МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Параллельных вычислительных технологий
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.
(фамилия, имя, отчество)

_____
(подпись)

«__» июня 2018 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Мошкиной Алёны Дмитриевны
(фамилия, имя, отчество студента – автора работы)

Разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах
(тема работы)

Факультет Прикладной математики и информатики
(полное название факультета)

Направление подготовки 01.03.02. Прикладная математика и информатика
(код и наименование направления подготовки бакалавра)

Руководитель от НГТУ	**Автор выпускной квалификационной работы**
Маркова В.П. (фамилия, имя, отчество)	Мошкина А.Д. (фамилия, И.О.)
к.т.н., доцент (ученая степень, ученое звание)	ФПМИ, ПМ-43 (факультет, группа)
..... (подпись, дата)	

Новосибирск, 2018 г.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Параллельных вычислительных технологий
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.
(фамилия, имя, отчество)

«16» марта 2018 г.

(подпись)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

студентке ***Мошкиной Алёне Дмитриевне***
(фамилия, имя, отчество студента)

Направление подготовки *01.03.02. Прикладная математика и информатика*

Факультет Прикладной математики и информатики

Тема ***Разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах***

Исходные данные (или цель работы):

разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах.

Структурные части работы:

1. Обзор подходов к организации взаимодействия пользователей с высокопроизводительными вычислительными системами;

2. Проектирование программной системы;

3. Реализация программной системы;

4. Подготовка демонстрационных приложений;

5. Написание отчета;

_____

_____

_____

_____

_____

_____

_____

Задание согласовано и принято к исполнению.

Руководитель от НГТУ	**Студент**
Маркова В.П.	*Мошкина А.Д.*
(фамилия, имя, отчество)	(фамилия, имя, отчество)
к.т.н., доцент	*ФПМИ, ПМ-43*
(ученая степень, ученое звание)	(факультет, группа)
15.03.2018 г.	*15.03.2018 г.*
(подпись, дата)	(подпись, дата)

Тема утверждена приказом по НГТУ № *1686/2* от «*16*» *марта 2018* г.

ВКР сдана в ГЭК № _____, тема сверена с данными приказа

16 марта 2018 г.
(подпись секретаря экзаменационной комиссии по защите ВКР, дата)

Задорожный А.Г.
(фамилия, имя, отчество секретаря экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Работа состоит из 96 страниц, 3 частей, 25 рисунков, 1 таблицы, списка литературы из 23 источников, 10 приложений.

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ, ПОЛЬЗОВАТЕЛЬСКИЕ ИНТЕРФЕЙСЫ, GUI, ГЕНЕРАЦИЯ ИНТЕРФЕЙСОВ, УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ, УНИФИЦИРОВАННЫЙ ДОСТУП, ФОРМАЛИЗАЦИЯ ЗНАНИЙ.

Объектом исследования является повышение уровня взаимодействия пользователей с высокопроизводительными вычислительными системами (ВВС).

Цель работы – разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах.

В процессе работы изучались способы взаимодействия с высокопроизводительными вычислительными системами, а также формализация знаний о предметных областях, разрабатывался программный комплекс, позволяющий осуществлять управление вычислениями на удаленных ВВС и генерировать интерфейсы приложений по их формальному описанию.

В результате разработана архитектура системы, позволяющая пользователям унифицированным образом организовать вычисления на удаленных ВВС; разработан прототип системы, включающий в себя сервер для управления работой пользователя и запуском задач на ВВС, веб-интерфейс для взаимодействия конечных пользователей с сервером, базу данных, хранящую данные пользователя и данные, связанные с запускаемыми им задачами, модуль запуска задач на выполнение на удалённых ВВС; разработан модуль, позволяющий автоматически генерировать интерфейсы приложений по предоставленному описанию на предложенном языке.

Степень внедрения – разработанный прототип планируется применять в дальнейших исследованиях по теме работы на кафедре ПВТ НГТУ.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Многие численные задачи характеризуются большим объёмом данных и вычислений и не всякая вычислительная система за приемлемое время способна выдать результат для таких задач. Для таких крупномасштабных вычислений возникает необходимость использовать высокопроизводительные вычислительные системы (*ВВС*), следовательно, возникает задача разработки программ для таких систем и необходимость организации работы пользователя с системами такого типа. Организация пользовательского взаимодействия с ВВС включает в себя следующие основные операции: передача данных на ВВС, запуск вычислительных задач, что, как правило, означает постановку в очередь некоторой системы управления прохождением задач (СУПЗ), отслеживание статуса выполнения запущенной задачи, извлечение результатов задачи и последующий анализ.

Большинству прикладных специалистов сложно пользоваться высокопроизводительными вычислительными системами, так как существует много различных видов таких систем и для того чтобы работать с каждым таким типом вычислительной системы, необходимо предварительно изучить соответствующую документацию по конкретному типу вычислительной системы, а также способы запуска задач на данной системе и только после этого осуществлять непосредственно запуск задачи.

Также довольно трудоёмко работать с ВВС при использовании нескольких вычислительных систем для работы, как например, делают пользователи из институтов Сибирского отделения Российской академии наук (СО РАН). В своей работе они применяют вычислительные кластеры Сибирского Суперкомпьютерного Центра (ССЦ), кластеры НГУ, МГУ, Межведомственного Суперкомпьютерного Центра РАН (МСЦ РАН), вычислительные машины, установленные в отдельных институтах и так далее.

Таким образом, возникает задача унификации доступа к ресурсам высокопроизводительных вычислительных систем.

Помимо проблем, связанных с выполнением основных операций по управлению задачами на ВВС, существует проблема низкого уровня интерфейсов самих приложений. На практике не наблюдается, чтобы пользователям предоставлялись высокоуровневые интерфейсы доступа к вычислительным приложениям, работающим на высокопроизводительных вычислительных системах. Существуют пакеты, например, MATLAB, OpenFOAM, ANSYS Fluent и другие, направленные на конечного пользователя, где пользователь буквально рисует область, к примеру, сетку или пакеты, позволяющие декларативно задавать уравнения и получать их решения. Да, используя такие пакеты можно решать сложные вычислительные задачи на ВВС, однако решение можно получить только в рамках конкретного пакета, и достаточно трудоёмок будет перенос вычислительной задачи с одного на другой пакет. При этом не все задачи могут быть решены с использованием таких частных пакетов, поэтому часто пользователи ВВС разрабатывают собственные вычислительные программы и отдельной трудностью для пользователей является разработка интерфейсов к таким программах, на разработку которых зачастую не хватает ресурсов. Одной из причин существования этой проблемы является отсутствие системного инструментария для разработки высокоуровневых интерфейсов и, в частности, визуализацию для результатов приложений, выполняющихся на ВВС.

Целью данной работы является разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах. Согласно рассмотренному выше такая программная система должна предоставить пользователям унифицированный пользовательский интерфейс для основных операций управления задачами на различных ВВС, а также должен быть предоставлен инструментарий для реализации интерфейсов приложений, выполняющихся на удалённых высокопроизводительных вычислительных системах.

Подытожив всё вышеперечисленное, можно сделать вывод о том, что ***актуальность работы*** обусловлена следующими пунктами:

- сложностью доступа к высокопроизводительным системам различных типов (ВВС) и отсутствием унифицированных инструментов взаимодействия пользователей с различными ВВС;
- отсутствием удобных для использования интерфейсов к задачам, работающим удалённо на ВВС и отсутствием инструментария для разработки интерфейсов и для визуализации результатов запуска задач удалённо на ВВС.

Предлагаемый подход к решению проблемы разработки интерфейсов приложений, выполняющихся на ВВС, подразумевающий использование в совокупности унифицирующего интерфейсного сервиса и средств генерации пользовательских интерфейсов по формальным спецификациям, а также соответствующих разработанный программный инструментарий обладают ***научно-технической новизной***.

Практической значимостью обладает разработанный прототип программного комплекса, включающий в себя графическую (интерфейсную) систему, веб-сервер, базу данных, а также скрипты для запуска задач на высокопроизводительных вычислительных системах. Программный комплекс позволяет исследовать проблему запуска различных вычислительных задач на высокопроизводительных вычислительных системах, и строить на его основе инструменты для упрощения доступа пользователей к высокопроизводительным вычислительным системам.

1. ОБЗОР И ПОСТАНОВКА ЗАДАЧИ

1.1. ОБЗОР СПОСОБОВ ВЗАИМОДЕЙСТВИЯ ПОЛЬЗОВАТЕЛЕЙ И/ИЛИ ИНТЕРФЕЙСНЫХ СИСТЕМ С ВЫСОКОПРОИЗВОДИТЕЛЬНЫМИ ВЫЧИСЛИТЕЛЬНЫМИ СИСТЕМАМИ

Существует множество различных типов высокопроизводительных вычислительных систем, на которых установлены различные и по-разному настроенные СУПЗ (системы управления прохождением задач) или же, наоборот, у некоторых систем СУПЗ отсутствует. Соответственно, для каждого типа системы запуск некоторой вычислительной задачи на такой системе будет отличаться. Рассмотрим несколько примеров запусков задач на высокопроизводительных вычислительных системах.

Для запуска вычислительной программы, написанной с применением библиотеки MPI [1], на одном из многоядерных серверов Института вычислительной математики и математической геофизики Сибирского отделения Российской академии наук (ИВМиМГ СО РАН) необходимо, используя сетевой протокол ssh зайти на сервер в формате:

```
ssh <логин>@<адрес_сервера> -p <порт>,
```

после чего ввести пароль. Далее необходимо выбрать каталог, в котором хранится задача, требующая вычисления или создать новую задачу, после чего предварительно собранный исполняемый файл *task.comp* можно запустить, используя команду *mpirun* вместе с необходимым набором параметров, каковыми являются параметр указания количества потоков (*-np*), максимальное время счёта (*-maxtime*) и другие, например,

```
mpirun -np 4 -maxtime 20 ./task.comp.
```

Для запуска вычислительной задачи на кластере НГУ [2], необходимо, используя язык системы Altair PBS Pro (система управления прохождением задач в ВВС [3, 4]) написать скрипт *script.sh*, определяющий параметры задачи, см. Приложение А. После чего, используя команду *qsub* поставить написанный ранее скрипт в очередь:

```
qsub script.sh,
```

которая после проверки скрипта и помещения его в очередь выводит на экран строку вида

```
123456.hpc-suvir1.hpc,
```

где «123456» – число, являющееся уникальным идентификатором задачи.

После завершения работы задачи, в текущей директории появляются файлы *submit.sh.e123456* и *submit.sh.o123456*, в которые перенаправляются стандартный поток ошибок и стандартный поток вывода соответственно, запускаемых в рамках задачи процессов параллельной программы.

Запустить вычислительную задачу удалённо на ВВС можно также с помощью MATLAB [5, 6, 7]. К примеру, имеется программа, написанная на MATLAB и находящаяся на локальном компьютере и некоторый кластер, на котором есть возможность запуска данной программы (установлен MATLAB). Тогда, после ввода пользователем логина и пароля к ВВС в локально установленном MATLAB, при запуске задачи локальный MATLAB обращается с помощью протокола *ssh* на кластер и запускает программу на этой ВВС, посредством её СУПЗ, например PBS Pro.

Рассмотрим ещё несколько примеров запуска некоторой вычислительной задачи на удалённой высокопроизводительной системе.

К примеру, запуск задач на суперкомпьютере МВС-10П Межведомственного Суперкомпьютерного Центра Российской Академии Наук (МСЦ РАН) [8] можно осуществлять несколькими способами, в зависимости от типа поддерживаемых СУПЗ. Данный суперкомпьютер поддерживает 2 типа СУПЗ:

СУППЗ (система управления прохождением параллельных заданий) и планировщик SLURM. Рассмотрим оба варианта запуска задачи на ВВС.

Для запуска задачи с использованием СУППЗ необходимо предварительно загрузить модуль СУППЗ:

```
module load launcher/suppz
```

и после выполнить ранее упоминаемую команду *mpirun*, например:

```
mpirun -np 4 ./task.comp.
```

Данная команда автоматически формирует паспорт задания и направляет его в очередь СУППЗ. Просмотр очереди СУППЗ осуществляется командой

```
mqinfo [-s system],
```

где system – имя логической системы.

Запуск задачи с использованием планировщика SLURM можно выполнить в пакетном или интерактивном режимах.

В *пакетном* режиме пользователь создает скрипт для запуска задачи и использует утилиту *sbatch*. Задача отправляется в очередь и будет выполнена как только будут доступны запрошенные ресурсы. Вывод результата выполнения задачи будет записан в файл *slurm-<номер задачи в очереди>.out* в директории, откуда осуществлялся запуск задачи. В качестве параметра утилите sbatch передаётся исполняемый скрипт, который будет запущен на первом из выделенных вычислительных узлов. Внутри скрипта осуществляется запуск задачи с помощью утилит *srun* или *mpiexec.hydra*. Утилита sbatch позволяет задавать необходимые ей опции внутри скрипта запуска. Формат задания опций:

```
#SBATCH <опция>
```

Для запуска с помощью утилиты *srun* сперва должен быть загружен модуль *launcher/slurm*. Скрипт для пакетного запуска X экземпляров MPI-программы task.comp на каждом из Y вычислительных узлов приведён в приложении Б.

Для запуска с помощью утилиты *mpiexec.hydra* сперва должен быть загружен модуль *launcher/intel*. Скрипт для пакетного запуска X экземпляров MPI-программы task.comp на каждом из Y вычислительных узлов приведён в приложении В.

В *интерактивном* режиме пользователь либо использует для запуска задачи утилиту srun, либо самостоятельно выделяет необходимое количество ВУ (вычислительных узлов) и запускает задачу с помощью утилиты mpiexec.hydra или srun. Вывод результата выполнения задачи будет произведен в консоль. Пример запуска задачи task.comp с помощью утилиты srun (предварительно необходимо загрузить модуль launcher/slurm, если не загружен):

```
srun -N Y --ntasks-per-node=X -n X*Y ./task.comp,
```

где -N – количество необходимых узлов, --ntasks-per-node – количество процессов запускаемых на каждом вычислительном узле.

Пример самостоятельного выделения ВУ и запуска задачи task.comp с помощью утилиты mpiexec.hydra (предварительно необходимо загрузить модуль launcher/intel, если не загружен):

Выделение Y вычислительных узлов с помощью команды *salloc*:

```
salloc -N Y,
```

где -N – количество необходимых узлов. При наличии ресурсов будет выведено сообщение об успешном выделении вычислительных узлов, после которого необходимо использовать команду mpiexec.hydra:

```
mpiexec.hydra -perhost X -n X*Y ./hello_mpi_new,
```

где -perhost <n> – запуск n процессов на каждом узле, -n – общее количество процессов. Далее задача будет запущена, а результат выполнения будет выведен на экран.

Все рассмотренные выше способы взаимодействия с вычислительными ресурсами могут применяться к виртуальным вычислительным ресурсам, создаваемым с помощью облачных сервисов, таких как Amazon Web Services (AWS) или Microsoft Azure. Однако это означает лишь то, что к проблемам, описанным

выше, добавляется ещё проблема организации управления виртуальными машинами.

Исходя из анализа приведенных выше систем организации вычислений на ВВС, можно сделать вывод о том, что проблема запуска прикладной задачи на ВВС решается, но решения этой проблемы зависят от типа ВВС и чаще всего требует изучения документации к выбранной ВВС. Однако некоторого универсального подхода, позволяющего запустить прикладную задачу на любой высокопроизводительной вычислительной системе не существует. Отсюда вытекает необходимость в промежуточном программном обеспечении (ПО), которое позволит пользователям запускать прикладные задачи на различных вычислительных системах и при этом данное ПО должно быть модульным для расширения числа высокопроизводительных вычислительных систем, на которых при использовании данного ПО есть возможность запускать различные вычислительные задачи. На основе анализа инструментария для управления задачами на различных ВВС, формируется требование к промежуточному ПО обеспечить унифицированный интерфейс для реализации следующих операций работы с ВВС:

- операция передачи данных на ВВС;
- операция запуска вычислительных задач;
- операция постановки в очередь некоторой системы управления прохождением задач (СУПЗ);
- операция отслеживания статуса выполнения запущенной задачи;
- операция извлечения результатов задачи;
- операция последующего анализа данных задачи.

В дальнейшем, операции из этого списка называются ***базовыми операциями*** работы с ВВС.

1.2. ФОРМАЛЬНОЕ ПРЕДСТАВЛЕНИЕ ЗНАНИЙ

Выделяют различные парадигмы программирования, рассмотрим некоторые из них:

- императивное программирование – парадигма программирования, которая задаёт процесс исполнения некоторого математического алгоритма в виде инструкций вычислительной машины, при этом переменные алгоритма отображаются в ресурсные переменные – то есть в ячейки памяти и т.д.
- декларативное программирование – парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается проблема и ожидаемый результат, противоположно императивному программированию не указываются шаги достижения результата.

Для вычислительных задач разрабатываются декларативные языки, например язык Норма [9, 10] в которых прикладному математику даётся возможность сформулировать свою задачу, например, записать некоторый численный метод решения задачи математической физики в привычных для него терминах, наподобие описания методов в различных учебниках и пособиях [11].

Если подобный метод из некоторого научного труда запрограммировать, используя императивный подход на каком-либо языке программирования, например на языке C++, то впоследствии, имея только запрограммированный алгоритм вычисления, невозможно определить автоматически, значения каких переменных алгоритма хранятся в тех или иных ячейках памяти вычислительной системы, в тех или иных позициях в выходных файлах программы [12].

Поэтому, если ставится цель автоматизировать работу математика-прикладника с высокопроизводительной вычислительной системой, необходимо предоставить для него возможность описывать алгоритмы в каких-либо более высокоуровневых терминах, нежели термины, используемые при императивном подходе программирования. В идеале, должна предоставляться воз-

можность описания предметной области на математическом языке, синонимичная описаниям в научных работах.

Идея заключается в следующем: знания о предметной области вычислительной задачи необходимо выразить в формальном языке. В качестве примера можно использовать язык вычислительных моделей [11] или онтологий [13].

В таком случае, если имеется достаточно полное формальное описание предметной области, связанной с задачей, это позволяет на высоком уровне ставить спецификацию вычислительной задачи и получать автоматически сгенерированные алгоритмы, а в идеале – программы, которые выполняют эти алгоритмы и позволяют решать, таким образом, задачи, специфицированные на высоком уровне. Например, если есть формальное представление некоторого вычислительного метода, то его можно автоматически анализировать и на его основе сгенерировать программу на каком-либо императивном языке программирования, тем самым избавляя математика-прикладника от написания таковой.

Тем не менее, данный подход весьма трудно реализуется на практике, так как не всегда возможно достаточно полно описать предметную область задачи, а для нетривиальных предметных областей такое полное описание неосуществимо. Генерация программы по математическому алгоритму подразумевает необходимость решать задачу отображения операций и переменных некоторого математического алгоритма на ресурсы распределённой вычислительной системы, которая является NP-полной. Поэтому решения для таких задач можно находить либо в виде эвристик, либо в виде оптимальных решений для отдельных классов задач, что составляет отдельную научную проблему [11].

Эти вопросы до сих пор открыты, и эффективное решение пока не найдено. Тем не менее, исследования могут продвигаться на основе ограничения проблематики некоторыми частными предметными областями.

Помимо автоматической генерации программы по полному описанию предметной области некоторой задачи и спецификации задачи, есть возможность автоматической визуализации различных характеристик задачи.

Рассмотрим следующую ситуацию: пусть имеется некоторая серия массивов, тогда её можно визуализировать, если в формальном языке, в котором описана предметная область текущей задачи, будет предоставлена пользователю возможность сказать, что это серия массивов. Тогда к системе всегда можно будет добавить модуль, который проанализировав описание предметной области и найдя в ней описание понятия серии массивов, после окончания работы пользовательской программы и формирования этой серии массивов предоставит нам визуализацию этой серии. Сама по себе визуализация серии массивов - задача известная, например, для серии двумерных массивов может быть сгенерировано видео, где каждый кадры соответствует массивам серии, а цвет пикселей определяется значениями элементов массивов. Или, имея значения элементов массива, среди них можно отыскать максимальный и минимальный элементы, и установив некоторую цветовую градацию в соответствии с найденными элементами сгенерировать изображение, на котором будут отображаться значения в виде графика или некоторой поверхности.

Однако если программа написана на некотором императивном языке программирования, например на языке C++, то не всегда известно что хранимые в памяти данные это именно необходимая нам серия массивов. Поэтому встаёт вопрос о том, чтобы позволить пользователю ввести некоторые понятия, то есть предложить такой декларативный язык, в котором пользователь вводит понятия, необходимые ему, и описывает их или использовать, например, язык OWL для описания онтологий [13]. Таким образом, если имеется формальное описание некоторого понятия, то можно автоматически сгенерировать представление этого понятия (например, проинициализированную соответствующую структуру данных или визуализацию), а также дополнить некоторыми операциями работы с данным понятием, при необходимости.

Так как сама по себе разработка такого языка для описания предметной области некоторой вычислительной задачи на языке близком к математическому языку довольно сложная задача, то для простоты на начальном этапе вместо описания самого алгоритма, в рамках ВКР принято решение ограничиться опи-

санием того, что имеет значение с внешней точки зрения относительно алгоритма – то, что подаётся ему на вход и то, что получается на выходе после прохождения данного алгоритма.

В рамках выполнения ВКР, в соответствии актуальностью класса численных задач, было произведено сужение класса предметных областей до класса вычислительных задач, которые моделируют некоторое явление в заданной для явления области моделирования.

