УДК 004.65

О РАЗРАБОТКЕ NOSQL СУБД GOLDENRACEDB КАК АЛЬТЕРНАТИВЫ GOOGLE FIREBASE

Р. В. Мосоло́в[0000-0002-4399-4397]

Сеть 70 салонов красоты «GRUSHKA» (Москва)

R.V.Mosolov@ya.ru

Аннотация

Описаны опыт разработки новой нереляционной системы управления базами данных, названной GoldenRaceDB, и предпосылки, располагавшие к ее созданию в контексте проблемы импортозамещения зарубежных технологий. Новая технология реализована на базе серверного окружения Node.js.

Материал статьи предполагает наличие у читателя опыта разработки серверной части системы минимум на одном высокоуровневом языке программирования либо опыта разработки СУБД. Описанная технология не является свободно распространяемой (open source) и была предназначена исключительно для решения задач внутри организации, места ее создания. Однако по прилагаемым в статье листингам можно понять общий вектор создания аналогичной СУБД для разработки собственной упраздненной СУБД на другом высокоуровнем языке программирования.

Ключевые слова*: система управления базами данных, СУБД, NoSQL СУБД, разработка СУБД, создание СУБД, оператор СУБД, СУБД примеры, запросы в СУБД, средства СУБД, импортозамещение софт, аналоги СУБД.*

О ПРЕДПОСЫЛКАХ СОЗДАНИЯ

В рамках курируемого автором в течение последних 2 лет международного хобби-проекта, над которым ведутся работы в качестве разработчика полного цикла (full-stack), встала задача замещения зарубежной технологии Google Firebase, являющейся нереляционной [3] системой управления базами данных (NoSQL СУБД). Таким временным решением стала разработка самописной СУБД, названной GoldenRaceDB в честь организации, в стенах которой она была создана.

Отметим, что разработка собственной СУБД виделась далеко не самым оптимальным решением[1], и данному решению предшествовал целый ряд причин:

1. **Периодические сбои в работе прежней СУБД.** Как отмечалось ранее, прежней СУБД была Google Firebase. На протяжении двух лет она служила достаточно исправно, однако после запуска спецоперации на Украине, примерно с февраля 2022 г., в ее работе начали наблюдаться довольно частые сбои, вплоть до того, что они начали препятствовать получению доступа к некоторым данным, важным для предметной области.

2. **Нарушение функциональной работы минимум 4 сервисов.** Данный пункт вытекает из п. 1 Списка предпосылок. Было замечено, что СУБД начала оказывать отрицательное влияние в той или иной степени на 4 сервиса (см. рис. 1) трех разрабатываемых веб-приложений организации, где требовалось взаимодействие серверной и клиентской частей.

3. **Нарушение работы 7 из 12 отделов организации хобби-проекта.** Данный пункт вытекает из п. 2 Списка предпосылок. На четырех вышеупомянутых сервисах, как оказалось, за прошедшие два года стала завязана работа свыше половины отделов организации, в рамках которой осуществлялось развитие хобби-проекта. Обнаружение подобного количества взаимосвязей стало точкой принятия решения о срочной *миграции данных*, т. к. доля времени, уделяемая работе с жалобами о неисправности отдельных сервисов, получаемых от пользователей, стала значительно превышать долю времени, уделяемую вопросам поставки новой функциональности для веб-приложений.

4. **Недоступность СУБД MongoDB.** В качестве альтернативы технологии Google Firebase автор рассматривал другую нереляционную СУБД, MongoDB (https://habr.com/ru/news/t/655291/), с которой у него имелся некоторый опыт работы ранее. Однако данная СУБД нас тоже «подвела» (см. рис. 2). Данная технология не просто возвращала сбои в работе, она напрочь остановила

1 Большинство коммерческих организаций предпочитает использовать промышленные решения вместо самописных.

поставки своих услуг в России, предупредив пользователей из России и Беларуси о «немедленном» удалении их данных.

5. Постоянная статья расходов. Одной из причин периодических сбоев в работе СУБД Google Firebase являлось то, что хобби-проект начал превышать ранее достаточную для организации квоту ресурсов сервера, где хранились данные организации. По мере роста организации за два года росло и число ее пользователей. Так, например, в первом квартале 2022 г. был снят документальный фильм, размещенный на видеохостинге «Ютуб», собравший за год около 2 млн. просмотров. Выпуск фильма увеличил среднюю посещаемость сайта в 2,5 раза относительно периода, когда аналитика только начала собираться, и была установлена система сбора метрик посещаемости («Яндекс.Метрика»). А на начало 2023 г. статистика посещаемости уже насчитывала 50 тыс. просмотров главной страницы за год при географии пользователей, исчисляемой 83 странами. Поскольку ресурсы VPS[2] еще не были полностью исчерпаны[3], оплата для использования зарубежной технологии казалось нерациональной с точки зрения экономичности процесса разработки. К тому же, даже несмотря на имеющиеся ресурсы, мы бы не смогли оплатить увеличение квот, поскольку Google Firebase, как и большинство других владельцев зарубежных технологий, стала запрещать оплату услуг с российских банковских карт.

6. Низкая скорость отклика либо полное его отсутствие на действия пользователей. Одним из стратегически важных веб-приложений организации являлась разработка Библиотеки 2.0 проекта, который позволит: 1) читать около 400 книг, переведенных в перспективе 5 лет примерно на 20 языков, онлайн; 2) вводить элементы монетизации[4] системы, которая будет осуществляться через продажу абонементов в онлайн-библиотеку. В данном проекте главным компонентом, для использования которого пользователь будет покупать абонемент, является компонент BookContent, отвечающий за генерацию (рендеринг) страниц книг. С системой Google Firebase главная проблема

2 Виртуального приватного сервера.

