ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Биоинформатика образована на стыке биологических направлений и информатики, как реакция на всё увеличивающий объем экспериментальных данных в связи с прогрессом в технологиях их получения. Биоинформатика как наука, предоставляющая множество сложных алгоритмов анализа и обработки данных, играет большую роль при решении задач фундаментальной биологии и медицине.

Одним из наиболее используемых экспериментальных методов является анализ генетической экспрессии, соответственно, выпускается все больше и больше публикаций, посвященных этой задаче, тем самым увеличивая объем общедоступных данных. Так, например, существует база *Gene Expression Omnibus*, агрегирующая и модерирующая множество опубликованных данных экспрессии генов.

Однако, не каждая команда исследователей или лаборатория имеет в своем составе человека, обладающего техническими и информатическими навыками, способного реализовывать алгоритмы анализа данных самостоятельно или программировать решения для команды, пользуясь уже созданными наработками. Таким образом, возникает необходимость в простых и понятных средствах, которыми могут пользоваться исследователи без информатической подготовки. На данный момент таких инструментов достаточно мало, а в тех, что есть, неполноценный функционал.

Таким образом, целью данной работы является разработка удобного и практичного инструмента для полноценного анализа данных экспрессии генов.

ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

В данной обзорной главе будет представлена предметная область биоинформатики, введено понятие экспрессии генов, рассмотрены существующие решения для анализа экспрессии генов, а также перечислен ряд технологий, которые используются или могут быть использованы для создания инструментов анализа экспрессии генов.

1.1. Биоинформатика

Биоинформатика — наука, объединяющая в себе методы прикладной математики, статистики, информатики для создания новых методов и алгоритмов для анализа разного рода биологических данных.

Биоинформатика занимается биохимией, биофизикой, экологией и многими другими областями биологии. Однако фокус в данной работе будет сосредоточен на конкретную задачу биоинформатики — анализ экспрессии генов.

1.1.1. Анализ экспрессии генов

Экспрессия генов — процесс преобразования наследственной информации от гена (в виде последовательности нуклеотидов ДНК) в функциональный продукт (РНК или белок).

Анализ экспрессии генов позволяет выяснить, как ведет себя каждый отдельный ген в разных условиях, тканях или организмах.

Экспрессия гена в образце характеризуется вещественным числом, которое также можно назвать некоторой мерой активности гена в данных условиях.

1.1.2. Используемые методы

Как было сказано ранее, биоинформатика использует в себе математику, информатику и статистику. Соответственно, задача анализа экспрессии генов сводится к исследованию путем статистических методов и алгоритмов числовой двумерной матрицы, где в виде вещественных чисел демонстрируется активность каждого гена в каждом образце. Пример такой матрицы можно увидеть в таблице 1.

Одним из первоочередных методов, применяемых для анализа экспрессии, является *визуальный анализ*. Числовая матрица представля-

Таблица 1 – Срез матрицы GSE14308. Строки матрицы соответствуют генам, столбцы — образцам.

	GSM357839	GSM357841	GSM357842	GSM357843	GSM357844
Rps29	16.32	16.30	16.25	16.32	16.30
Rpl13a	16.27	16.23	16.32	16.30	16.27
Rps3a1	16.23	16.19	16.30	16.25	16.25
Rpl38	16.21	16.25	16.27	16.27	16.21
Tmsb4x	16.30	16.32	16.23	16.21	16.32

ется в виде *тепловой карты*, где цветом показана активность каждого гена.

На рисунке 1 можно увидеть визуализацию матрицы экспрессии из таблицы 1 в виде тепловой карты.

Также к основным методам анализа относятся:

— кластеризация:
  – иерархическая: метод упорядования данных таким образом, чтобы их можно было визуализировать в виде дерева (дендрограммы);
  – вероятностная: метод разбиения данных на несколько групп (кластеров);
— дифференциальная экспрессия: метод, позволяющий сравнивать поведение генов в разных условиях и искать закономерности;
— метод главных компонент: метод для уменьшения размерности данных с наименьшей потерей информации.

1.2. Существующие решения для анализа экспрессии генов

1.2.1. R/Bioconductor

R — язык программирования для статистического анализа данных и работы с графикой [1].

Bioconductor — библиотека, содержащая в себе множество реализаций биоинформатических алгоритмов и методов обработки биологических данных на *R*. Она постоянно обновляется, пополняется новыми библиотеками, модерируется сообществом [2]. *R* и *Bioconductor* очень популярны в биоинформатической среде ввиду предоставляемых возможностей.

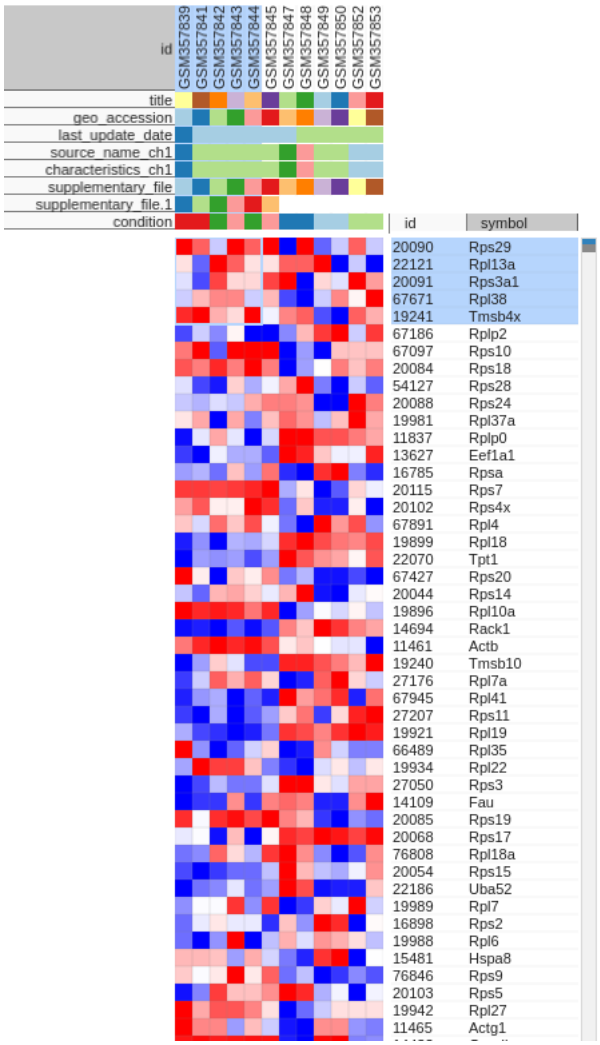

Рисунок 1 – Визуализация матрицы GSE14308 в виде тепловой карты в веб-приложении Morpheus

Однако для качественного и полноценного анализа с помощью этих инструментов, нужно иметь навыки программирования на *R*, что весьма неудобно для исследователей биологических специальностей.

1.2.1.1. Biobase и ExpressionSet

Необходимый минимум функций для работы с геномными данными содержится в *R*-пакете *Biobase* [2].

Класс `ExpressionSet` [3] также содержится в *Biobase*. Он помогает представлять данные об экспрессии генов в удобном формате:

- `assayData` — описание матрицы:
  - `features` — количество генов;
  - `samples` — количество образцов;
  - `exprs` — числовая матрица экспрессии;

- `phenoData` — аннотация к образцам:
  - `sampleNames` — идентификаторы образцов;
  - `varLabels` — названия характеристик;
  - `varMetadata` — описание характеристик;
  - `pData` — матрица характеристик;
- `featureData` — аннотация к генам:
  - `featureNames` — идентификаторы генов;
  - `fvarLabels` — названия характеристик;
  - `fvarMetadata` — описание характеристик;
  - `fData` — матрица характеристик.

Для доступа к каждому из элементов есть одноименная функция, что позволяет удобно взаимодействовать с экземплярами класса. Также многие из функций обработки данных в *Bioconductor* и в *Biobase* в частности завязаны на использование `ExpressionSet`.

1.2.2. GENE-E

GENE-E — Платформа для анализа данных и визуального исследования данных, созданная на *Java* и *R* [4]. Содержит в себе множество полезных для исследования инструментов: тепловые карты, кластеризацию, фильтрацию, построение графиков и т.д. Позволяет исследовать любые данные в виде матрицы. К тому же, содержит дополнительные инструменты для данных генетической экспрессии.

Недостатки:

- чтобы использовать, необходимо устанавливать на свой компьютер;
- поддержка данного приложения прекратилась в связи с созданием *morpheus.js* [5];
- не имеет открытого исходного кода, а только *API* для взаимодействия и создания новых приложений на его основе.

1.2.3. morpheus.js

Morpheus.js — веб-приложение для визуализации и анализа матриц от создателя *GENE-E* [5]. В отличие от предшественника, создано на *JavaScript* и с открытым исходным кодом. Удобно для использования исследователями без навыков программирования и так же, как и *GENE-E*, применимо к любым матрицам.

Недостатки:

— ограниченный набор функций, которых недостаточно для полноценного анализа;
— для расширения биоинформатическими алгоритмами, не прибегая к дополнительным инструментам, требуется реализовывать их заново на *JavaScript*;

Далее будут описаны основные компоненты, классы и методы, реализованные в *morpheus.js*, с которыми необходимо работать в случае расширения данного приложения.

1.2.3.1. Чтение данных

В *morpheus.js* данные могут быть загружены из файла одним из следующих путей:

— Из компьютера;
— По *URL*-ссылке;
— Из *Dropbox*.

Допустимые форматы загружаемых файлов:

— *txt*-файл с *tab*-разделителями;
— *Excel*-таблица;
— *Mutation Annotation Format* (*MAF*) [6];
— *Gene Cluster Text* (*GCT*) [7];
— *Gene Matrix Transposed* (*GMT*) [8].