Помимо визуализации выходных параметров, рассмотренной выше, разрабатываемый язык должен включать в себя возможность описания области моделирования с последующей визуализацией. То есть пользователь на вход алгоритму может подать геометрию предметной области, в которой будет происходить моделирование, а система по предоставленной геометрии сама сформулирует параметры, требуемые вычислительному алгоритму на вход.

1.3. ПОСТАНОВКА ЗАДАЧИ

В заключение обзора можно сделать вывод, что поставленные проблемы унификации доступа к ВВС и создания некоторого декларативного языка для описания предметных областей задач и последующей генерации алгоритмов и программ выполнения алгоритмов на основе описания и формальных спецификаций задач решаются, но для очень узких областей и конкретных систем. Однако же, если начать рассматривать данные проблемы шире и в совокупности, то универсального решения такой общей проблемы пока не существует. В рамках выпускной квалификационной работы ставится задача практически исследовать возможность решения этой проблемы на основе создания программной системы, в соответствии со следующими требованиями:

- разрабатываемая система должна обеспечивать реализацию базовых операций работы с ВВС, введённых в первом разделе обзора;
- в ходе работы необходимо:

- предусмотреть возможность разбиения пользователей на группы с различными требованиями,
- предусмотреть возможность расширения системы для других классов пользователей;

- разрабатываемая система должна быть проста в использовании: под этим в рамках настоящей работы подразумевается, что для каждой операции из списка базовых операций работы с ВВС предусматривается отдельное окно или область пользовательского интерфейса, при этом на каждом окне:
  - доступны все элементы управления, необходимые для осуществления операции,
  - отсутствуют элементы управления, не имеющие отношения к данной операции;
- система должна иметь декларативный язык, позволяющий по формальной спецификации входных и выходных параметров генерировать интерфейсы для задания параметров задачи и для анализа, в том числе, посредством визуализации результатов расчётов;
- система должна обладать открытой архитектурой, позволяющей расширять список возможных вычислительных задач для запуска на ВВС, а также самих ВВС, на которых будет осуществляться запуск непосредственно;
- разрабатываемая система должна сохранять всю получаемую информацию от пользователей и вычислительных задач, а также иметь возможность отобразить её пользователю при соответствующем запросе.

2. ПРОЕКТ

2.1. СТРУКТУРНЫЕ КОМПОНЕНТЫ ПРОГРАММНОГО КОМПЛЕКСА

Рассмотрим следующую ситуацию: пусть имеется необходимость того, чтобы пользователи или некоторые программные системы имели возможность такие операции, как постановка задачи на выполнение на ВВС, отслеживание их состояния и получения результатов выполнения, независимо от особенностей различных ВВС. Тогда необходим некоторый сервис, позволяющий выполнять базовые операции, определённые в обзоре. Такой сервис будет принимать указания (команды) от пользователя через некоторый пользовательский интерфейс и осуществлять передачу этих указаний определённым ВВС в соответствии с особенностями интерфейсов конкретных ВВС. Кроме того, необходимо отделить всю функциональность по управлению вычислениями от конкретных реализаций пользовательского интерфейса, например, веб-, мобильных или десктопных интерфейсов. Рассматриваемая функциональность по управлению вычислениями также должна быть централизованной, то есть предоставлять возможность многопользовательского обслуживания как единичных пользователей, так и некоторых групп, сформированных пользователями. Отсюда вытекает идея реализовать функциональность по управлению вычислениями такого типа в рамках данной выпускной квалификационной работы в рамках отдельного ***сервера***.

Итак, ***сервер*** – это программная система, позволяющая запускать вычислительные задачи на удалённых высокопроизводительных вычислительных системах пользователям и/или другим программным системам, а также отслеживать состояние запущенных задач и имеющая возможность отправлять результат вычисления в удобном для запрашивающего виде. Для дальнейших рассуж-

дений введём ещё несколько определений, связанных с сервером и постановкой задач на выполнение.

Приложение – некоторая вычислительная программа, для которой известны параметры, требуемые ей на вход, а также которую можно запустить на одной из доступных ВВС, явно указав входные параметры программы и параметры запуска на конкретной ВВС. Разрабатываемая система должна обладать возможностью накопления различных вычислительных приложений.

Задача – конкретное приложение с заданными пользователем конкретными параметрами и входными данными, учитываемое сервером от момента указания пользователя о запуске этого приложения на одной из доступных ВВС до удаления сведений о такой задаче из учетных записей сервера по указанию пользователя. Задача проходит состояния: поставлена в очередь на исполнение, исполняется, завершена.

Пользователи – агенты, от имени которых сервером осуществляются операции по организации вычислений. В качестве агентов могут выступать программные системы и/или люди.

Для запуска вычислительной задачи на удалённой ВВС также необходимы данные, такие как параметры текущей задачи, параметры запуска для задачи, файлы, требуемые на вход задаче и так далее. Следовательно, для каждого отдельного запуска такие данные будут различны, а значит, появляется необходимость сохранять эти данные, например, в так называемой *файловой системе*. В файловой системе помимо данных для запуска можно хранить и сами приложения, а также доступные для вычислений на них ВВС.

Файловая система – хранилище всех данных, поступающих от пользователей или приложений.

Так как пользователей, желающих запустить ту или иную задачу может быть несколько, то необходимо сделать так, чтобы каждый отдельный пользователь не конфликтовал с другими. Значит, необходимо каким-то образом разделять пользователей, тогда, при запуске одной и той же задачи на удалённой ВВС будет учитываться то, каким именно пользователем она была запущена.

Помимо прочего, каждому пользователю должен быть доступен персональный список приложений, располагающих возможностью к запуску, список ВВС, на которых этот запуск можно осуществить и так далее, то есть возникает необходимость хранения таких списков. Общий список пользователей также нужно сохранять, для установления личности, например с помощью логина и пароля, отдельного пользователя при использовании клиентской части. Следовательно, необходима база данных, в которой будут храниться эти данные и при обращении к которой будет доступна та или иная информация.

Как было сказано выше, запуск задач может осуществляться на одной из доступных ВВС, кроме того, исходя из обзора способов взаимодействия пользователей с ВВС, можно сделать вывод, что запуск задач на различных ВВС осуществляется в зависимости от особенностей таких систем. Следовательно, необходим модуль, который в дальнейшем называется ***модулем запуска***, содержащий инструкции для запуска прикладных задач на конкретной ВВС, и таких модулей должно быть несколько, под каждый тип ВВС, доступный пользователю.

Таким образом, разрабатываемый программный комплекс должен состоять из следующих частей:

- сервер, отвечающий за выполнение всех запросов, поступающих от пользователя, таких как постановка задачи на выполнение и так далее, и взаимодействующая с базой данных;
- база данных, хранящая в себе всю информацию о пользователях и запускаемых задачах;
- клиентская часть, с помощью которой осуществляется работа пользователя и отправка команд серверу;
- модули запуска прикладных задач для вычислительных систем.

Схематически взаимодействие всех необходимых частей программного комплекса представлено на рисунке (рисунок 1; примечание: здесь и далее в разделе используется нотация UML [14]):

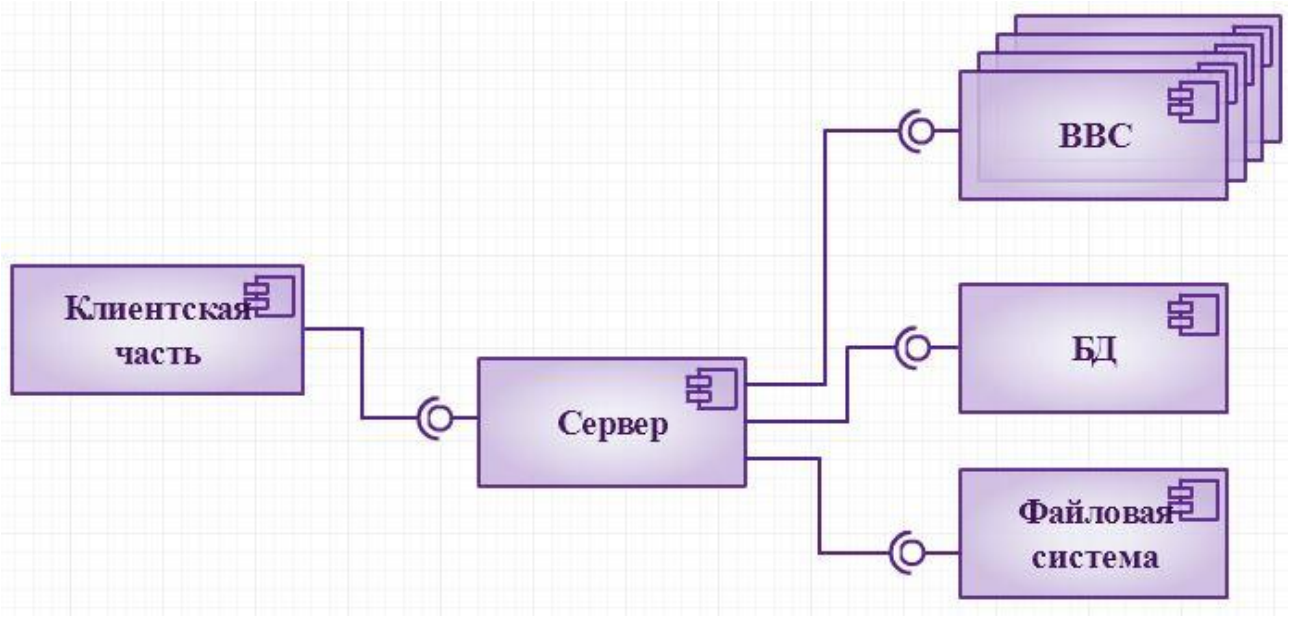


Рисунок 1 – Модель взаимодействия систем в программном комплексе

Исходя из вышеизложенного, разрабатываемый сервер должен иметь следующую функциональность:

- сервер должен позволять запустить задачу на ВВС, что подразумевает под собой постановку задачи на выполнение на ВВС, отслеживание состояния задачи и получение результата выполнения;
- сервер должен уметь отображать на клиентский интерфейс информацию о структуре интерфейсов приложений, которые можно запустить на ВВС;
- сервер должен поддерживать аутентификацию пользователей, а также авторизацию;
- должен иметь возможность добавления нового пользователя в общий список пользователей, например посредством регистрации в системе;
- сервер должен уметь формировать и отправлять список доступных пользователю приложений или высокопроизводительных вычислительных систем, а также информацию о конкретной задаче;
- должен позволять добавлять новое приложение в список приложений, доступных пользователю.

Рассмотрим функциональность сервера более подробно.

2.2. ОПИСАНИЕ СЕРВЕРНОЙ ЧАСТИ СИСТЕМЫ

2.2.1. Основная функциональность сервера

Для того чтобы запустить вычислительную задачу на высокопроизводительной вычислительной системе (см. Приложение Г) пользователю необходимо сперва выбрать требуемую задачу из списка доступных ему приложений, после чего установить все параметры, требуемые на вход данному приложению, либо прикрепить требуемые файлы. Также необходимо выбрать ВВС, на которой будет осуществлён запуск и установить параметры, такие как количество ядер, на которых будет запущена задача, и другие, доступные для данной ВВС.

Далее, все эти данные необходимо отправить вместе с соответствующим запросом на сервер, который, обработав их, добавит текущий запуск как новую задачу в базу данных и вызовет модуль запуска, соответствующий выбранной ВВС, отправляя, таким образом, задачу на выполнение.

Данная задача будет считаться успешно завершённой, если после выполнения её на ВВС сформировался файл, сигнализирующий об окончании вычислений и вместе с результатами переданный на обработку серверу. Сервер же, после того как получил файл с результатом и файл-признак завершения работы, уведомляет пользователя о завершении выполнения поставленной им задачи и отправляет клиенту результаты вычислений для последующей визуализации результатов, при запросе таковых пользователем.

Помимо постановки задачи на выполнение, сервер должен разделять пользователей и отличать запуски задач одного пользователя от другого. Например, это можно сделать с помощью авторизации пользователей в системе с помощью уникального сочетания данных – логина и пароля (см. Приложение Д). Изначально пользователю необходимо сохранить свои данные в базе данных, с помощью так называемой «регистрации», после чего ввести свои данные для проверки их на уникальность. Сервер, обработав полученные данные, обратит-

ся к базе данных и запросит информацию о пользователе с такими данными. Если такой пользователь существует, то на основании логина и пароля сервером будет возвращена и сохранена в системе зашифрованная строка – токен, позволяющая пользователю дальнейший доступ к функциональности системы и избавляющая его от необходимости в многократном вводе своих личных данных для доступа к какой-либо операции в системе. Данная мера позволяет обезопасить работу пользователей внутри системы.

Под задачей мы понимаем некоторое приложение с определёнными параметрами, запущенное на конкретной ВВС. Следовательно, сервер должен уметь работать с различными приложениями, что предполагает необходимость добавления новых приложений в систему (Приложение Е).

Для добавления нового приложения в систему, пользователь должен послать соответствующий запрос серверу, содержащий необходимую информацию, например, имя добавляемого приложения, описанные параметры, требуемые данному приложению, и прочее. Необходимым параметром для успешного добавления нового приложения в систему является путь до реализации этого приложения, включающий имя ВВС и путь в файловой системе ВВС. В рамках настоящей работы подразумевается, что данный путь хранит исполняемый файл добавляемого приложения. Другие варианты реализации приложения могут быть исследованы в дальнейшем. Если предполагается, что приложение должно иметь графический интерфейс, в качестве одного из параметров передается спецификация интерфейса (см. раздел 2.4). Сервер, обработав введённые данные, попытается добавить их в базу данных как новое приложение и в случае успеха вернёт идентификатор добавленного приложения или ошибку, в противном случае.

Пользователь должен иметь возможность просматривать доступные ему приложения, ВВС, а также видеть предыдущие запуски этих приложений. Следовательно, сервер должен уметь возвращать пользователю запрашиваемые данные в виде некоторого списка необходимых данных (Приложение Ж).

2.2.2. REST-ресурсы, которыми управляет сервер

Исходя из рассуждений в предыдущем разделе, можно сделать вывод о том, что под управлением сервера должны находиться следующие объекты: приложения, учетные записи пользователей, ВВС, задачи и объекты авторизации (*токены*). При создании программного интерфейса сервера, оперирующего такими объектами, целесообразно выбрать архитектурный стиль REST – структурированный подход к проектированию API (сокр. от application programming interface – интерфейс программирования приложений) [15], так как объекты будут прямо соответствовать понятиям REST-ресурсов.

Суть этого подхода состоит в том, ресурсы (любые объекты), которыми управляет сервер, имеют идентификаторы, которыми на практике часто выступают URL (сокр. от uniform resource locator – единообразный локатор ресурса), по которым соответственно отправляются запросы. Смысл необходимого действия над ресурсами при таком смысле URL будет передаваться через типы запросов. В основном используют 4 типа запросов:

- GET - запрос ресурса (объекта) в некотором представлении, идентифицируемого указанным URL;
- POST - создание ресурса в рамках коллекции ресурсов, идентифицируемой указанным URL;
- PUT - модификация ресурса, идентифицируемого указанным URL;
- DELETE - удаление указанного ресурса.

Преимуществом разработки API в стиле REST является то, что интерфейс формируется в рамках некоторой концепции, предполагающей ресурсы и действия, таким образом, на выходе мы имеем единообразную структуру всех запросов к серверу, что упрощает его разработку и работу прикладных программистов, которые его используют.

Для описания серверного API разрабатываемой системы необходимо ввести понятия REST-ресурсов разрабатываемого API.

Первое множество ресурсов – ресурсы типа ***«token»***. Данный ресурс связан с авторизацией пользователя в системе. Вся коллекция токенов называется *tokens*. Данному типу ресурсов соответствует URL */tokens*. По URL такого вида можно сделать запрос POST, означающий запрос на аутентификацию пользователя в системе и создание токена. Параметрами к такому запросу являются логин и пароль, введённые пользователем. Результатом запроса является отправка токена пользователю, в случае успешной аутентификации, или сообщение об ошибке. Также по URL вида */tokens/*<*token_value*>, где <token_value> – токен, ранее полученный сервером от пользователя можно сделать запрос DELETE, позволяющий удалить токен пользователя.

Все остальные запросы к системе, исключая запрос на создание пользователя, требуют передачи токена в качестве параметра запроса.

Следующий ресурс – ресурс ***«users»***. Данный ресурс определяет коллекцию ресурсов типа *user*, которые отвечают за представление всей необходимой информации о пользователе системы. Данному ресурсу соответствует URL */users* и позволяет сделать запрос POST, означающий регистрацию нового пользователя в системе. Необходимыми параметрами такого запроса являются фамилия, имя, логин, пароль и электронный адрес пользователя. На параметр логин накладывается требование уникальности значения. Необязательными параметрами являются отчество, дата рождения и пол пользователя. Результатом успешного выполнения такого запроса является возврат идентификатора добавленного пользователя в систему, а также создание его персонального каталога в файловой системе, в противном случае будет возвращена ошибка.

Ресурс ***«applications»*** отвечает за список приложений, доступных пользователю. Данному ресурсу соответствует URL */applications* и позволяет сделать запрос POST, означающий добавление нового приложения в систему. Необходимыми параметрами такого запроса являются имя, спецификация входных и выходных параметров для генерации интерфейса приложения и путь до реали-

зации нового приложения. Необязательным параметром являются описание приложения. Результатом успешного выполнения такого запроса является возврат идентификатора добавленного приложения в систему, в противном случае будет возвращена ошибка. Также, URL позволяет сделать запрос GET, означающий вывод списка всех приложений, доступных пользователю. Запрос не требует параметров и возвращает названия доступных приложений. Запрос DELETE, параметром которого является идентификатор приложения, позволяет удалить выбранное приложение.

Ресурс ***«systems»*** определяет коллекцию ресурсов типа *system*, которые отвечают за представление информации о высокопроизводительных вычислительных системах. Данному ресурсу соответствует URL */systems*. Поддерживает запросы типа GET и DELETE, по своим возможностям аналогичные запросам предыдущего ресурса.

И, наконец, ресурс ***«jobs»***, определяющий коллекцию ресурсов типа *job*, которые отвечают за представление информации о задачах (приложение с конкретными параметрами). Данному ресурсу соответствует URL */jobs*, поддерживающий запросы типа GET, аналогичные предыдущим ресурсам и POST, позволяющий отправить некоторую задачу для вычисления на ВВС. Параметрами запроса POST являются имя запускаемой задачи и параметры, требуемые задаче для корректного запуска, а также идентификатор приложения, устанавливающийся в зависимости от принадлежности задачи некоторому приложению. Результатом успешного выполнения такого запроса является возврат идентификатора запускаемой задачи на ВВС, в противном случае будет возвращена ошибка.

2.3. ОПИСАНИЕ ИНТЕРФЕЙСНОЙ ЧАСТИ СИСТЕМЫ

Для работы как программных систем с сервером, так и конечных пользователей, необходимо создать пользовательскую интерфейсную систему, которая будет обращаться к серверу по его API для отработки указаний пользователя. Предлагается в проекте реализовать интерфейсную систему в виде веб-приложения таким образом, что весь интерфейс доступен пользователю в браузере. Преимуществами такого подхода, в отличие от специальных десктопных приложений, является отсутствие необходимости установки программного обеспечения на компьютер пользователя, а также доступ к системе практически из любого места, где есть доступ в Internet и браузер. Данный раздел посвящён проектированию пользовательского интерфейса, называемого ***WebGUI***.

2.3.1. Описание основных интерфейсных окон WebGUI

Изначально пользователь, перейдя по специальному URL, попадает на начальную страницу (рисунок 2), где ему предлагается ввести логин и пароль для доступа к своему *«личному кабинету»*, после чего, нажав кнопку «Войти» оказаться в нём.

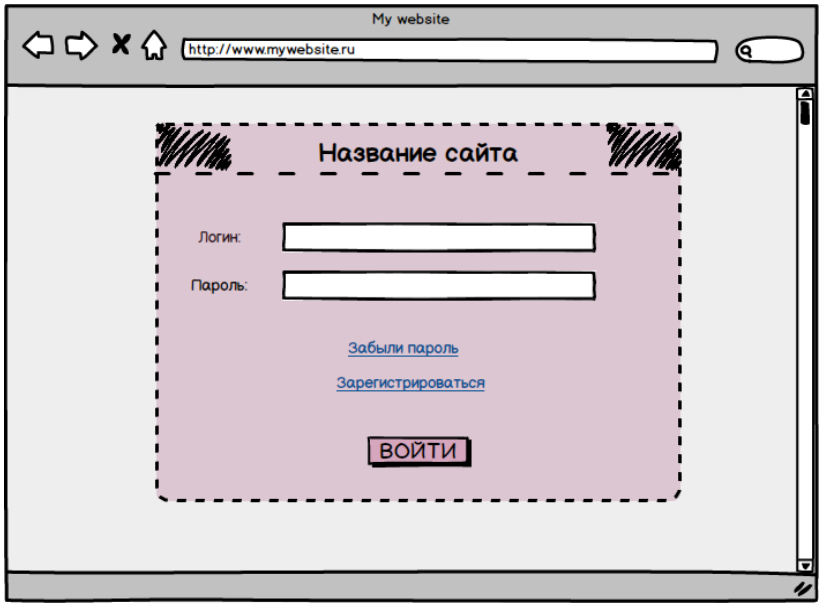


Рисунок 2 – Окно входа в систему

Личный кабинет – основная страница взаимодействия пользователя с программной системой, позволяющая ему использовать функциональность системы для запуска вычислительных задач на удалённых ВВС.

Если же пользователь ещё не создал свой личный кабинет, то ему предлагается это сделать, перейдя на вкладку «Зарегистрироваться» (рисунок 3) или же перейти на вкладку «Забыли пароль», в случае потери пользователем своих личных данных.

