МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра параллельных вычислений

Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Александрова Евгения Александровича

Тема работы:

РАЗРАБОТКА ПРОТОТИПА ОБЛАЧНОЙ IDE ДЛЯ СИСТЕМЫ LUNA

«К защите допущена»
зав. каф. ПВ ФИТ НГУ,
д.т.н., профессор
Малышкин В. Э./……..
(ФИО) / (подпись)
«27» мая 2021 г.

Руководитель ВКР
д.т.н, профессор,
зав. каф. ПВ ФИТ НГУ
Малышкин В. Э./……...
(ФИО) / (подпись)
«27» мая 2021 г.

Соруководитель
ст. преп. каф. ПВ ФИТ НГУ
Перепёлкин В. А. /……...
(ФИО) / (подпись)
«27» мая 2021 г.

Новосибирск, 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)
Факультет информационных технологий
Кафедра параллельных вычислений
Направление подготовки 09.03.01 Информатика и вычислительная техника
Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

зав. каф. ПВ ФИТ НГУ
Малышкин В. Э.
(фамилия, И., О.)
…….
(подпись)
«17» декабря 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Александрову Евгению Александровичу, группы 17201
(фамилия, имя, отчество, номер группы)

Тема: Разработка прототипа облачной IDE для системы LuNA
(полное название темы выпускной квалификационной работы)

утверждена распоряжением проректора по учебной работе от 17.12.2020 № 0451

Срок сдачи студентом готовой работы 31 мая 2021 г.

Исходные данные (или цель работы): разработать облачную IDE для системы фрагментированного программирования LuNA, снижающей порог вхождения в ее использование для новых пользователей.

Структурные части работы: Обзор существующих IDE. Разработка требований к IDE. Разработка функциональности IDE. Разработка архитектуры IDE. Реализация LuNA Web Server и Front-end Application.

Консультанты по разделам ВКР (при необходимости, с указанием разделов):
……
(раздел, ФИО)

Руководитель ВКР
зав. каф. ПВ ФИТ НГУ,
д.т.н., профессор
Малышкин В. Э./……...
(ФИО) / (подпись)
«17» декабря 2020 г.

Соруководитель
ст. преп. каф. ПВ ФИТ НГУ
Перепёлкин В. А. /……...
(ФИО) / (подпись)
«17» декабря 2020 г.

Задание принял к исполнению
Александров Е. А. /……..
(ФИО студента) / (подпись)
«17» декабря 2020 г.

Содержание

Определения

HPC Community Cloud - интерфейс, посредством которого пользователи могут получить доступ к высокопроизводительным вычислительным ресурсам в соответствии с разрешенным уровнем доступа, поставить вычислительное задание в очередь, посмотреть статусы выполняющихся задач, визуализировать результаты завершенных задач или же загрузить их себе на компьютер.

LuNA Web Server - отдельный слабосвязанный компонент вычислительной системы с установленной системой LuNA, который взаимодействует по API с облачной IDE.

Web RTS - система распределенного исполнения в веб-браузерах параллельных программ на базе системы LuNA. Для запуска программы в Web RTS необходим файл “program.ja” скомпилированной LuNA программы, атомарные фрагменты кода реализуются на JavaScript.

Введение

В наше время существует много систем для разработки программ, которые сложны в использовании. Каждый желающий познакомиться с такой системой, должен потратить достаточное количество времени, чтобы разобраться с её интерфейсом. Когда пользователь видит какой объем работы ему нужно сделать, чтобы запустить самый простой тестовый пример, это отталкивает его от изучения данной технологии. Не все разработчики систем, языков программирования, создают онлайн IDE для них. Онлайн IDE решает проблемы установки необходимого ПО, хранения файлов, обновление софта. Создание облачной IDE — это комплексная проблема, включающая в себя разработку архитектуры облачной IDE, требований, которые удовлетворяют потребностям пользователей, а также удобного, интуитивно понятного для пользователя интерфейса.