3 Оставалось в свободном распоряжении около 59 Гб.

4 Организация, будучи некоммерческой, на март 2023 г. финансируется исключительно на членские взносы и пожертвования.

состояла в том, что она: 1) либо возвращала контент только по прошествии 20 секунд – 2 минут времени; 2) либо не возвращала его совсем. Иначе говоря, в случае публикации данного веб-приложения, мы даже не могли бы гарантировать, что книга, которую решил почитать пользователь, будет загружена. Подобная перспектива была крайне непривлекательной, это было равносильно интернет-магазину, у которого онлайн-продажи являются базисом бизнеса, но при этом технически не работает возможность положить товар в корзину. Механизм, вследствие которого получался такой исход, представлен на рис. 3. Если кратко, данные следствия имели место по той причине, что каждая книга переносится из формата данных DOCX/PDF в JSON не как один модуль, а как совокупность таковых, т. е. каждая книга для удобства хранения делится на главы для последующего упрощения сопровождения. Следовательно, если книга содержит, например, 21 модуль (реальный пример одной из книг), то каждый такой модуль запрашивается у сервера *асинхронно*. Получается, что клиентская часть инициирует 21 запрос на сервер[5], вероятнее всего, вставая в очередь с другими разработчиками, подключившими данную СУБД к своим проектам. Как следствие ожидание по 21 запросу от сервера вполне может достигать 2 минут, а то и вовсе не завершаться ответом всех модулей книги. В связи с этим мы решили изменить способ получения данных с асинхронного на синхронный и сделать так, чтобы клиент в очереди был всего один[6]. Перевод решения с асинхронного на синхронное и стал решением проблемы ожидания[7]. К данному выводу мы пришли, изучая в [7, с. 160]

5 Поиск в API Google Firebase способа, как одним запросом выгружать сразу все содержимое файлов директории, результата не дал.

6 На самом деле, клиентов как минимум 4, поскольку запросы к серверу используются минимум в 4 сервисах. Добавим к этому, что клиентами могут выступать разные пользователи, что также увеличит объем очереди. Но второй фактор (очереди) пока не столь проблематичным, поскольку посещаемость основного сайта организации обычно не превышает 100 визитов в день, т. е. на данном этапе высокая нагрузка (*англ.* high load) отсутствует.

7 Наблюдательный читатель мог бы заметить, что нами не был описан способ т. н. «ленивой загрузки», при котором главы книг могли бы загружаться при пролистывании экрана монитора вниз. На самом деле, его мы тоже пробовали внедрять, но он оказал несущественное влияние. Если автору не изменяет память, при «ленивой загрузке» или сохра-

один из возможных механизмов распределения трафика. По ряду косвенных признаков было похоже, что Google Firebase, вероятнее всего, придерживается его, а именно, реализует очередь FIFO (*англ*. first in – first out).

7. Отсутствие в команде экспертизы, связанной с разработкой серверных частей веб-приложения. В группе разработчиков имелось всего 2 специалиста, оба из которых имеют больше практики с разработкой клиентской части веб-приложений, чем с работой над серверной частью. В связи с этим, делегирование данной задачи было возможным только в случае найма подрядчика. Учитывая некоммерческий статус организации, данного специалиста было бы возможно привлечь лишь на разовой основе[8], что все равно бы означало впоследствии необходимость освоения концепций программирования серверной части веб-приложений штатными разработчиками.

8. Отсутствие достаточного объема времени на освоение новой технологии. Помимо освоения MongoDB как альтернативы Google Firebase, автор также рассматривал вариант подключения PostgreSQL (SQL СУБД). Однако данный вариант, по его оценкам, потребовал бы минимум около 3 месяцев времени на освоение новой технологии, тогда как исправлять проблему с периодическими сбоями в работе 4 сервисов нужно было в срочном порядке. Срок в виде минимум 3 месяцев предполагал:

а. **изучение литературы по основам работы СУБД.** Поскольку автор по специализации является разработчиком клиентской части веб-приложений, что редко требует в профессиональной деятельности взаимодействия с СУБД напрямую (в основном подобное взаимодействие осуществляется через API, предоставляемый разработчиками серверной части), а

нялась проблема 20-секундного ожидания, или начинали блокироваться некоторые запросы вследствие квот Google Firebase. Иначе говоря, подобное решение оставалось бы лишь частичным.

8 При средней заработной плате серверного разработчика по Москве на март 2023 г. ежегодных денежных ресурсов, поступивших в бюджет организации с марта 2022 по март 2023, хватило бы лишь на 5 месяцев содержания 1 разработчика серверной части. Учитывая, что разработка ПО не является ее предметной областью (область – издательская деятельность), данную идею было бы крайне сложно инициировать и отстаивать на уровне Совета организации.

также предпочитает начинать освоение любой технологии с теоретической основы, данный шаг для него виделся необходимым. В качестве источника теоретической базы планировалось изучение [3];

b. **затем планировалось освоение дополнительной книги** уже непосредственно по синтаксису новой СУБД, приведенной в [5];

c. **после этого потребовалась бы небольшая практическая работа** по применению данной технологии в целях закрепления теории на практике.

Предлагаемая к рассмотрению новая технология будет базироваться на наборе классических CRUD-операций (*аббр. от англ.* Create, Read, Update, Delete), в целом характерных для СУБД.