При переходе в окно регистрации (рисунок 3), пользователю необходимо в обязательные поля, отмеченные красным цветом, ввести свои данные, необязательные поля заполняются по желанию, и нажать кнопку «Зарегистрироваться» для создания собственного профиля и получения доступа к личному кабинету и дальнейшей работе с данным веб-сервером. В случае успешной регистрации, пользователь попадает на вкладку с соответствующим сообщением (рисунок 4), после чего ему предлагается вернуться в начало (рисунок 2) и ввести свои данные для входа в личный кабинет (рисунок 5).

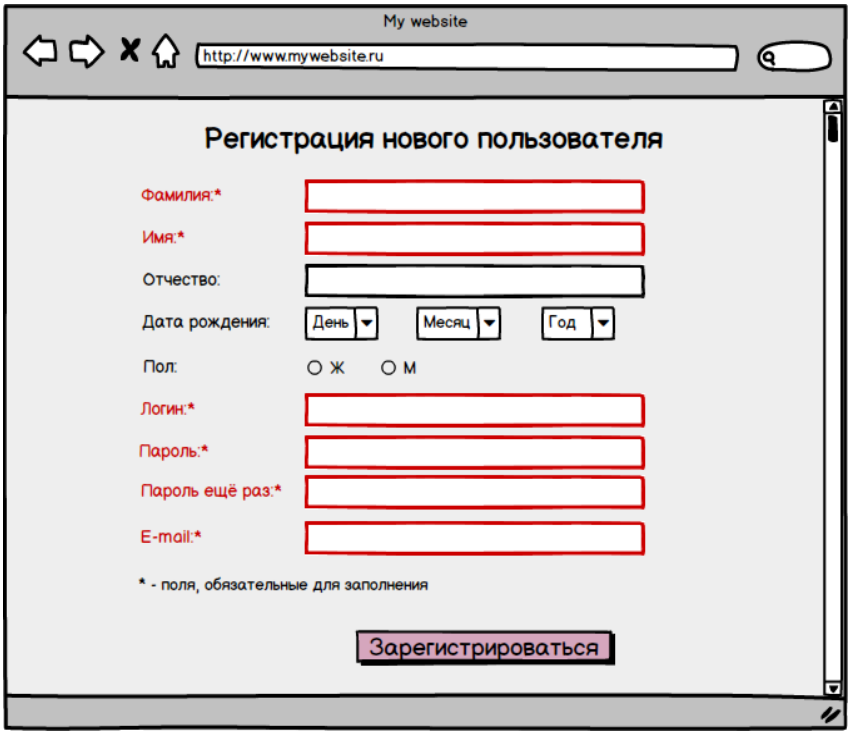


Рисунок 3 – Окно регистрации нового пользователя

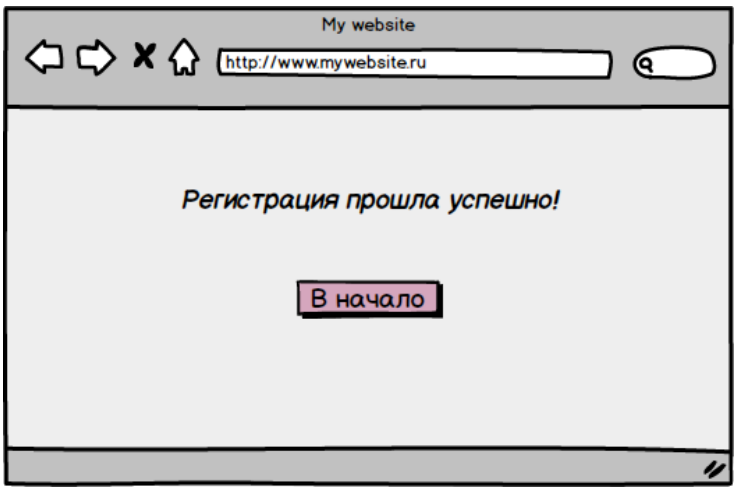


Рисунок 4 – Окно завершения регистрации

После выполнения одной из вышеперечисленных процедур, пользователь оказывается в личном кабинете (рисунок 5), который функционально разделён на несколько областей, с каждой из которых можно взаимодействовать тем или иным способом.

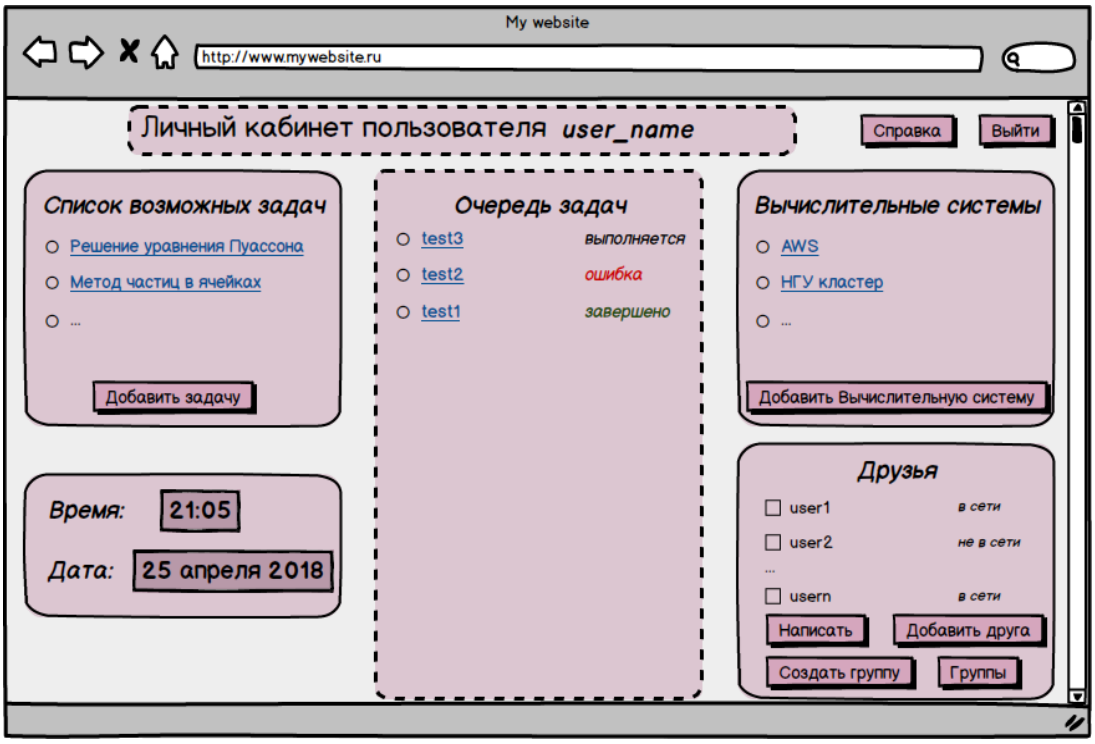


Рисунок 5 – Окно основной страницы – личного кабинета

Сверху расположены кнопки «Справка», для получения справочной информации о возможном функционале личного кабинета, «Выход», для выхода из личного кабинета (рисунок 5) и возвращения к стартовой странице, а также

поле, информирующее нас о том, что пользователь находится в своём личном кабинете и соответствующий данному пользователю логин.

В левой части страницы расположены элементы управления списком имеющихся у пользователя задач. Функционал этой области позволяет просматривать описания каждой задачи, отображать список последних запусков и результатов запусков, а также добавлять новые задачи в личный кабинет (рисунок 5) пользователя с веб-сервера или самостоятельно загружать задачи. Чуть ниже расположена область, информирующая пользователя о текущей дате и времени.

В правой части личного кабинета (рисунок 5) располагается область работы с вычислительными системами, доступными пользователю, которая имеет возможности просмотра основной информации о той или иной вычислительной системе, а также добавления новой вычислительной системы в личный кабинет пользователя для дальнейшего использования. Также в этой части пользовательской страницы расположена область, отображающая друзей пользователя. В этой области доступны такие функции, как добавка нового друга (коллеги), обмен сообщениями с уже имеющимися друзьями, формирование групп пользователей и просмотр сформированных групп с целью «расшаривания» задач, имеющихся у данного пользователя.

В центре личного кабинета (рисунок 5) расположена так называемая очередь задач. Здесь отображаются все задачи, запущенные пользователем во время текущего сеанса работы, а также статус выполнения каждой задачи. Статус может принимать одно из трёх доступных значений:

- *задача выполняется* – данный статус лишь информирует о выполнении некоторой задачи в данный промежуток времени на некоторой выбранной пользователем вычислительной системе, для просмотра результата запуска задачи, необходимо дождаться статуса «завершена»;
- *поставлена в очередь* – ожидается освобождение необходимых ресурсов для выполнения задачи на ВВС, имеющей СУПЗ;
- *задача завершена* – вычисления завершены, и пользователю доступен просмотр результатов или ошибки в ходе выполнения задачи.

Клиент запрашивает у сервера статус задачи с некоторой периодичностью (Приложение 3). В случае, когда статус задачи «завершена», пользователь, кликнув по имени задачи, сможет увидеть результат решения данной задачи на определённой им заранее вычислительной системе с определёнными параметрами запуска, а также увидеть графическое отображение решения. В случае, когда задача завершилась с ошибкой, кликнув по имени задачи, пользователь увидит все параметры, которые были использованы для запуска данной задачи, а также текстовую строку с ошибкой.

Рассмотрим все функциональные области подробнее, начиная с области управления вычислительными системами.

2.3.2. Описание области управления вычислительными системами

При клике на любую из доступных пользователю систем, он попадает на вкладку информации о данной вычислительной системе (рисунок 6), где характеристики данной системы расписаны более подробно или предоставлена ссылка на официальную страницу ресурса, если такая имеется, с актуальными для него характеристиками. Данная информация позволит пользователю оценить возможности каждой из доступных вычислительных систем, чтобы выбрать подходящую для запуска систему, для некоторой задачи, которую необходимо выполнить.

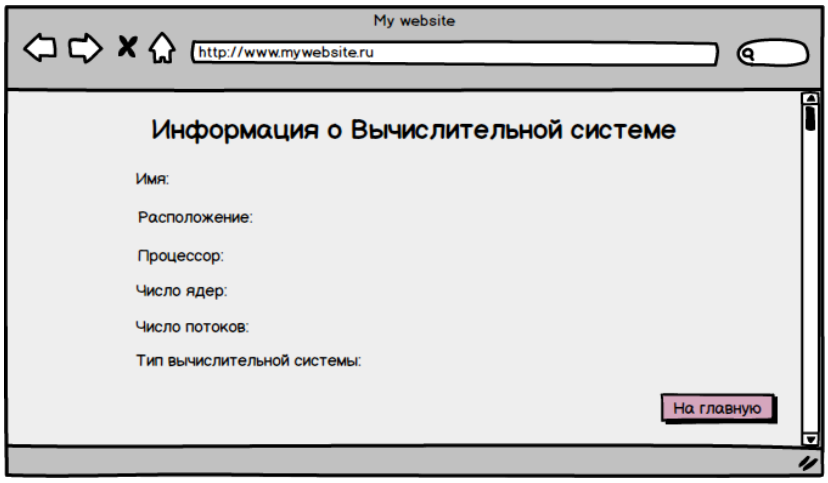


Рисунок 6 – Окно информации о выбранной вычислительной системе

Если в области управления вычислительными системами кликнуть на кнопку «Добавить ВС», то открывается вкладка, позволяющая добавить необходимую систему в свой личный кабинет (рисунок 7).

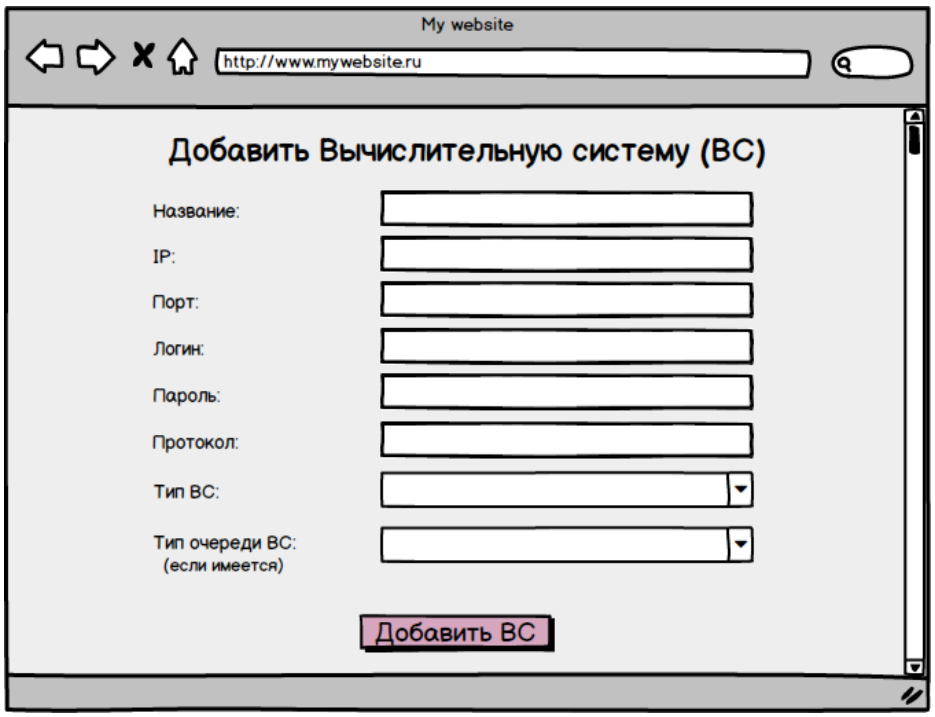


Рисунок 7 – Окно добавления новой вычислительной системы

В данном окне, пользователю необходимо ввести параметры добавляемой вычислительной системы, которые в дальнейшем будут использоваться сервером для подключения к ней и отправки на неё задач. В зависимости от типа системы, на веб-сервере, будет вызываться тот или иной модуль запуска задач, если выбранный тип вычислительной системы присутствует в таблице типов, которые поддерживает наш сервер. Помимо этого, существуют вычислительные системы с разными типами очередей, в зависимости от которых модуль запуска будет иметь другое строение, по сравнению с системами без очередей, поэтому, если очередь в системе имеется, то необходимо указать её тип для корректной отправки задачи на кластер с помощью модуля запуска.

2.3.3. Описание области управления пользовательскими задачами

Теперь рассмотрим область управления задачами, доступными пользователю. При клике на одну из доступных пользователю задач, осуществляется переход на вкладку, содержащую информацию о данной задаче (рисунок 8). На данной вкладке выводится имя выбранной пользователем задачи, описание задачи в текстовой форме (определения, формулы и пр.) или ссылка на некоторую веб-страницу, на которой находится описание задачи. Также на вкладке находится история запусков данной задачи, для которой указываются дата и время последних нескольких запусков задачи, а также кнопка, позволяющая посмотреть результаты и соответственно параметры этих запусков.

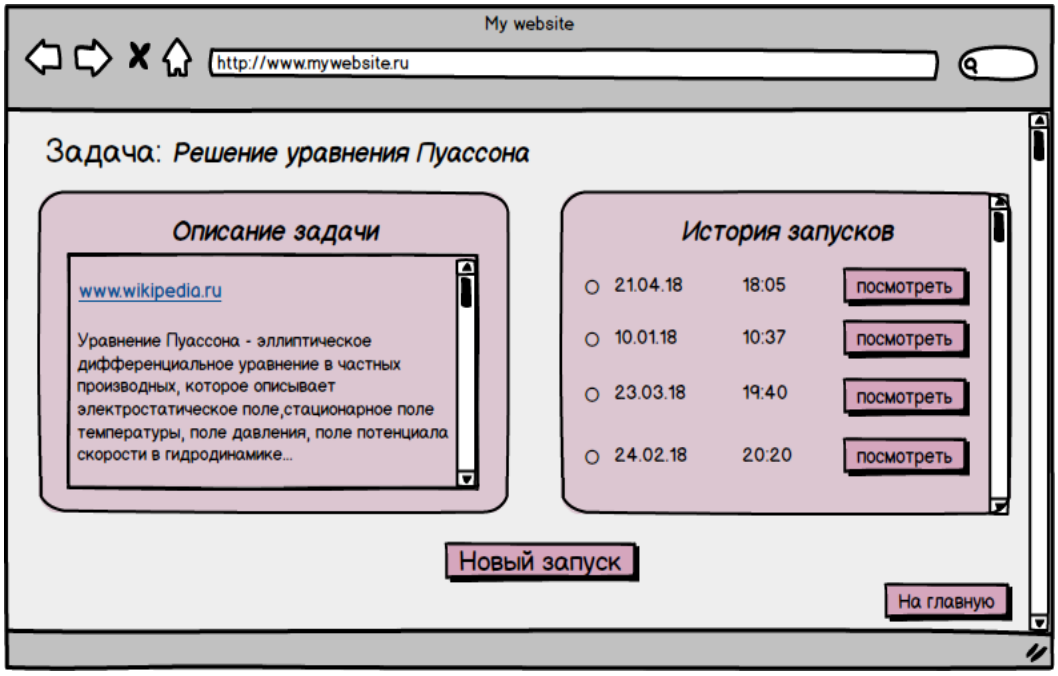


Рисунок 8 – Окно информации о выбранной задаче

При клике на кнопку «Новый запуск», пользователь переходит на страницу добавления параметров (рисунок 9), необходимых для нового запуска текущей задачи на некоторой вычислительной системе и добавления этого запуска в очередь задач в личном кабинете для мониторинга её статуса. Для добавления нового запуска необходимо указать 3 типа параметров. Все необходимые параметры указываются при формировании новой задачи и добавлении её в базу

всевозможных задач, доступных пользователям, поэтому здесь необходимо лишь присвоить им конкретные значения.

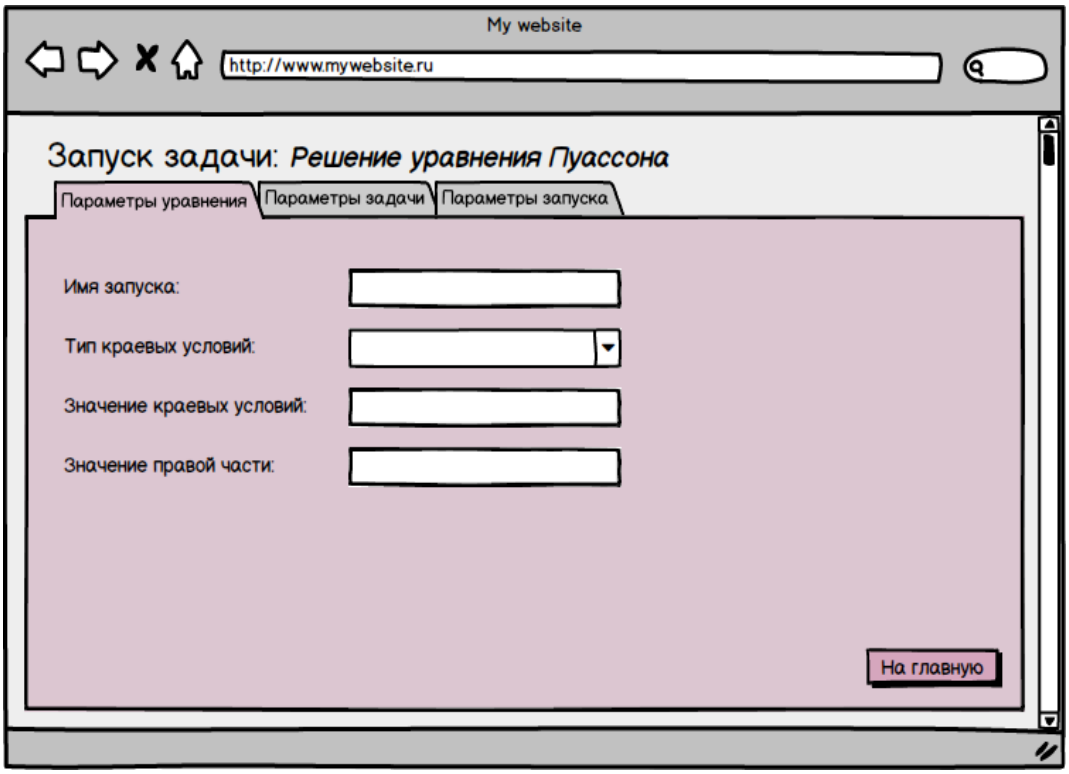


Рисунок 9 – Окно установки параметров задачи для её запуска

Сначала, пользователь оказывается на вкладке «Параметры уравнения» (рисунок 9), где необходимо указать такие параметры как имя запуска, которое будет отображаться в области очереди задач, краевые условия, значение правой части и пр.

После добавления параметров уравнения, необходимо перейти на вкладку «Параметры задачи» (рисунок 10), для добавления параметров, необходимых для нахождения решения задачи (например: шаг, точность решения, количество итераций и пр.)

