Санкт-Петербургский политехнический университет Петра Великого
Институт Компьютерных Наук и Технологий
Кафедра «Высшая школа программной инженерии»

Курсовой проект

по дисциплине: «Микропроцессорные системы»

Выполнил
студент гр.33534/21 _____ С.А. Фомин

Проверил
преподаватель _____ С.К. Круглов

Санкт-Петербург
2018

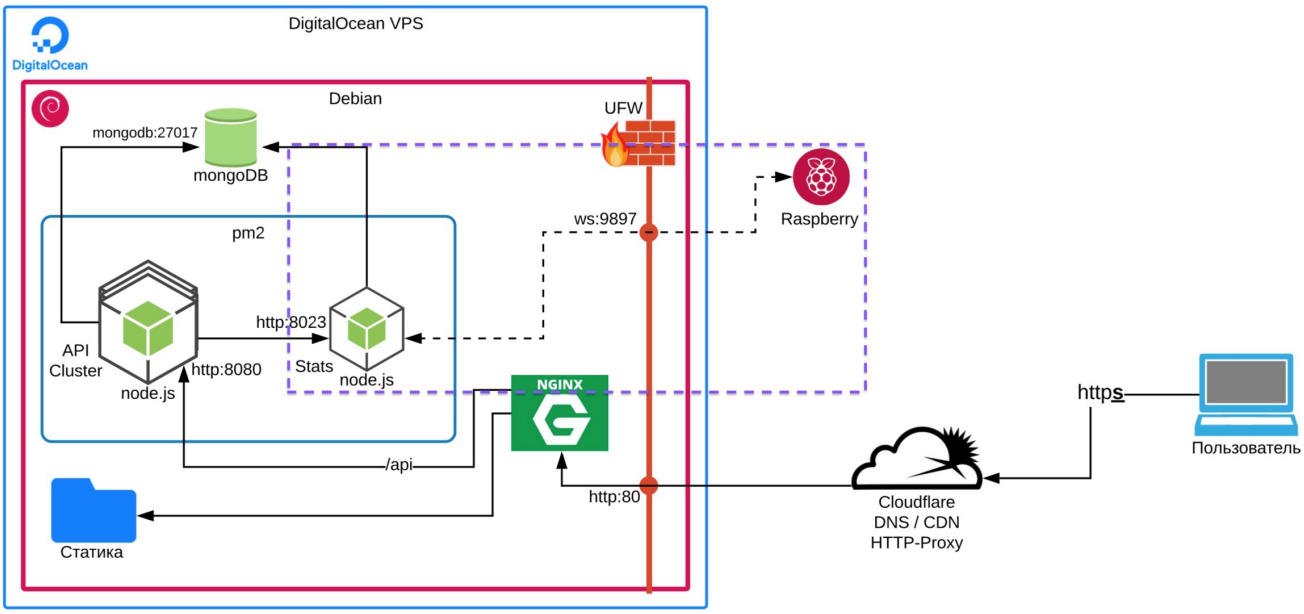

Рис. 1: Схема архитектуры крупным планом.

Постановка задачи

Разработать клиент-серверное приложение для вывода на внешний жидкокристаллический дисплей (с помощью Raspberry Pi) количества сокращенных ссылок из базы данных сервиса short.taxnuke.ru в режиме реального времени.

Ход работы

На рисунке 1 изображена схема сервиса short.taxnuke.ru в целом, а также интегрированного в него разрабатываемого клиент-серверного приложения (в пунктирной фиолетовой рамке).

В качестве программной среды было принятно решение использовать Node.js - кроссплатформенное решение, использующее в качестве JavaScript-движка V8, написанный на C. Для Node.js имеется множество готовых библиотек на JavaScript с привязками к нативным модулям на C++ и C и с многопоточностью. Выбор обусловлен тем, что изначально сервис short.taxnuke.ru тоже написан Node.js и использоание одной программной среды и языка позволяет минимизировать количество зависимостей и упростить интерфейс между компонентами. Также Node.js позволяет порождать дочерние процессы и передавать в них аргументы командной строки, управляющие последовательности и использовать потоки ввода/вывода.

