Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
<<Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)>>
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ <<**Информатика и системы управления**>>

КАФЕДРА <<**Программное обеспечение ЭВМ и информационные технологии**>>

ОТЧЁТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Студент Романов Семен Константинович

Группа ИУ7-65Б

Тип практики Производственная

Название предприятия ООО <<СитиСофт>>

Студент	_____	**Романов С. К.**
	подпись, дата	*фамилия, и.о.*
Руководитель практики (от университета)	_____	**Толпинская Н. Б.**
	подпись, дата	*фамилия, и.о.*
Руководитель практики (от предприятия)	_____	**Батин Р. Е.**
	подпись, дата	*фамилия, и.о.*

Оценка _____

2023 г.

Содержание

ВВЕДЕНИЕ

BOB --– это распределённая система хранения данных типа ключ-значение[**?**]. Боб проектировался для быстрой и надёжной работы с бинарными данными средних размеров (например, с изображениями). Боб постоянно развивается и обладает множеством запланированных задач разной сложности.

Цель производственной практики --- овладеть навыками разработки систем хранения данных, Web-приложений[**?**], open source-проектов[**?**]

Задачи проекта:

Разработать Web-приложение[**?**], включающее в себя:

1) страницу авторизации;
2) домашнюю страницу с основными параметрами кластера;
3) страницу с нодами;
4) страницу с подробной информацией о ноде;
5) страницу с раскладкой виртуальных дисков.

Индивидуальные задачи:

1) получить навыки командной работы;
2) спроектировать и разработать клиент для взаимодействия с API[**?**] Боба;
3) спроектировать собственный API[**?**];
4) разработать серверную часть веб-приложения, интегрируя в него ранее разработанный клиент и API[**?**];
5) провести тестирование итогового Web-приложения.

1 Характеристика предприятия

???

2 Характеристика проекта для производственной практики

Требуется разработать Web-приложение, включающее в себя следующее:

– Страница авторизации. На странице необходимо задавать адрес входной ноды и опционально логин и пароль.
– Домашняя страница с основными параметрами кластера: число нод в кластере, число рабочих нод, число нод с проблемами, число нерабочих нод, информация по физическим дискам аналогична. Также необходимо включить общее число обрабатываемых запросов на кластере в секунду с разбивкой далее по put, get, exist, delete, общее место на дисках кластера, занятое данными место на дисках всего кластера.
– Страница с нодами. Страница должна отображать все ноды с именем и адресом, по каждой ноде её состояние -- работает, есть проблемы, не работает; операции в секунду на ноде; занятое дисковое пространство. Перечень проблем на ноде: не доступен один из дисков, есть alien'ы, есть поврежденные БЛОБы, осталось свободного места менее 10%, виртуальная память, выделенная ноде, превышает доступную физическую память, нагрузка на CPU выше 90%. С данной страницы должна быть реализована возможность перехода на страницу с подробной информации о ноде.
– Страница с подробной информацией о ноде. Страница должна демонстрировать основные метрики ноды, параметры системы, на которой запущена нода -- загрузка CPU, RAM, дескрипторы; список дисков на ноде с их состоянием, список виртуальных дисков внутри каждого диска. Также дать возможность просмотра целиком метрик ноды в виде таблицы.
– Страница с раскладкой виртуальных дисков. Страница должна отображать список виртуальных дисков с их доступностью в кластере.

3 Характеристика индивидуального задания для производственной практики

В ходе практики предстоит выполнить следующие индивидуальные задачи:

1) Студент должен овладеть навыками командной работы, что позволит ему эффективно сотрудничать с другими разработчиками в рамках проекта.
2) Спроектировать и разработать клиент для взаимодействия с API[**?**] Боба. Это важная задача, так как клиент будет обеспечивать сервер данными о распределенной системе хранения данных.
3) Спроектировать собственный API[**?**], который позволит серверной части Web-приложения[**?**] передавать данные внешнему интефейсу. Это API должно быть хорошо продумано, чтобы обеспечить удобный и эффективный доступ к данным Боба.
4) Разработать серверную часть Web[**?**]-приложения, включив в него ранее разработанный клиент и API. Это будет являться основной частью работы, так как сервер должен обеспечивать функциональность авторизации, отображение информации о кластере, нодах, детальной информации о нодах и раскладке виртуальных дисков.
5) Провести тестирование итогового Web-приложения[**?**], чтобы убедиться в его работоспособности, надежности и соответствии поставленным задачам.