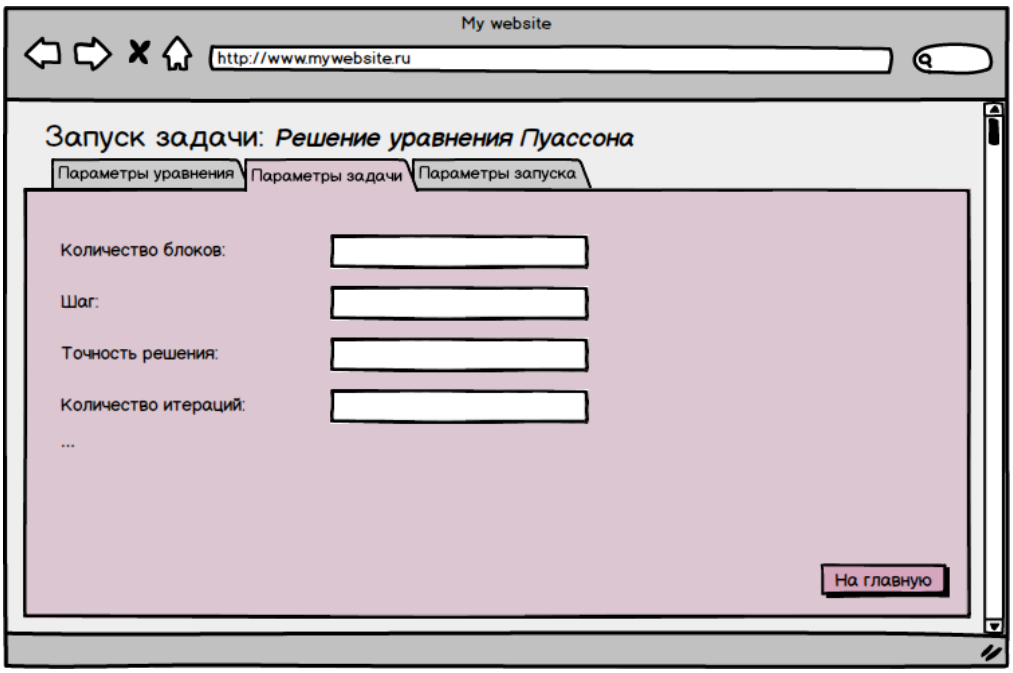


Рисунок 10 – Окно установки параметров решения для запуска задачи

И, наконец, необходимо на вкладке «Параметры запуска» (рисунок 11) указать вычислительную систему, на которой будет запущена задача, а также количество ядер и потоков, необходимых пользователю для работы задачи. После указания имени вычислительной системы, в области расположенной справа, отображаются актуальные характеристики выбранной системы, в качестве справочной информации, для корректного ввода необходимых параметров и согласования их с выбранной вычислительной системой. После заполнения всех типов параметров, необходимо нажать на кнопку «Выполнить», которая запустит задачу на некотором кластере и добавит её в область очереди задач, которая расположена в центре личного кабинета, откуда можно узнать статус выполнения задачи и посмотреть результат в случае успешного или неуспешного выполнения задачи.

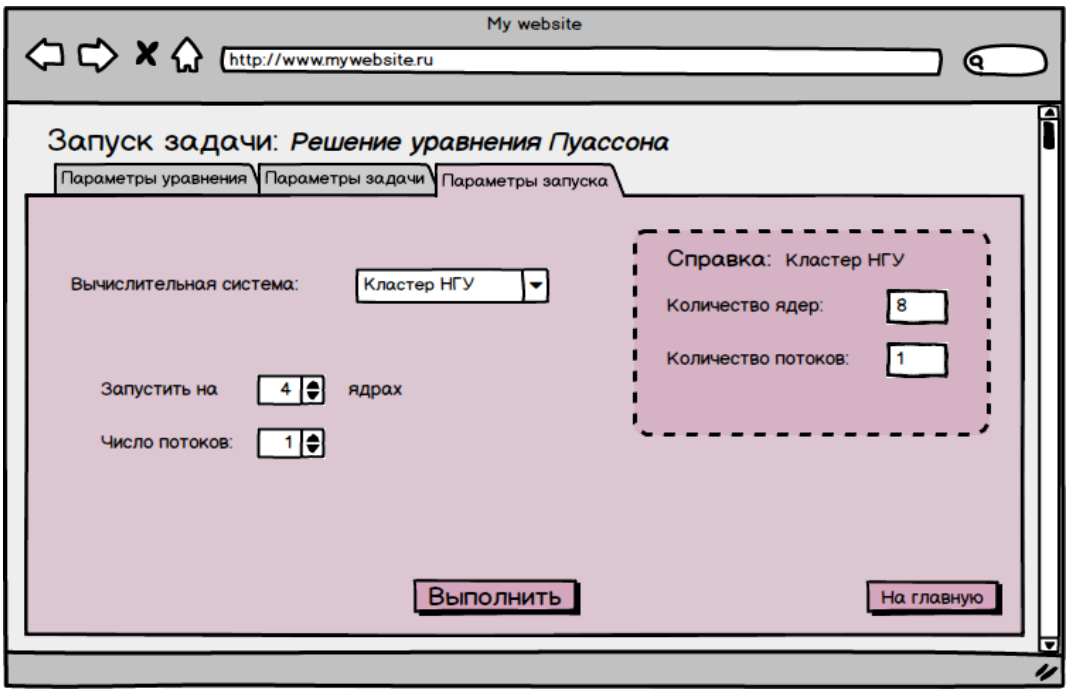


Рисунок 11 – Окно установки параметров системы для запуска задачи

В области управления задачами, помимо списка доступных задач, есть кнопка «Добавить задачу», позволяющая добавить новую задачу в личный кабинет пользователя и в дальнейшем запускать её на некоторых доступных вычислительных системах.

При клике на данную кнопку, пользователь переходит на страницу добавления новой задачи (рисунок 12). Здесь необходимо добавить имя задачи, которое будет отображаться в списке возможных задач в личном кабинете, описание задачи, которую необходимо решить. Подразумевается, что пользователь введёт словесное описание задачи, либо оставит ссылку на ресурс, где эта задача подробно описана. Далее требуется прикрепить код решателя данной задачи, с помощью которого мы и будем получать решение задачи на вычислительной системе.

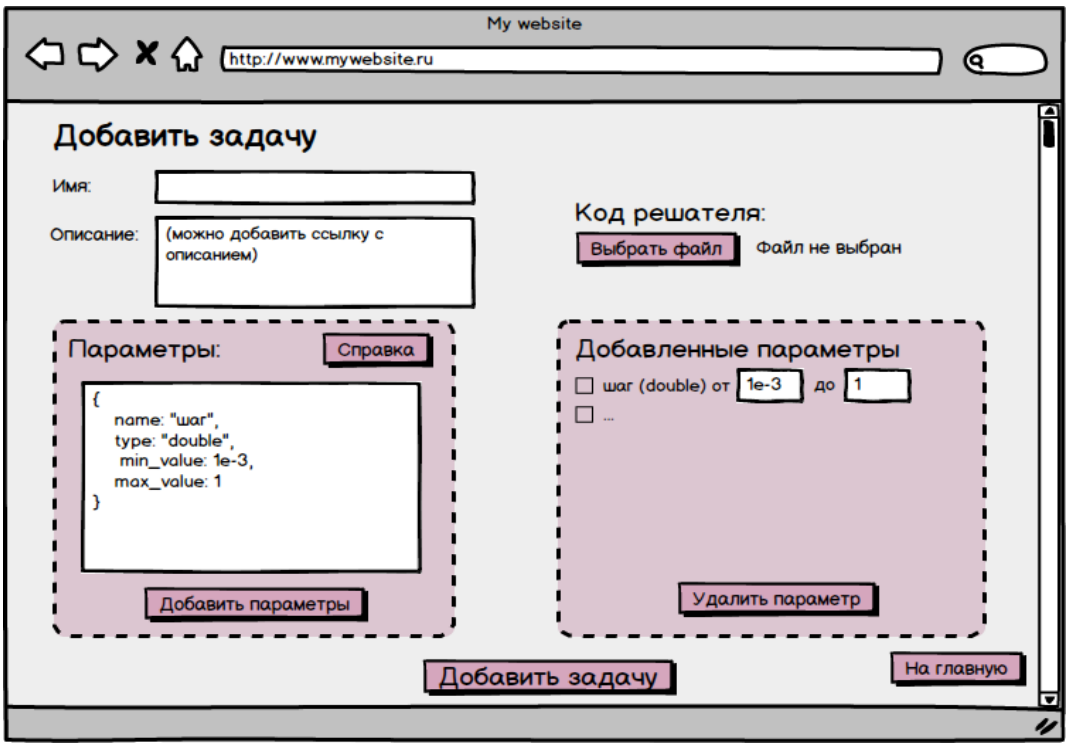


Рисунок 12 – Окно добавления новой задачи в систему

После этого, в области «Параметры» пользователь должен будет ввести список параметров, которые необходимы задаче для корректного запуска и выполнения. Эти параметры необходимо ввести в формате JSON, если пользователь не знаком с этим форматом, то при клике на кнопку «Справка», ему откроется краткое описание синтаксиса, необходимого для ввода параметров, с примером ввода. В качестве полей параметров обязательно должны присутствовать имя параметра, его тип, максимальное и минимальное значения. Подробнее параметры и результаты добавления задачи будет рассмотрены в следующем разделе.

После ввода параметров и нажатия на кнопку «Добавить параметры» введённые параметры отобразятся в области добавленных параметров, где их можно выделить и удалить, в случае необходимости. Выполнив все вышеперечисленные действия и нажав на кнопку «Добавить задачу», пользователь увидит сообщение о том, что задача успешно добавлена или что невозможно добавить задачу, в связи с отсутствием некоторых из вышеперечисленных пунктов.

2.3.4. Описание области управления списком коллег

Вернёмся к рассмотрению страницы личного кабинета, а точнее к области, отображающей коллег пользователя и рассмотрим её подробнее.

При клике на кнопку «Добавить друга», пользователь переходит на страницу добавления нового друга (рисунок 13). Сама процедура очень проста: необходимо ввести логин или фамилию, или имя, или отчество, или ФИО полностью, затем среди найденных совпадений выбрать нужного нам пользователя и нажать кнопку «Добавить в друзья». Далее всплывает сообщение о том, что выбранный пользователь добавлен в список друзей и ему приходит системное сообщение о том, что пользователь N добавил его в друзья.

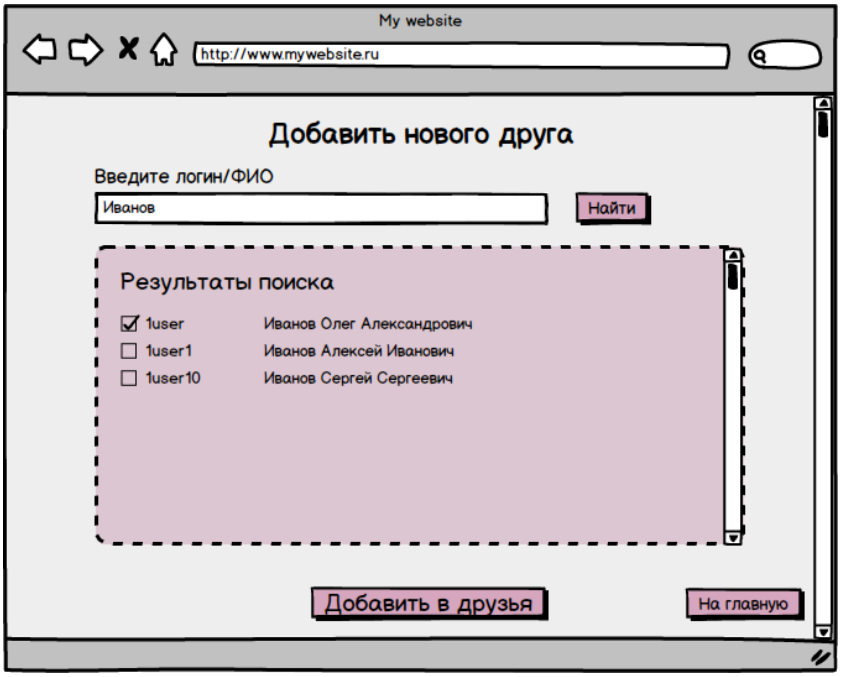


Рисунок 13 – Окно добавления нового друга

При клике на кнопку «Создать группу», пользователь переходит на страницу создания группы пользователей (рисунок 14). Необходимо ввести имя новой группы и выбрать пользователей, которые будут входить в эту группу, после чего следует нажать на кнопку «Создать новую группу», после чего всплывает сообщение о том, что группа N создана.

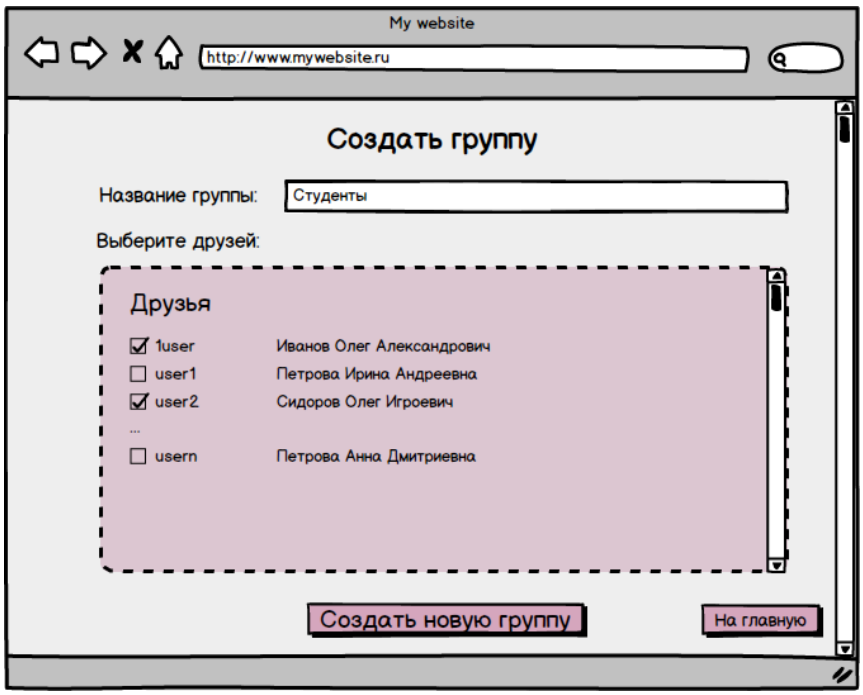


Рисунок 14 – Окно создания группы пользователей

При клике на кнопку «Группы», пользователь переходит на страницу со списком групп пользователей (рисунок 15), где он может поделиться задачей с одной или несколькими группами. Необходимо выбрать из списка возможных групп те группы, которым надо отослать задачу и выбрать саму задачу, которой пользователь будет делиться в группой. При нажатии на кнопку «Расшарить», всплывает сообщение о том, что выбранная задача «расшарена» выбранной(-ым) группе(-ам) пользователей.

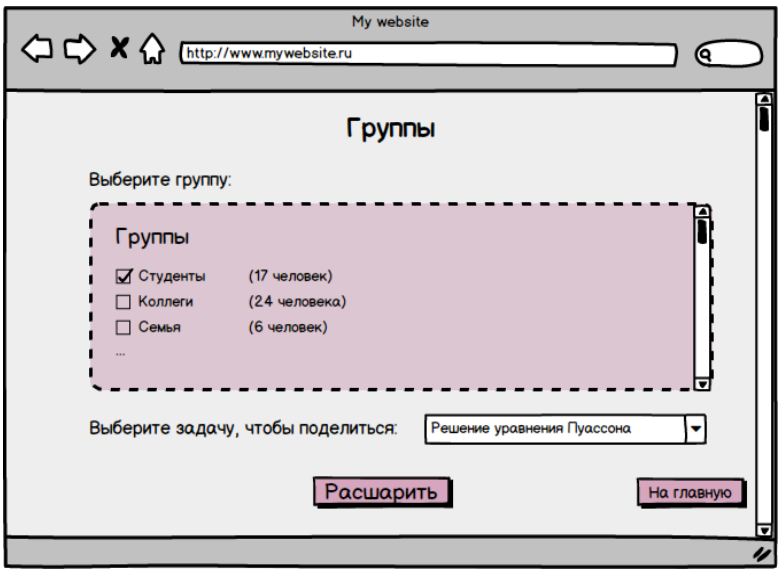


Рисунок 15 – Окно «расшаривания» задачи группе пользователей

2.4. ОПИСАНИЕ ГЕНЕРАЦИИ ЧАСТИ ИНТЕРФЕЙСА

Одним из требований, на основе которых разрабатывается данный программный комплекс, является наличие функционального языка, который позволит описывать понятия, связанные с вычислительной задачей и на основе описания генерировать пользовательский интерфейс для запуска такой задачи на ВВС.

Другими словами, при добавлении в систему нового приложения, пользователю необходимо специфицировать параметры приложения, если они имеются. После ввода спецификации параметров, пользователю предоставляется возможность отредактировать введённые параметры при необходимости и добавить приложение в систему.

Система должна уметь распознавать различные типы параметров, специфицированные с помощью разработанного языка, и в соответствии с этим типом генерировать соответствующий интерфейс, то есть различать, например, целочисленные или вещественные параметры, а также отличать их от строковых и так далее. Или, к примеру, система должна отличать такой тип как *«геометрия области»*. После того как пользователь специфицирует параметр с таким типом и добавит приложение с таким параметром, то в интерфейсе постановки приложения на выполнение на ВВС, должна появиться автоматически сгенерированная форма ввода, позволяющая пользователю указать, как именно должна выглядеть область моделирования для решения данной задачи, например, нарисовав её в предложенной форме или описав в некотором формальном текстово-численном представлении.

В вырожденном, простейшем случае, если есть описание каких-либо параметров, должен генерироваться интерфейс, где для всех параметров предусмотрены поля ввода, соответствующие типам этих параметров. Таким образом, по описанию в разработанном языке все объекты создаются автоматически, следовательно, нет необходимости каждый раз изменять интерфейс под

конкретную задачу вручную и такая генерация избавляет пользователя от создания интерфейса задачи для запуска вручную.

В рамках ВКР предлагается в качестве языка спецификации входных и выходных параметров использовать язык JSON, по причине того, что данный язык хорошо интерпретируется веб-приложениями, а также удобен для описания различных коллекций данных. Кроме того, данный язык быстр и компактен, что позволяет необходимые данные легко сериализовать и передать на хранение в файловую систему.

Так как данный текстовый формат применяется в качестве языка спецификации, то он должен быть структурирован и применять его предполагается так: каждый из описываемых параметров заключается в фигурные скобки, разделяемые между собой знаком «;», за исключение последнего параметра в списке. Внутри каждый параметр имеет специальные поля для заполнения: поле «name» – отвечающее за имя описываемого параметра, «type» – тип параметра, «min_value» – минимальное значение параметра и поле «max_value» – максимальное значение. Поле type различает несколько типов параметров: double/float – вещественное число, int – целое число и string –текстовая строка. Для текстовой строки отсутствует необходимость указания минимального и максимального значения. Внутри фигурных скобок запись полей для описания параметров выглядит следующим образом:

```
"<название_параметра>":"<значение_параметра>".
```

Поля между собой разделяются запятыми, за исключением последнего поля. Пример описания спецификации параметров на таком структурированном языке представлен в приложении И.

В результате, специфицировав таким образом параметры, требуемые задаче (рисунок 17), будет автоматически сгенерирован интерфейс, позволяющий задать значения описанных параметров (рисунок 18, 19, 20).

3. РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1. ВЫБОР СРЕДСТВ ДЛЯ РЕАЛИЗАЦИИ ПРОГРАММНОГО КОМПЛЕКСА

В рамках данной работы были использованы следующие средства для реализации программного комплекса: для разработки WebGUI были использованы технологии HTML и CSS, являющиеся основными средствами разработки веб-приложений. Кроме того, для обеспечения интерактивности страниц веб-приложения был использован язык JavaScript (JS). Для разработки архитектуры системы была использована программная платформа Node.js с фреймворком Express.js, преимуществами которых являются возможность использования JS как на стороне клиента, так и на стороне сервера, а также взаимодействие через свой API с клиентскими. Кроме того, приложения, разрабатываемые с помощью данных средств, являются кроссплатформенными. Для разработки базы данных была использована библиотека SQLite. Преимуществом такой библиотеки являются обращения СУБД напрямую к файлам, в которых хранятся данные, а также широкий набор инструментов для работы с ней.

3.2. ОПИСАНИЕ МОДЕЛЬНОГО ПРИЛОЖЕНИЯ

В качестве модельного приложения для демонстрации корректной работы разработанного программного комплекса было выбрано приложение, позволяющее численно решить уравнение Пуассона [16]. Пусть нам необходимо решить уравнение Пуассона:

$$\nabla^2 \phi = f \tag{1}$$

$$\frac{\partial^2 \phi}{\partial x^2} + \frac{\partial^2 \phi}{\partial y^2} + \frac{\partial^2 \phi}{\partial z^2} = f \tag{2}$$

в трёхмерной области с краевыми условиями первого рода, накладываемыми на границы областей. Сама область представлена в виде прямоугольного параллелепипеда, размером endX * endY * endZ (рисунок 21).

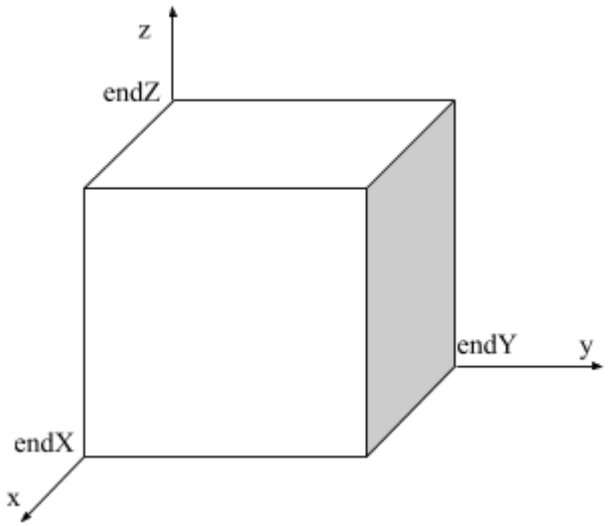


Рисунок 16 – Область решения численной задачи

В области задана равномерная сетка с шагами hx, hy, hz. В этой сетке уравнение (2) аппроксимируется семиточечной конечно-разностной схемой:

$$\frac{\tilde{\phi}_{i-1,j,k} - 2\tilde{\phi}_{i,j,k} + \tilde{\phi}_{i+1,j,k}}{h_x^2} + \frac{\tilde{\phi}_{i,j-1,k} - 2\tilde{\phi}_{i,j,k} + \tilde{\phi}_{i,j+1,k}}{h_y^2} + \frac{\tilde{\phi}_{i,j,k-1} - 2\tilde{\phi}_{i,j,k} + \tilde{\phi}_{i,j,k+1}}{h_z^2} = f_{i,j,k} \quad (3)$$

Множество уравнений вида (3) для всех значений i,j,k с некоторыми граничными условиями составляют СЛАУ, которая может быть решена различными методами, например, итерационным.