Для каждого формата файла в исходном коде morpheus.js присутствует соответствующий обработчик данных.

Также, *morpheus.js* предлагает набор предзагруженных данных из базы *The Cancer Genome Atlas* (*TCGA*) [9].

1.2.3.2. Класс Dataset

Одним из ключевых классов всего веб-приложения является класс `Dataset`. В каждом экземпляре этого класса хранится вся необходимая информация о данных, в которую входят:

— числовая матрица, характеризующая уровень экспрессии всех генов во всех образцах;
— количество строк и столбцов в матрице;
— аннотация к образцам, например:

  - пол, возраст, контактную информацию испытуемых, если образцы были взяты с людей;
  - есть или нет инфекция в данном образце;
  - способ лечения;
  - контакты ответственного за взятие данного образца и пр.;
- аннотация к генам, например:
  - идентификатор гена в том или ином стандарте;
  - числовые характеристики гена (средний уровень экспрессии по образцам, номер кластера) и пр.

Аннотация реализована в классе `MetadataModel`, который представляет собой не что иное, как набор именованных векторов с характеристиками. В каждом векторе хранятся:

- название;
- формат (строка, число);
- массив значений.

Для векторов так же предусмотрены утилиты для визуализации. Так, например, есть возможность показать аннотацию в виде текста и/или цветом, что удобно для категориальных характеристик.

1.2.3.3. Класс SlicedDatasetView

Чаще всего во время работы программы экземпляры класса `Dataset` становятся обернуты в оболочку из `SlicedDatasetView`. Этот дополнительный класс дает возможность не пересоздавать каждый раз `Dataset`, а просто добавляет к данным информацию о том, какие индексы строк и столбцов выбраны и используются в данный момент и в каком порядке.

1.2.3.4. Класс HeatMap

Данный класс предназначен для визуализации данных, обернутых в класс `Dataset` или `SlicedDatasetView`. Он дает возможность выбирать, какая аннотация будет представлена на экране, цветовой код, выбирать строки и столбцы, с которыми будут работать те или иные инструменты.

1.2.3.5. Класс DatasetUtil

Класс `DatasetUtil` содержит в себе утилиты для обработки и чтения данных в `Dataset`:

- обработка входных файлов с данными и отправка их на соответствующий класс чтения в зависимости от формата;
- поиск по данным;
- преобразование данных в *JSON* и обратно.

1.2.3.6. Реализованные методы

В *morpheus.js* имеются реализации следующих методов обработки и анализа данных:

- `Adjust` — инструмент для корректировки данных:
  - логарифмическое преобразование;
  - обратное логарифмическое преобразование;
  - квантиль-нормализация;
  - z-тест;
  - устойчивый Z-тест;
- `Collapse` — инструмент, позволяющий агрегировать строки или столбцы с одинаковыми значениями с помощью функции: $\min$, $\max$, $mean$, $median$, sum, максимум 25-го и 75-го перцентилей;
- `CalculatedAnnotation` — добавление вычисленной аннотации для строк или столбцов;
- `Similarity Matrix` — построение матрицы соответствия строк или столбцов друг другу;
- `Transpose` — транспонирование матрицы;
- `t-SNE` — реализация алгоритма снижения размерности;
- `ChartTool` — построение графиков.

Также присутствуют фильтрация по строкам и столбцам и сортировка.

1.2.4. ProjectX

ProjectX — веб-приложение, созданное в рамках выпускной квалификационной работы [10], которое соединяло в себе возможности для веб-разработки, предоставляемые фреймворком *Django* и методы из библиотеки *Bioconductor* с помощью *OpenCPU*. Преимуществом данного

приложения перед актуальным на тот момент *GENE-E* была вопроизводимость исследований (на каждом этапе пользователь мог скачать исполняемый *R*-код, эквивалентный коду, выполненному в сервисе), также он содержал в себе большее число методов анализа и обработки данных.

Недостатки:

— работа над проектом была завершена до того, как у него появились пользователи, так что оно осталось невостребованным.

1.3. Использование R в веб-разработке

Как было сказано ранее, *R* удобно использовать для статистического анализа данных. Но чтобы использовать методы *R* в веб-приложениях, возникает необходимость в дополнительных технологиях для интеграции инструментов веб-разработки и *R*.

1.3.1. Shiny

Shiny — фреймворк для создания веб-приложений, используя только язык *R* [11].

1.3.2. OpenCPU

HTTP (*HyperText Transfer Protocol*) — протокол прикладного уровня передачи данных на основе технологии «клиент-сервер» [12].

HTTP API - набор процедур и функций, вызов которых и возвращение их результа осуществляется посредством протокола *HTTP*.

RPC (*Remote Procedure Call*) — класс технологий, позволяющих компьютерным программам вызывать функции или процедуры в другом адресном пространстве.

Веб-сервер — сервер, принимающий *HTTP*-запросы от клиентов, обычно веб-браузеров, и выдающий им *HTTP*-ответы.

OpenCPU — система для встроенных научных вычислений и воспроизводимых исследований, предоставляющая *HTTP API* для вызова R-функций и взаимодействия с R-объектами с помощью `POST` и `GET` запросов [13].

1.3.2.1. OpenCPU-сервер

OpenCPU-сервер можно запустить одним из следующих способов:

— однопользовательский сервер: сервер запускается из активной *R*-сессии и предназначен в основном для разработки и локального использования. Такой сервер не поддерживает параллельных запросов, так как *R*-сессии работают в одном потоке;
— облачный сервер: этот сервер можно запустить на *Ubuntu 16.04* и выше. Запуск и настройка облачного сервера осуществляется с помощью *Apache* или *Nginx*. В отличие от однопользовательского сервера, здесь поддерживаются параллельные запросы и настройка безопасности.

Однопользовательский сервер в большинстве случаев работает значительно быстрее, чем облачный, так как последний управляется с помощью *rApache* [14], что добавляет удобства в использовании, но замедляет доступ к серверу. Первым же можно пользоваться как многопользовательским, используя несколько экземпляров сервера, доступ к которым осуществляется с помощью *Apache* и балансировщика, который тот предоставляет.

1.3.2.2. OpenCPU API

Входной точкой *API* является `/ocpu/`. `GET`-запрос используется для получения некоторого ресурса, а `POST`-запрос используется для *RPC*. На таблице 2 можно увидеть более подробное описание запросов.

Таблица 2 – Запросы к *OpenCPU*-серверу, их аргументы и действия

Запрос	Действие	Аргументы	Пример
`GET`	посмотреть объект	формат представления объекта	GET /ocpu/library/MASS/R/cats/json
`POST`	вызвать функцию	аргументы функции	POST /ocpu/library/stats/R/rnorm
`GET`	прочитать файл	-	GET /ocpu/library/MASS/NEWS
`POST`	запустить скрипт	аргументы запуска скрипта	POST /ocpu/library/MASS?scripts/ch01.R

На *OpenCPU*-сервере могут быть доступны:
— библиотеки и содержащиеся в них пакеты: `/ocpu/library/pkgname/`;
— пакеты из *git*-репозиториев: `/ocpu/apps/gituser/reponame/`;
— пакеты, установленные в домашней директории *Linux*-пользователя: `/ocpu/user/username/library/pkgname/`;

— временные сессии, содержащие вывод от запуска функции или скрипта: `/ocpu/tmp/key/`.

1.3.2.3. Opencpu.js

Так как зачастую *OpenCPU* используется разработчиками для использования *R* в веб-приложениях для анализа и визуализации данных, для удобной интеграции *JavaScript* и *R* существует библиотека *opencpu.js*, которая реализует *RPC*-вызовы по принципу *Asynchronous JavaScript and XML* (*Ajax* [15]). Таким образом запросы обрабатываются в фоновом режиме, тем самым не замедляя работу графического интерфейса и вычислений, осуществляемых на стороне клиента.

В данной библиотеке реализован класс `Session`, содержащий в себе ключ сессии, адреса на ссылки, файлы и переменные, существующие внутри сессии.

Для подключения к *R*-пакету на *OpenCPU*-сервере удобно использовать код, представленный на листинге 1. Для успешного подключения *R*-пакет должен быть предварительно установлен на *host*-машину, на которой располагается сервер.

```
1  ocpu.seturl("//hostname/ocpu/library/{pkgname}/R");
```

Листинг 1 – Подключение к *R*-пакету

После этого можно вызывать и запускать функции, содержащиеся в данном *R*-пакете, например, как в листинге 2.

```
1  var req = ocpu.rpc("function.name", arguments, callback(session) {
2    \\ Handling result
3  });
```

Листинг 2 – Шаблон вызова *R*-функции из *JavaScript*

1.4. Инфраструктура

1.4.1. Docker

Docker — программное обеспечение для автоматизации запуска и внедрения приложений внутри контейнеров [16].

Для дальнейшего описания данного инструмента введем несколько определений.

Образ — отдельный исполняемый пакет, включающий себя все необходимое для запуска единицы программного обеспечения, в том числе исходный код, библиотеки, переменные окружения, конфигурационные файлы. Зачастую образ построен на основе другого образа с дополнительной конфигураций. Образ компилируется по *Dockerfile*, каждая команда в котором соответствует новому слою. При перекомпиляции обновляются только те слои, которые изменились.

Контейнер — запущенный экземпляр образа. Контейнер обычно исполняется изолированно от окружения, имея доступ к файлам или портам хост-системы только при наличии соответствующей конфигурации.