Проработка архитектуры и регламента взаимодействия между сервисами

В первую очередь необходимо было продумать, какие механизмы и средства будут использоваться для взаимодействия сервисом между собой и сервиса статистики и его потребителей. В качестве протокола для общения сервиса статистики и его клиентов было решено использовать websocket, чтобы избежать опрашивания основного сервиса (даже "длинного"опрашивания), которое неизбежно повлекло бы за собой ухудшение производительности основного сервиса при количестве клиентов (потребителей статистики оного), стремящемся к бесконечности.

В качестве механизма взаимодействия сервисов между собой был выбран webhook, то есть один сервис будет отправлять http-запрос любым методом в любой эндпоинт другого сервиса с целью просто "дернуть"его и уведомить о том, что данные обновились. Было решено также не передавать с webhook никаких данных от одного сервиса к другому, это позволило избежать лишних развертываний основного сервиса по причинам изменений в интерфейсе сервиса статистики или формата данных.

Вместо этого, сервис статистики открывает свое личное соединение с базой данных и по пришествии webhook сам параллельно отправляет к базе запрос и реализует там своими средствами всю бизнес логику, инкапсулируя ее от основного сервиса.

Создание фундамента

Было принято решение использовать монорепозиторий git для управления контролем версий проекта, ввиду того, что логически это один т.н. микросервис, в дополнение к тому, что клиент и сервер изоморфны и используют одни библиотеки и написаны на одном языке.

В качестве линтера (утилиты контроля качества кода) используется eslint, его конфигурационный файл можно увидеть в приложении 4.

В качестве сиситемы доставки и развертки используется стандартная для Node.js связка из pm2 и git. pm2 на машине-разработчике подключается к машине-исполнителю по ssh, выполняет git clone, разворачивает проект из исходого кода и переставляет символьную ссылку на последний релиз, а так же берет на себя управление переменными окружения. Конфигурационный файл pm2 можно увидеть в приложении 3

Разработка сервиса-сервера

В качестве библиотеки для работы с websocket была выбрана самая популярная - "ws". Драйвер для общения с базой данных MongoDB было решено использовать официальный - "mongodb".

При первом старте, сервер пытается подключиться к базе данных, если все произошло успешно, то создается http-сервер, который слушает на порт из переменной окружения, и аналогично далее создается websocket-сервер.

При установлении нового соединения, сервер websocket отпрвляет тестовые данные для подтверждения соединения, и через 3 секунды отправляет клиенту актуальную информацию о статистике. Далее, websocket-сервер оповещает своих клиентов только после получения webhook от основного сервиса, при этом обновляя информацию о статистике.

Разработка сервиса-клиента

В качестве библиотеки для работы с websocket была также выбрана "ws". При старте сервис-клиент пытается установить соединение с сервисом-сервером по websocket и далее выводит информацию на жидкокристаллический экран с помощью библиотеки "lcdi2c которая позволяет абстрагироваться от низкоуровневой логики общения по протоколу I2C.

Результат работы

В результате выполнения курсового проекта было разработано серверное и клиентское приложения, работающие в паре и выполняющие поставленную задачу.

ПРИЛОЖЕНИЕ 1. Код сервера

```
const WebSocket = require('ws')
const hookPort = process.env.HOOK_PORT

const url = `mongodb://localhost:${process.env.MONGO_PORT}`

require('mongodb').MongoClient
  .connect(url)
  .then(mongoConnection => {
    let latestData = null

    const httpServer = require('express')()
    const websocketServer = new WebSocket.Server({
      port: process.env.WS_PORT
    })

    websocketServer.broadcast = function broadcast(data) {
      console.info(`broadcasting ${JSON.stringify(data)}...`)

      websocketServer.clients.forEach(function each(client) {
        if (client.readyState === WebSocket.OPEN) {
          client.send(data)
        }
      })
    }

    websocketServer.on('connection', function connection(ws) {
      ws.send('hi')

      setTimeout(() => {
        ws.send(latestData)
      }, 3000)
    })

    httpServer.use('*', (req, res) => {
      mongoConnection.db('url-shortener').collection('aliases')
        .countDocuments()
        .then(count => {
          res.end()

          const data = JSON.stringify({
            link_count: count
          })

          websocketServer.broadcast(data)

          latestData = data
        })
        .catch(console.error)
```

```
  })

  httpServer.listen(hookPort)
    .on('listening', () => {
      console.log(`HTTP server listening on port ${hookPort}`)
    })
})
.catch(console.error)
```