Итерационный процесс продолжается до тех пор, пока норма разности решения на текущей итерации с решением на предыдущей итерации не станет меньше некоторого заданного малого числа:

$$\left\| \phi^k - \phi^{k-1} \right\| < \varepsilon \quad (4)$$

3.3. ДЕМОНСТРАЦИЯ РАБОТЫ МОДЕЛЬНОГО ПРИЛОЖЕНИЯ

3.3.1. Спецификация интерфейсов

В результате работы при добавлении модельного приложения в систему для спецификации параметров модельного приложения, описанных в приложении Е, был автоматически сгенерирован следующий интерфейс, состоящий из трёх вкладок. Пользователю необходимо ввести описание параметров (см. раздел 2.4), а также имя приложения, путь к исполняемому файлу и описание, при необходимости. Кнопка «Добавить приложение» проинформирует об успешной добавке в систему и автоматически перенесёт на страницу личного кабинета, где новое приложение отобразится в списке доступных приложений (рисунок 17, 21).

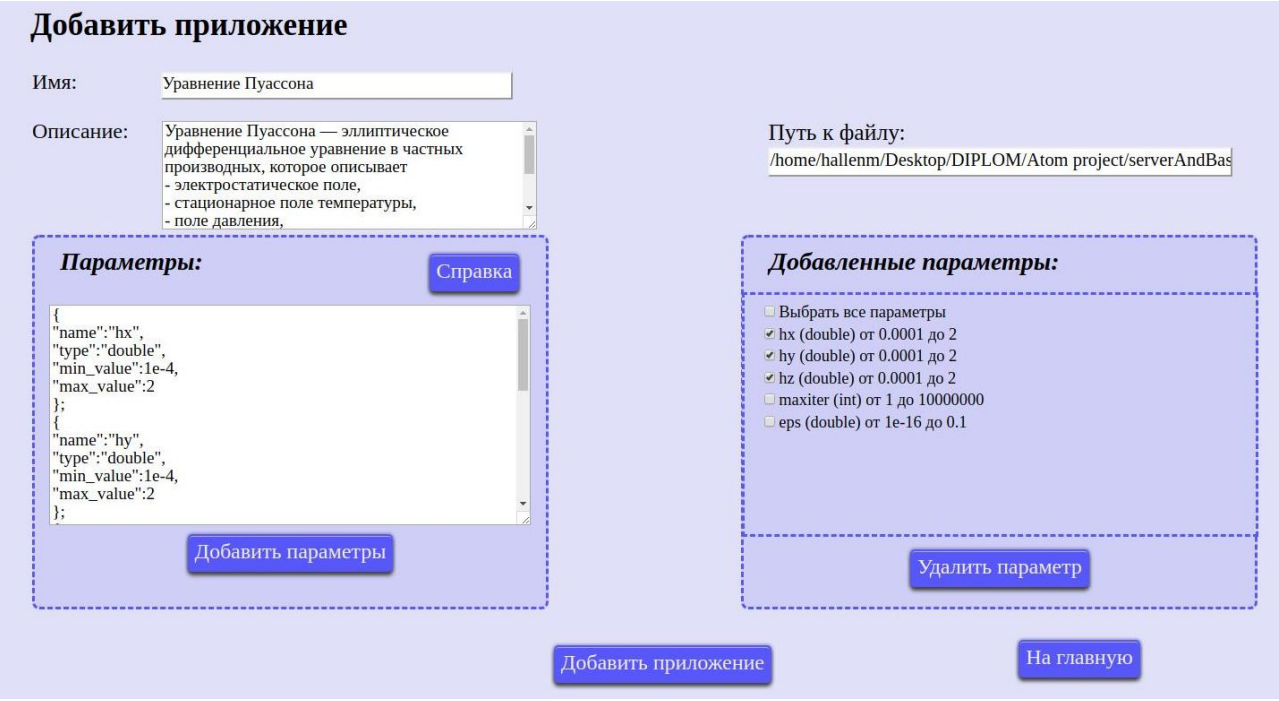


Рисунок 17 – Добавление модельного приложения в систему

При клике на название приложения пользователь перейдёт на страницу с описанием приложения, где будет предложено запустить приложение на ВВС. Для этого необходимо заполнить параметры, требуемые на вход приложению

(рисунок 19), а также имя запускаемой задачи, необходимые на вход задаче файлы и данные для ВВС (рисунок 18, 20).

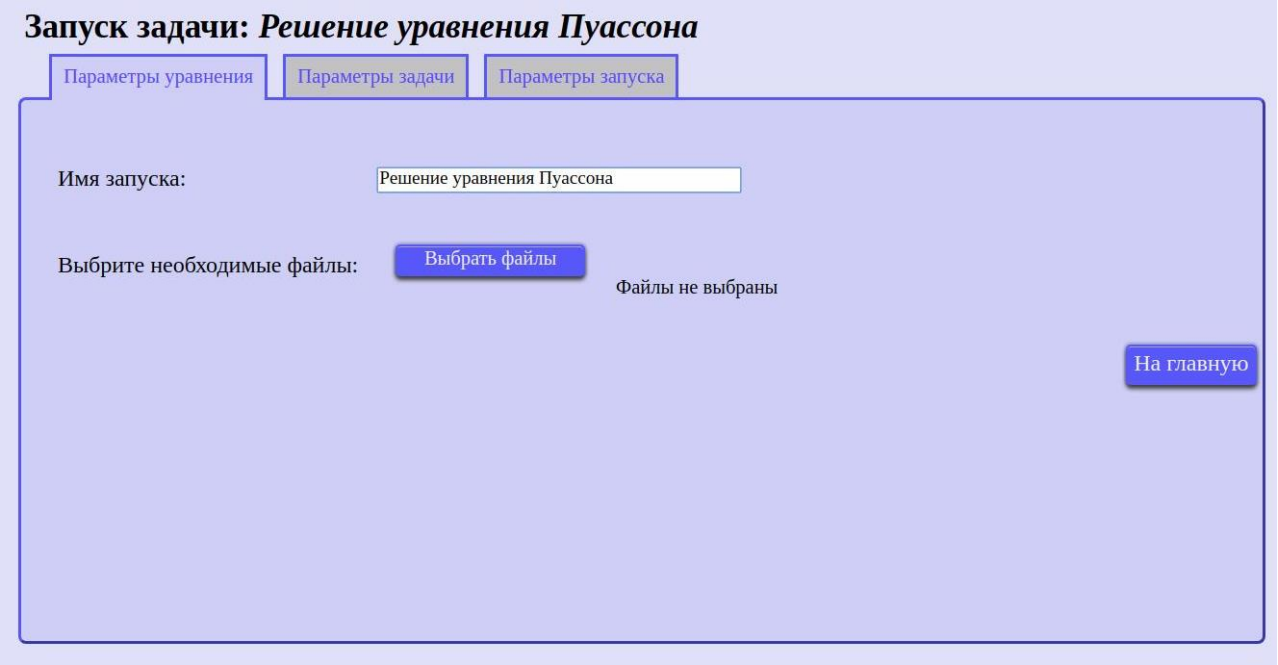


Рисунок 18 – Сгенерированный интерфейс приложения, вкладка 1

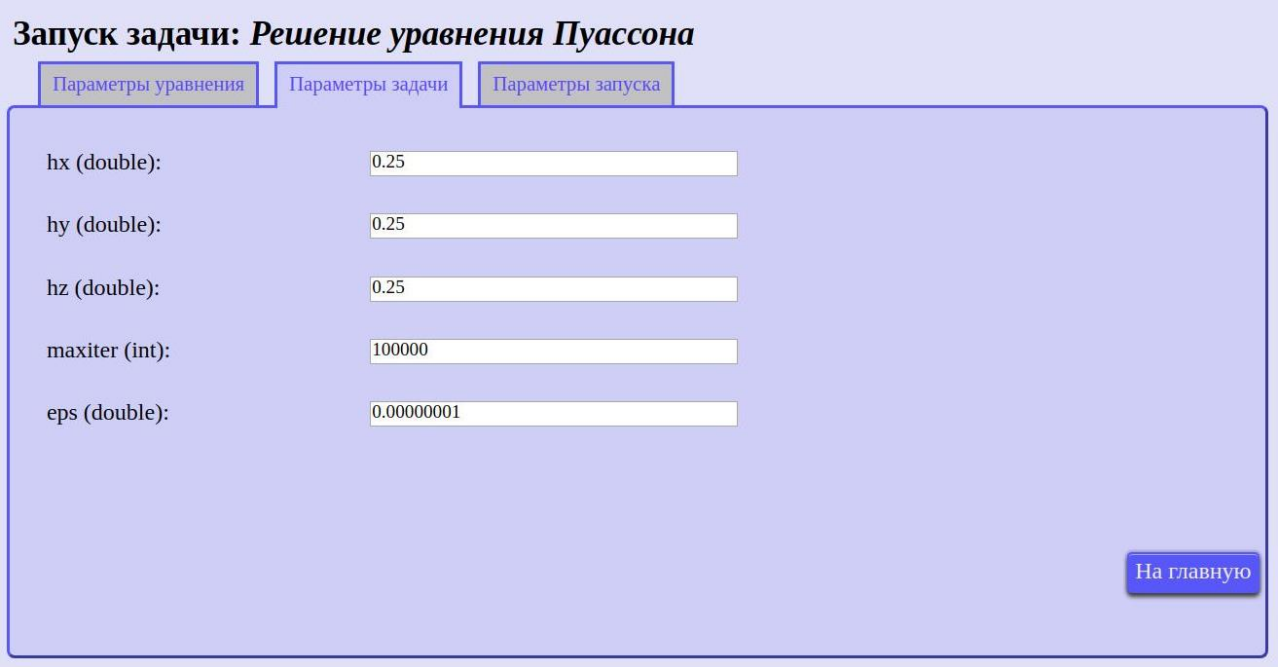


Рисунок 19 – Сгенерированный интерфейс приложения, вкладка 2

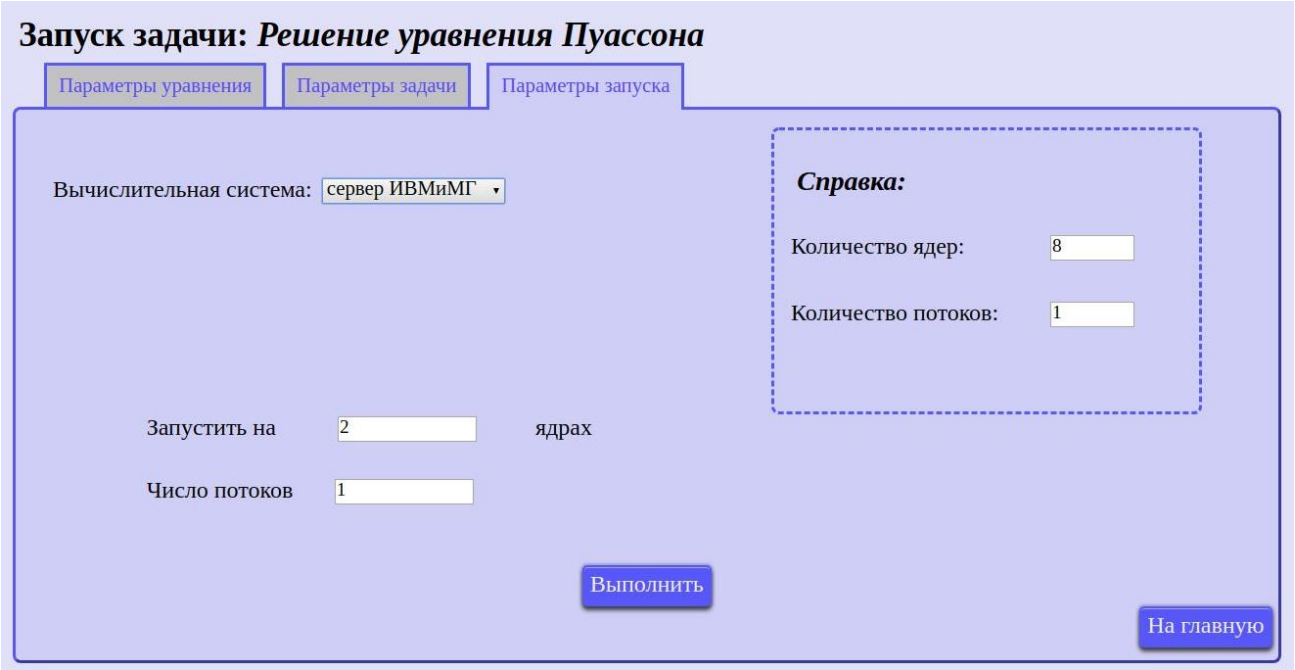


Рисунок 20 – Сгенерированный интерфейс приложения, вкладка 3

3.3.2. Пример постановки модельного приложения на выполнение на ВВС

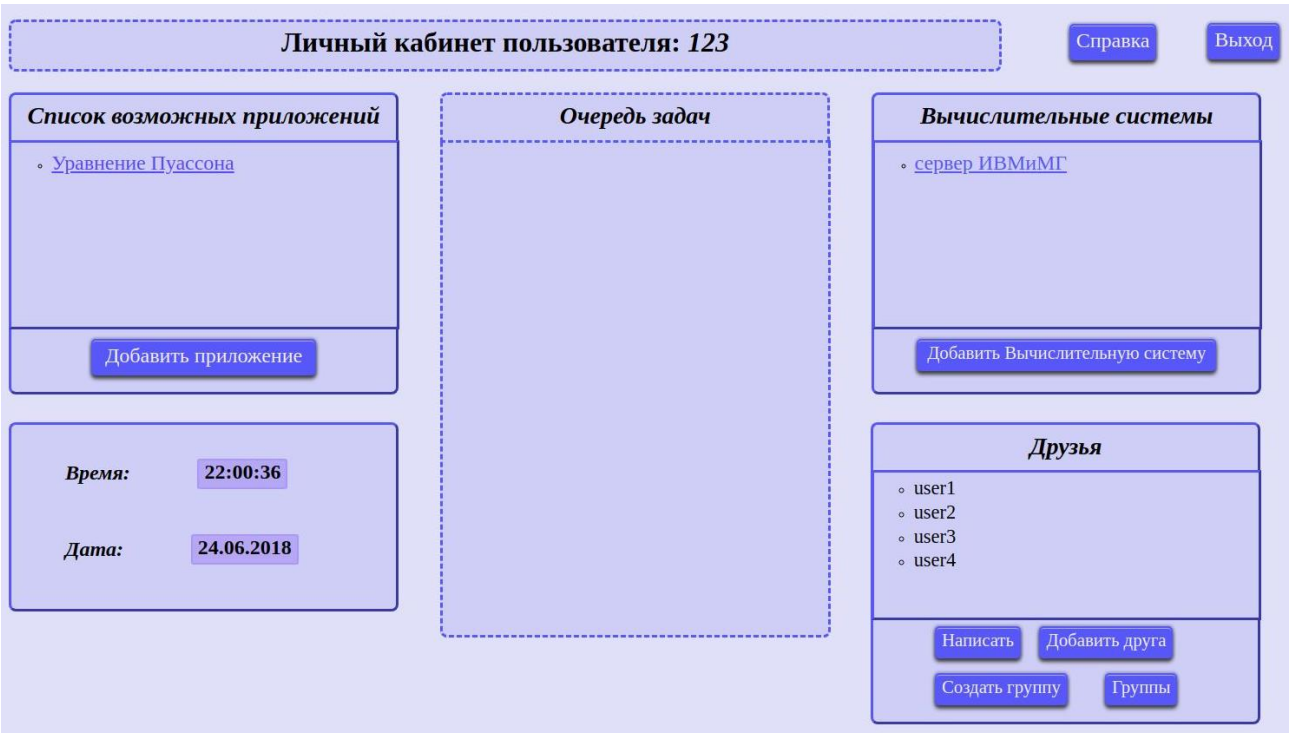


Рисунок 21 – Вид личного кабинет после добавления приложения

После добавления приложения (рисунок 17, 21) пользователь оказывается в личном кабинете и далее, при клике на имя приложения, переходит на страницу с информацией о приложении (рисунок 22).

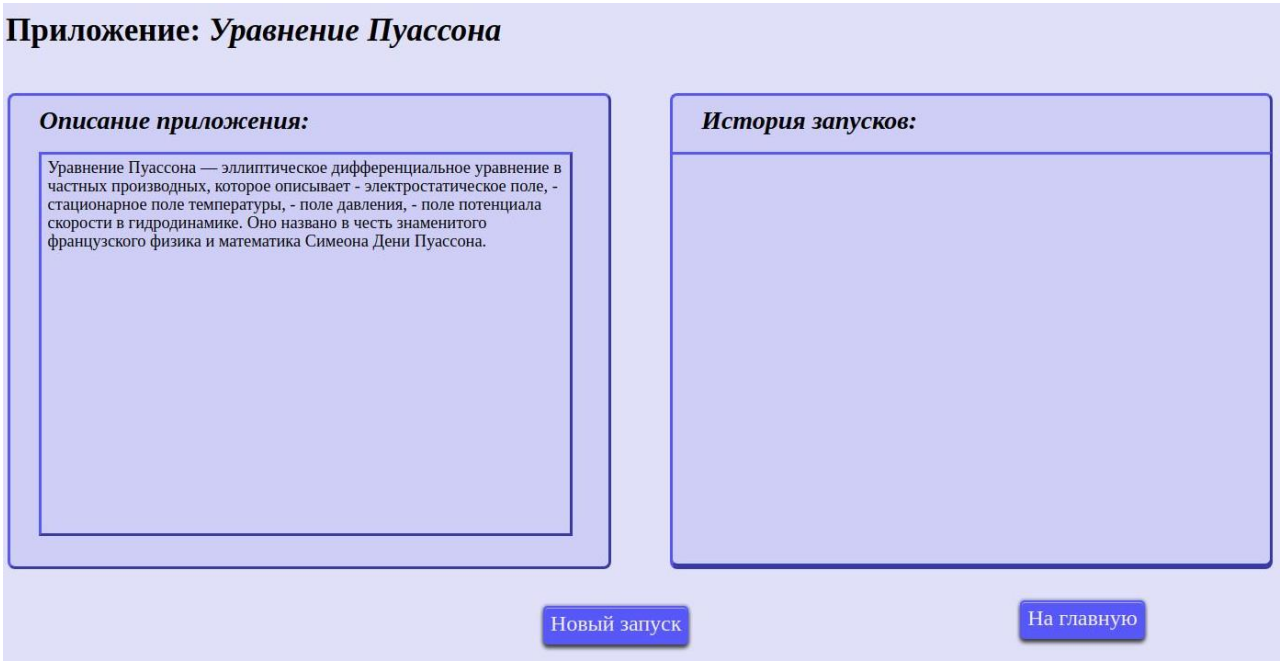


Рисунок 22 – Информация о добавленном приложении

Для постановки этого приложения на выполнение на ВВС, необходимо, нажав кнопку «Новый запуск» перейти на вкладки ввода параметров (рисунок 18, 19, 20) и, определив их, нажать на кнопку «Выполнить», после чего пользователя проинформируют об успешной постановке задачи на выполнение и перенаправят в личный кабинет, где в области отображения запускаемых задач будет отображён новый запуск (рисунок 23).

Рисунок 23 – Выполнение задачи на ВВС

На рисунке, расположенном выше видно, что состояние задачи «выполняется (running)». После завершения выполнения задачи, состояние «выполняет-

ся» будет изменено на «завершено (done)» (рисунок 24) и пользователь получит возможность просмотра результата выполнения задачи (рисунок 25).

