации процессов сбора данных и их импорта.

**Приложение А. Листинг кода**

TaskLogModel.js

const { DataTypes} = require('sequelize');

const sequelize = require('./database');

const TaskStatusLog = sequelize.define('TaskStatusLog', {

    task\_id: {

      type: DataTypes.INTEGER,

      allowNull: false

    },

    task\_type: {

      type: DataTypes.STRING,

      allowNull: false

    },

    status: {

      type: DataTypes.STRING,

      allowNull: false

    },

    created\_at: {

      type: DataTypes.DATE,

      defaultValue: DataTypes.NOW

    }

  }, {

    tableName: 'task\_status\_log',

    timestamps: false

  });

  module.exports = TaskStatusLog;

TaskLogFunc.js

const TaskStatusLog = require('./taskLogModel');

const { fork } = require('child\_process');

const { time } = require('console');

const schedule = require('node-schedule');

const path = require('path');

const avitoParserPath = path.resolve(\_\_dirname, '../request\_avito\_parser');

const cianParserPath = path.resolve(\_\_dirname, '../request\_cian\_parser');

const mirkvParserPath = path.resolve(\_\_dirname, '../request\_mirkv\_parser');

const moveParserPath = path.resolve(\_\_dirname, '../request\_move\_parser');

const importPath = path.resolve(\_\_dirname, './import');

async function logTaskStatus(taskId, taskType, status) {

  try {

    await TaskStatusLog.create({

      task\_id: taskId,

      task\_type: taskType,

      status: status

    });

    console.log(`Task ${taskId} status updated to: ${status}`);

  } catch (error) {

    console.error('Error logging task status:', error);

  }

}

async function getNextTaskId() {

  try {

    // Находим максимальное значение taskId в базе данных

    const maxTask = await TaskStatusLog.findOne({

      attributes: [[TaskStatusLog.sequelize.fn('MAX', TaskStatusLog.sequelize.col('task\_id')), 'maxTaskId']]

    });

    // Извлекаем максимальное значение

    const maxTaskId = maxTask && maxTask.get('maxTaskId') !== null

      ? parseInt(maxTask.get('maxTaskId'), 10)

      : 0;

    // Возвращаем новый taskId

    return maxTaskId + 1;

  } catch (error) {

      console.error('Error fetching max task ID:', error);

      throw new Error('Не удалось получить taskId');

  }

}

function scheduleTask(taskType, scheduledTime, scriptPath) {

    schedule.scheduleJob(scheduledTime, async () => {

      const taskId = await getNextTaskId();

      // Логируем начало выполнения задачи

      await logTaskStatus(taskId, taskType, 'выполняется');

      try {

        console.log(`Running task ${taskId}...`);

        const child = fork(scriptPath); // Запускаем указанный файл

        child.on('message', message => {

          console.log(`Message from task ${taskId}:`, message);

        });

        child.on('exit', async code => {

          if (code === 0) {

            console.log(`Task ${taskId} completed successfully.`);

            await logTaskStatus(taskId, taskType, 'выполнено');

          } else {

            console.error(`Task ${taskId} failed with exit code ${code}.`);

            await logTaskStatus(taskId, taskType, 'ошибка');

          }

        });

        child.on('error', async error => {

          console.error(`Error in task ${taskId}:`, error);

          await logTaskStatus(taskId, taskType, 'ошибка');

        });

      } catch (error) {

        console.error(`Error while running task ${taskId}:`, error);

        await logTaskStatus(taskId, taskType, 'ошибка');

      }

    });

    console.log(`Task ${taskType} scheduled for ${scheduledTime}`);

}

const newTime = new Date(Date.now() + 10\*1000);

//Примеры задач для выполнения

scheduleTask('Парсинг', newTime, avitoParserPath);

scheduleTask('Импортирование данных', newTime, importPath);