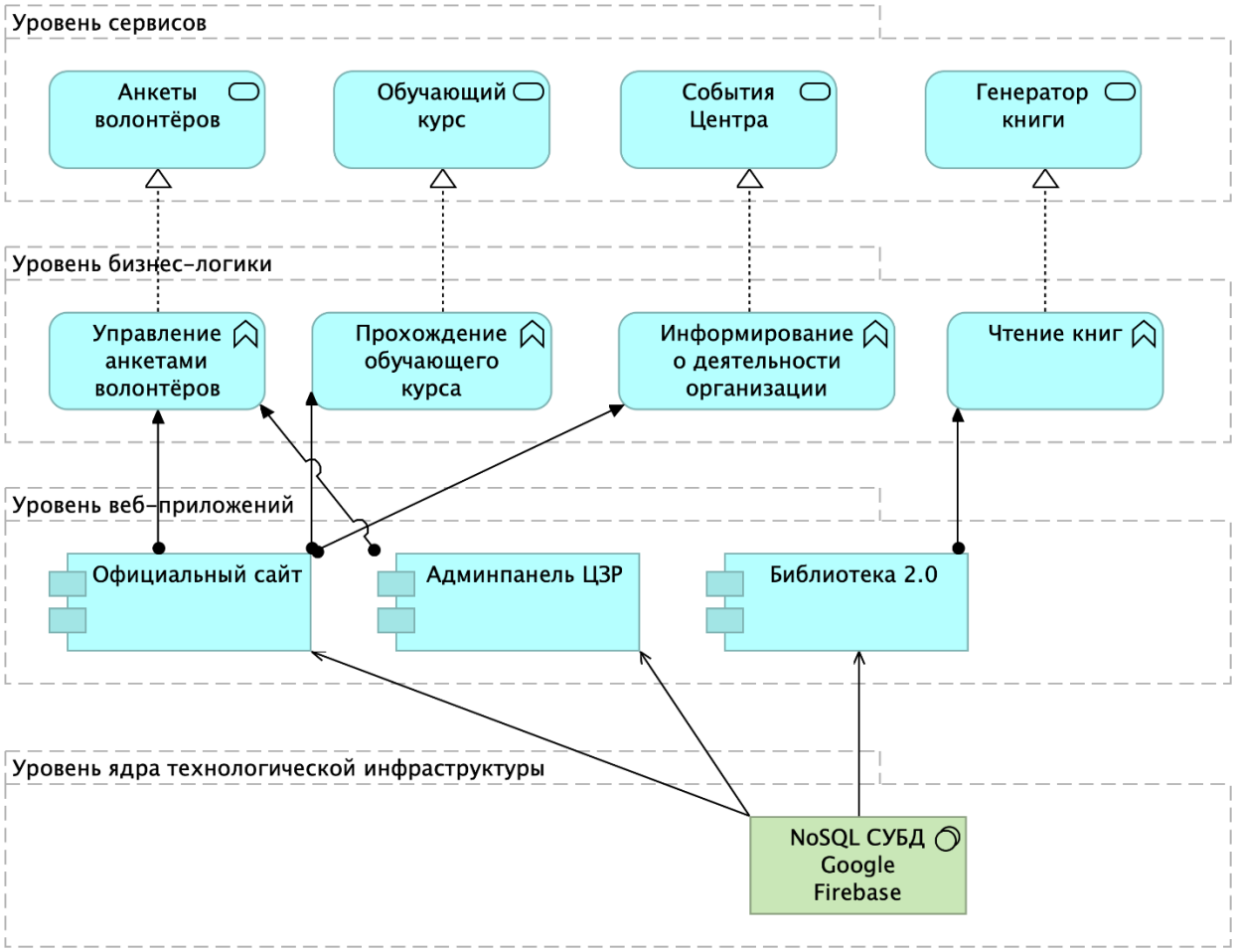


Рис. 1. Высокоуровневая архитектура веб-приложений и сервисов, роль СУБД в ней

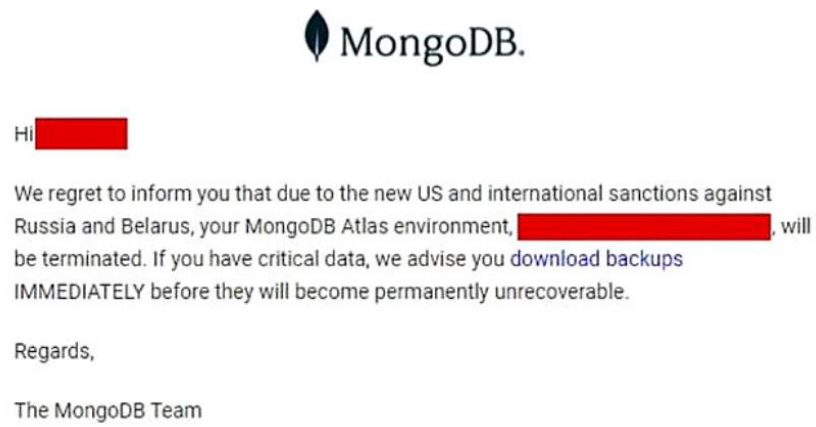


Рис. 2. Уведомление, отправленное администрацией MongoDB разработчикам из России и Беларуси