В отличие от виртуальных машин, которые запускают гостевую операционную систему в каждом экземпляре, контейнеры могут разделять общее ядро, и вся информация, которая должна быть в контейнере, это исполняемый процесс и его зависимости. Исполняемые процессы из контейнеров работают как нативные процессы, и могут управляться по отдельности.

Для контроля версий и хранения образов в открытом доступе используется *Docker Hub* [17]. В этом хранилище можно как добавлять репозитории, управляемые вручную, так и поддерживать автоматические сборки (*Automated Build*), которые привязаны к репозиториям на в популярных системах контроля версий: GitHub [18] и Bitbucket [19].

1.4.2. Apache

Apache HTTP Server Project — устойчивый, полностью открытый *HTTP*-сервер [20].

Apache позволяет конфигурировать веб-сервисы, переадресацию, балансировку нескольких экземпляров серверов.

1.5. Форматы сериализации данных

1.5.1. JSON

JSON — текстовый формат представления данных, основанный на *JavaScript*, который, среди прочих достоинств, легко читается человеком.

JSON-текст представляет собой одну из следущих структур:

— пара *ключ*: *значение*, где ключом может быть только регистрозависимая строка, а в качестве значения может выступата массив, число, строка, литералы или другой *JSON*-объект;
— набор значений.

1.5.2. Protocol Buffers

Protocol Buffers (Protobuf) — гибкий, универсальный и автоматизированный механизм для сериализации структурированных данных [21].

Структура информации задается с помощью `*.proto`-файлов в форме сообщений (`message`).

ProtoBuf-формат не является человекочитаемым, данные хранятся в двоичном формате. Для десериализации и дальнейшего чтения необходим соответствующий `*.proto`-файл. Файл с форматом компилируется соответствующим выбранному языку программирования компилятором, таким образом будет создан класс доступа к данным. С помощью этого класса уже можно сериализовать/десериализовать данные, получать данные с помощью `get/set`-методов и пр.

1.6. Открытые источники данных экспрессии генов

1.6.1. Gene Expression Omnibus

Gene Expression Omnibus (GEO) — международный публичный репозиторий, агрегирующий и распространяющий различные формы геномных данных от исследовательского сообщества [22].

GEO предоставляет устойчивую базу данных для эффективного хранения геномных данных, содержит их в качественно аннотированном формате и дает возможность удобно как добавлять новые данные для публикации, так и запрашивать интересующие данные для исследований.

В *GEO* содержатся следующие типы и форматы данных:
— информация о платформе, на которой производилось секвенирование генов (*Platform records*): `GPLxxx`;
— информация об образцах и условиях, в которых производились исследования этих образцов (*Sample records*): `GSMxxx`;
— информация о непосредственно исследованиях, полученные данные и выводы (*Series records*): `GSExxx`.

— обработанная и подготовленная для дальнейшего статистического анализа *GEO*-кураторами информация об исследованиях (*(*DataSet records)): `GDSxxx`.

В библиотеке *Bioconductor* есть *R*-пакет *GEOquery* для удобной загрузки данных из *GEO* [23].

1.6.2. The Cancer Genome Atlas

The Cancer Genome Atlas (TCGA) — проект, нацеленный на систематизацию данных о генетических мутациях, приводящих к возникновению рака. На данный момент участниками проекта отсеквенировано и проанализировано 33 вида рака [9].

1.7. Постановка задачи

Рассмотрев существующие решения для анализа экспрессии генов и инструментов, которые могли бы пригодиться для будущих решений, можно сформулировать цель и основные задачи данной работы

1.7.1. Цель работы

Целью работы является создание веб-приложения, интегрирующего существующие возможности веб-приложения *morpheus.js* и методы анализа, реализованные в *Bioconductor*.

1.7.2. Основные задачи

Для достижения поставленной цели были сформулированы следующие задачи:

а) разработать способ взаимодействия между *js*-клиентом и *R* и встроить его в *morpheus.js*, чтобы избежать реализации с нуля уже существующих алгоритмов;
б) реализовать графический интерфейс в *js*-клиенте и серверную реализацию в *R*-пакете;
в) соединить все составляющие в одном веб-приложении *phantasus*;
г) запустить веб-приложение в открытый доступ для исследователей.

1.7.3. Требования к веб-приложению phantasus

1.7.3.1. Доступность

Необходимо, чтобы веб-приложение *phantasus* было доступно для исследователей независимо от их местоположения и времени суток. Варианты действий:

а) сделать его доступным по определенному веб-адресу, и тогда пользователь сможет продолжать исследования из любой точки, где есть подключение к интернету;
б) предоставить возможность запускать приложение локально, например, с помощью *Docker* или внутри *R*.

1.7.3.2. Возможность дальнейшего расширения функционала

Как уже было сказано выше, библиотека *Bioconductor* постоянно обновляется и пополняется новыми алгоритмами, а исследователи находят новые методы для анализа экспрессии генов, так что необходимо не только реализовать дополнительные методы, но также отладить и описать алгоритм действий для добавления новых.

Выводы по главе 1

В данной главе были введены понятия биоинформатики и анализа экспрессии генов, обозначена цель работы и задачи, выполнение которых необходимо для ее достижения, представлены существующие решения с их достоинствами и недостатками, а также перечислен ряд инструментов, которые могут быть использованы для решения поставленных задач работы.

ГЛАВА 2. АРХИТЕКТУРА ПРОЕКТА PHANTASUS

В этой главе будет рассмотрена архитектура проекта *phantasus*, общая схема, ключевые компоненты и их взаимодействие между ними. Также будут описаны сопутствующие инструменты и их предназначение в системе, и ключевые для архитектуры выдержки из исходного кода.

2.1. Общая схема

Проект *phantasus* состоит из следующих компонент:

— клиентская сторона проекта: *phantasus.js* — модифицированный *morpheus.js*, дополненный интерфейсами для новых инструментов, поддержкой `ExpressionSet` и сериализации в *ProtoBuf*;

— серверная сторона проекта: *R*-пакет *phantasus* — пакет, включающий в себя вычисления и реализации новых инструментов.

Общую схему взаимодействия можно увидеть на диаграмме 2.

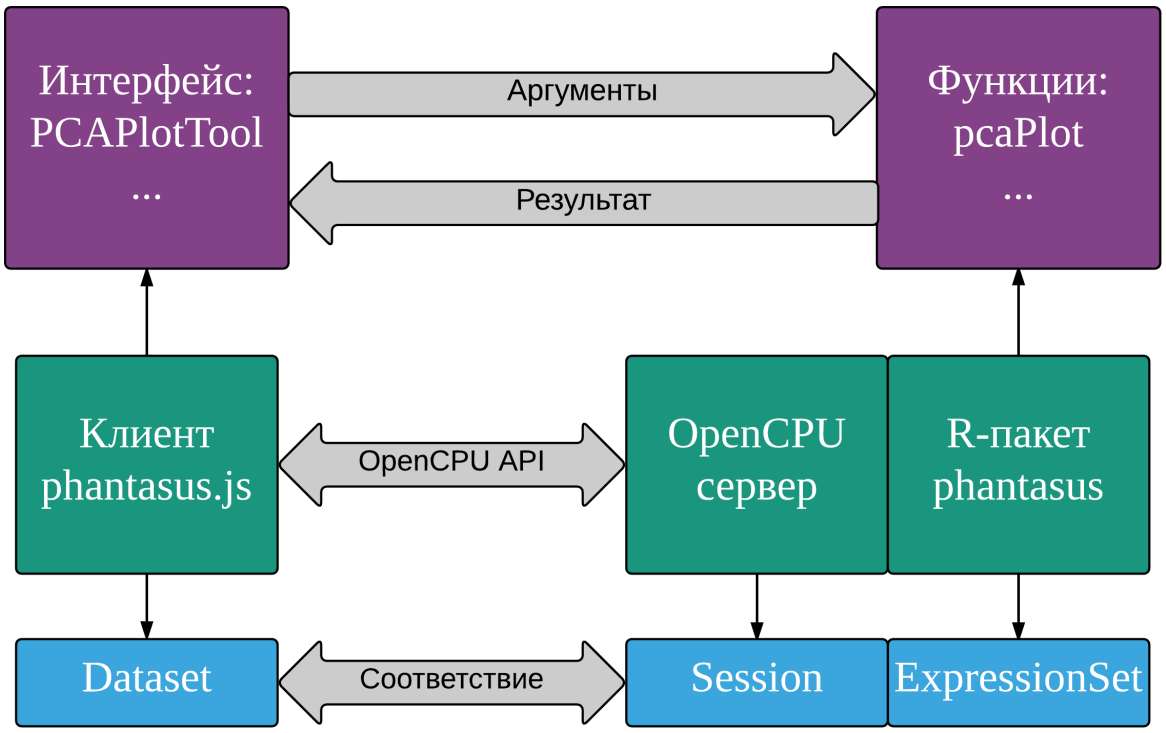

Рисунок 2 – Общая схема проекта *phantasus*

2.2. Взаимодействие между клиентом и сервером

Все инструменты, реализованные в проекте, имеют две компоненты:

— графический интерфейс в *phantasus.js*;
— вычислительную реализацию в *R*-пакете *phantasus*.

Соответственно, для связи этих компонент необходим своеобразный мост между клиентом, написанном на *JavaScript*, и функциями, написанными на *R*. В качестве такого моста используется описанный в обзоре *OpenCPU*.

Помимо наличия моста также необходимо, чтобы общение между клиентом и сервером происходило достаточно быстро, что приводит к потребности в дополнительной сериализации сообщений. В проекте для этого используется протокол *ProtoBuf*. В данном разделе будет подробно описано об использовании данных технологий при реализации взаимодействия между клиентом и сервером.