4 Описание выполнения задания

4.1 Командная разработка

4.2 Анализ используемых технологий

4.2.1 Основное приложение

В качестве языка программирования основного приложения используется функциональный язык Clojure [5], являющийся подмножеством языка LISP [6] и поддерживающий библиотеки JVM [7], подходящий для обработки больших объёмов данных и построения высоконагруженных систем.

4.2.2 Сервис по созданию отчётов

Для передачи данных от приложения сервису была написана библиотека, определяющая транспортные типы данных. Данные передаются в виде сериализованных JSON-объектов[10]. Сервис при передаче критерия отчёта приложению также использует JSON-объект.

4.2.3 База данных приложения

В качестве системы управления базой данных[11] (СУБД) используется Datomic[12] -- распределённая база данных класса NoSQL[13], реализующая язык логических операций Datalog [14]. В качестве сервиса хранения используется база данных[15] PostgreSQL[16].

Особенностью рассматриваемой СУБД является связь каждой транзакции с текущим временем, база данных хранит историю всех операций, что позволяет получить её состояние в любой момент времени, такой подход позволяет реализовать режим просмотра истории изменения данных, являющийся аналогом журналирования.

Datomic предоставляет транзакции, соответствующие принципам ACID[17] и состоит из механизма изменения состояния базы данных -- транзактора, сервиса хранилища данных и пользовательского сервиса, который отсылает данные на транзактор. Оптимальная скорость работы достигается за счёт зарезервированных инструкций и консистентности данных, встроенного

в транзактор механизма кэширования запросов как на чтение, так и на запись, и отсутствия планировщика транзакций. Унификация данных структурой вида <<ключ-значение>> позволяет достичь согласованности, в качестве ключа может быть как зарезервированное слово, так и атрибут сущности или хранимая процедура.

Главной особенностью является возможность пользователя полностью влиять на системные настройки и отсутствие явных правил формирования запроса. Неумелое обращение с данной СУБД может привести к результатам на порядок хуже, чем в традиционных реализациях, например, отсутствие планировщика транзакций может значительно замедлить выполнение составного запроса.

4.3 Создание запроса

База данных представлена в виде фактов и отношений[18], каждая запись имеет следующую структуру: [сущность атрибут значение транзакция '_], запрос представлен хеш-картой[19] вида <<ключ-значение>>.

Структура запроса состоит из трёх ключей:

- `:find` -- содержит свободный набор переменных, которые будут использованы для итогового формирования выборки;
- `:in` -- содержит источники данных, которые будут использованы в запросе;
- `:where` -- содержит набор правил, которые записывают значения атрибутов в свободные переменные и используют источники данных для постановки логического вопроса.

Например, для того, чтобы в базе данных музыкантов и их музыкальных релизов найти все релизы по имени музыканта, можно использовать запрос, представленный в листинге 1.

Листинг 1: Пример запроса

```
1 (d/q {:find ?release-name
2       :in $ ?artist-name
```

```
 3       :where [?artist :artist/name ?artist-name]
 4              [?release :release/artists ?artist]
 5              [?release :release/name ?release-name]}
 6       db "John Lenon")
 7
 8 => #{["Power to the People"]
 9      ["Unfinished Music No. 2: Life With the Lions"]
10      ["Live Peace in Toronto 1969"]
11      ["Live Jam"]
12      ...}
```

Для формирования запроса используется хеш-карта, каждый ключ запроса будет формироваться на основе наличия или отсутствия параметров критерия. Обновление параметров запроса осуществляется с помощью оператора условного перехода `cond` вместе с потоковым макросом[20] `->`. Для обновления значения ключей используются стандартные операторы `update` и `conj`. Пример формирования запроса представлен в листинге 2. Результатом запроса является коллекция сущностей, отфильтрованная с помощью критерия.