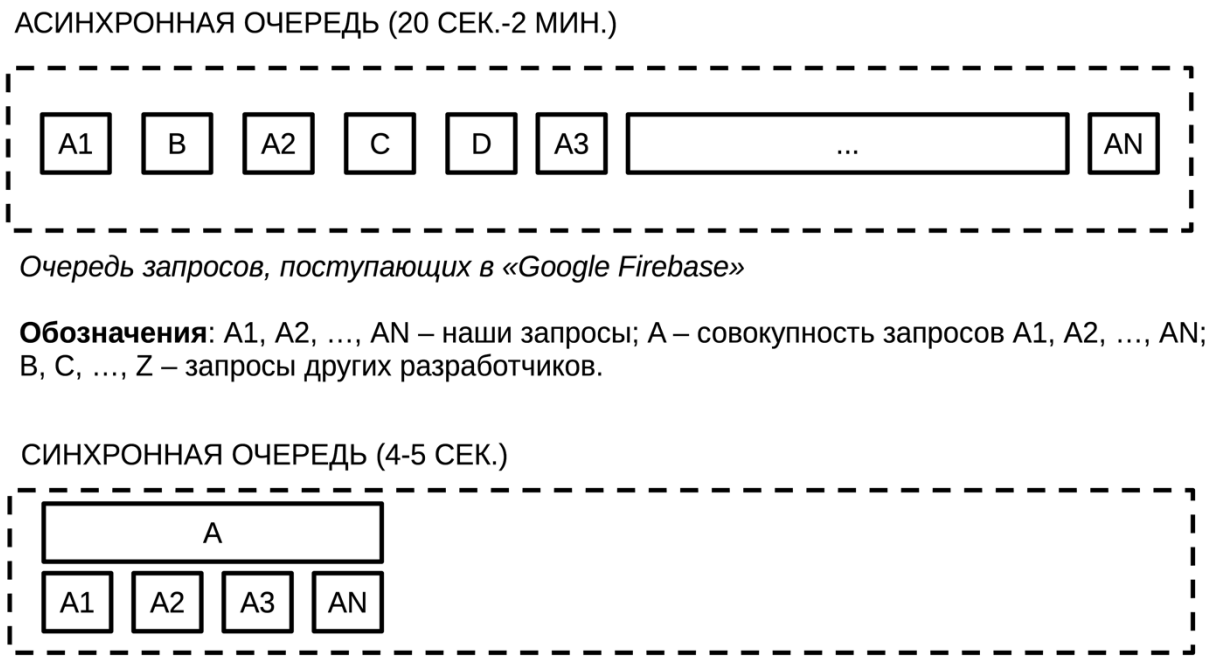


Рис. 3. Отличие асинхронной очереди от синхронной

СТРУКТУРА КОДОВОЙ БАЗЫ

Как можно видеть из структуры репозитория (рис. 4), файловая структура делится на две основных части:

1. **Директорию dbms**, которая содержит бизнес-логику работы СУБД. В модели MVC ее обычно представляют слоем C (*англ.* controller). В ней содержатся методы для манипуляции данными, хранящимися в другой директории.

2. **Директорию db**, хранящую непосредственно базы данных. Именно в данной директории впоследствии отображаются все актуальные данные.

|----constants	Константные значения
|--------paths.js	Переменные для ускорения доступа к БД
|----db	Корневая директория баз данных
|--------docs	Аналог Firestore в Google Firebase, базируется на концепции коллекции документов [2]
|------------admin	
|------------library	
|----------------readers.json	
|------------www	
|----------------courseSubscribers.json	
|----------------volunteers.json	
|--------files	
|------------admin	
|------------library	
|----------------books	
|--------------------creatingMan	
|------------------------creatingMan01.json	
|------------------------creatingMan02.json	
|------------------------…	
|------------www	
|----dbms	
|----node_modules	Содержит основные зависимости проекта
|----.gitignore	
|----index.js	
|----package.json	
|----README.md	
|----sandbox.js	
|----yarn-lock.js	

Рисунок 4. Структура директорий и модулей в репозитории СУБД

Обратите внимание, что после перезаписей внутри баз данных сервер не подтягивает их сразу, поскольку сервер был реализован не на высокоуровневом языке программирования JavaScript (JS), а посредством использования надстройки над JS, т. е. с использование TypeScript (TS). Последнее означает, что данные могут быть подтянуты только после повторной транспиляции из TS в JS.

Чтобы автоматизировать процесс транспиляции данных, нами используется инструмент pm2 (см. подр. https://pm2.keymetrics.io/), позволяющий создавать «демона» (робота), осуществляющего автоматическую процедуру транспиляции раз в час. В принципе, можно запускать его и чаще, например, раз в минуту. Однако следствием слишком частого запуска будут являться более частая недоступность сервера и как следствие работа со сбоями 4 сервисов веб-приложений организации, поскольку каждая такая процедура занимает около 10–15 секунд.

О МЕТОДАХ КЛАССА, СОДЕРЖАЩЕГО ОСНОВНУЮ ЛОГИКУ

Как упоминалось ранее, логика работы СУБД, отвечающая за манипуляцию данными (см. листинг 1), хранится в директории dbms и представляет собой два класса – Docs и Files – с CRUD-методами. Именуя данные классы, мы старались сделать их привычными для возможных будущих разработчиков, уже имеющих опыт работы с Google Firebase. Тогда как понятие «документ» активно используется в данной СУБД[9], в его дочернем продукте Firestore, а понятие «файл» – в Storage.

Отличия между двумя данными понятиями, в рамках новой СУБД, достаточно условны, в частности, их сложно привязать к формату данных[10]. Например, данные о пользователях (логин, email и др.) хранятся в формате JSON, а манипулирование ими происходит посредством методов класса Docs.