2.2.1. OpenCPU

Как было сказано выше, *OpenCPU* используется для связи между *JavaScript*-клиентом и *R*-функциями.

2.2.1.1. Использование OpenCPU со стороны клиента

Как было сказано в обзоре, для удобной интеграции *JavaScript* и *R* существует *JavaScript*-библиотека *opencpu.js*, в которой реализованы *RPC*-вызовы. С помощью этой библиотеки в проекте реализована взаимосвязь между графическими интерфейсом инструментов и соответствующих функций из *R*-пакета.

Каждый из инструментов, реализованных в *phantasus.js* действует по следующему принципу:

а) обработка аргументов, полученных в интерфейсе;
б) подготовка данных и аргументов к отправке на сервер;
в) *RPC*-вызов функции, как на листинге 2;
г) обработка содержащегося в полученной сессии результата в виде глобальной переменной, ответа функции или сохраненного в сессии файла;
д) передача ответа в интерфейс.

2.2.1.2. Использование OpenCPU со стороны R-пакета

Со стороны *R*-функций никаким специальным образом не обозначается, что результат работы будет передан именно в *JavaScript*-клиент,

все реализованные функции достаточно универсальны. Функция возвращает результат, в некоторых случаях дополнительно сохраняя его в файл или глобальную переменную, что в дальнейшем использует клиент, который заранее знает, в каком формате получит ответ.

2.2.2. Protocol Buffers

OpenCPU API поддерживает передачу сообщений в *ProtoBuf*-формате. Однако, *JavaScript*-библиотека не предусматривает такой возможности, так что в процессе работы над проектом была добавлена сериализация данных в *ProtoBuf* на стороне клиента и поддержка сериализованных сообщений в *opencpu.js*.

2.2.2.1. Сериализация данных со стороны клиента — protobuf.js

Чаще всего обрабатываемые матрицы содержат от 10000 до 40000 строк и от 12 до 40 столбцов. Соответственно, пересылать их между клиентом и сервером в *JSON*-формате слишком долго.

Как было сказано в обзоре, технология *Protocol Buffers* позволяет лучше сериализовать данные, чтобы уменьшить размер пересылаемого пакета.

К сожалению, *Google Developers* официально поддерживают только *Java*, *Python*, *C++*, *Go*, *Objective-C*, *Ruby*, *JavaNano* и *C#*. Для *JavaScript* сообщество создает поддержку самостоятельно. После анализа существующих решений, было решено выбрать библиотеку *ProtoBuf.js* [24].

Данная библиотека реализует класс `Builder`, который компилируется из `*.proto`-файлов и позволяет получить доступ к сериализованным данным. С помощью экземпляра данного класса можно закодировать соответствующий *JSON*-объект в `Uint8Array`, чтобы после переслать его в сжатом виде на сервер.

Для преобразования экземпляров классов `Dataset` и `SlicedDatasetView`, которые были представлены в обзоре приложения *morpheus.js*, в сериализованнный `Uint8Array` используется протокол, описанный в приложении А. Данный протокол используется также в *R*-пакете *protolite* [25], с помощью которого данные десериализуются на сервере.

Таким образом в класс `DatasetUtil` была добавлена утилита, сериализующая `Dataset`, которая используется перед отправкой данных

на сервер, и десериализующая результат работы *R*-функций, если те сохраняют результат в бинарном файле.

2.2.2.2. Сериализация данных со стороны сервера — protolite

Внутри *R*-функций нет необходимости вручную явно десериализовавать входные данные, так как *OpenCPU* автоматически разбирает аргументы до входа в функцию.

Однако, если функция возвращает матрицы больших размеров, используется *R*-пакет *protolite* [25] для сериализации результата работы, который после сериализованный записывается в бинарный файл. В дальнейшем этот файл считывается клиентом из директории возвращенной временной сессии.

2.3. Поддержка ExpressionSet

Как было сказано ранее, а также продемонстрировано на схеме 2, в ходе работы приложения постоянно поддерживается соответствие между экземпляром класса `Dataset` на клиенте и экземпляром класса `ExpressionSet` на сервере. В данном разделе будет подробнее разобрана реализация данного соответствие.

2.3.1. Поддержка ExpressionSet на стороне клиента

В *phantasus.js* в Dataset добавлено дополнительное поле `esSession`, в котором находится объект класса `Promise` для асинхронного обновления ключа сессии в этом поле.

При загрузке или обновлении `Dataset` осуществляется следующий ряд действий:

а) в поле `dataset.esSession` записывается экземпляр класса `Promise`, который позволяет продолжать загрузку данных в фоновом режиме, а также ждать, когда данные будут обработаны прежде чем запускать функции использующие `ExpressionSet` в качестве аргумента (`pcaPlot`, `kmeans`, `limma`). При создании `Promise` в аргументах указывается две функции: `reject` и `resolve`;

б) актуальное содержимое экземпляра класса `Dataset` вместе с матрицей и аннотацией сериализуется в *ProtoBuf* по протоколу, описанному в приложении А;

в) с помощью *opencpu.js* отправляется *RPC* за функцией `createES` с аргументом в виде сериализованных данных;
г) данные поступают на сервер, десериализуются там автоматически и функция `createES` создает `ExpressionSet`, являющийся копией `Dataset` из клиента;
д) функция `createES` объявляет данный `ExpressionSet` глобальной переменной, таким образом имеется доступ к этому объекту по *API-entrypoint* `/ocpu/tmp/key/R/es`;
е) по завершении *RPC* получает ключ временной сессии, содержащий созданный `ExpressionSet` и завершает `Promise` с `resolve(session)`;
ж) если во время одного из этапов произошла ошибка, то `Promise` завершается с `reject(error)` с текстом ошибки.

2.3.2. Создание ExpressionSet из внешних данных на стороне сервера

В начале работы с *phantasus* необходимо загрузить данные. Если данные загружены из файла, то они будут сначала обработаны на клиенте, а после пересланы на сервер для создания *ExpressionSet* из них с помощью кода на листинге 3.

Функция `createES` принимает следующие аргументы:

— `data` — непосредственно матрица экспрессии;
— `pData` — аннотация к образцам;
— `varLabels` — названия характеристик описания образцов;
— `fData` — аннотация к генам;
— `fvarLabels` — названия характеристик описания генов.

По завершении функция отправляет `es` в глобальные переменные, чтобы созданный `ExpressionSet` был доступен по адресу: `/ocpu/tmp/key/R/es`. Таким образом, получив ключ данной сессии, можно иметь доступ и к `ExpressionSet`, находящемуся в ней.

Ключ сессии обновляется каждый раз при изменении `Dataset` в *phantasus.js*. Чаще всего изменения происходят в результате работы одного из следующих инструментов: `Adjust`, `Collapse`, `new HeatMap`, `Transpose`. Изменные данные, точно так же ,как и новые, пересылаются на сервер и ключ сессии обновляется в поле `esSession` в `Dataset`.

```
createES <- function(data, pData, varLabels, fData, fvarLabels) {
  exprs <- t(data)
  phenoData <- AnnotatedDataFrame(data.frame(pData))
  varLabels(phenoData) <- varLabels

  featureData <- AnnotatedDataFrame(data.frame(fData))
  varLabels(featureData) <- fvarLabels

  es <- ExpressionSet(assayData = exprs, phenoData=phenoData, featureData =
     featureData)
  assign("es", es, envir = parent.frame())
  es
}
```

Листинг 3 – Функция создания ExpressionSet из исходных данных

2.4. Способы визуализации

В данном разделе будут показаны достоинства и недостатки различных способов визуализации, а также применимость в различных ситуациях.

2.4.1. Отрисовка графиков на стороне сервера

Первый вариант визуализации графиков и схем состоит в отрисовке их там же, где и происходит вычисление всех необходимых для них данных, то есть на сервере внутри *R*-функции. Данный способ подразумевает отрисовку и сохранение изображений в виде *png* или *svg*-файла, который сохраняется внутри временной *OpenCPU*-сессии. Клиент после забирает из сессии изображение и показывает в графическом интерфейсе приложения.

Достойнства:

— возможность пользоваться проверенными *R*-пакетами для визуализации, например, *ggplot2* [26];

— файл можно переиспользовать при необходимости, нужно знать только ключ сессии, где он находится.

Однако таким образом можно использовать только статичные изображения.

2.4.2. Визуализация на стороне клиента

Для отображения интерактивных графиков удобно использовать библиотеку *plotly.js* [27], которая предоставляет *API*, где описание графика строится в *JSON*-формате. Таким образом можно строить интерак-

тивные изображения, переложить всю работу по визуализации с сервера на клиент.

Именно этот способ на данный момент используется в проекте, подробнее об инструменте будет рассказано в главе 3.

Выводы по главе 2

В данной главе были рассмотрены основные составляющие проекта:

— *phantasus.js* — расширенный *morpheus.js*;
— *R*-пакет *phantasus* — *R*-пакет, содержащий в себе серверные реализации всех добавленных методов и инструментов.

Также были представлены следующие подробности архитектуры проекта:

— общая схема всего проекта, где показаны компоненты и связи между ними;
— способ взаимодействия между компонентами с помощью *OpenCPU*;
— сериализация данных в *ProtoBuf* на стороне клиента и на стороне сервера;
— поддержка `ExpressionSet`;
— способы визуализации.

ГЛАВА 3. РЕАЛИЗАЦИЯ И ИСПОЛЬЗОВАНИЕ

В этой главе будут подробно описаны реализованные методы анализа в проекте *phantasus*, их реализация на стороне сервера и на стороне клиента, будет рассказано о способах запуска приложения и другие инфраструктурные подробности.