Листинг 2: Создание запроса

```
1 (cond-> '{:find [?x]
2           :in [$]
3           :where []}
4   report-param-1
5   (-> (update :in conj '[?report-param-1...])
6       (update :where
7               conj
8               '[?rp-1 ?attr ?report-param-1]))
9   ...
```

4.4 Обработка и конвертация данных

Чтобы сконвертировать выборку в коллекцию транспортных объектов необходимо предварительно получить все необходимые значения полей. Каждая сущность представляет собой хеш-карту, достаточно извлечь необходимые значения атрибутов. Пример деструктуризации[21] сущности представлен в листинге 3.

Листинг 3: Деструктуризация сущности

```
(defn- rasterize
  [{:entity/keys [key-1 key-2 key-3]}]
  (assoc-some
    {:key-1 key-1
     :key-2 key-2
     :key-3 key-3}))

(->> query-result
     (map rasterize))
```

Для конвертации был написан макрос преобразования типов полей. Конвертация происходит согласно заранее определённому типу транспортного объекта. Для каждого типа отчёта, создаётся своя функция для конвертации. Данный подход позволяет расширять список поддерживаемых типов отчётов.

Листинг 4: Макрос конвертации в транспортный объект

```
(defmacro def-converter
  [report-type]
  (let [dto-class (symbol (csk/->PascalCaseString report-type))
        type-info (clojure.reflect/reflect (resolve dto-class))
        members (:members type-info)
        setters (filter #(string/starts-with? (:name %) "set")) members)
        fn-name (symbol (str "make-" report-type "-record"))]
    `(defn ~(with-meta fn-name {:report-type report-type})
       [~'item]
       (let [~'record (new ~dto-class)]
         ~@(map (fn [setter]
                  (let [setter-name (str (:name setter))
                        item-key (csk/->kebab-case-keyword
                                  (subs setter-name (count "set")))
                        setter-fn (symbol (str "." setter-name))
                        [field-type] (:parameter-types setter)
                        converter-fn (or (get field->converter item-key)
                                         (case field-type
                                           java.lang.Long 'long
                                           java.lang.Float 'float
                                           dto.type-1 'coverter-type-1
                                           nil))]
                    (if (= field-type 'java.util.List)
                      (let [converter-fn (get field->converter item-key)]
                        `(let [~'l (ArrayList.)]
                           (doseq [~'v (~item-key ~'item)]
                             (.add ~'l ~(if converter-fn
                                          `(~converter-fn ~'v)
                                          'v)))
                           (~setter-fn ~'record ~'l)))
                      `(~setter-fn ~'record
                                   ~(if converter-fn
                                      `(~converter-fn (~item-key ~'item))
                                      `(~item-key ~'item))))))
                setters)
         ~'record))))
```

4.5 Сериализация объектов

Для передачи данных в сервис была написана функция сериализации в JSON-объект. Исходный код функции представлен в листинге 5.

Листинг 5: Сериализация объектов

```
(fn [stream]
  (try
    (with-open [writer (io/writer stream)]
      (let [^ObjectMapper object-mapper (make-object-mapper)]
        (let [report-info (doto (ReportRequestInfo.) ...)
              json-string (.writeValueAsString object-mapper report-info)]
          (.write writer json-string)
          (.write writer "\r\n"))

        (doseq [item items]
          (let [json-string (.writeValueAsString object-mapper
                                                 (make-record item))]
            (.write writer json-string)
            (.write writer "\r\n")))

        (.write writer ".")
        (.write writer "\r\n")))
    (catch Throwable t
      (log/error t "Unable to transfer report data")
      (throw t))))
```

ЗАКЛЮЧЕНИЕ

В рамках производственной практики был реализован функционал получения выборки по заданному критерию, обработки, конвертации и сериализации данных.