9 Кстати, не только в ней, но и в других нереляционных СУБД тоже, например, в MongoDB. Понятие «документ», в целом, является основополагающим при работе с NoSQL СУБД.

10 Основным форматом обмена данных в GoldenRaceDB является формат JSON, широко распространенный в вебе. Причиной выбора этого формата как основного стало то, что все три системы, разрабатываемые в рамках хобби-проекта, предполагают размещение систем (или как минимум одной из ее версий) в интернет-браузерах.

Тогда как главы книг тоже хранятся в JSON, но манипулирование ими происходит уже посредством методов класса Files, поскольку данные файлы предполагаются довольно объемными[11]. Иначе говоря, основными критериями разделения файлов являются следующие два фактора: 1) формат файла; 2) размер файла.

```
const fs = require("fs");
const { __reponame } = require("../constants/paths");

class Docs {
  constructor() {}

  readOne(subdomain, collection, id) {
    return require(`../db/docs/${subdomain}/${collection}`).find(
      (item) => item.id === id
    );
  }

  readAll(subdomain, collection) {
    return require(`../db/docs/${subdomain}/${collection}`);
  }

  create(subdomain, collection, data) {
    fs.writeFileSync(
      __reponame + `/db/docs/${subdomain}/${collection}.json`,
      JSON.stringify([...this.readAll(subdomain, collection), data]),
      "utf8",
      (error, data) => {
        if (error) {
          return error;
        }
        return data;
      }
    );
  }
}

module.exports = Docs;
```

Листинг 1. Содержимое одного из основных модулей СУБД, отвечающего за манипуляцию данными

МЕХАНИЗМ РЕЗЕРВНОГО КОПИРОВАНИЯ ДАННЫХ

В процессе разработки новой СУБД мы столкнулись с тем, что данные в процессе ее работы будут храниться на VPS, а не в системах, к которым можно

11 Одна книга со всеми главами, объединенными в JSON, в Библиотеке 2.0 «весит» примерно 2–4 Мб, т. е. близка по весу к размеру изображения, снятого на зеркальный фотоаппарат.

было бы обеспечить доступ при наличии интернета вроде Git. Последнее создавало риск, что в ряде случаев мы могли просто в одночасье потерять ценные данные, аккумулируемые на протяжении двух лет. В ряде случаев такие данные могли обходиться в весомую сумму денег, например, выступая результатом всевозможных маркетинговых кампаний, на которые выделялся рекламный бюджет.

Развилки, при которых была возможна утеря данных:

1. **Прерывание работы сервера в случае неудачного продления оплаты за содержание VPS**. Автор был наблюдателем прецедента, когда за неуплату VPS все веб-приложения просто выходили из строя (при отсутствии диверсификации веб-приложений по отдельным серверам из-за ограничений в бюджете). При слишком длительном поиске причины их вывода из строя, а такое может случиться, если разработчиков по какой-то причине в команде нет, есть риск, что владельцы VPS просто удалят все наработки организации и продадут очищенный VPS другому заказчику;

2. **Случайный ввод ошибочных команд**. Примером такой команды является rm -r /home, введенная сразу после авторизации на VPS, которая начисто удаляет все файлы и директории внутри директории home, где на данный момент располагаются репозитории веб-приложений организации. Хотя такой случай маловероятен, для полноты картины он был включен тоже;

3. **Физическая поломка сервера (при отсутствии резервного копирования, осуществляемого провайдером)**. В данном случае имеется риск появления форс-мажорных обстоятельств (пожара, наводнения и др.), которые могут оказать физическое отрицательное воздействие на работу сервера и вывести его из строя.

В связи с названными факторами нами было принято решение позаботиться также о резервном копировании данных с новой СУБД. Для этого была выбрана система GitHub, обычно используемая для управления версиями исходного кода, поскольку содержимое директории db – набор исходных кодов.

На листинге 2 показана реализация интеграции GoldenRaceDB с GitHub. Главной в нем является строка:

```
exec("git add . && git commit -m \"Create copy of databases\" && git push", …),
```

она отвечает за ввод в Терминале команды изнутри модуля на Node.js для создания резервной копии баз данных в GitHub.

```
const http = require('node:http');
const exec = require('child_process').exec;
const { v4: uuidv4 } = require("uuid");
const Docs = require("./dbms/Docs");

const hostname = '127.0.0.1';
const port = 3000;

const server = http.createServer((req, res) => {
  res.statusCode = 200;
  res.setHeader('Content-Type', 'text/plain');
  res.end('Hello, World!\n');
});

setInterval(() => {
  const docs = new Docs();
  docs.create("test-db", {
    id: uuidv4(),
    firstName: "Test",
    email: "test@mail.ru",
  });
  exec(
    "git add . && git commit -m \"Create copy of databases\" && git push",
    function(err, stdout, stderr) {
    if (err) {
      console.log(err);
    }
    console.log(stdout);
  });
  return console.log("The setInterval is finished");
}, 5000);

server.listen(port, hostname, () => {
  console.log(`Server running at http://${hostname}:${port}/`);
});
```

Листинг 2. Логика, отвечающая за резервное копирование

О РАБОТЕ С БОЛЬШИМИ ФАЙЛАМИ, ГЕНЕРАЦИИ ПУБЛИЧНЫХ URL И РЕГИСТРАЦИИ ПОЛЬЗОВАТЕЛЕЙ

В процессе разработки кастомной СУБД мы столкнулись с рядом сложностей.