3.1. Реализованные методы анализа экспрессии

В ходе работы над проектом были реализованы следующие функции и методы анализа экспрессии:

- `loadGEO` — загрузка и визуализация данных из *Gene Expression Omnibus*;
- `pcaPlot` — реализация метода главных компонент и визуализация результата;
- `kmeans` — реализация кластеризации методом *kmeans*;
- `limmaAnalysis` — анализ дифференциальной экспрессии для сравнения образцов.

В этом разделе будут один за другим описаны все вышеперечисленные методы.

3.1.1. Загрузка данных из GEO

В разделе 1.6.1 обзора были описаны форматы данных в репозитории *Gene Expression Omnibus*. В *phantasus* загрузка данных из *GEO* осуществляется следующим образом:

а) функция `loadGEO` принимает на вход идентификатор *GEO*;
б) в зависимости от его вида (*GSE* или *GDS*) запускаются дополнительные функции (`getGSE` на листинге 4 и `getGDS` на листинге 5);
в) в каждой из двух функций с помощью `GEOquery::getGEO` загружаются данные с аннотацией (или подгружаются из кэша, если он указан или если их уже загружали);
г) результат обрабатывается, создается `ExpressionSet` и отправляется в глобальные переменные;
д) в файл записываются сериализованные в *ProtoBuf* данные, в том же формате, что и при создании `ExpressionSet` из внешних данных (смотри раздел 2.3.2), которые после считает и обработает клиент.

```
getGSE <- function(name, destdir = tempdir()) {
  es <- getGEO(name, AnnotGPL = T, destdir = destdir)[[1]]
  featureData(es) <- featureData(es)[,grepl("symbol", fvarLabels(es), ignore.case = T)]
  phenoData(es) <- phenoData(es)[,grepl("characteristics", varLabels(es), ignore.case = T)
                                 | (varLabels(es) %in% c("title", "id", "geo_accession"))]
  chr <- varLabels(es)[grepl("characteristics", varLabels(es), ignore.case = T)]
  take <- function(x, n) {
    sapply(x, function(x) { x[[n]] })
  }
  rename <- function(prevName, x) {
    splitted <- strsplit(x, ": ")
    sumlength <- sum(sapply(as.vector(splitted), length))
    if (sumlength != 2 * length(x)) {
       return(list(name = prevName, x = x))
    }
    splittedFirst <- unique(take(splitted, 1))
    if (length(splittedFirst) == 1) {
       res = list(name = splittedFirst[1], x = take(splitted, 2))
    }
    else {
       res = list(name = prevName, x = x)
    }
    res
  }
  renamed <- lapply(chr, function(x) { rename(x, as.vector(pData(es)[,x])) })
  phenoData(es) <- phenoData(es)[, !(varLabels(es) %in% chr)]
  pData(es)[,take(renamed,1)] <- take(renamed,2)
  es
}
```

Листинг 4 – Загрузка данных типа GSE из Gene Expression Omnibus

Основной код загрузки и дополнительные утилиты к нему можно увидеть на листинге 6.

Такая реализация загрузки данных из *GEO* позволяет избежать проблемы *cross-origin request*, которая служила препятствием для загрузки из *GEO* в исходном приложении *Morpheus*.

3.1.2. Метод главных компонент и визуализация его результата

Данный инструмент предназначен для построения графиков в соответствие с методом главных компонент. В качестве аргументов на вход к инструменту подается:

— номера образцов для сравнения;

— категориальная аннотация для различения точек по цвету (если не указана, то стандартный цвет);

```
getGDS <- function(name, destdir = tempdir()) {
 l <- getGEO(name, destdir = destdir)
 table <- slot(l, 'dataTable') # extracting all useful information on dataset
 data <- Table(table)  # extracting table ID_REF | IDENTIFIER/SAMPLE | SAMPLE1 |
     ...
 columnsMeta <- Columns(table) # phenoData
 sampleNames <- as.vector(columnsMeta[["sample"]])
 rownames <- as.vector(data[["ID_REF"]])
 symbol <- as.vector(data[["IDENTIFIER"]])
 data <- data[,sampleNames] # expression data
 exprs <- as.matrix(data)
 row.names(exprs) <- rownames
 row.names(columnsMeta) <- sampleNames
 # columnsMeta <- columnsMeta[,!(colnames(columnsMeta) %in% c('sample'))]
 pData <- AnnotatedDataFrame(data.frame(columnsMeta, check.names = F))
 fData <- data.frame(matrix(symbol, nrow(exprs), 1));
 colnames(fData) <- "symbol"
 fData <- AnnotatedDataFrame(fData)
 featureNames(fData) <- rownames
 ExpressionSet(assayData = exprs, phenoData = pData, featureData = fData)
}
```

Листинг 5 – Загрузка данных типа GDS из Gene Expression Omnibus

— числовая аннотация для различения точек по размеру (если не указана, то стандартный размер);
— аннотация для подписей к точкам (если не указана, то без подписи);
— функция замены NA в данных при вычислении матрицы *PCA* (`mean` или `median`).

Далее по алгоритму, описанному в разделе 2.2.1.1, на *OpenCPU*-сервер отправляется *RPC*-вызов с аргументами: ключ сессии, содержащий актуальный `ExpressionSet`, и функция замены NA.

Данные аргументы приходят на вход к функции `pcaPlot`, реализованной в *R*-пакете *phantasus*, код которой можно увидеть на листинге 7. Предварительно все NA-значения заменяются в соответствии с переданной функцией, если этого не сделать, то дальнейшие вычисления будут невозможны. Матрица экспрессии из входного *ExpressionSet* передается в стандартную функцию `prcomp` из *R*-пакета *stats* [28], которая и вычисляет результирующую матрицу.

На клиент в *JSON*-формате приходит вычисленная матрица *PCA*.

После, по дополнительным аргументам и вычисленной матрице, строится интерактивный график с помощью *plotly.js*, пример которого можно увидеть на рисунке 3.

```
loadGEO <- function(name, type = NA) {
  es <- getES(name, type, destdir = "/var/phantasus/cache")
  assign("es", es, envir = parent.frame())
  data <- as.matrix(exprs(es)); colnames(data) <- NULL; row.names(data) <- NULL

  pdata <- as.matrix(pData(es)); colnames(pdata) <- NULL; row.names(pdata) <-
     NULL

  participants <- colnames(es)
  rownames <- rownames(es)

  fdata <- as.matrix(fData(es))
  colnames(fdata) <- NULL
  row.names(fdata) <- NULL

  res <- list(data = data, pdata = pdata,
              fdata = fdata, rownames = rownames,
              colMetaNames = varLabels(phenoData(es)),
              rowMetaNames = varLabels(featureData(es)))

  f <- tempfile(pattern = "gse", tmpdir = getwd(), fileext = ".bin")
  writeBin(protolite::serialize_pb(res), f)
  f
}
getES <- function(name, type = NA, destdir = tempdir()) {
  if (is.na(type)) {
     type = substr(name, 1, 3)
  }
  if (type == 'GSE') {
    es <- getGSE(name, destdir)
  }
  else if (type == 'GDS') {
    es <- getGDS(name, destdir)
  }
  else {
    stop("Incorrect name or type of the dataset")
  }
  es
}
```

Листинг 6 – Загрузка данных из Gene Expression Omnibus

3.1.3. Кластеризация методом kmeans

Этот инструмент осуществляет разбиение генов на указанное пользователем число кластеров по алгоритму *kmeans*.

На клиенте в инструменте `KmeansTool`, который показан на рисунке 4, считываются следующие аргументы:

— количество кластеров, на которые нужно разбить данные;

— функция замены *NA* в данных.

Данные аргументы и ключ сессии актуального `ExpressionSet` отправляются на сервер в соответствующую функцию `kmeans`, код которой можно увидеть на листинге 8. Также как и в `pcaPlot`, здесь снача-