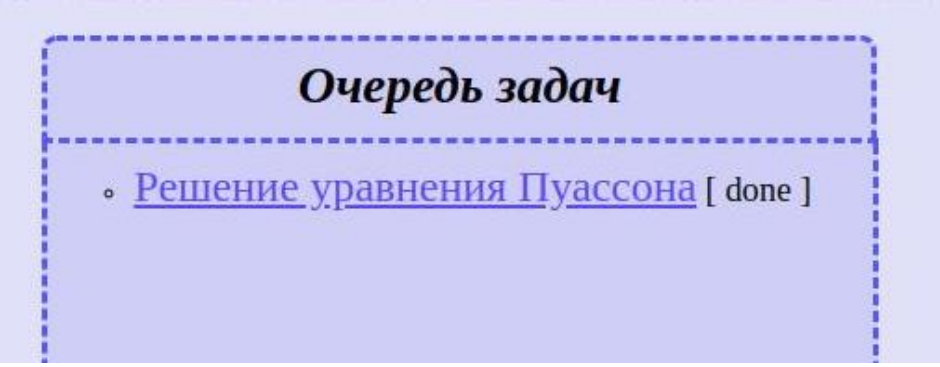


Рисунок 24 – Изменение состояния выполняемой задачи

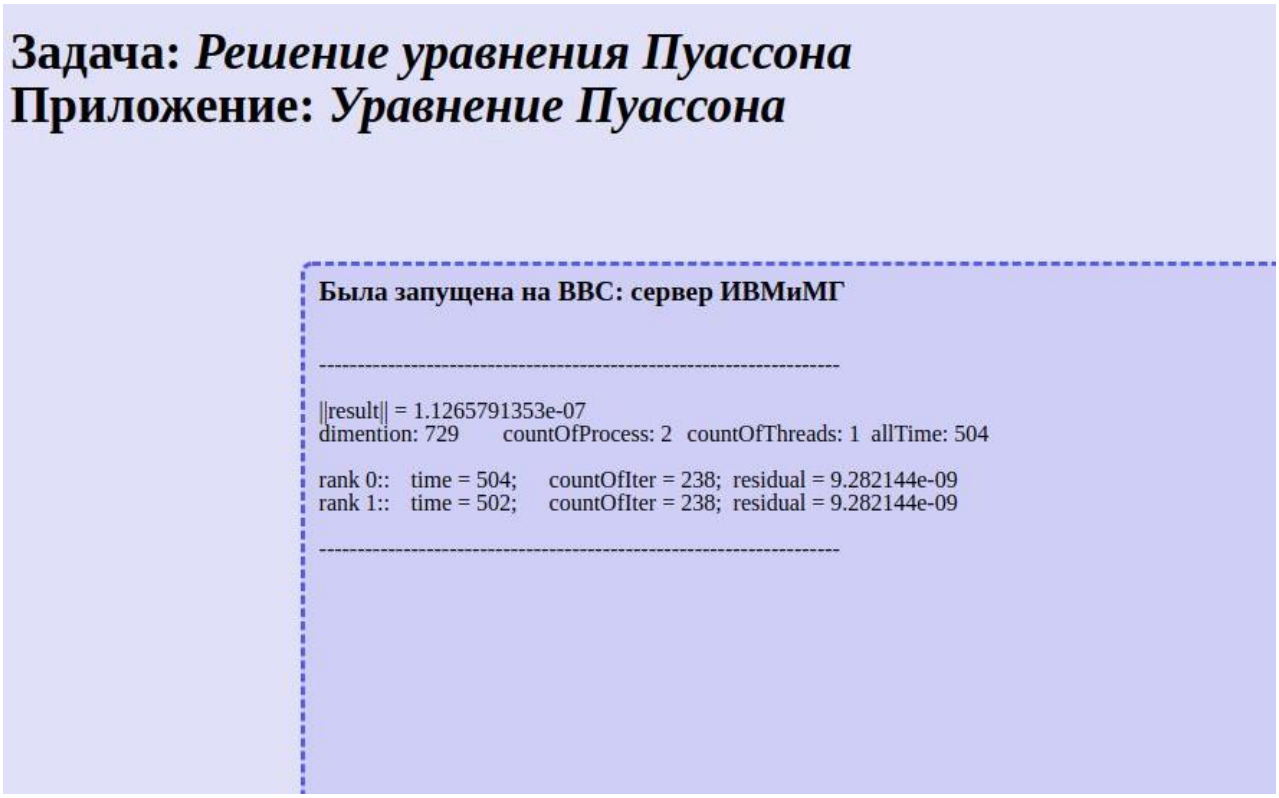


Рисунок 25 – Результат выполнения задачи

3.3.3. Характеристики производительности системы

Выполнение модельного приложения проводилось на процессоре Intel Xeon E5-2620 (6-ядерный процессор с частотой 2.10ГГц) сервера ИВМиМГ. Тестирование проводилось на сетках с шагом 0.25 и 0.5, на 2, 4 и 8 процессах системы. Результаты вычислений отображены в таблице 1.

Таблица 1 – Результаты решений уравнения Пуассона на ВВС с различными характеристиками

Размерность задачи	Количество процессов	Количество итераций	Время работы, мс	Ускорение	Эффективность, %
125	2	57	11	1.000000000	50.00
	4	57	18	0.611111111	15.28
	8	57	274	0.040145985	0.50
729	2	242	504	1.000000000	50.00
	4	242	570	0.884210526	22.11
	8	242	2 124	0.237288136	2.97

ЗАКЛЮЧЕНИЕ

В результате работы были получены следующие результаты:

1. разработана архитектура системы, позволяющая пользователям унифицированным образом организовать вычисления на удаленных высокопроизводительных вычислительных системах;
2. разработан прототип системы, включающий в себя следующие компоненты:
    а) сервер для управления работой пользователя и запуском задач на ВВС,
    б) веб-интерфейс для взаимодействия конечных пользователей с сервером,
    в) базу данных, хранящую данные пользователя и данные, связанные с запускаемыми им задачами,
    г) модуль запуска задач на выполнение на удалённых ВВС;
3. разработан модуль, позволяющий автоматически генерировать интерфейсы приложений по предоставленному описанию на предложенном языке;
4. разработанный прототип протестирован на прикладной численной задаче решения уравнения Пуассона.

Полученные результаты соответствуют поставленным целям.

В дальнейшем планируется расширение функциональности системы, путём добавления новых типов ВВС для запуска приложений, а также модуля управления списком коллег. Также планируется разработать язык, позволяющий формально описывать понятия, связанные с приложениями и генерировать формы к таким понятиям.

СПИСОК ЛИТЕРАТУРЫ

1. MPI: A Message-Passing Interface Standard. Version 3.1. University of Tennessee, Knoxville, Tennessee, june 4, 2015. – 868 p.
2. Информационно-вычислительный центр Новосибирского государственного университета. [Электронный ресурс]. URL: http://nusc.nsu.ru/wiki/doku.php/doc/index (дата обращения: 18.06.2018).
3. Altair PBS Pro: Altair PBS Professional User's Guide. Version 13.0. Altair Engineering, Inc. 2003-2015. – 339 p.
4. Altair PBS Pro: Altair PBS Professional Reference Guide. Version 13.0. Altair Engineering, Inc. 2003-2015. – 534 p.
5. Оленёв Н.Н., Печёнкин Р.В., Чернецов А.М. Параллельное программирование в MATLAB и его приложения: монография / Н.Н. Оленёв, Р.В Печёнкин, А.М. Чернецов — Москва: ВЦ РАН, 2007. – 120 с.
6. Пономарева, И.С. Некоторые аспекты создания web-приложения на базе MATLAB Web Server / И.С. Пономарева, В.А. Зелепухина, Ю.Ю. Тарасевич // Информационные технологии, № 9, 2006 – с. 68–72.
7. MATLAB Distributed Computing Server: Perform MATLAB and Simulink computations on clusters, clouds, and grids. [Электронный ресурс]. URL: https://uk.mathworks.com/products/distriben.html (дата обращения: 18.06.2018).
8. Межведомственный Суперкомпьютерный Центр Российской Академии Наук (МСЦ РАН). [Электронный ресурс]. URL: http://www.jscc.ru/scomputers.html (дата обращения: 18.06.2018).
9. Технологии параллельного программирования: НОРМА | PARALLEL.RU – Информационно-аналитический центр по параллельным вычислениям. [Электронный ресурс]. URL: https://parallel.ru/tech/norma/ (дата обращения: 18.06.2018).

10. Andrianov A. NONPROCEDURAL NORMA LANGUAGE AND ITS TRANSLATION METHODS FOR PARALLEL ARCHITECTURES / A.N. Andrianov, T.P. Baranova, A.B. Bugerya, K.N. Efimkin // University News. North-Caucasian Region. Technical Sciences Series. 2017. № 3. С. 5 – 12.
11. Малышкин В.Э. Параллельное программирование мультикомпьютеров. / В.Э.Малышкин, В.Д.Корнеев. – В серии «Учебники НГТУ», Новосибирск, изд-во НГТУ, 2006 – 296 с.
12. Нариньяни А.С. Модель или алгоритм – новая парадигма информационной технологии. Информационные технологии, №4, 1997. с.18 – 22.
13. OWL 2 Web Ontology Language Document Overview (Second Edition). [Электронный ресурс]. URL: https://www.w3.org/TR/2012/REC-owl2-overview-20121211/ (дата обращения: 18.06.2018).
14. About the Unified Modeling Language (UML) Specification. Version 2.5.1. [Электронный ресурс]. URL: https://www.omg.org/spec/UML (дата обращения: 18.06.2018).
15. R. T. Fielding Architectural Styles and the Design of Network-based Software Architectures: Doctoral dissertation / Roy Thomas Fielding – University of California, Irvine, 2000 – p. 162.
16. Тамм И. Е. Основы теории электричества: Учеб. пособие для вузов. – 11-е изд., испр. и доп. – М.: ФИЗМАТЛИТ, 2003. – 616 с.
17. Гергель В.П. Основы параллельных вычислений для многопроцессорных вычислительных систем: учеб. пособие / В. П. Гергель, Р. Г. Стронгин – Н. Новгород: Изд-во ННГУ им. Н.И. Лобачевского, 2003. – 184 с.
18. Antonov A.S., Volkov N.I. An AlgoView Web-visualization System for the AlgoWiki Project / A.S. Antonov, N.I. Volkov // Parallel Computational Technologies (PCT'2017) – Kazan, Russia, 2017. – p. 3–13.
19. Грибова В.В. Автоматизация разработки пользовательских интерфейсов с динамическими данными / В.В. Грибова, Н.Н. Черкезишвилли // Открытые семантические технологии проектирования интеллектуаль-

ных систем OSTIS-2011, Материалы I Международной научно-технической конференции 10 февраля 2011 г. Минск. – Минск: БГУИР, 2011. – с. 287–292.

20. Генаев М. А., Комышев Е. Г., Гунбин К. В., Афонников Д. А. BioInfoWF – система автоматической генерации Web-интерфейсов и Web-сервисов для биоинформационных исследований / Вавиловский журнал генетики и селекции. 2012. Т. 16. №4/1. – с. 849.
21. Зелепухина В.А. Разработка систем управления содержимым интернет-ресурсов на основе автоматической генерации WEB-интерфейса и SQL-запросов / В.А. Зелепухина // Информационные технологии, № 8, 2008 – с. 20–22.
22. Грибова В.В., Клещёв А.С. Методы и средства разработки пользовательского интерфейса: современное состояние // Программные продукты и системы, 2001. №1. – с. 2–6.
23. Грибова В.В., Клещёв А.С. Управление проектированием и реализацией пользовательского интерфейса на основе онтологий // Проблемы управления №2. 2006. – с.58–62.

ПРИЛОЖЕНИЕ А

Запуск вычислительной задачи на кластере НГУ

(примечание – цветом выделены комментарии)

```
# Указывается путь к bash-интерпретатору
#!/bin/bash
# Строки, начинающиеся с «#PBS», содержат директивы, используемые планировщиком. Ресурсы, относящиеся к серверу PBS или к очереди, используются всей задачей и в запросе указываются в следующем виде: «-l Ресурс=Значение»
# Время, необходимое задаче для работы
#PBS -l walltime=00:01:00
# Ресурсы, относящиеся к узлам (или виртуальным узлам), запрашиваются и выделяются в виде блоков (chunk). Запрос в большинстве случаев можно представить в следующем виде: «-l select=N:chunk», где строка «chunk» имеет вид «Ресурс1=Значение1[:Ресурс2=Значение2 …]», а число «N» - сколько таких блоков требуется.
# Задаче необходимы два блока, каждый из которых содержит 4 ядра и 2000 МБ памяти, и на каждом из этих блоков будут запущены 4 MPI процесса, причём  расположение выделяемых блоков будет произвольно (place=free)
#PBS -l select=2:ncpus=4:mpiprocs=4:mem=2000m,place=free
# Переход в каталог, на который указывает переменная окружения $PBS_O_WORKDIR
cd $PBS_O_WORKDIR
# Подсчёт суммарного количества MPI процессов, требующих запуска
MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')
# Вывод на экран числа MPI процессов, хранящегося в переменной $MPI_NP
echo "Number of MPI process: $MPI_NP"
# Вывод на экран сообщения «File $PBS_NODEFILE:»
echo 'File $PBS_NODEFILE:'
# Вывод на экран содержимого файла, на который указывает переменная окружения $PBS_NODEFILE
cat  $PBS_NODEFILE
```

```
# Вывод на экран пустой строки
echo
# Запуск программы с помощью команды mpirun, которая узнаёт выделенные задаче узлы из файла, на который указывает переменная окружения $PBS_NODEFILE (для intel_mpi ключ «-hostfile» необходимо заменить на «-machinefile»)
mpirun -hostfile $PBS_NODEFILE -np $MPI_NP ./task.comp
```

ПРИЛОЖЕНИЕ Б

Запуск задач на суперкомпьютере МВС-10П МСЦ РАН с использованием планировщика SLURM в пакетном режиме с помощью утилиты srun

(примечание – цветом выделены комментарии)

```
# Указывается путь к sh-интерпретатору
#!/bin/sh
# Указание ограничения по времени, равное 1 минуте
#SBATCH --time=1
# Запуск задачи с помощью утилиты srun, где на каждом вычислительном узле необходимо выделить X узлов, при общем количестве узлов X*Y
srun --ntasks-per-node X -n X*Y ./task.comp
```

ПРИЛОЖЕНИЕ В

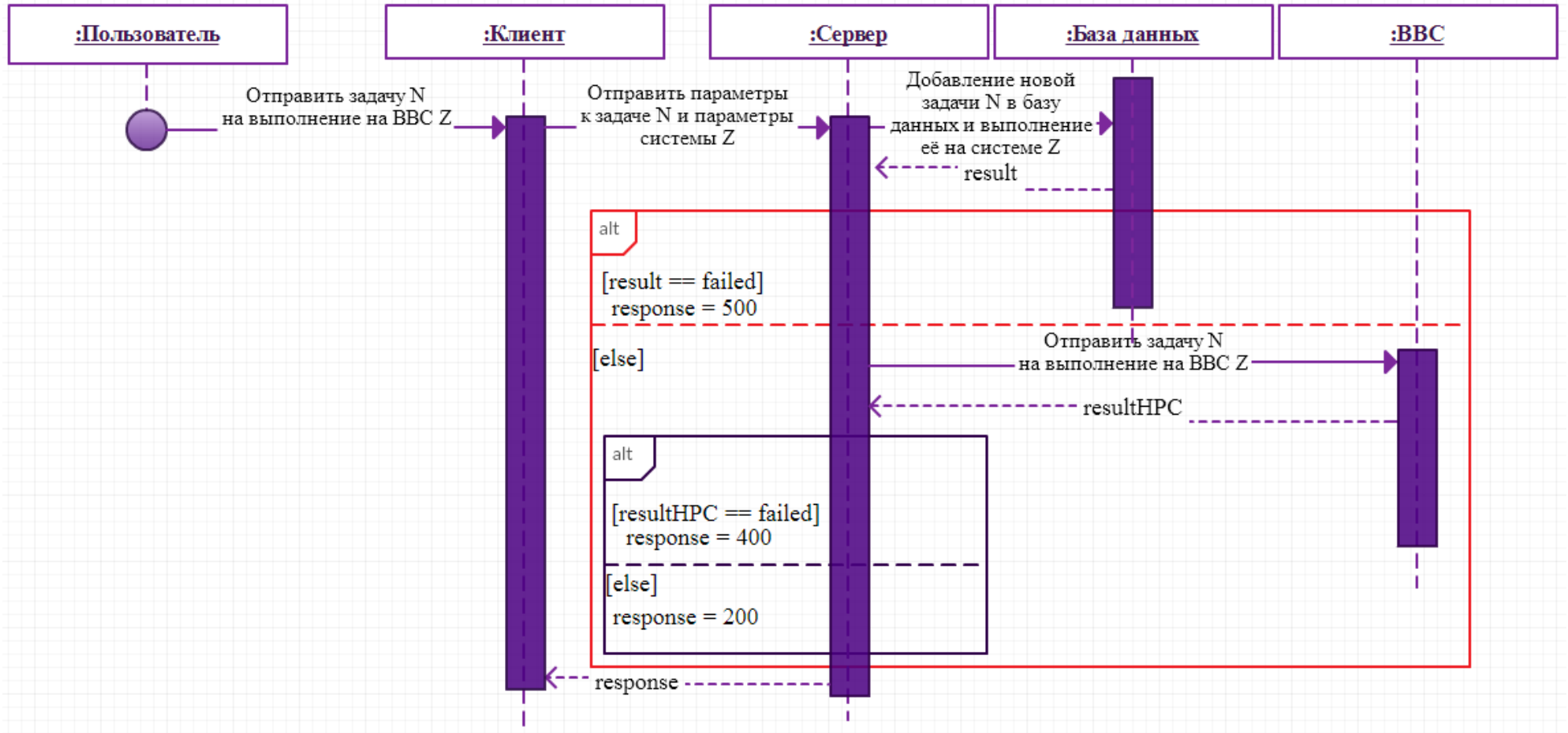
Запуск задач на суперкомпьютере МВС-10П МСЦ РАН с использованием планировщика SLURM в пакетном режиме с помощью утилиты mpiexec.hydra

(примечание – цветом выделены комментарии)

```
# Указывается путь к sh-интерпретатору
#!/bin/sh
# Запуск задачи с помощью утилиты mpiexec.hydra, где на каждом вычисли-
тельном узле необходимо выделить X узлов, при общем количестве узлов X*Y
mpiexec.hydra –perhost X -n X*Y ./task.comp
```