Первое – работа с большими файлами. Было обнаружено, что при резервном копировании данных возникают сложности на стадии резервного копирования. Так, например, GitHub устанавливает требование, согласно которому файлы размером свыше 100 Мб нельзя сохранять в репозитории. Иначе говоря, имеется запрет на использование GitHub в качестве хостинга. Данная проблема была решена посредством следующей команды, вводимой в окне терминала:

scp -C <PATH_TO_OLD_PLACE> <PATH_TO_NEW_PLACE>,

она позволяет скопировать файл с локального компьютера на VPS (для VPS дополнительно необходимо вводить логин и IP-адрес).

Второе – генерация публичных URL. Для решения определенных задач нам требовалось формировать публичные ссылки на размещенные в GoldenRaceDB файлы. Решением в данном случае стало использование метода res.sendFile() (https://expressjs.com/ru/api.html#res.sendFile), предоставляемое серверным фреймворком Express.js [12, с. 117]. Пример реализации логики с использованием данного метода см. в Листинге 3.

```
module.exports.getFilePublicLink = (req, res, next) => {
  try {
    const { path } = req.query;

    if (!req.query.path) {
      res.status(400).json({
        error: 'Please set the path query for the end point',
      });
    }

    const pathChunks = path.split('/');
    const fileFullName = pathChunks[pathChunks.length - 1];
    const pathWithoutFileName = path.split('/' + fileFullName)[0];
    const options = {
      root: '../../gr-dbms-videos' + pathWithoutFileName,
      dotfiles: 'deny',
      headers: {
        'x-timestamp': Date.now(),
        'x-sent': true,
      },
    };

    res.sendFile(fileFullName, options, function (error) {
      if (error) {
        next(error);
      } else {
        console.log('Sent:', fileFullName);
      }
    });
  } catch (error) {
    res.status(500).send(error);
  }
};
```

Листинг 3. Реализация логики для генерации публичных URL с использованием Node.js

Третье – регистрация пользователей. В процессе разработки новой СУБД мы столкнулись с тем, что пользователей требовалось каким-то образом регистрировать в системе. Причем речь здесь не просто о добавлении регистрационных данных в БД, а, скорее, о том, как регистрировать email и пароль, гарантируя, что пароль не станет известен кому-либо из тех, у кого есть/будет доступ к СУБД (разработчикам, хакерам [10] и т. д.).

Здесь можно было внедрить реализацию системы шифрования данных, разработанную нами и формализованную в [6, с. 25]. Однако, поскольку время на миграцию данных в новую СУБД было ограничено, а функционал Google Firebase для регистрации новых пользователей ранее уже был апробирован и не имел сбоев в работе, было принято решение пока ограничиться реализацией регистраций учетных записей, имеющейся в Google Firebase.

ПРЕИМУЩЕСТВА И НЕДОСТАТКИ НОВОЙ СУБД

Преимущества:

1. **Простота реализации, возможность самостоятельного внедрения.** Для понимания степени простоты вышеприведенного листинга отметим лишь, что главный модуль технологии Redux [13], широко известной в сообществе разработчиков клиентской части веб-приложений как инструмент для управления состояниями (англ. state), содержал на начальной стадии становления СУБД примерно такой же объем кода.
2. **Были решены основные задачи, стоявшие перед отделом сроком в 1 месяц.** В это время входит не только стадия разработки новой СУБД, но и перевод нуждающихся в ней сервисов на новую СУБД.
3. **Знакомый синтаксис (JS).** Хотя Node.js [1, с. 36] является относительно молодой технологией и сильно уступает ЯП (языку программирования) Java по численности сообщества, тем не менее, синтаксис данной технологии уже был знаком автору, что уменьшало порог входа для ее подключения.
4. **Увеличена скорость отклика основной функции веб-приложения Библиотека 2.0 в 4 раза.**

5. **Небольшой размер кодовой базы.** Если приложения на React.js «славятся» необходимостью переноса за собой директории с обширным списком зависимостей (node_modules) размером около 250 Мб, что, бесспорно, являлось бы недостатком в средах с высокими требованиями к ограничениям вычислительных ресурсов [8], то Node.js, хотя также требует переноса данной директории, тем не менее, требует для переноса всего около 12 Мб.

6. **Отсутствие необходимости в регулярных расходах за пользование зарубежными технологией и сервером, что критично для общественных организаций.**

Недостатки:

1. **Отсутствие механизма авторизации на уровне СУБД.** Если при использовании других СУБД, например, MongoDB, обычно перед началом использования требуется ввести специальную команду авторизации в кодовой базе, то в предложенном на рассмотрение решении такой авторизации нет, поскольку решение не проектировалось нами как переиспользуемое другими разработчикам ввиду кажущейся простоты его самостоятельной реализации.

2. **Более низкая производительность Node.js как высокоуровневой технологии [11].** Поскольку Node.js изначально строилось как высокоуровневое серверное окружение, то оно в некоторой степени проигрывает СУБД, построенным на низкоуровневых ЯП.

3. **Степень задокументированности.** Технология, будучи новой, пока слабо задокументирована, что может в будущем усложнить сопровождение серверных модулей, завязанных на нее. Частично данную проблему можно решить за счет подключения к проекту технологии TypeScript [9], позволяющей привнести в динамически типизированный JavaScript статическую типизацию. При добавлении данной технологии IDE (см., например, https://code.visualstudio.com/) тогда будет подсвечивать API методов.