LuNA (Language for Numerical Algorithms) - система фрагментированного программирования, предназначенная для поддержки параллельной реализации больших численных моделей для суперкомпьютеров. LuNA нуждается в такой облачной IDE. Разработка онлайн IDE для LuNA – практически значимо, а также внесет вклад в будущую проработку онлайн IDE для аналогичных систем.

В группу пользователей суперкомпьютеров входят специалисты в своих предметных областях, которые используют численное моделирование для решения поставленных перед ними задач. Это может быть военная, медицинская, банковская, научная область. Для специалистов в таких сферах недостаточно будет запустить программу и посмотреть ее результат, нужен более продвинутый функционал. Например: подсветка синтаксиса, отображение ошибок компиляции/программы, пошаговое исполнение программы, взаимодействие с системой визуализации фрагментированной программы, запуск программ на кластере.

Цель работы - разработать облачную IDE для системы фрагментированного программирования LuNA.

Задачи

1. Разработать требования к IDE.
2. Разработать функциональность.
3. Разработать архитектуру IDE.
4. Реализовать front-end часть IDE.
5. Разработать вычислительную систему.
6. Провести тестирование.

Глава 1. Разработка требований и архитектуры IDE

1.1 Обзор существующих IDE

Проанализируем существующие характерные примеры облачных IDE, выявим общие черты и особенности, которые можно использовать при разработке облачной IDE для системы LuNA.

Для языка Python существует сайт с консолью, где каждый может попробовать запустить какие-либо команды. Основное преимущество такой среды разработки – это простота использования, для работы с ней не нужно ничего устанавливать, необходим лишь доступ в интернет. Рассмотрим ключевые из них.

Python - высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

python.org – официальный сайт, где можно познакомиться с этим языком. На рисунке 1 можно увидеть начальное изображение сайта. Сразу бросается в глаза самый большой элемент на сайте – командная строка с пятью примерами, с помощью которых, можно познакомиться с синтаксисом, операциями, работой со списками, определением функций. А нажав на желтую кнопку «Launch interactive shell», загружается командная строка, где уже можно попробовать данные примеры или поэкспериментировать со своими программами.