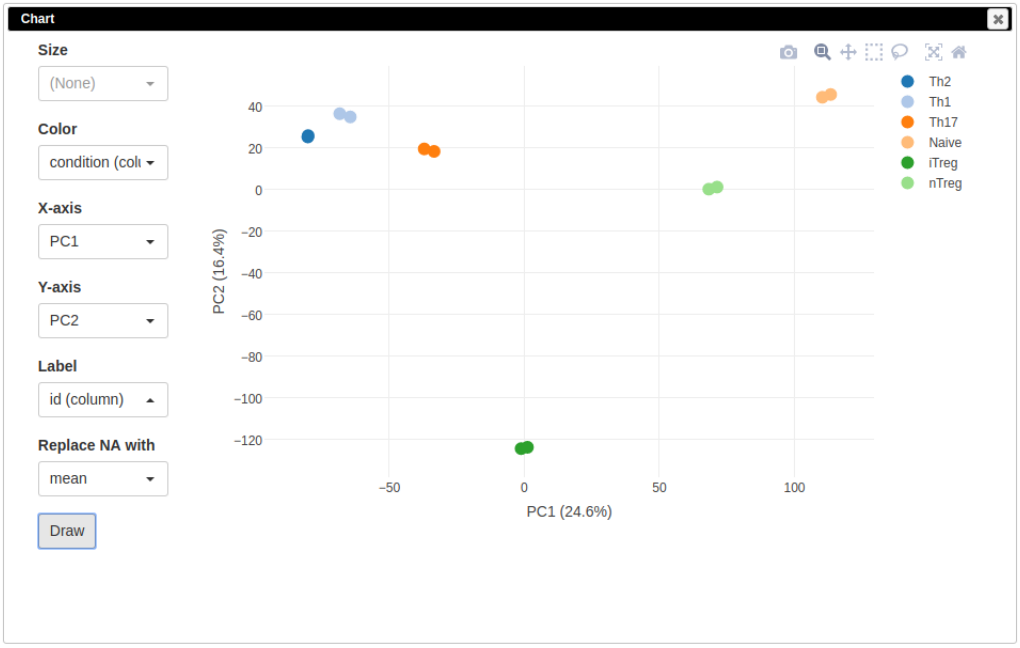

Рисунок 3 – Инструмент `PcaPlotTool` и отрисованный график по данным GSE14308

```
  pcaPlot <- function(es, rows=c(), columns = c(), replacena = "mean") {
  rows <- getIndicesVector(rows, nrow(exprs(es)))
  columns <- getIndicesVector(columns, ncol(exprs(es)))
  data <- data.frame(exprs(es))[rows, columns]

  ind <- which(is.na(data), arr.ind = T)
  if (nrow(ind) > 0) {
    data[ind] <- apply(data, 1, replacena, na.rm = T)[ind[,1]]
  }
  ind1 <- which(!is.nan(as.matrix(data)), arr.ind = T)
  left.rows <- unique(ind1[,"row"])
  data <- data[left.rows,]
  data <- t(data)

  pca <- prcomp(data)
  explained <- (pca$sdev)^2 / sum(pca$sdev^2)

  xs <- sprintf("PC%s", seq_along(explained))
  xlabs <- sprintf("%s (%.1f%%)", xs, explained * 100)

  pca.res <- as.matrix(pca$x); colnames(pca.res) <- NULL; row.names(pca.res) <-
      NULL
  return(jsonlite::toJSON(list(pca = t(pca.res), xlabs=xlabs)))
}
```

Листинг 7 – Вычисление матрицы главных компонент

ла заменяются NA на соответствующие данной функции значения, так как в этом случае NA-значения мешают вычислить среднее среди векторов. После данные отправляются в стандартную функцию `kmeans` па-

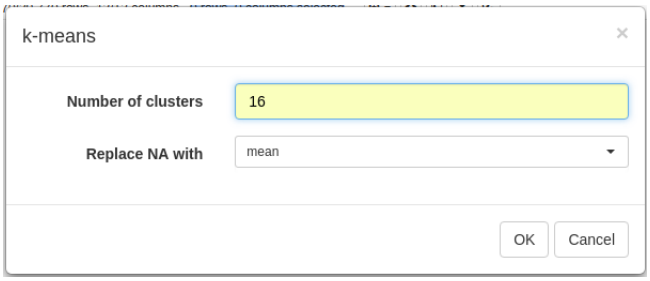

Рисунок 4 – Графический интерфейс инструмента KmeansTool

кета *stats* [28]. Результат возвращается как список соответствия каждого гена определенному кластеру, который приходит на клиент в *JSON*-формате и отрисовывается как новая цветовая аннотация к строкам. Пример такой аннотации можно увидеть на рисунке 5.

```
kmeans <- function(es, columns = c(), rows = c(), k, replacena = "mean") {
  assertthat::assert_that(k > 0)

  rows <- getIndicesVector(rows, nrow(exprs(es)))
  columns <- getIndicesVector(columns, ncol(exprs(es)))
  data <- replacenas(data.frame(exprs(es))[rows, columns], replacena)

  data <- t(scale(t(data)))
  while (sum(is.na(data)) > 0) {
    data <- replacenas()
    data <- t(scale(t(data)))
  }

  km <- stats::kmeans(data, k)
  res <- data.frame(row.names = row.names(exprs(es)))
  res[["cluster"]] <- NA
  res[names(km$cluster), "cluster"] <- as.vector(km$cluster)
  return(toJSON(as.vector(km$cluster)))
}
```

Листинг 8 – Кластеризация методом kmeans

3.1.4. Анализ дифференциальной экспрессии

Инструмент предназначен для анализа дифференциальной экспрессии: экспрессия генов сравнивается в двух группах образцов, и вычисляются несколько статистических характеристик, показывающих, насколько случайны различия этих групп.

На клиенте, в инструменте, показанном на рисунке 6, осуществляется получение следующих аргументов:

— какие аннотации образцов участвуют в сравнении;
— какая комбинация значений указанных выше аннотаций обозначает класс A;

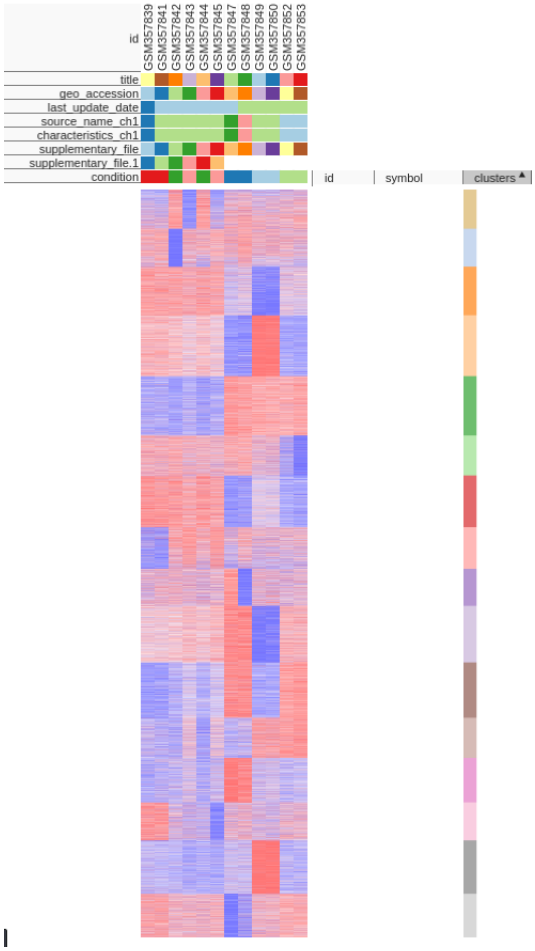

Рисунок 5 – Результат работы инструмена `KmeansTool` на данных GSE14308

— аналогично для класса B.

Далее происходит подготовка аргументов к отправке на сервер: образцы разбиваются по выбранным аннотациям на три группы: A, B и не участвующие в сравнении. Список соответствия образцов классам и ключ сессии, содержащей актуальный `ExpressionSet`, отправляются на сервер.

Аргументы приходят в функцию `limmaAnalysis`, код которой можно увидеть на листинге 9. Прежде чем использовать функцию de (*differential expression*), реализованную в пакете *limma* [29], которая помогает увидеть, насколько случайны различия между образцами, гены которых находятся в разных условиях, необходимо дополнить аннотацию образцов списком, содержащим идентификаторы классов сравнения.

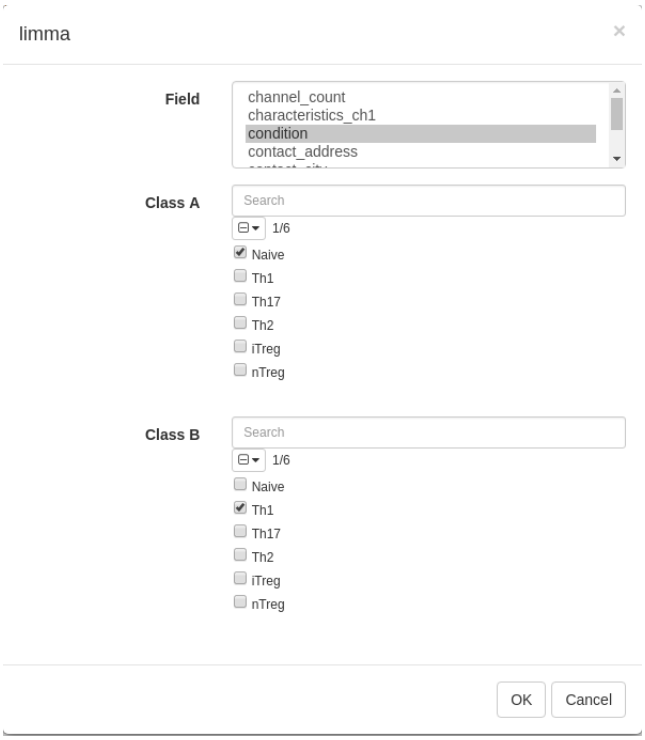

Рисунок 6 – Графический интерфейс инструмента `LimmaTool`

Построенный дизайн сравнения передается в функцию `de`, которая возвращает матрицу статистических характеристик к каждому гену. Эта матрица далее сериализуется в *ProtoBuf* и записывается в файл.

Клиент, получив ключ временной *OpenCPU*-сессии, читает файл с сериализованной в *ProtoBuf* матрицей результатов, которые с помощью *protobuf.js* разбираются и после отрисовываются в виде аннотации к строкам. Результат работы можно увидеть на рисунке 7.

3.2. Инфраструктура проекта phantasus

3.2.1. Структура git-репозитория

Как следует из главы об архитектуре проекта, проект состоит из двух составляющих:

— *phantasus.js* — *fork* репозитория *morpheus.js* [5];

— *phantasus* — репозиторий для *R*-пакета.

Внутри репозитория *phantasus* находится подмодуль для репозитория *phantasus.js*. Соответственно, чтобы загрузить целиком весь проект достаточно вызвать команду из листинга 10.