4. **Отсутствие сообщества разработчиков.** Поскольку технология нова, поиск готовых решений по ней в интернете (например, на портале Stack Overflow) будет затруднительным из-за ограниченного круга работающих с ней разработчиков.

5. **Зависимость от другого иностранного ресурса (GitHub; см. https://github.com/).** Как упоминалось выше, механизм резервного копирования данных обеспечивается с завязкой на стороннюю зарубежную систему GitHub, что создает некоторые риски утери резервных копий БД. Однако данное опасение видится избыточным ввиду того, что резервное копирование данных осуществляется в 3 местах: 1) локальная машина автора; 2) VPS; 3) GitHub. Иначе говоря, в случае вывода из строя одного из трех мест хранения данных высоковероятно, что останутся доступными другие два.

6. **Отсутствие дополнительных условий-проверок при манипуляции данными.** Новая СУБД не осуществляет никаких проверок данных, поступающих ей на вход (например, проверку на дублирование данных), что перекладывает написание всех необходимых проверок на разработчика.

7. **Ограниченность небольшим масштабом проекта.** Предложенное решение хорошо подходит для небольших проектов, однако в крупных корпоративных системах, где чаще предпочитают использовать СУБД на базе SQL, оно будет менее предпочтительно.

8. **Задержка при резервном копировании данных может составлять до 24 ч.**

9. **Задержка при обновлении подтягиваемых сервером данных из СУБД может составлять 1 час.**

ЗАКЛЮЧЕНИЕ

Признаем, что описанное выше решение частично наследуется от технологии Google Firebase. Хотя нам неизвестно внутреннее содержимое данной технологии на уровне кодовой базы, полагаем, что рассмотренное решение вполне могло бы стать хорошей альтернативой на российском рынке для небольших организаций. **Главное – оно позволило бы решить две проблемы, стоявшие перед нами: 1) предоставление гарантии получения данных, часть которых является критически важными для организации; 2) а также значительно ускорить работу сервисов (в 1,5–4,5 раза).**

После того, как мы уже внедрили самописную СУБД, нами была обнаружена возможность использования API Яндекс.Диска (см. подробнее

https://yandex.ru/dev/disk/), позволяющая решить задачу, как говорят в среде разработчиков, «более элегантным способом», за счёт хранения некоторых данных на сервере(-ах) Яндекса. Маловероятно, что Яндекс.Диск будет подвержен геополитическому контексту в отношении пользователей из России и Беларуси. Квот на частоту использования API нами также замечено не было (API не апробировалось).

Тем не менее, мы пока решили повременить с данным решением, поскольку есть риск, что впоследствии в одностороннем порядке Яндексом может начать взыматься плата, например, за объем обращений по API, как это уже имело место с Google Firebase. А учитывая некоммерческий статус организации и нестабильность работы волонтеров[12], **практика работы после февраля 2022 г. показывает, что хотя самописные решения и более сложны на стадии сопровождения, они являются более надежными и менее зависимыми от действий подрядчиков (создателей др. СУБД и т. п.).**

СПИСОК ЛИТЕРАТУРЫ

1. *Аквино К., Ганди Т.* Front-end. Клиентская разработка для профессионалов. Node.js, ES6, REST. СПб.: Питер, 2017. 512 с.: ил. (Серия «Для профессионалов»).

2. *Бэнкер К.* MongoDB в действии. / Пер. с англ. А.А. Слинкина. М.: ДМК Пресс, 2012. 394 с.: ил.

3. *Дейт К. Дж.* Введение в системы баз данных, 8-е издание: Пер. с англ. М.: Издательский дом «Вильяме», 2005. 1328 с.: ил. Парал. тит. англ.

4. *Джуба С., Волков А.* Изучаем PostgreSQL 10 / пер. с анг. А.А. Слинкина. М.: ДМК Пресс, 2019. 400 с.: ил.

5. *Лоре А.* Проектирование веб-API / Пер. с англ. Д.А. Беликова. М.: ДМК Пресс, 2020. 440 с.

12 Порядка 40% волонтеров, приходящих в отдел ИТ организации, теряет интерес к работам над проектами в течение 1–4 недель. Полагаем, основной причиной потери мотивации является невозможность использования материальных способов мотивации, оклада и премий, при работе с ними.

6. *Мосолов Р. В.* Программа «История гениального открытия»: автоматизация рутинной деятельности ученых: дис. маг. комп. наук 09.04.04 / Казань, 2021. 47 с.
URL: https://kpfu.ru/student_diplom/10.160.178.20_29GG9GW6GT5OBPD9N0DGSIVNS0WANG9_PEMUXEOD82DR2X2Y26_F_Mosolov.pdf.

7. *Олифер В., Олифер Н.* Компьютерные сети. Принципы, технологии, протоколы: Юбилейное издание. СПб.: Питер, 2021. 1008 с.: ил. (Серия «Учебник для вузов»).

8. *Портянкин И.* Программируем Cloud Native: микросервисы, Docker и Kubernetes. 2022. 177 с.: ил.

9. *Розенталс Н.* Изучаем TypeScript 3. М.: ДМК Пресс, 2019. 624 с.

10. *Скабцов Н. В.* Аудит безопасности информационных систем. СПб.: Питер, 2018. 272 с.: ил. (Серия «Библиотека программиста»).

11. *Таненбаум Э., Остин Т.* Архитектура компьютера. 6-е изд. СПб.: Питер, 2022. 816 с.: ил. (Серия «Классика computer science»).