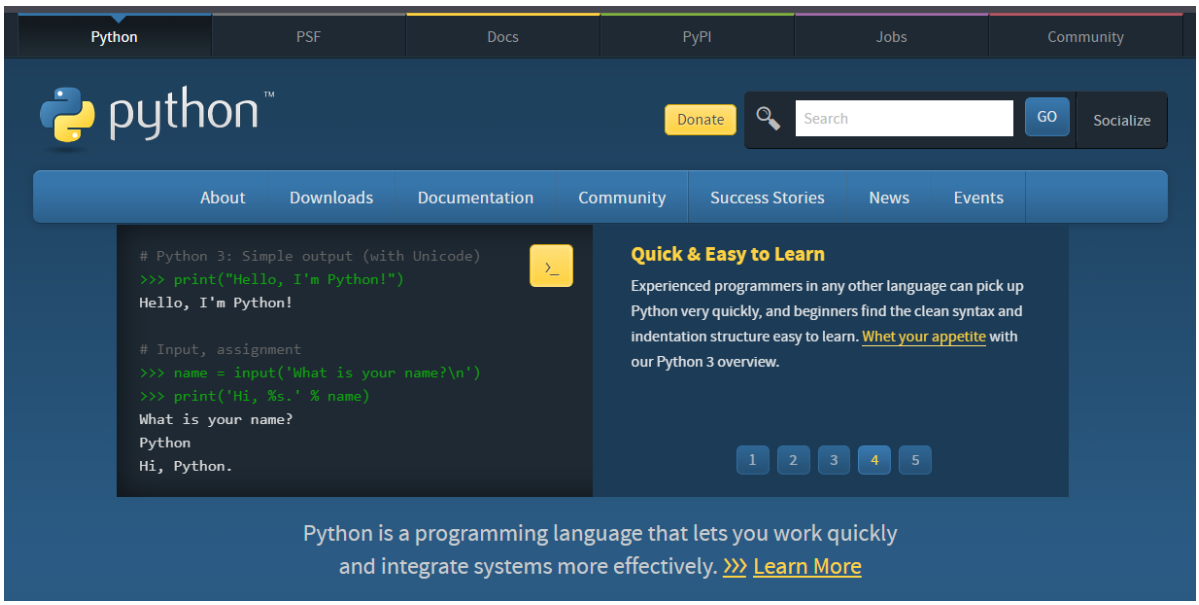

Рисунок 1 - Изображение сайта python.org

Над командной строкой можно увидеть несколько активных кнопок: About, Downloads, Documentation, Community, Success Stories, News, Events. В первой вкладке About можно узнать где используется язык Python, в каких framework является основой, что предоставляет стандартная библиотека. В Downloads можно найти файл установки для нужной операционной системы (Windows, MacOS, Linux и т.д.) различных версий от 2.2 до 3.8. Documentation содержит официальную документацию для версий 2.x и 3.x, tutorials, guides. В Community можно найти ответы на часто задаваемые вопросы. Success Stories содержит статьи, где рассказывается об опыте применения Python в образовании, инженерии, науке и бизнесе. News и Events содержат новости и различные мероприятия, проводимые разработчиками, соответственно.

Golang – компилируемый многопоточный язык программирования, разработанный внутри компании Google. Язык Go разрабатывался как язык программирования для создания высокоэффективных программ, работающих на современных распределённых системах и многоядерных процессорах. Основными его особенностями являются: простой и понятный синтаксис, скорость и компиляция, богатая стандартная библиотека.

play.golang.org – сайт, где можно познакомиться с возможностями этого языка и запустить предлагаемые примеры. На рисунке 2 представлен снимок

сайта. Интерфейс гораздо проще чем у python.org, снизу текстовой поле, куда нужно вписывать код, а на панели сверху – несколько управляющих кнопок. Но и тут мы можем увидеть, что есть несколько примеров, которые помогают пользователю познакомиться с синтаксисом, решают стандартные проблемы, используя стандартную библиотеку. Стоит отметить, что, нажав на кнопку About, можно найти подробную информацию о языке, стандартной библиотеке, инструкцию по установке.

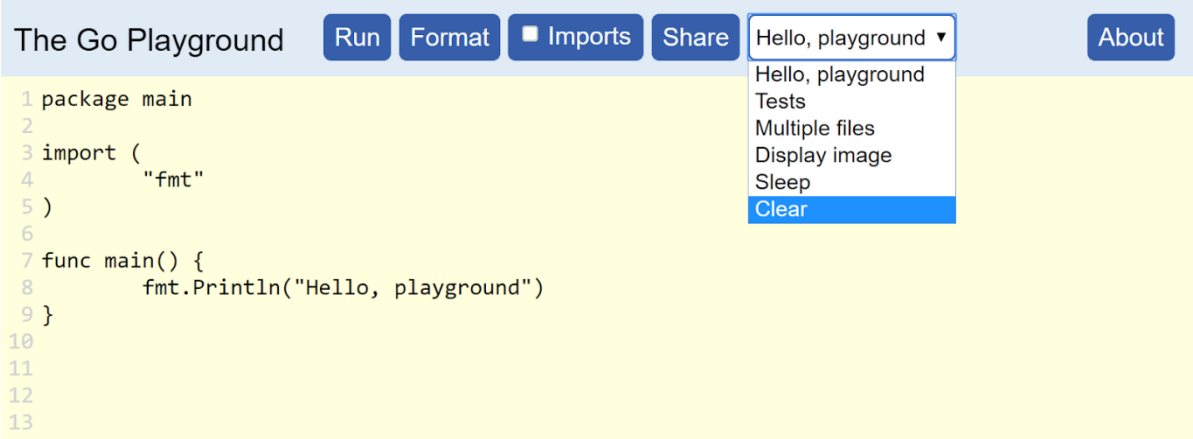

Рисунок 2 - Изображение сайта play.golang.org

Посмотрим на полноценную среду IntelliJ IDEA и выявим какие возможности она предоставляет. IntelliJ IDEA — интегрированная среда разработки программного обеспечения для многих языков программирования, разработанная компанией JetBrains.