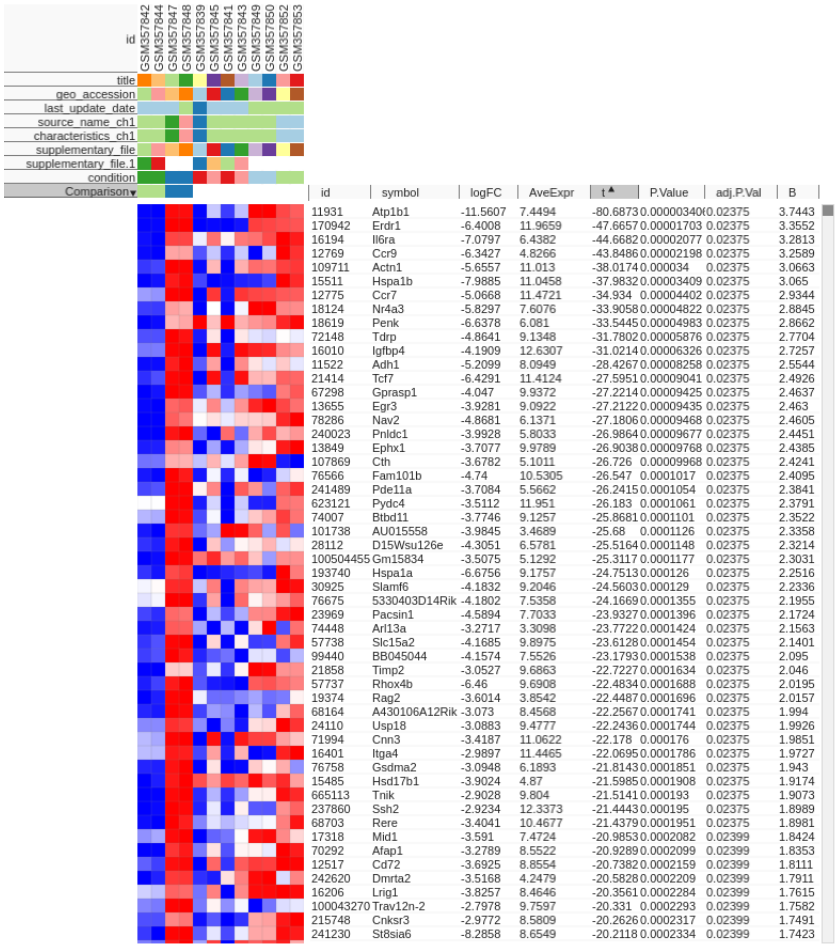

Рисунок 7 – Результат работы инструмента `LimmaTool` на данных GSE14308

3.2.2. Запуск приложения

На данный момент существует два варианта запуска приложения *phantasus*. В данном разделе будут описаны оба.

3.2.2.1. Единый R-пакет phantasus

Так как проект теперь существует в виде единого *git*-репозитория, его легко можно использовать как полноценный R-пакет, содержащий в себе в том числе и файлы для веб-приложения. С помощью функции, представленной на листинге 11, можно запускать веб-приложение phantasus непосредственно из R.

Соответственно, чтобы запустить приложение, необходимо выполнить код, представленный на листинге 12 и перейти на `http://localhost:8000` в браузере.

```
limmaAnalysis <- function(es, rows = c(), columns = c(), fieldValues) {
  assertthat::assert_that(length(columns) == length(fieldValues) || length(
     columns) == 0)
  rows <- getIndicesVector(rows, nrow(exprs(es)))
  columns <- getIndicesVector(columns, ncol(exprs(es)))
  fieldName <- "Comparison"
  fieldValues <- replace(fieldValues, fieldValues == '', NA)
  new.pdata <- pData(es)[columns,]
  new.pdata[[fieldName]] <- as.factor(fieldValues)
  new.pdata <- new.pdata[!is.na(new.pdata[[fieldName]]),]
  new.sampleNames <- row.names(new.pdata)
  es.copy <- es[rows, new.sampleNames]
  pData(es.copy) <- new.pdata
  fData(es.copy) <- data.frame(row.names=rownames(es.copy))
  es.design <- model.matrix(~0 + Comparison, data = pData(es.copy))
  colnames(es.design) <- gsub(pattern = fieldName,
                              replacement = '',
                              x = make.names(colnames(es.design)))
  fit <- lmFit(es.copy, es.design)
  fit2 <- contrasts.fit(fit, makeContrasts(B - A,
                                           levels=es.design))
  fit2 <- eBayes(fit2)
  de <- topTable(fit2, adjust.method="BH", number=Inf)
  de <- de[row.names(fData(es.copy)),]
  f <- tempfile(pattern = "de", tmpdir = getwd(), fileext = ".bin")
  writeBin(protolite::serialize_pb(as.list(de)), f)
  f
}
```

Листинг 9 – Реализация анализа дифференциальной экспрессии в R-пакете phantasus

```
  git clone --recursive https://github.com/ctlab/phantasus.git
```

Листинг 10 – Клонирование репозитория проекта phantasus

3.2.2.2. Docker-образ phantasus

На `hub.docker.com` существует автоматический репозиторий, привязанный к *git*-репозиторию *phantasus*. Для каждой перекомпиляции он использует Dockerfile с листинга Б, расположенный в репозитории.

На данный момент существуют две ветки Docker-образа:

— *master* — компиляция происходит из *master*-веток составляющих проекта, чаще всего эти скомпилированные образы стабильны и отправляются в открытый доступ для использования;

— *develop* — компиляция происходит из *develop*-веток составляющих проекта, эти образы используются для тестирования всего прило-

```
servePhantasus <- function(host, port,
                           staticRoot=system.file("www/phantasus.js", package="
                              phantasus"),
                           cacheDir=tempdir()) {
  options(phantasusCacheDir=cacheDir)
  app <-
    Rook::URLMap$new(
      "/ocpu"=opencpu:::rookhandler("/ocpu"),
      "/?"=Rook::Static$new(
        urls = c('/'),
        root = staticRoot
      ))

  httpuv::runServer(host,
                    port,
                    app=app)

}
```

Листинг 11 – Функция для запуска приложения из R

```
library(phantasus)
example(servePhantasus)
```

Листинг 12 – Запуск приложения в качестве R-пакета

жения в целом, тестирования нового функционала и не предназначены для использования на серверах.

Чтобы загрузить *Docker*-образ, нужно воспользоваться командой с листинга 13

```
  docker pull dzenkova/phantasus
```

Листинг 13 – Загрузка Docker-образа phantasus

Для запуска *Docker*-контейнера, необходимо воспользоваться командой с листинге 14.

```
  docker run -t -d -p "800$tag:80" -v /mnt/data/phantasus-cache:/var/phantasus/
     cache dzenkova/phantasus
```

Листинг 14 – Запуск Docker-контейнера

3.2.3. Кэш для данных из GEO

Независимо от способа запуска, данные, загруженные из *GEO*, кэшируются в определенной папке (сейчас это неизменяемое значение: `/var/phantasus/cache`), чтобы не было необходимости перескачивать их заново.

3.3. Настройка с помощью Apache

В обоих из представленных вариантах запуска, приложение запускается в формате *single-user*, соответственно, настройка деятельности приложения в формате *multi-user* осуществляется возможностями *Apache*. Конфигурационный файл можно увидеть в приложении В.

3.3.1. Переадресация OpenCPU-сервера

Как было сказано в разделе 1.3.2.1 обзора, *OpenCPU*-сервер работает быстрее в однопользовательском режиме. Сервер запускаетстся на определенном порту (например, 8001), после для корректной работы приложения, происходит переадресация запросов с `/ocpu` на `//localhost:8001/ocpu` и наоборот.

3.3.2. Балансировщик для multi-user соединения

Из-за того, что используется однопользовательский *OpenCPU*-сервер, несколько людей, использующих веб-приложение *phantasus* одновременно, вынуждены ждать, пока закончится запрос для одного.

Чтобы избежать такой ситуации, запускается четыре одинаковых экземпляра *OpenCPU*-сервера и с помощью *Apache*-балансировщика можно получать доступ к *R*-серверу параллельно.

Выводы по главе 3

В данной главе были рассмотрены реализованные инструменты и методы анализа:

— загрузка данных из *Gene Expression Omnibus*;
— метод главных компонент и его визуализация;
— кластеризация методом *kmeans*;
— анализ дифференциальной экспрессии.

Также были описаны технические подробности реализации веб-приложения: инструкции для запуска, варианты использования и детали настройки веб-приложения на сервере.

ЗАКЛЮЧЕНИЕ

В ходе данной работы веб-приложение *Morpheus* было расширено дополнительным функционалом с помощью реализованных в *R* и в *Bioconductor* статистических методов и методов анализа экспрессии генов. Веб-приложение *phantasus* было выпущено в открытый доступ и им можно пользоваться по адресу `https://artyomovlab.wustl.edu/phantasus`, либо же запускать самостоятельно в виде *R*-пакета или *Docker*-контейнера, как это было описано в главе 3. Проект *phantasus* используется в:

— лаборатории Максима Артемова в Washington University in St. Louis;
— лаборатории Laurent Yvan-Charvet в Université Nice Sophia Antipolis.

Также проект был продемонстрирован на семинаре по системной биологии в Сиднее (10-13 апреля 2017) и в Санкт-Петербурге (14-19 мая 2017).

Полученное веб-приложение соответствует всем поставленным требованиям.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 *R Core Team*. R: A Language and Environment for Statistical Computing / R Foundation for Statistical Computing. — Vienna, Austria, 2014. — URL: http://www.R-project.org/.

2 Orchestrating high-throughput genomic analysis with Bioconductor / Huber [и др.] // Nature Methods. — 2015. — Т. 12, № 2. — С. 115–121. — URL: http://www.nature.com/nmeth/journal/v12/n2/full/nmeth.3252.html.

3 *Falcon S.*, *Morgan M.*, *Gentleman R.* An Introduction to Bioconductor's ExpressionSet Class. — 2006. — URL: https://www.bioconductor.org/packages/devel/bioc/vignettes/Biobase/inst/doc/ExpressionSetIntroduction.pdf. [Электронный ресурс].