12. *Холмс С.* Стек MEAN. Mongo, Express, Angular, Node. СПб.: Питер, 2017. 496 с.: ил. (Серия «Библиотека программиста»).

13. *Garreau M., Faurot W.* 2018. Redux in Action. Manning Publications. E-book. Accessed: 04 November 2022. Hagos, T. 2021.

ABOUT “GOLDEN RACE DB” DEVELOPMENT (NOSQL DBMS) AS AN ALTERNATIVE OF “GOOGLE FIRABASE”

R. V. Mosolov[0000-0002-4399-4397]

The chain of 70 beauty branches “GRUSHKA” (Moscow)

R.V.Mosolov@ya.ru

Abstract

In this article, we have described expirience of the new NoSQL Database Management System (DBMS) development named as “GoldenRaceDB”. Also, we have described prerequisites that needed to create it in the context of the Russian

software's import problem. The new technology is realised as based on server environment Node.js. To understand this article, a reader should have expirience of server-side development by using one of high-level progrmming language (as minimum) or expirience of development custom DBMS. This technical solution is not open source, we have created it to solve our local tasks at our organization exactly, the technology birth place. But a reader can understand a general vector of creating custom resized DBMS by using other high-level programming language.

Keywords*: database management system, DBMS, NoSQL DBMS, DBMS development, DBMS creation, DBMS operator, DBMS example, DBMS queries, Russian sowtware import problem, DBMS analogs.*

REFERENCES

1. *Akvino K., Gandi T.* Front-end. Klientskaja razrabotka dlja professionalov. Node.js, ES6, REST. SPb.: Piter, 2017. 512 s.: il. (Serija "Dlja professionalov").

2. *Bjenker K.* MongoDB v dejstvii / Per. s angl. A.A. Slinkina. M.: DMK Press, 2012. 394 s.: il.

3. *Dejt K. Dzh.* Vvedenie v sistemy baz dannyh, 8-e izdanie.: Per. s angl. M.: Izdatel'skij dom "Vil'jame", 2005. 1328 s.: il. Paral. tit. angl.

4. *Dzhuba S., Volkov A.* Izuchaem PostgreSQL 10 / per. s ang. A.A. Slinkina. M.: DMK Press, 2019. 400 s.: il.

5. *Lore A.* Proektirovanie veb-API / Per. s angl. D.A. Belikova. M.: DMK Press, 2020. 440 s.

6. *Mosolov R. V.* Programma "Istorija genial'nogo otkrytija": avtomatizacija rutinnoj dejatel'nosti uchenyh : dis. mag. komp. nauk 09.04.04 / Kazan', 2021. 47 s.
URL: https://kpfu.ru/student_diplom/10.160.178.20_29GG9GW6GT5OBPD9N0DGSIVNS0WANG9_PEMUXEOD82DR2X2Y26_F_Mosolov.pdf.

7. *Olifer V., Olifer N.* Komp'juternye seti. Principy, tehnologii, protokoly: Jubilejnoe izdanie. SPb.: Piter, 2021. 1008 s.: il. (Serija "Uchebnik dlja vuzov").

8. *Portjankin I.* Programmiruem Cloud Native: mikroservisy, Docker i Kubernetes. 2022. 177 s.: il.

9. *Rozentals N.* Izuchaem TypeScript 3. M.: DMK Press, 2019. 624 s.

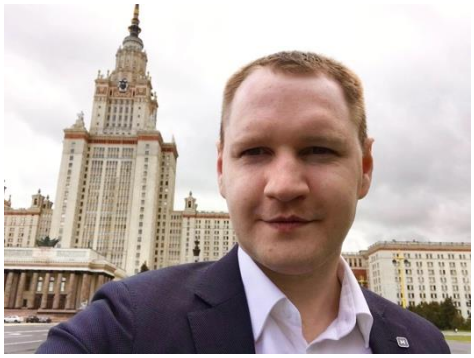
10. *Skabcov N.V.* Audit bezopasnosti informacionnyh sistem. SPb.: Piter, 2018. 272 s.: il. (Serija “Biblioteka programmista”).

11. *Tanenbaum Je., Ostin T.* Arhitektura komp'jutera. 6-e izd. SPb.: Piter, 2022. 816 p.: il. (Serija “Klassika computer science”).

12. *Holms S.* Stek MEAN. Mongo, Express, Angular, Node. SPb.: Piter, 2017 496 s.: il. (Serija “Biblioteka programmista”).

13. *Garreau M., Faurot W.* 2018. Redux in Action. Manning Publications. E-book. Accessed: 04 November 2022. Hagos, T. 2021.

СВЕДЕНИЯ ОБ АВТОРЕ



МОСОЛО́В Роман Валерьевич – бакалавр социологии, магистр программной инженерии, разработчик CRM-системы в сети салонов красоты «GRUSHKA» (70 филиалов по России). Альма-матер: Казанский (Приволжский) федеральный университет. Специализация в программировании: разработка клиентской части веб-приложений с использованием технологического стека TS, SASS, React.js, Redux Toolkit, Ant Design, Webpack, Storybook.

Roman Valerievich MOSOLOV – Bachelor in Sociology, Master in Computer Science, developer of CRM system at chain of beauty salons “GRUSHKA” (70 branches by Russia). Kazan Federal University is the alma mater. The specialization in software engineering is using the following technological stack: TS, SASS, React.js, Redux Toolkit, Ant Design, Webpack, Storybook.

email: R.V.Mosolov@ya.ru

ORCID: 0000-0002-4399-4397

Материал поступил в редакцию 8 июля 2023 года