На рисунке 3 в текстовом поле для кода можем увидеть подсветку синтаксиса, что сильно упрощает восприятие программы. В панели слева находится иерархия проекта, внизу вкладки: Git, Problems, TODO, Terminal и, в данном случае, TypeScript, потому что проект на скриншоте реализован на этом языке программирования, а в верхней правой панели находятся кнопки для запуска программы, а также дебага, позволяющего пошагово исполнять программу и видеть значение переменных. В этой среде программирования есть много инструментов для рефакторинга кода, которые можно установить с помощью плагинов, например, подсветка ошибки синтаксиса, дублирование

кода, подсветка неиспользуемых переменных. LuNA — это система, которая нуждается в IDE с такими возможностями, упрощающими восприятие и написание кода, что повышает производительность разработчиков и снижает порог вхождения для новых пользователей.

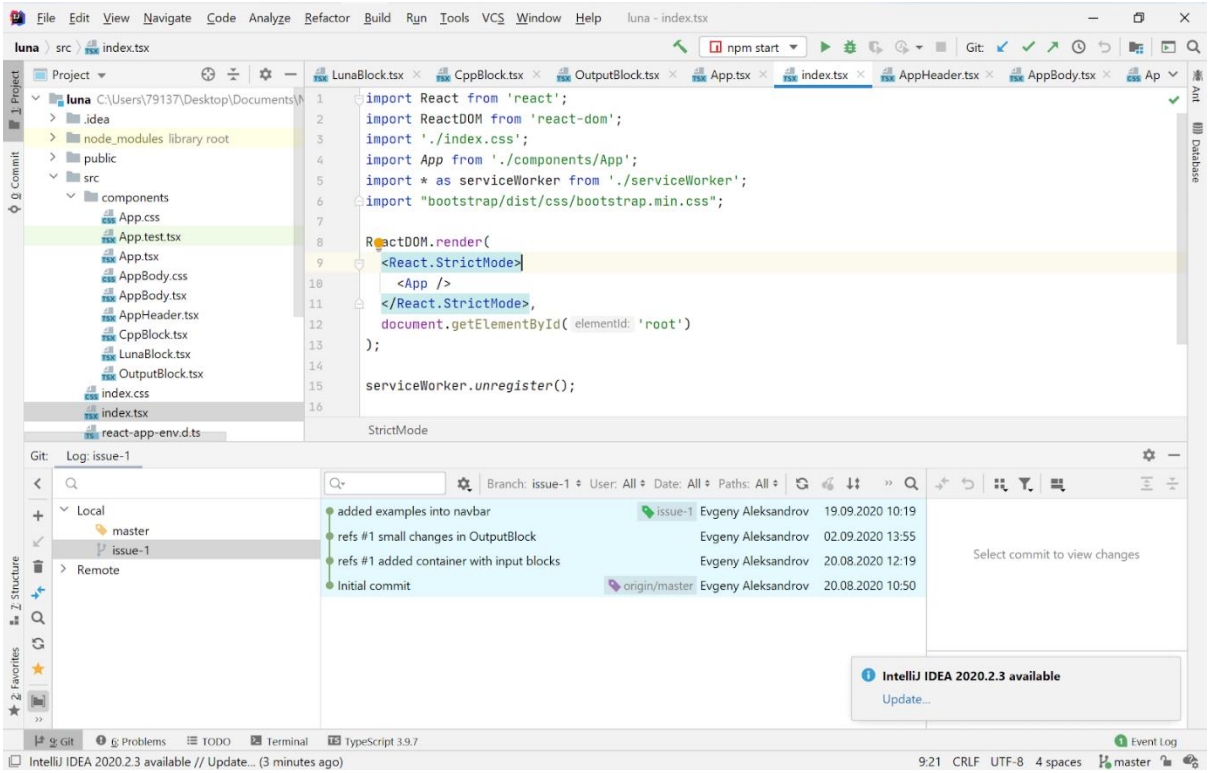

Рисунок 3 - Изображение полноценной среды разработки IntelliJ IDEA

Выводы по существующим IDE:

1. Наличие поля ввода исходного кода
2. Возможность просмотра тестовых примеров
3. Подсветка синтаксиса
4. Возможность перехода к документации
5. Запуск программы и возможность видеть результат её исполнения

1.2 Устройство работы системы LuNA

На вход системе LuNA подается набор входных файлов одного из двух типов: файл с расширением “.fa” (может быть несколько), в нем описывается LuNA программа, то есть перечисляющая часть, описывающая множество триплетов (<in, mod, out> - входные/выходные данные и модуль, которые способен вычислить из входных данных выходные), и файл с расширением

“.cpp”, который реализует модули, то есть последовательные процедуры, осуществляющие вычисление выходных из входных данных, также возможна реализация на другом языке, например JavaScript.

В процессе компиляции создается много промежуточных файлов, одним из таких является “program.ja” - JSON-представление LuNA программы, необходимый для запуска программы в Web RTS.

Результат компиляции — это файл libucodes.so - исполняемое представление, скомпилированное обычным компилятором.

Непосредственный запуск программы осуществляется командой: “luna *.fa”.

1.3 Сведения о работе Web RTS

Для запуска программы на Web RTS нужен файл “.ja”, то есть JSON-представление LuNA программы. Атомарные фрагменты кода на JavaScript загружаются в систему вместе с прикладной программой ее автором, они попадают на каждый узел вычислительной сети.

1.4 Разработка требований к IDE

Первой задачей ВКР является разработка требований к облачной IDE. Требования можно разделить на 3 группы: очевидные, необходимые в IDE, специфичные для системы LuNA, требующие подробного описания и дополнительные, облегчающие восприятие и разработку программ для системы LuNA.

К очевидным требованиям относится: возможность писать исходный код программы, компиляция и запуск программы, возможность видеть результат исполнения программы, получение сообщения об ошибке компиляции/выполнения программы.

Опишем подробнее специфичные требования, необходимые для системы LuNA:

1. Запуск на различных вычислительных системах. Система LuNA предназначена для запуска на различных кластерах, с разным количеством узлов, следовательно облачной IDE должен быть обеспечить эту потребность. Облачная IDE должна взаимодействовать с различными вычислительными

системами, это снизит ее зависимость от запуска программ на конкретных вычислительных системах. Необходимо обеспечить простую заменяемость вычислительных систем и предоставить возможность добавления других вычислительных систем для запуска программ на них. Таким образом, облачная IDE будет обладать точкой расширения для интеграции других вычислительных систем и гибкостью выбора этих вычислительных систем.

2. Требования к количеству одновременных подключений к IDE. Облачная IDE будет использоваться разработчиками программ на LuNA, в текущих реалиях порядка 10 одновременных подключений хватает с запасом.

3. Просмотр и запуск тестовых примеров. Облачная IDE разрабатывается для снижения порога вхождения новых пользователей, поэтому нужно предоставить им примеры программ на языке LuNA, тем самым они быстрее освоятся и начнут выполнять свои задачи.

4. Интеграция существующего инструментария - Web RTS и визуализатор исполнения фрагментированных программ.

5. Отображение результатов производительности. Разработчику на LuNA необходимо знать не только что программа успешно выполнилась, ему нужно знать характеристики производительности. Поэтому, в течение сессии, разработчик должен будет видеть график, в котором будет представлена информация о времени исполнения программы.

6. Возможность расширения. Необходимо обеспечить расширение IDE для следующих возможностей:

- Отладчик программ
- Пошаговое исполнение программ запускаемых на Web RTS
- Запуск на других ИС и запуск сгенерированного кода
- Визуализатор ФА (визуальный язык)

К дополнительным требованиям относим скачивание исходного кода фрагментированной программы, для того чтобы пользователь мог продолжить писать свои программы в другом месте, а также предоставление ссылку на документацию языка LuNA.

1.5 Разработка функциональности IDE

Далее необходимо разработать функциональность облачной IDE, т. е. набор предоставляемых функций. Исходя из требований набор состоит из такого списка: возможность писать исходный код программы, компилировать и запускать программу, возможность видеть результат исполнения программы, получать сообщения об ошибке компиляции/выполнения программы, скачивать исходный код, просматривать и запускать тестовые примеры, иметь доступ к документации, видеть визуализацию фрагментов программы, отображать результаты производительности.