ПРИЛОЖЕНИЕ 2. Код клиента

```
const I2CLCDConnection = require('lcdi2c')
const WebSocket = require('ws')

/**
 * i2cdetect 1
 */
const I2C_ADDR = 0x3f
const lcdConnection = new I2CLCDConnection(1, I2C_ADDR, 16, 2)
lcdConnection.println('short.taxnuke.ru', 1)

const ws = new WebSocket(
  `ws://${process.env.WS_HOST}:${process.env.WS_PORT}`
)

ws.on('open', () => {
  lcdConnection.println('connected', 2)
  console.info('WebSocket connection established')
})

ws.on('close', function close() {
  console.warn('WebSocket connection closed')
  lcdConnection.println('conn closed', 2)
})

ws.on('message', data => {
  console.log(data)

  lcdConnection.clear()

  try {
    data = JSON.parse(data)
  } catch (e) {
    console.error(data)
  } finally {
    lcdConnection.println('short.taxnuke.ru', 1)
    lcdConnection.println(
      `links: ${data.link_count || 'Error =('}`,
      2
    )
  }
})
```

ПРИЛОЖЕНИЕ 3. Содержимое конфигурационного файла для развертки

```
module.exports = {
  // Options reference: https://pm2.io/doc/en/runtime/reference/ecosystem-f
  apps: [
    {
      /**
       * SERVER
       */
      name: 'url-shortener-status-server',
      script: 'source/server.js',
      env: {
        watch: true,
        HOOK_PORT: 8813,
        WS_PORT: 9897,
        MONGO_PORT: 28017,
        NODE_ENV: 'development'
      },
      env_vps: {
        watch: false,
        HOOK_PORT: 8813,
        WS_PORT: 9897,
        MONGO_PORT: 27017,
        NODE_ENV: 'production'
      }
    },
    {
      /**
       * CLIENT
       */
      name: 'url-shortener-status-client',
      script: 'source/client.js',
      env: {
        watch: true,
        WS_PORT: 9897,
        WS_HOST: 'localhost',
        NODE_ENV: 'development'
      },
      env_raspberry: {
        watch: false,
        WS_PORT: 9897,
        WS_HOST: '138.68.183.160',
        NODE_ENV: 'production'
      }
    }
  ],

  deploy: {
    vps: {
```

```
      user: 'adminus',
      host: '138.68.183.160',
      ref: 'origin/master',
      repo: 'git@github.com:taxnuke/url-shortener-status.git',
      path: '/var/www/stat.short.taxnuke.ru',
      'post-deploy': 'npm i && pm2 reload ecosystem.config.js --env vps --c
    },
    raspberry: {
      user: 'pi',
      host: '192.168.1.70',
      ref: 'origin/master',
      repo: 'git@github.com:taxnuke/url-shortener-status.git',
      path: '/var/www/stat.short.taxnuke.ru',
      'post-deploy': 'npm i && pm2 reload ecosystem.config.js --env raspber
    }
  }
}
```

ПРИЛОЖЕНИЕ 4. Конфигурация линтера

```
module.exports = {
  'env': {
    'es6': true,
    'node': true
  },
  'extends': 'eslint:recommended',
  'parserOptions': {
    'ecmaVersion': 2015
  },
  'rules': {
    'indent': [
      'error',
      2
    ],
    'linebreak-style': [
      'error',
      'unix'
    ],
    'quotes': [
      'error',
      'single'
    ],
    'semi': [
      'error',
      'never'
    ],
    'no-console': 0
  }
};
```