4 *Gould J.* GENE-E. — URL: http://www.broadinstitute.org/cancer/software/GENE-E/. [Электронный ресурс].

5 *Gould J.* morpheus.js. — URL: https://clue.io/morpheus.js/. [Электронный ресурс].

6 Mutation Annotation Format. — URL: https://wiki.nci.nih.gov/display/TCGA/Mutation+Annotation+Format+%28MAF%29+Specification/. [Электронный ресурс].

7 GCT. — URL: http://software.broadinstitute.org/cancer/software/genepattern/file-formats-guide#GCT/. [Электронный ресурс].

8 Gene Matrix Transposed file format. — URL: http://software.broadinstitute.org/cancer/software/gsea/wiki/index.php/Data_formats#GMT:_Gene_Matrix_Transposed_file_format_.28.2A.gmt.29/. [Электронный ресурс].

9 The Cancer Genome Atlas Pan-Cancer analysis project / T. C. G. A. R. Network [и др.] // Nat Genet. — 2013. — Окт. — Т. 45, № 10. — С. 1113–1120. — ISSN 1061-4036. — URL: http://dx.doi.org/10.1038/ng.2764 ; Commentary.

10 *Зайцев К.* Проектирование и реализация веб-сервиса для анализа экспрессии генов. — 2015. — URL: http://is.ifmo.ru/diploma-theses/2015/bachelor/zaycev/zaycev.pdf.

11 shiny: Web Application Framework for R / W. Chang [и др.]. — 2017. — URL: https://CRAN.R-project.org/package=shiny ; R package version 1.0.3.

12 Hypertext transfer protocol–HTTP/1.1: тех. отч. / R. Fielding [и др.]. — 1999.

13 *Ooms J.* The OpenCPU System: Towards a Universal Interface for Scientific Computing through Separation of Concerns // arXiv:1406.4806 [stat.CO]. — 2014. — URL: http://arxiv.org/abs/1406.4806.

14 *Horner J.* rApache: Web application development with R and Apache. — 2013. — URL: http://www.rapache.net/.

15 *Holdener III A. T.* Ajax: The Definitive Guide. — First. — O'Reilly, 2008. — ISBN 9780596528386.

16 *Merkel D.* Docker: Lightweight Linux Containers for Consistent Development and Deployment // Linux J. — Houston, TX, 2014. — Март. — Т. 2014, № 239. — ISSN 1075-3583. — URL: http://dl.acm.org/citation.cfm?id=2600239.2600241.

17 Docker Hub. — URL: https://hub.docker.com/. [Электронный ресурс].

18 GitHub. — URL: https://github.com/. [Электронный ресурс].

19 Bitbucket. — URL: https://bitbucket.org/. [Электронный ресурс].

20 Apache HTTP Server Project. — URL: https://httpd.apache.org. [Электронный ресурс].

21 *Varda K.* Protocol buffers: Google's data interchange format // Google Open Source Blog, Available at least as early as Jul. — 2008.

22 *Edgar R.*, *Domrachev M.*, *Lash A. E.* Gene Expression Omnibus: NCBI gene expression and hybridization array data repository // Nucleic acids research. — 2002. — Т. 30, № 1. — С. 207–210.

23 *Davis S.*, *Meltzer P.* GEOquery: a bridge between the Gene Expression Omnibus (GEO) and BioConductor // Bioinformatics. — 2007. — Т. 14. — С. 1846–1847.

24 *dcode*. Protocol Buffers for JavaScript (& TypeScript). — URL: https://github.com/dcodeIO/ProtoBuf.js/. [Электронный ресурс].

25 *Ooms J.* protolite: Fast and Simple Object Serialization to Protocol Buffers. — 2017. — URL: https://CRAN.R-project.org/package=protolite ; R package version 1.6.

26 *Wickham H.* ggplot2: Elegant Graphics for Data Analysis. — Springer-Verlag New York, 2009. — ISBN 978-0-387-98140-6. — URL: http://ggplot2.org.

27 *Inc. P. T.* Collaborative data science. — 2015. — URL: https://plot.ly.

28 *R Core Team*. R: A Language and Environment for Statistical Computing / R Foundation for Statistical Computing. — Vienna, Austria, 2017. — URL: https://www.R-project.org/.

29 limma powers differential expression analyses for RNA-sequencing and microarray studies / M. E. Ritchie [и др.] // Nucleic Acids Research. — 2015. — Т. 43, № 7. — e47.

ПРИЛОЖЕНИЕ А. ПРОТОКОЛ СЕРИАЛИЗАЦИИ В PROTOBUF

R-пакет protolite использует стандартный протокол сериализации, представленный на листинге А. Этот же протокол было решено использовать и при сериализации на клиенте для однообразия и для корректного разбора сообщений как на клиенте от сервера, так и на сервере от клиента.

```
package rexp;

option java_package = "org.godhuli.rhipe";
option java_outer_classname = "REXPProtos";

message REXP {
    enum RClass {
        STRING = 0;
        RAW = 1;
        REAL = 2;
        COMPLEX = 3;
        INTEGER = 4;
        LIST = 5;
        LOGICAL = 6;
        NULLTYPE = 7;
        NATIVE = 8;
    }
    enum RBOOLEAN {
        F=0;
        T=1;
        NA=2;
    }

    required RClass rclass = 1;
    repeated double realValue = 2 [packed=true];
    repeated sint32 intValue = 3 [packed=true];
    repeated RBOOLEAN booleanValue = 4;
    repeated STRING stringValue = 5;

    optional bytes rawValue = 6;
    repeated CMPLX complexValue = 7;
    repeated REXP rexpValue = 8;

    repeated string attrName = 11;
    repeated REXP attrValue = 12;
    optional bytes nativeValue = 13;
}
message STRING {
    optional string strval = 1;
    optional bool isNA = 2 [default=false];
}
message CMPLX {
    optional double real = 1 [default=0];
    required double imag = 2;
}
```

ПРИЛОЖЕНИЕ Б. DOCKERFILE

```
FROM ubuntu
RUN apt-get -y update && apt-get -y dist-upgrade && apt-get -y install \
        software-properties-common \
        git \
        libcairo2-dev \
        libxt-dev \
        libssl-dev \
        libssh2-1-dev \
        libcurl4-openssl-dev \
        apache2 \
        locales && \
    apt-add-repository -y ppa:opencpu/opencpu-1.6 && \
    apt-get -y update && apt-get -y install opencpu-lib

RUN touch /etc/apache2/sites-available/opencpu2.conf
RUN printf "ProxyPass /ocpu/ http://localhost:8001/ocpu/\nProxyPassReverse /ocpu/
    http://localhost:8001/ocpu\n" >> /etc/apache2/sites-available/opencpu2.conf
RUN a2ensite opencpu2

RUN sh -c 'echo "deb http://cran.rstudio.com/bin/linux/ubuntu trusty/" >> /etc/
   apt/sources.list'
RUN gpg --keyserver keyserver.ubuntu.com --recv-key E084DAB9
RUN gpg -a --export E084DAB9 | apt-key add -
RUN apt-get -y update && apt-get -y install \
    r-base \
    libprotobuf-dev \
    protobuf-compiler \
    r-cran-xml

RUN git clone --recursive https://github.com/ctlab/phantasus /root/phantasus
RUN cp -r /root/phantasus/inst/www/phantasus.js /var/www/html/phantasus
RUN R -e 'source("https://bioconductor.org/biocLite.R"); install.packages("XML",
   repo = "http://cran.gis-lab.info"); biocLite("Biobase"); biocLite("limma");
   biocLite("org.Mm.eg.db")'
RUN R -e 'install.packages("devtools", repo = "http://cran.gis-lab.info");
   library(devtools); install_github("hadley/scales"); install_github("assaron/
   GEOquery"); install("/root/phantasus")'

RUN a2enmod proxy_http
EXPOSE 80
EXPOSE 443
EXPOSE 8004
RUN locale-gen en_US.UTF-8
ENV LANG en_US.UTF-8
ENV LANGUAGE en_US:en
ENV LC_ALL en_US.UTF-8

RUN mkdir -p /var/phantasus/cache
VOLUME ["/var/phantasus/cache"]

CMD service apache2 start && \
    R -e 'opencpu::opencpu$start(8001)' && \
    tail -F /var/log/opencpu/apache_access.log
```

Листинг Б.1 – Dockerfile для Docker-образа веб-приложения phantasus

ПРИЛОЖЕНИЕ В. КОНФИГУРАЦИОННЫЙ ФАЙЛ APACHE

```
1     ProxyPass /phantasus balancer://phantasus-servers/phantasus
2     ProxyPassReverse /phantasus balancer://phantasus-servers/phantasus
3
4     <Location "/phantasus">
5         Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=
              BALANCER_ROUTE_CHANGED
6     </Location>
7
8     <Proxy balancer://phantasus-servers/>
9         BalancerMember http://localhost:8000/ route=1
10        BalancerMember http://localhost:8001/ route=2
11        BalancerMember http://localhost:8002/ route=3
12        BalancerMember http://localhost:8003/ route=4
13        ProxySet stickysession=ROUTEID
14    </Proxy>
15
16    <Location "/ocpu">
17        Header add Set-Cookie "ROUTEID=.%{BALANCER_WORKER_ROUTE}e; path=/" env=
              BALANCER_ROUTE_CHANGED
18    </Location>
19
20    ProxyPassMatch ^/ocpu/(.*/.*)$ balancer://phantasus-servers/ocpu/$1
21    ProxyPassReverse /ocpu/ balancer://phantasus-servers/ocpu
```