1.6 Разработка архитектуры IDE

В рамках данной ВКР разрабатывается облачная IDE, следовательно, необходим такой компонент, как веб-сервер, обрабатывающий запросы пользователя.

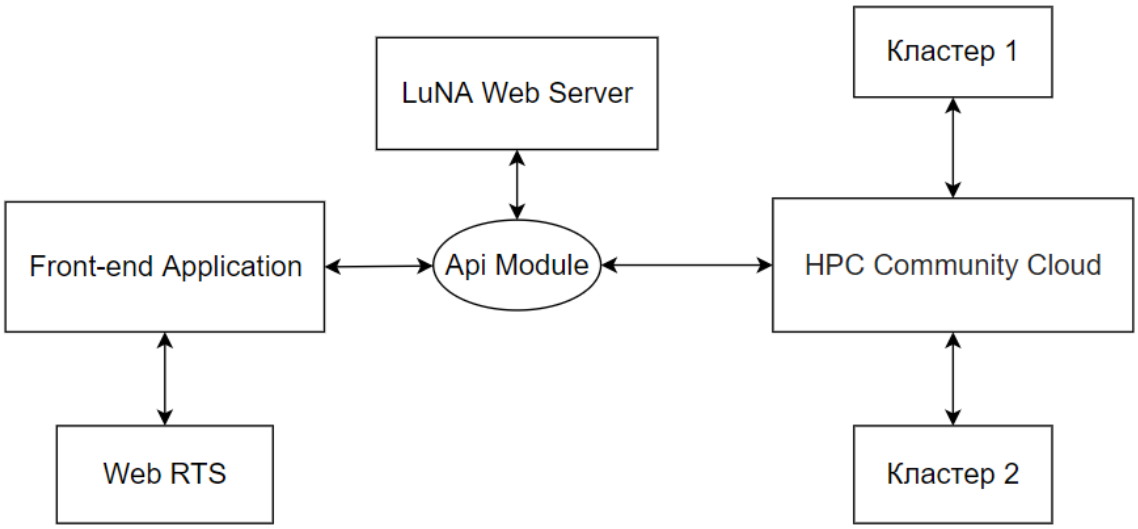

Рисунок 4 - Основные модули архитектуры: **Api Module -** модуль, через который облачная IDE взаимодействует с вычислительными системами, например, LuNA Web Server. Api Module является точкой расширения, для интеграции других вычислительных систем.

Для обеспечения бесперебойной работы около 10 пользователей веб-сервер, на котором развернут Front-end Application, должен обрабатывать запросы асинхронно, то есть запросы будут исполняться одновременно. Чтобы запросы не выполнялись по очереди, должен быть пул потоков, в каждом потоке

которого исполняется конкретный запрос, что удовлетворяет требованию об обеспечении одновременной работы 10 пользователей.