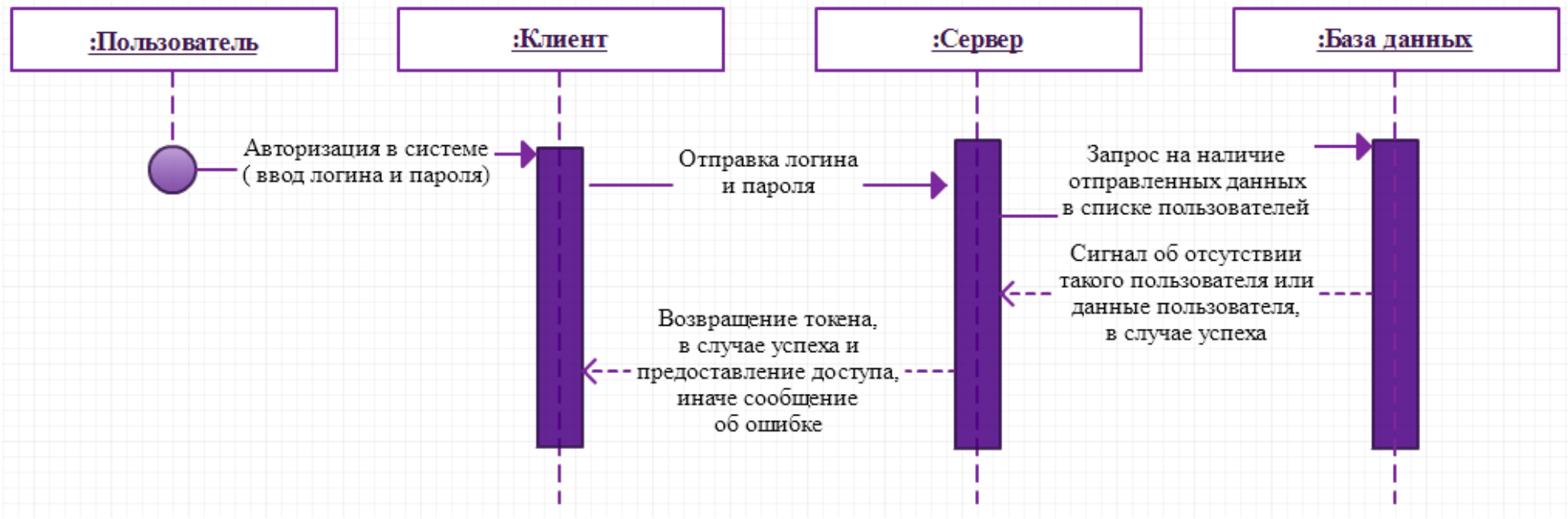
ПРИЛОЖЕНИЕ Г

Пример взаимодействия при отправке задачи на выполнение



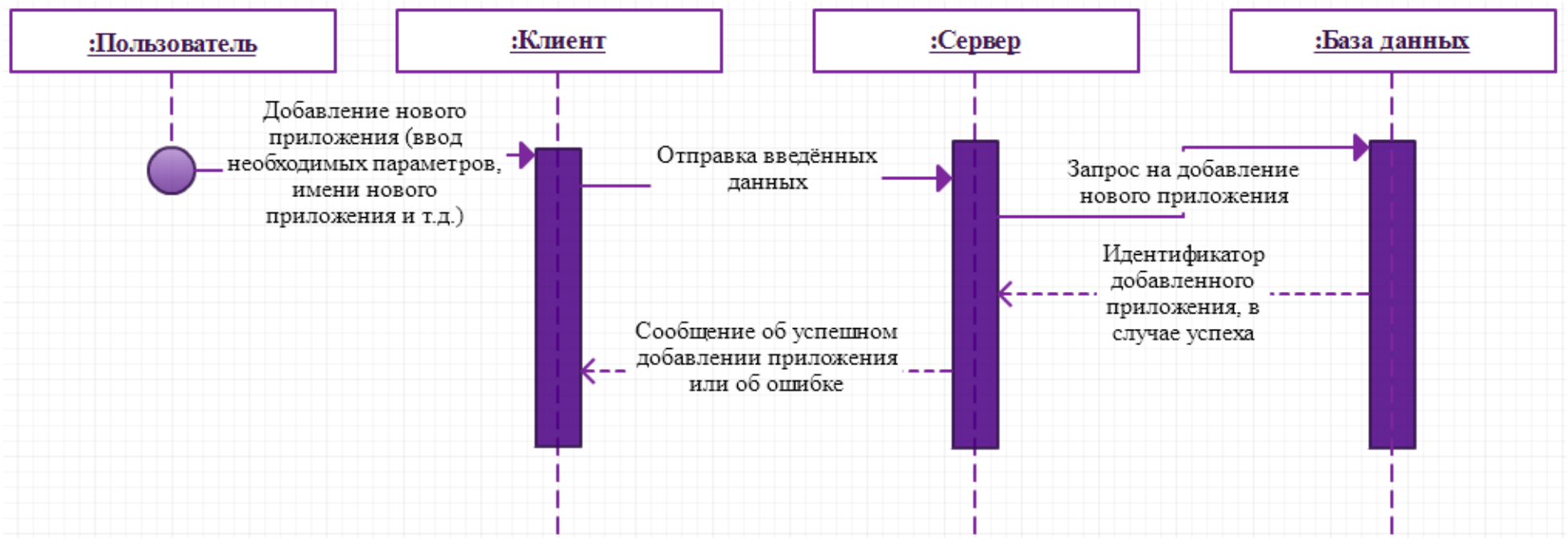
ПРИЛОЖЕНИЕ Д

Пример взаимодействия при авторизации пользователя



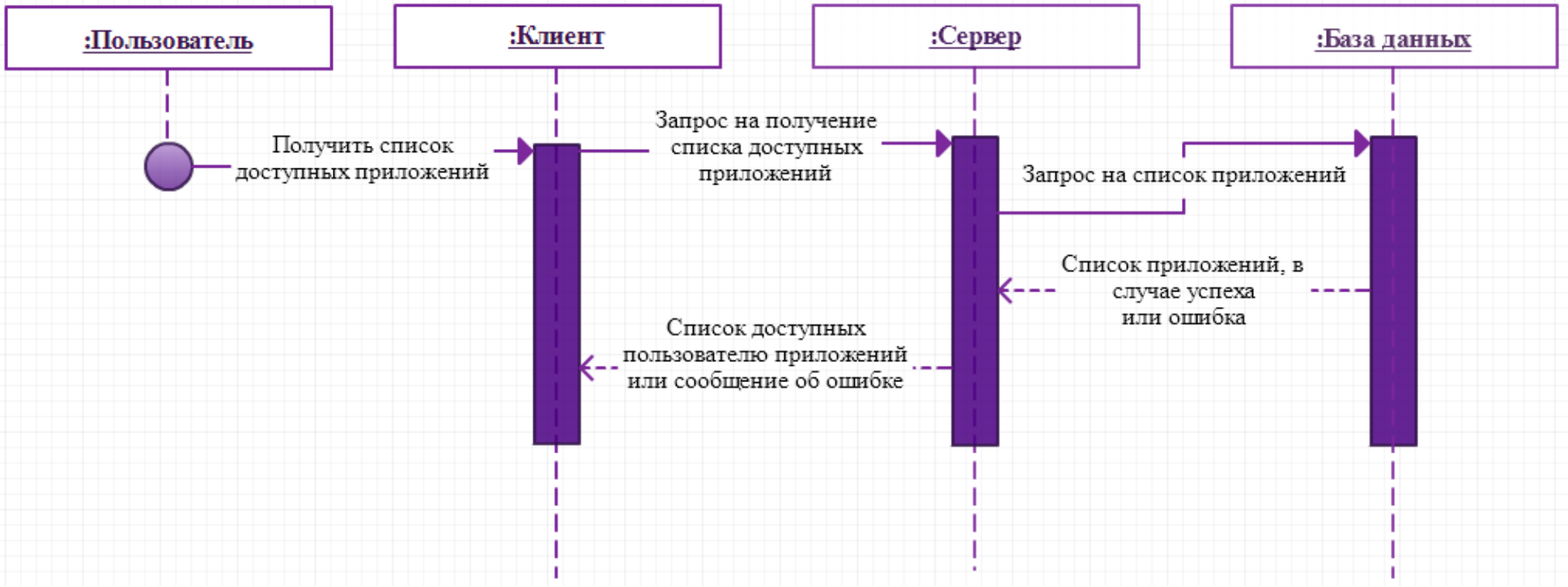
ПРИЛОЖЕНИЕ Е

Пример взаимодействия при добавлении нового приложения



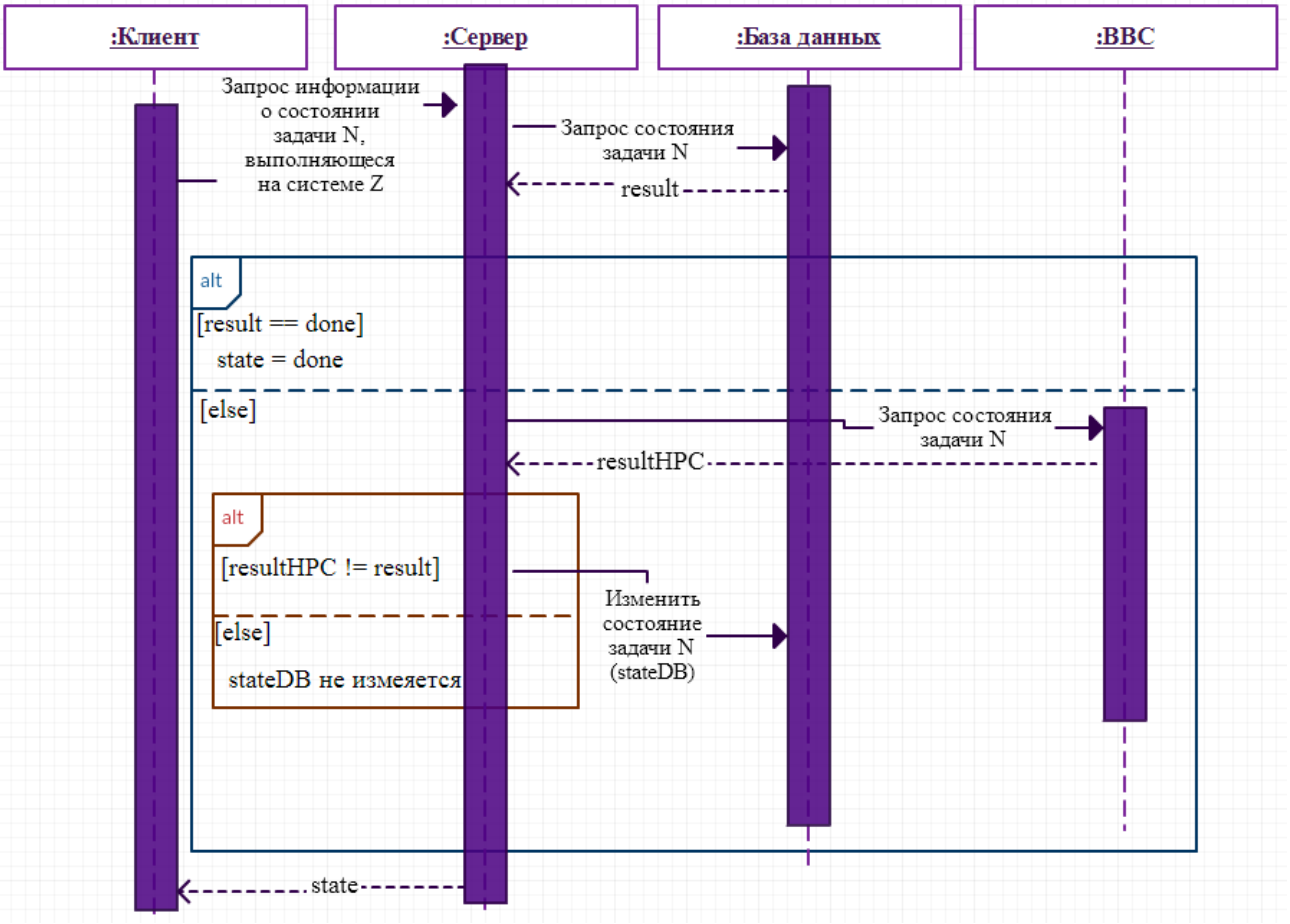
ПРИЛОЖЕНИЕ Ж

Пример взаимодействия при получении списка доступных приложений/задач/ВВС



ПРИЛОЖЕНИЕ З

Пример взаимодействия при запросе состояния выполняющейся задачи



ПРИЛОЖЕНИЕ И

Пример описания спецификации параметров в текстовом формате JSON для последующей генерации интерфейса задачи

```
{
"name":"hx",
"type":"double",
"min_value":0.0001,
"max_value":2
};
{
"name":"hy",
"type":"double",
"min_value":0.0001,
"max_value":2
};
{
"name":"hz",
"type":"double",
"min_value":0.0001,
"max_value":2
};
{
"name":"maxiter",
"type":"int",
"min_value":1,
"max_value":10000000
};
{
"name":"eps",
"type":"double",
"min_value":1e-16,
"max_value":0.1
}
```

```
// Описание параметров, определяющих «шаги сетки» (hx, hy, hz) имеющих тип вещественное число и находящихся в диапазоне [0.0001, 2]

// Описание параметра «максимальное количество итераций», имеющего тип целое число, находящееся в диапазоне [1, 10000000]

// Описание параметра «точность сходимости», имеющего тип вещественное число и находящегося в диапазоне [1e-16, 0.1]
```

ПРИЛОЖЕНИЕ К

1. Серверная часть

```
// Подключение express
const express = require("express");
const jwt = require('jsonwebtoken');
const jwt_express = require('express-jwt');
const basicAuth = require('basic-auth');
const sqlite3 = require('sqlite3').verbose();
const fs = require("fs");
const cookieParser = require('cookie-parser');
const mkdirp = require('mkdirp');
const child_process = require('child_process');

var jwtCheck = jwt_express({
  secret: 'supersecret'
});

// Создаём объект приложения
const app = express();

app.use(express.static('../react.js'));
app.use(express.json());
app.use(cookieParser());
app.use(jwtCheck.unless({path:['/favicon.ico', '/token','/users',
'/jobslist', '/allusers', 'jobslistss']}));

// Подключение базы данных
let db = new sqlite3.Database('./myBase.db', (err) => {
  if (err) {
    console.error(err.message);
  }
  console.log('Connected to the database.');
});
```

```
// Авторизация пользователя и создание для него токена
app.get("/token", function(request, response){
    var auth = basicAuth(request);
    let sql = "SELECT * FROM users WHERE login='" + auth.name + "' AND
password='" + auth.pass + "'";
    db.all(sql, [], (err, row) => {
        if (err) {
            console.log("Error in get users selecting from db");
            throw err;
            response.status(500);
        };
        if(row.length === 0) {
            response.status(401);
            //response.json({id:0});
            console.log("Error: empty request");
        }
        else {
            response.status(200);
            var token = jwt.sign({username:row[0].login}, 'superse-
cret', {expiresIn: 12000});
            response.json({token:token});
            console.log('user exist: '+row[0].id+' '+row[0].login+'
'+row[0].surname+' '+row[0].first_name+'\n');
        }
    });
});
// Регистрация нового пользователя
app.post("/users", function(request, response){
    db.run("INSERT INTO users (surname, first_name, second_name,
birthdate, gender, login, password, email) VALUES (?,?,?,?,?,?,?,?)",
request.body['surname'], request.body['first_name'], re-
quest.body['second_name'], request.body['birthdate'], re-
quest.body['gender'], request.body['login'], request.body['password'],
request.body['email'], function(err){
        if (err){
            if(err.errno === 19) {
                response.status(409);
```

```
                response.json({err:"UnUNICUE"});
            }
            else {
                response.status(500);
                response.json({err:"Inner error"});
            }
        }
        else {
            response.status(201);
            console.log("Add new user: " + this.lastID + ' ' + re-
quest.body['login'] + ' ' + request.body['email'] + ' ' + re-
quest.body['surname'] + ' ' + request.body['first_name']);
            response.json({id:this.lastID});
            mkdirp(process.cwd() + '/users/' + request.body['login'],
function(err) {
               if (err){
                   console.log("Error: imposible create this folder: "
+ request.body['login']);
               }
               else
                 console.log("Folder was created. " + re-
quest.body['login']);
            });
            mkdirp(process.cwd() + '/users/' + request.body['login'] +
'/applications/', function(err) {
               if (err){
                  console.log("Error: imposible create this folder: "
+ request.body['login'] + '/applications/');
              }
              else
                 console.log("Folder was created. " + re-
quest.body['login'] + '/applications/');
            });
        }
    });
});
// Добавление нового приложения
```

```
app.post("/applications", function(request, response){
    db.run("INSERT INTO applications (name, describe, parameters,
path_file) VALUES (?,?,?,?)", request.body['name'],
        request.body['describe'], request.body['parameters'], re-
quest.body['path_file'], function(err){
        if (err) {
            response.status(500);
            console.error(err.message);
            response.json({err:"Inner error"});
        }
        else {
            response.status(201);
            console.log("Add new application: " + this.lastID + ' ' +
request.body['name']);
            response.json({id:this.lastID});
            mkdirp(process.cwd() + '/users/' + request.body['login'] +
                '/applications/' + request.body['name'], function(err)
{
                if (err){
                  console.log("Error: imposible create this folder: "
+ 'applications/' + request.body['name']);
                }
                else {
                    console.log("Folder was created. " + 'applica-
tions/' + request.body['name']);
                    fs.writeFile(process.cwd() + '/users/' + re-
quest.body['login'] + '/applications/'
                        + request.body['name'] + "/field.json", re-
quest.body['parameters'], function(err) {

                        if(err) console.error(err.message);
                    });
                }
            });
        }
    });
});
```

```
// Получение списка доступных приложений для вывода в ЛК
app.get("/applications", function(request, response){
    let sql = `SELECT * FROM applications`;
    db.all(sql, [], (err, rows) => {
        if (err) {
            console.log("Error in get jobs selecting from db");
            throw err;
            response.status(500);
        }
        else {
            response.status(200);
        }
        console.log("Getting information about all available tasks");
        response.json(rows);
    });
});
app.get("/jobs", function(request, response){
    let sql = `SELECT * FROM jobs`;
    db.all(sql, [], (err, rows) => {
        if (err) {
            console.log("Error in get jobs selecting from db");
            throw err;
            response.status(500);
        }
        else {
            response.status(200);
        }
        console.log("Getting information about all available tasks");
        response.json(rows);
    });
});
// Информация о статусе задачи
app.get("/jobs/:id", function(request, response){
    var result = 'done';
    id = request.params["id"];
    var workerProcess = child_process.exec('scp serv-
er:~/webcloud/users/123/Poisson/results.txt
```

```
./users/123/jobs/myjob/results.txt',
        function(error, stdout, stderr) {
        if (error) {
            console.log(error.stack);
            console.log('Error code: '+error.code);
            console.log('Signal received: '+error.signal);
            result = 'error';
        }
        console.log('stdout: ' + stdout);
        console.log('stderr: ' + stderr);
        response.end(result);
    });
});
// Изменение статуса задачи
app.post("/jobs/:id", function(request, response){
    db.run("UPDATE jobs SET status = ? WHERE id = ?", re-
quest.body['status'],
        request.body['id'], function(err){
        if (err){
            console.error(err.message);
            response.status(500);
        }
        change = this.changes;
        response.json({update:change});
    });
});
// Поставить задачу на выполнение на ВВС
app.post("/jobs", function(request, response){
    console.log(request.body);
    db.run("INSERT INTO jobs (name, appID, input_param) VALUES
(?,?,?)", request.body['name'],
        request.body['appID'], request.body['input_param'], func-
tion(err){
        if (err){
            console.error(err.message);
            response.status(500);
        }
```

```
        else {
                var workerProcess = child_process.exec('cd scripts/;
./runjob.sh', function(error, stdout, stderr) {
                    if (error) {
                        console.log(error.stack);
                        console.log('Error code: '+error.code);
                        console.log('Signal received: '+error.signal);
                    }
                    console.log(`\nresults:\n`);
                    console.log('stdout: ' + stdout);
                    console.log('stderr: ' + stderr);
                    });
        }
        newrowID = this.lastID;
        response.json({id:newrowID});
    });
});
// Получение информации о конкретной задаче
app.post("/jobinfo", function(request, response){

    db.each('SELECT * FROM jobs WHERE id = ?', request.body['id'],
function(err, row) {
        if (err) {
            console.log("Error in get job selecting from db\n");
            response.status(500);
            console.error(err.message);
        }
        else {
            response.status(200);
            response.json(row);
        }
    });
});
app.get("/result", function(request, response){

    fs.readFile(process.cwd() + "/users/123/jobs/myjob/results.txt",
"utf8", function(err,data){
```

```
        if(err) console.error(err.message);
        response.json(data);
    });
});
app.get("/systems", function(request, response){
    let sql = `SELECT * FROM systems`;
    db.all(sql, [], (err, rows) => {
        if (err) {
            console.log("Error in get users selecting from db");
            throw err;
        }
        console.log("Getting information about all available computing
systems");
        response.json(rows);
    });
});

app.post("/systems", function(request, response){
    console.log(request.body);
    db.run("INSERT INTO systems (name) VALUES (?)", re-
quest.body['name'], function(err){
        if (err){
            console.err(err);
            response.status(500);
        }
        else {
            response.status(201);
        }
        newrowID = this.lastID;
        response.json({id:newrowID});
    });
});

// Начинаем прослушивать подключения на 3000 порту
app.listen(3000, function () {
   console.log('Server listening on port 3000!');
});
```

2. Функции работы с сервером

```
// Получение кукисов
function getCookie(name) {
  var value = "; " + document.cookie;
  var parts = value.split("; " + name + "=");
  if (parts.length == 2) return parts.pop().split(";").shift();
};
function authorizedRequest(requestType, string) {
    var xhr = new XMLHttpRequest();
    xhr.open(requestType, string, false);
    var token = getCookie("token");
    xhr.setRequestHeader("Authorization", "Bearer " + token);
    return xhr;
}
// Получение списка задач и/или вычислительных систем
function GetList (string) {
    // Создаём новый объект XMLHttpRequest
    var xhr = authorizedRequest('GET', string);
    xhr.send();

    if (xhr.status !== 200) {
        alert( xhr.status + ': ' + xhr.statusText );
    }
    else {
        var list = JSON.parse(xhr.responseText);
        return list;
    }
};
// Авторизация пользователя
function DataForInput (login, password) {
    // Создаём новый объект XMLHttpRequest
    var xhr = new XMLHttpRequest();
    xhr.open('GET', '/token', false);
    xhr.setRequestHeader("Authorization", "Basic " +
btoa(login+":"+password));
```

```
    xhr.send();

    if (xhr.status !== 200 && xhr.status !== 401) {
        alert( xhr.status + ': ' + xhr.statusText );
    }
    else {
        if (xhr.status === 401)
            alert("Error: wrong login/password ");
        else {
            console.log("token user:
"+JSON.parse(xhr.responseText).token);
            document.cookie = "token=" +
JSON.parse(xhr.responseText).token;
              document.cookie = "login=" + login;
            window.location.href = "personalAccount.html";
        }
    }
};
// Регистрация нового пользователя
function AddNewUser (jsonstr) {
    // Создаём новый объект XMLHttpRequest
    var xhr = new XMLHttpRequest();

    xhr.open('POST', '/users', false);
    xhr.setRequestHeader("Content-Type", "application/json");

    var data = JSON.stringify(jsonstr);
    xhr.send(data);

    if (xhr.status !== 201 && xhr.status !== 409) {
        console.log( xhr.status + ': ' + xhr.statusText + '\n' +
JSON.parse(xhr.responseText).err);
        console.log("Error: internal server error.");
    }
    else {
        if(xhr.status === 409) {
            console.log( xhr.status + ': ' +
```

```
JSON.parse(xhr.responseText).err);
            alert("Error: this login/email already exist.");
        }
        else {
            var result = JSON.parse(xhr.responseText);
            console.log('id new user: ' + result.id);
            window.location.href="success.html";
        }
    }
}
// Добавление нового приложения
function AddNewApplication(jsonstr) {
    var xhr = authorizedRequest('POST', '/applications');
    xhr.setRequestHeader("Content-Type", "application/json");
    jsonstr.login = getCookie("login");
    var data = JSON.stringify(jsonstr);
    xhr.send(data);

    if (xhr.status !== 201) {
        alert( xhr.status + ': ' + xhr.statusText );
    }
    else {
        var result = JSON.parse(xhr.responseText);
        console.log('id new application: ' + result.id);
        alert('Ваше приложение успешно добавлено!');
        window.location.href="personalAccount.html";
    }
}
// Получение данных о приложении
function GetTask(id) {
    var xhr = authorizedRequest('POST', '/appinfo');
    xhr.setRequestHeader("Content-Type", "application/json");
    var data = JSON.stringify(id);
    xhr.send(data);

    if (xhr.status !== 200) {
        alert( xhr.status + ': ' + xhr.statusText );
```

```
        console.log("Error: internal server error.");
    }
    else {
        var result = JSON.parse(xhr.responseText);
        return result;
    }
}
function CheckStatus(status, id) {

    var xhr = authorizedRequest('GET', '/jobs/:'+id);
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.send();

    if(xhr.responseText === 'done') {
        status = 'done';
        var xhr2 = authorizedRequest('POST', '/jobs/:'+id);
        xhr2.setRequestHeader("Content-Type", "application/json");

        var jsonstr = new Object();
        jsonstr.id = id;
        jsonstr.status = 'done';
        var data = JSON.stringify(jsonstr);
        xhr2.send(data);

        if (xhr2.status !== 200) {
            alert( xhr.status + ': ' + xhr2.statusText );
            console.log("Error: internal server error.");
        }
        else {
            var result = JSON.parse(xhr2.responseText);
            alert(result.update);
        }
    }
    return status;
}
// Поставить выполнение задачи на ВВС
function StartJob(jsonstr) {
```

```
    var xhr = new XMLHttpRequest();
    xhr.open('POST', '/jobs', true);
    var token = getCookie("token");
    xhr.setRequestHeader("Authorization", "Bearer " + token);

    xhr.setRequestHeader("Content-Type", "application/json");
    var data = JSON.stringify(jsonstr);
    xhr.send(data);

    // callback для асинхронного запроса
    xhr.onreadystatechange = function() {
        if (xhr.readyState != 4) return;

        if (xhr.status !== 200) {
            alert( xhr.status + ': ' + xhr.statusText );
            console.log("Error: internal server error.");
        }
        else {
            var result = JSON.parse(xhr.responseText);
            return result;
        }
    }
}
// Получение данных о задаче
function GetJob(id) {
    var xhr = authorizedRequest('POST', '/jobinfo');
    xhr.setRequestHeader("Content-Type", "application/json");
    var data = JSON.stringify(id);
    xhr.send(data);

    if (xhr.status !== 200) {
        alert( xhr.status + ': ' + xhr.statusText );
        console.log("Error: internal server error.");
    }
    else {
        var result = JSON.parse(xhr.responseText);
        return result;
```

```
    }
}
function ViewResult() {
    var xhr = authorizedRequest('GET', '/result');
    xhr.setRequestHeader("Content-Type", "application/json");
    xhr.send();
    var result = JSON.parse(xhr.responseText);
    return result;
}
```