Так же были выполнены следующие задачи:

1) утверждено техническое задание;
2) проведён анализ используемых предприятием технологий;
3) ознакомился с правилами создания запроса для функционально-логической базы данных;
4) изучил инструменты для обработки и конвертации данных;
5) изучил способы сериализации данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Что такое микросервисная архитектура: простое объяснение | MCS Mail.ru [Электронный ресурс]. Режим доступа: https://mcs.mail.ru/blog/prostym-jazykom-o-mikroservisnoj-arhitekture (дата обращения: 25 июля 2023 г.).

2. Serialization in Java - Java Serialization | Digital Ocean [Электронный ресурс]. Режим доступа: https://www.digitalocean.com/community/tutorials/serialization-in-java (дата обращения: 25 июля 2023 г.).

3. Arrival | Zero-emission Solutions [Электронный ресурс]. Режим доступа: https://arrival.com/world/en (дата обращения: 25 июля 2023 г.).

4. Pros and Cons of Data Transfer Objects | Microsoft Docs [Электронный ресурс]. Режим доступа: https://docs.microsoft.com/en-us/archive/msdn-magazine/2009/august/pros-and-cons-of-data-transfer-objects (дата обращения: 25 июля 2023 г.).

5. Clojure [Электронный ресурс]. Режим доступа: https://clojure.org/ (дата обращения: 25 июля 2023 г.).

6. Common Lisp [Электронный ресурс]. Режим доступа: https://lisp-lang.org/ (дата обращения: 25 июля 2023 г.).

7. What is the JVM? Introducing the Java Virtual Machine | InfoWorld [Электронный ресурс]. Режим доступа: https://www.infoworld.com/article/3272244/what-is-the-jvm-introducing-the-java-virtual-machine.html (дата обращения: 25 июля 2023 г.).

8. Kotlin Programming Language [Электронный ресурс]. Режим доступа: https://kotlinlang.org/ (дата обращения: 25 июля 2023 г.).

9. FasterXML/jackson [Электронный ресурс]. Режим доступа: https://github.com/FasterXML/jackson (дата обращения: 25 июля 2023 г.).

10. JSON | MDN [Электронный ресурс]. Режим доступа: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON (дата обращения: 25 июля 2023 г.).

11. Что такое СУБД - RU-CENTER [Электронный ресурс]. Режим доступа: https://www.nic.ru/help/chto-takoe-subd_8580.html (дата обращения: 25 июля 2023 г.).

12. Overview | Datomic [Электронный ресурс]. Режим доступа: https://docs.datomic.com/on-prem/overview/overview.html (дата обращения: 25 июля 2023 г.).

13. Что такое NoSQL? | Amazon AWS [Электронный ресурс]. Режим доступа: https://aws.amazon.com/ru/nosql/ (дата обращения: 25 июля 2023 г.).

14. Datalog: Deductive Database Programming [Электронный ресурс]. Режим доступа: https://docs.racket-lang.org/datalog/ (дата обращения: 25 июля 2023 г.).

15. Что такое база данных | Oracle Россия и СНГ [Электронный ресурс]. Режим доступа: https://www.oracle.com/ru/database/what-is-database/ (дата обращения: 25 июля 2023 г.).

16. PostgreSQL: Документация. [Электронный ресурс]. Режим доступа: https://postgrespro.ru/docs/postgresql/ (дата обращения: 25 июля 2023 г.).

17. Транзакции, ACID, CAP | GeekBrains [Электронный ресурс]. Режим доступа: https://gb.ru/posts/acid_cap_transactions (дата обращения: 25 июля 2023 г.).

18. Datomic Queries and Rules | Datomic [Электронный ресурс]. Режим доступа: https://docs.datomic.com/on-prem/query/query.html (дата обращения: 25 июля 2023 г.).

19. HashMap Under the Hood | Baeldung [Электронный ресурс]. Режим доступа: https://www.baeldung.com/java-hashmap-advanced (дата обращения: 25 июля 2023 г.).

20. Clojure - Threading Macros [Электронный ресурс]. Режим доступа: https://clojure.org/guides/threading_macros (дата обращения: 25 июля 2023 г.).

21. Clojure - destructuring [Электронный ресурс]. Режим доступа: https://clojure.org/guides/destructuring (дата обращения: 25 июля 2023 г.).