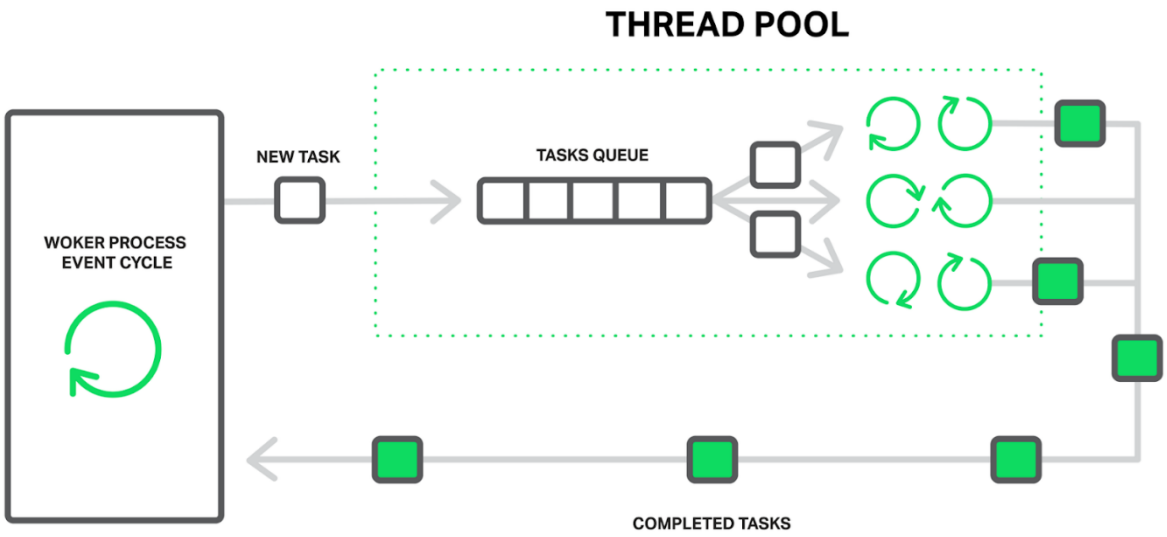

Рисунок 5 - Изображен механизм работы Thread Pool,
использующийся Front-end Application

Чтобы возможность запуска программ на различных вычислительных системах была обеспечена, было решено сделать отдельный слабосвязанный компонент вычислительных системы - LuNA Web Server с установленной системой LuNA, который должен взаимодействовать по API с облачной IDE. Есть еще много вычислительных кластеров, с которыми можно взаимодействовать, обеспечивать подключение к каждому из них не беремся, но есть специальная система, которая выступает промежуточным звеном для этого подключения - HPC Community Cloud. Взаимодействие с вычислительными системами через HPC Community Cloud нужно обеспечить. Для подключения к HPC Community Cloud предоставляется гостевой аккаунт с фиксированными вычислительными ресурсами, и ограниченной библиотекой атомарных фрагментов кода. Это экономит время для нового пользователя, не имеющего аккаунта на HPC Community Cloud, и предоставляет возможность сразу запустить тестовую программу, но для гостевого аккаунта ограничивается библиотека атомарных фрагментов кода, во избежании запуска потенциально опасного кода. Если пользователь имеет свой аккаунт на HPC Community Cloud,

то он может зайти в него и пользоваться доступными ему вычислительными ресурсами. Исходя из анализа, были сформированы следующие варианты запуска:

- Компиляция и запуск на LuNA Web Server
- Компиляция и запуск на гостевом аккаунте HPC Community Cloud
- Компиляция и запуск на своем аккаунте HPC Community Cloud после успешной авторизации

Взаимодействие с внешними вычислительными системами происходит через отдельный интерфейс - Api Module. Api Module реализован на HTTP основой которого является технология “клиент-сервер”, HTTP в настоящее время повсеместно используется во Всемирной паутине для получения информации с веб-сайтов, а значит хорошо поддержан инструментально и будет поддерживаться разработчиками технологий для создания веб-инструментов. Api Module реализует подмножество запросов уже существующего HTTP API HPC Community Cloud, так как оно удовлетворяет требованиям по запуску программы. Таким образом, Front-end Application через Api Module взаимодействует с отдельными модулями вычислительных систем LuNA Web Server и HPC Community Cloud по HTTP API, что обеспечит заменяемость модулей вычислительных систем и удовлетворит требованию по запуску программ на различных вычислительных системах.

Пользователь в веб-интерфейсе Front-end Application может просмотреть и выбрать тестовый пример для запуска программы, что удовлетворяет требованию о просмотре и запуске тестовых примеров.

Для переключения между вычислительными системами, предусмотрено использовать конфигурацию, описывающая адрес вычислительной системы, что облегчает установку адресов для вычислительных систем. Для визуализации фрагментированных программ используются готовые скрипты хранящиеся на LuNA Web Server, в результате работы которых создаются изображения с визуализацией программы. При успешном выполнении программы изображения скачиваются. Web RTS и визуализация программ будут интегрированы, что удовлетворяет требованию по использованию существующего инструментария.