3. База данных

```
CREATE TABLE users (
    id integer PRIMARY KEY,
    surname text NOT NULL,
    first_name text NOT NULL,
    second_name text,
    birthdate text,
    gender text,
    login text NOT NULL UNIQUE,
    password text NOT NULL,
    email text NOT NULL UNIQUE
);
INSERT INTO users (surname, first_name, second_name, birthdate, gen-
der, login, password, email) VALUES
    ('Иванов', 'Иван', 'Иванович', '10.03.1997', 'м', 'username',
'qwerty', 'username@users.ru'),
    ('Сидорова', 'Мария', null, null, null, 'user1', 'qwerty123',
'user1@users.ru'),
    ('1123', '1223', '1233', null, null, '123', '123',
'123@users.ru');
CREATE TABLE applications (
    id integer PRIMARY KEY,
    name text NOT NULL,
    describe text,
    parameters blob NOT NULL,
    path_file text NOT NULL
```

```
);
CREATE TABLE jobs (
    id integer PRIMARY KEY,
    name text NOT NULL,
    appID integer NOT NULL,
    input_param text NOT NULL
);
CREATE TABLE systems (
    id integer PRIMARY KEY,
    name text NOT NULL
);
INSERT INTO systems (name) VALUES
    ('кластер'),
    ('локальная машина');
```

4. Веб-страницы интерфейса

4.1 index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="UTF-8" />
    <title>Вход</title>
    <script
src="https://unpkg.com/react@16/umd/react.development.js"></script>
    <script src="https://unpkg.com/react-dom@16/umd/react-
dom.development.js"></script>
    <script src="https://unpkg.com/babel-
standalone@6.15.0/babel.min.js"></script>
    <script src="funcForServer.js"></script>
    <link rel="stylesheet" type="text/css" href="myStyle.css">
  </head>
  <body>

    <div id="example"></div>

    <script type="text/babel">
```

```
    function Input() {
        var login = document.getElementById("login").value;
        var password = document.getElementById("password").value;
        DataForInput(login, password);
    }

    const app = document.getElementById('example');
    ReactDOM.render(
    <div>
        <div className = "boxTitle">
            Web cloud
        </div>
        <div className = "boxBody">
        <br />
            <p className="in"> Логин:
                <input type="text" className="start" id="login"/>
            </p>
            <p className="in">Пароль:
                <input type="password" class-
Name="start"id="password"/>
            </p> <br />
            <a href="">Забыли пароль</a>
            <a href="registration.html">Зарегистрироваться</a>
            <button className="btn in" onClick={Input}>Войти</button>
        </div>
    </div>
    , app);
    </script>
  </body>
</html>
```

Примечание – остальные страницы содержат основные модули, используемые в тексте

4.2 registration.html

```
function SuccessReg() {
    var surname = document.getElementById("surname").value;
    var name = document.getElementById("first_name").value;
```

```
    var sec_name = document.getElementById("second_name").value;
    var birthdate;
    if(document.getElementById("day").value == docu-
ment.getElementById("month").value == docu-
ment.getElementById("year").value == 0)
       birthdate = "";
    else
       birthdate = document.getElementById("day").value + '.' + docu-
ment.getElementById("month").value + '.' + docu-
ment.getElementById("year").value;
   var gender;
   if(document.getElementById("female").checked)
      gender = 'ж';
    else
      if(document.getElementById("male").checked)
           gender = 'м';
      else
           gender = "";
    var login = document.getElementById("login").value;
    var password = document.getElementById("password").value;

    var email = document.getElementById("email").value;
    var check = email.split('@');
    if(check.length == 1)
        alert('Error: check your email!');
    else {
        var jsonstr = new Object();
        jsonstr.surname = surname;
        jsonstr.first_name = name;
        jsonstr.second_name = sec_name;
        jsonstr.birthdate = birthdate;
        jsonstr.gender = gender;
        jsonstr.login = login;
        jsonstr.password = password;
        jsonstr.email = email;
        AddNewUser(jsonstr);
    }
```

```
}
function InS () {
   var id = "surname";
   InputData(id);
}
function InF () {
   var id = "first_name";
   InputData(id);
}
function InL () {
    var id = "login";
    InputData(id);
}
function InE () {
    var id = "email";
    InputData(id);
}
function InputData (id) {
    if( document.getElementById(id).value.length == 0)
         document.getElementById(id).className = 'start registr';
    else
         document.getElementById(id).className = 'start';
}
function CorrectPassword() {
    var password = document.getElementById("password").value;
    var password_two= document.getElementById("password_two").value;
    if(password == password_two && password.length!=0 && pass-
word_two.length!=0) {
       document.getElementById("password").style.borderColor='green';
       docu-
ment.getElementById("password_two").style.borderColor='green';
     }
     else {
        document.getElementById("password").style.borderColor='red';
        docu-
ment.getElementById("password_two").style.borderColor='red';
        }
```

```
}
```

4.3 personalAccount.html

```
function NameUser() {
    var name = getCookie("login");;
    document.getElementById('nameUser').innerHTML = name;
    ListApplications();
    QueueTasksFirst();
    ListSystems();
}
function ListApplications () {
    var list = GetList('/applications');
        for (var i = 0; i < list.length; i++) {
            var name = list[i].name;
            var myLi = document.createElement('li');
            var linkA = document.createElement('a');
            var textElem = document.createTextNode(name);
            linkA.setAttribute('href', 'prototypeTasks.html?id=' +
list[i].id);
            linkA.innerHTML = name;
            myLi.appendChild(linkA);
            document.getElementById('myTasks').appendChild(myLi);
        }
}
function QueueTasksFirst () {
    document.getElementById('queueTasks').innerHTML = '';
    var list = GetList('/jobs');
    for (var i = 0; i < list.length; i++) {
            var name = list[i].name;
            var status = list[i].status;
            var myLi = document.createElement('li');
            var linkA = document.createElement('a');
            var textElem = document.createTextNode(' [ '+status+' ]');
            linkA.setAttribute('href', 'viewTask.html?id=' + list[i].id);
            linkA.innerHTML = name;
            myLi.appendChild(linkA);
            myLi.appendChild(textElem);
```

```
            document.getElementById('queueTasks').appendChild(myLi);
     }
}
function QueueTasks () {
    document.getElementById('queueTasks').innerHTML = '';
    var list = GetList('/jobs');
    for (var i = 0; i < list.length; i++) {
       var name = list[i].name;
       var status = list[i].status;
       var myLi = document.createElement('li');
       var linkA = document.createElement('a');
       if(status === 'in_queue' || status === 'running')
            status = CheckStatus(status, list[i].id);
            var textElem = document.createTextNode(' [ '+status+' ]');
            linkA.setAttribute('href', 'viewTask.html?id=' + list[i].id);
            linkA.innerHTML = name;
            myLi.appendChild(linkA);
            myLi.appendChild(textElem);
            document.getElementById('queueTasks').appendChild(myLi);
      }
}
setInterval(()=>{QueueTasks()}, 10000);
function ListSystems () {
   var list = GetList('/systems');
   for (var i = 0; i < list.length; i++) {
        var name = list[i].name;
        var myLi = document.createElement('li');
        var linkA = document.createElement('a');
        var textElem = document.createTextNode(name);
        linkA.setAttribute('href', "newSystem.html");
        linkA.innerHTML = name;
        myLi.appendChild(linkA);
        document.getElementById('listSystem').appendChild(myLi);
   }
}
class Clock extends React.Component {
   constructor (props) {
```

```
  super (props);
  this.state = {
      hour: this.correctHours(),
       minute: this.correctMinutes(),
       second: this.correctSeconds()
   };
   this.dynamicClock();
};
dynamicClock=()=> {
    setInterval(()=>{
            this.setState({
            hour: this.correctHours(),
            minute: this.correctMinutes(),
            second: this.correctSeconds()
            });
   }, 1000);
};
correctHours=()=> {
   var h=(new Date()).getHours();
   if (h<10)
       return '0'+h;
   else return h;
}
correctMinutes=()=> {
    var m=(new Date()).getMinutes();
     if (m<10)
         return '0'+m;
     else return m;
}
correctSeconds=()=> {
        var s=(new Date()).getSeconds();
        if (s<10)
            return '0'+s;
        else return s;
}
render () {
   return (
```

```
        <h2 className="ttime">
           {this.state.hour}:
           {this.state.minute}:
           {this.state.second}
        </h2>
        );
    }
}
class MyDate extends React.Component {
   constructor (props) {
        super (props);
         this.state = {
            day: this.correctDay(),
            month: this.correctMonth(),
            year: (new Date()).getFullYear()
         };
        this.dynamicData();
    };
    dynamicData=()=> {
            setInterval(()=>{
                this.setState({
                    day: this.correctDay(),
                    month: this.correctMonth(),
                    year: (new Date()).getFullYear()
                })
            }, 1000);
    };
    correctMonth=()=> {
        var m = (new Date()).getMonth() + 1;
        if (m<10)
            return '0'+m;
        else return m;
    }
    correctDay=()=> {
        var d=(new Date()).getDate();
        if (d<10)
            return '0'+d;
```

```
        else return d;
     }
     render () {
       return (
          <h2 className="ttime">
             {this.state.day}.
             {this.state.month}.
             {this.state.year}
           </h2>
        );
    }
}
function GoToTask() {
   window.location.href = "newTask.html";
}
function GoToStart() {
      document.cookie = "token=''";
      document.cookie = "login=''";
      window.location.href = "index.html";
}
function GoToSystem() {
      window.location.href = "newSystem.html";
}
```

4.4 newTask.html

```
function GoToPersonalAcc() {
        window.location.href="personalAccount.html"
}
function AddApp() {
   var name = document.getElementById("nameTask").value;
   var describe = document.getElementById("taskArea").value;
   var path_file = document.getElementById("pathFiles").value;
   var jsonstr = new Object();
   jsonstr.name = name;
   jsonstr.describe = describe;
   jsonstr.path_file = path_file;
   jsonstr.parameters = JSON.stringify(parameters);
```

```
    AddNewApplication(jsonstr);
}
// Добавление параметров
function AddParam() {
    var text = document.getElementById("paramTask").value;
    var mas = text.split(';');
    var data = [];
    for (var i = 0; i < mas.length; i++) {
            data[i] = JSON.parse(mas[i]);
    }
    var check = document.getElementsByName('param');
    var all = document.getElementsByName('allParam');
    var count;
    for (var i = 0; i < check; i++) {
        if(check[i].checked) count ++;
     }
     check = check.length;
     if(document.getElementById('pp').checked && (check == 0 || check
!= count)) {
            all[0].checked = false;
     }
     /* Добавление введённых параметров и вывод их на экран */
     for (var i = 0; i < data.length; i++) {
        var label = document.createElement('label');
         var checkbox = document.createElement('input');
         var textElem = document.createTextNode(data[i].name + '
('+data[i].type+') от ' + data[i].min_value + ' до
'+data[i].max_value);
          var br = document.createElement('br');
          var param = new Object();
          param.name = data[i].name;
          param.type = data[i].type;
          param.min_value = data[i].min_value;
          param.max_value = data[i].max_value;
          parameters.push(param);
          checkbox.type = "checkbox";
          checkbox.name = "param";
```

```
            checkbox.className = "addParameters";
            checkbox.id = "p" + i;
            checkbox.value = textElem;
            label.appendChild(checkbox);
            label.appendChild(textElem);
            document.getElementById('newParam').appendChild(label);
            document.getElementById('newParam').appendChild(br);
        }
}
// Выбор всех параметров
function CheckAll () {
    var elements = document.getElementsByName('param');
    if(document.getElementById('pp').checked) {
       for (var i = 0; i < elements.length; i++) {
           elements[i].checked = true;
       }
    }
    else
      for (var i = 0; i < elements.length; i++)
        elements[i].checked = false;
}
function DeleteParam () {
     var allcheckboxes = document.getElementById('newParam');
      var label = allcheckboxes.getElementsByTagName('label');
      // Удаление всех чекбоксов, если выбран параметр "все"
      if(document.getElementById('pp').checked) {
         allcheckboxes.innerHTML = '';
         parameters = [];
      }
      else {
          var newcheckboxes = [];
          var param = [];
          // Формирование не выбранных чекбоксов
          for (var i = 0; i < label.length; i++) {
             if(!label[i].children[0].checked) {
                newcheckboxes.push(label[i]);
                param.push(parameters[i]);
```

```
            }
        }
        allcheckboxes.innerHTML = '';
        parameters = param;
        for (var i = 0; i < newcheckboxes.length; i++) {
            allcheckboxes.appendChild(newcheckboxes[i]);
            allcheckboxes.appendChild(document.createElement('br'));
        }
    }
}
```

4.5 prototypeTask.html

```
function TaskInfo() {
   var id = new Object();
   id.name = window.location.search.replace( '?', '').split('=')[0];
   id.id = window.location.search.replace( '?', '').split('=')[1];
   var task = GetTask(id);
   document.getElementById('nameTask').innerHTML = task.name;
   document.getElementById('description').innerHTML = task.describe;
   document.cookie = "taskname=" + task.name;
   document.cookie = "taskid=" + task.id;
}
function StartJob() {
   window.location.href="firstTab.html?id=" + getCookie('taskid');
}
function GoToPersonalAcc() {
   document.cookie = "taskname='";
   document.cookie = "taskid='";
   window.location.href="personalAccount.html"
}
```

4.6 firstTab.html

```
var save = new Object();
function CountFiles() {
    var allfiles = document.getElementById("newFiles");
    if (allfiles == 0)
        document.getElementById("info").innerHTML='Файлы не выбраны';
```

```
    else {
        var str = ' ';
        for(var i = 0; i < allfiles.files.length; i++) {
            str += allfiles.files[i].name;
            str += ', ';
        }
        if (allfiles.files.length === 0)
           document.getElementById("info").innerHTML='Файлы не
выбраны';
        else
           document.getElementById("info").innerHTML='Выбранный(-е)
файл(-ы):' + str;
        save.files = str;
     }
}
function Data() {
        var login = getCookie('login');
        var data = JSON.parse(sessionStorage.getItem(login + 'data')); //
получение данных из сессии
        if(data != null) {
            document.getElementById('name').value = data.name;
        }
}
// Сохранение введённых параметров
function Save() {
    save.name = document.getElementById('name').value;
    var data = JSON.stringify(save);
    var login = getCookie('login');
    sessionStorage.setItem(login + 'data', data);
}
setInterval(()=>{ Save(); }, 1000);
function GoToPersonalAcc() {
   document.cookie = "taskname=''";
   document.cookie = "taskid=''";
   sessionStorage.clear();
   window.location.href="personalAccount.html"
```

```
}
```

4.7 secondTab.html

```
var save = new Object();
var params;
// Сохранение введённых параметров
function SaveParam() {
   var data;
   var name = [];
   var value = [];
   for( var i = 0; i < params.length; i++) {
        var param = params[i].split(/[,":]+/);
        data = document.getElementById("p" + i).value;
        name.push(param[2]);
        value.push(data);
    }
    save.name = name;
    save.value = value;
    var login = getCookie('login');
    sessionStorage.setItem(login + 'param', JSON.stringify(save));
}
setInterval(()=>{ SaveParam(); }, 1000);
// Генерация параметров, необходимых для запуска задачи
function ParametersInfo() {
    var id = new Object();
    id.id = getCookie('taskid');
    var task = GetTask(id);
    params = task.parameters.split('[{');
    params = params[1].split('}]');
    params = params[0].split('},{');
    for (var i = 0; i < params.length; i++) {
        var param = params[i].split(/[,":]+/);
        var myP = document.createElement('p');
        var myInput = document.createElement('input');
        myP.className = "task status next";
        myP.id = "i" + i;
        myP.innerHTML = param[2] + ' (' + param[4] + '):';
```

```
        myInput.name = "param";
        myInput.className = "start startup next";
        myInput.id = "p" + i;
        if(param[4] === 'double' || 'int') {
           myInput.type = "number";
           myInput.min = param[6];
           myInput.max = param[8];
           if (param[4] === 'double')
               myInput.step = 0.01;
           else
               myInput.step = 1;
        }
        if(param[4] === 'string') {
           myInput.type = "text";
        }
        myP.appendChild(myInput);
        document.getElementById('Params').appendChild(myP);
   }
}
function GoToPersonalAcc() {
   document.cookie = "taskname='';
   document.cookie = "taskid='';
   sessionStorage.clear();
   window.location.href="personalAccount.html"
}
```

4.8 thirdTab.html

```
// Сохранение введённых параметров
function SaveSystems() {
   var save = new Object();
   save.system = docu-
ment.getElementById("system").options[document.getElementById("system").
selectedIndex].text;
   save.core = document.getElementById("core").value;
   save.thread = document.getElementById("thread").value;
   var login = getCookie('login');
   sessionStorage.setItem(login + 'system', JSON.stringify(save));
```

```
}
setInterval(()=>{ SaveSystems(); }, 1000);
function SystemsInfo() {
   document.getElementById('system').innerHTML = '';
   var list = GetList('/systems');
   var myOptionZero = document.createElement('option');
   myOptionZero.setAttribute('value', 0);
   myOptionZero.innerHTML = 'Выберите систему';
   var login = getCookie('login');
   var systems = JSON.parse(sessionStorage.getItem(login + 'system'));
   if(systems != null) {
       document.getElementById("core").value = systems.core;
       document.getElementById("thread").value = systems.thread;
    }
    else {
       myOptionZero.setAttribute('selected', name);
    }
    document.getElementById('system').appendChild(myOptionZero);
    for (var i = 0; i < list.length; i++) {
        var name = list[i].name;
        var myOption = document.createElement('option');
        if(name === system.system) {
            myOption.setAttribute('selected', name);
        }
         myOption.setAttribute('value', i + 1);
         myOption.innerHTML = name;
         document.getElementById('system').appendChild(myOption);
     }
}
function GoToPersonalAcc() {
    document.cookie = "taskname=''";
    document.cookie = "taskid=''";
    sessionStorage.clear();
    window.location.href="personalAccount.html"
}
function InfoAboutSystem () {
    var name = document.getElementById('system');
```

```
    var id = name.options[name.selectedIndex].value;
    if(id === '0') {
        document.getElementById('infcore').value = "";
        document.getElementById('infthread').value = "";
     }
     if(id === '1') {
        document.getElementById('infcore').value = "8";
        document.getElementById('infthread').value = "1";
     }
     if(id === '2') {
        document.getElementById('infcore').value = "20";
        document.getElementById('infthread').value = "8";
     }
}
// Выполнить задачу
function RunJob() {
   var login = getCookie('login');
   var jsonstr = new Object();
   var data = JSON.parse(sessionStorage.getItem(login + 'data'));
   var param = JSON.parse(sessionStorage.getItem(login + 'param'));
   var system = JSON.parse(sessionStorage.getItem(login + 'system'));
   jsonstr.name = data.name;
   jsonstr.appID = getCookie('taskid');
   if (param == null)
       jsonstr.input_param = JSON.stringify("{'param':''}");
   else {
       jsonstr.input_param = JSON.stringify("{'param':'aaaa'}");
   }
   jsonstr.core = system.core;
   jsonstr.thread = system.thread;
   console.log(jsonstr);
   var name = document.getElementById('system');
   var id = name.options[name.selectedIndex].value;
   if(id === '0') {
       alert('Выберите систему для запуска задачи!');
   }
   if(id === '1' || id === '2') {
```

```
        if(id === '1') {
            jsonstr.status = 'running';
            jsonstr.hpcsystem = 'сервер ИВМиМГ';
        }
        else {
            jsonstr.status = 'in_queue';
            jsonstr.hpcsystem = 'кластер';
        }
        StartJob(jsonstr);
        alert('Ваша задача успешно запущена!');
        document.cookie = "taskname=''";
        document.cookie = "taskid=''";
        sessionStorage.clear();
        window.location.href="personalAccount.html"
    }
}
```

4.8 viewTask.html

```
function JobInfo() {
    var id = new Object();
    id.name = window.location.search.replace( '?', '').split('=')[0];
    id.id = window.location.search.replace( '?', '').split('=')[1];
    var job = GetJob(id);
    id.id = job.appID;
    var task = GetTask(id);
    document.getElementById('nameJob').innerHTML = job.name;
    document.getElementById('nameTask').innerHTML = task.name;
    document.getElementById('nameHPC').innerHTML = 'Была запущена на
ВВС: ' + job.hpcsystem;
    document.cookie = "taskid=" + task.id;
    document.cookie = "jobid=" + job.id;
    var text = ViewResult('res');
    document.getElementById('res').innerHTML = text;
}
```