Функция LuNA Web Server и HPC Community Cloud в данной архитектуре одинакова: скомпилировать и запустить полученную программу с Web Server, затем результат отправить обратно. Web Server после получения результата, отображает вывод программы и обновляет график по времени выполнения программ. Это удовлетворяет требованию об отображении результатов производительности.

Таким образом, требования к облачной IDE удовлетворены, при условии реализации компонентов системы с приведенными к ним требованиям.

Глава 2. Реализация Front-end Application, Api Module и LuNA Web Server

2.1 Front-end Application

В качестве **front-end** библиотеки был использован ReactJS. ReactJS - javascript библиотека для разработки пользовательских веб-интерфейсов. Эта библиотека была выбрана, потому что она разрабатывается крупной компанией Facebook, а значит будет иметь долгую поддержку, также для первого использования она проста в использовании.

Страница строится из набора компонентов ReactJS. Каждый компонент обладает состоянием, и при его изменении, у пользователя изменяется только этот компонент. Это значительно ускоряет работу интерфейса, так как не нужно обновлять страницу полностью.

Для управления состоянием приложения выбран Redux. Redux — библиотека для JavaScript, предназначенная для управления состоянием приложения, она хорошо адаптирована для написания клиентских веб-приложений в связке с React.

Ниже представлена схема и скриншот расположения элементов пользовательского интерфейса.

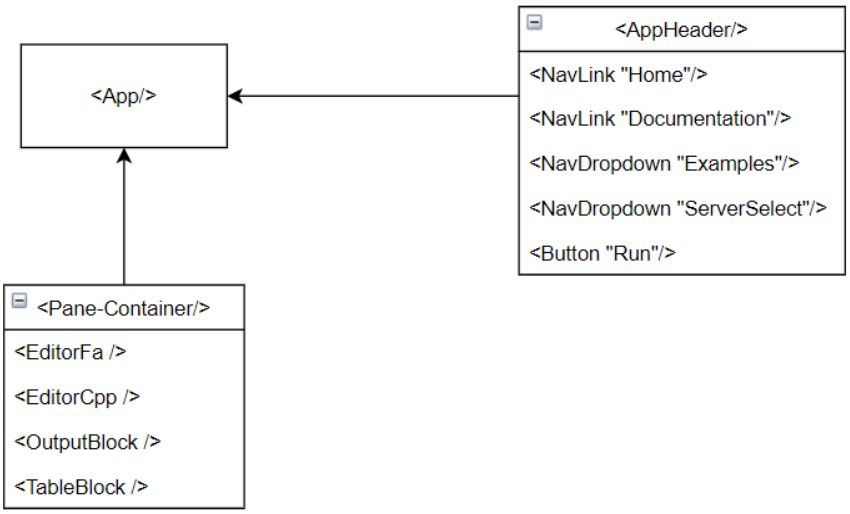

Рисунок 6 - Схема расположения элементов Front-end Application

Структурный блок App состоит из двух блоков: AppHeader и Pane-Container. Внутри AppHeader слева располагаются кнопки возврата на домашний экран облачной IDE, ссылки на документацию, что удовлетворяет дополнительным требованиям облачной IDE, меню выбора примера программы, а справа располагается меню выбора вычислительной системы для запуска и кнопка запуска программы. Меню выбора вычислительной системы удовлетворяет требованию по запуску программ на различных вычислительных системах.

Рисунок 7 - Скриншот, иллюстрирующий расположение блоков ввода исходного кода, вывода программы и таблицы результатов времени.

Также на панели справа расположена кнопка запуска программы, при нажатии на которую открывается модальное окно с выбором количества потоков.

Внутри блока Pane-Container расположены блоки EditorFa, EditorCpp - поля ввода исходного кода, необходимые для выполнения требования по написанию программ, ниже расположен блок вывода результата программы - Output Block и таблица результатов времени исполнения программы, удовлетворяющие очевидным требованиям по выводу результата программы и специфичным требованиям отображению результатов производительности соответственно.

При нажатии на кнопку Run реализовано открытие модального окна с заданием количества потоков для исполнения, что удовлетворяет требованиям по возможностям расширения, в частности закладывается возможность задания других аргументов программы для выполнения других задач, например, получения изображений визуализации программы.

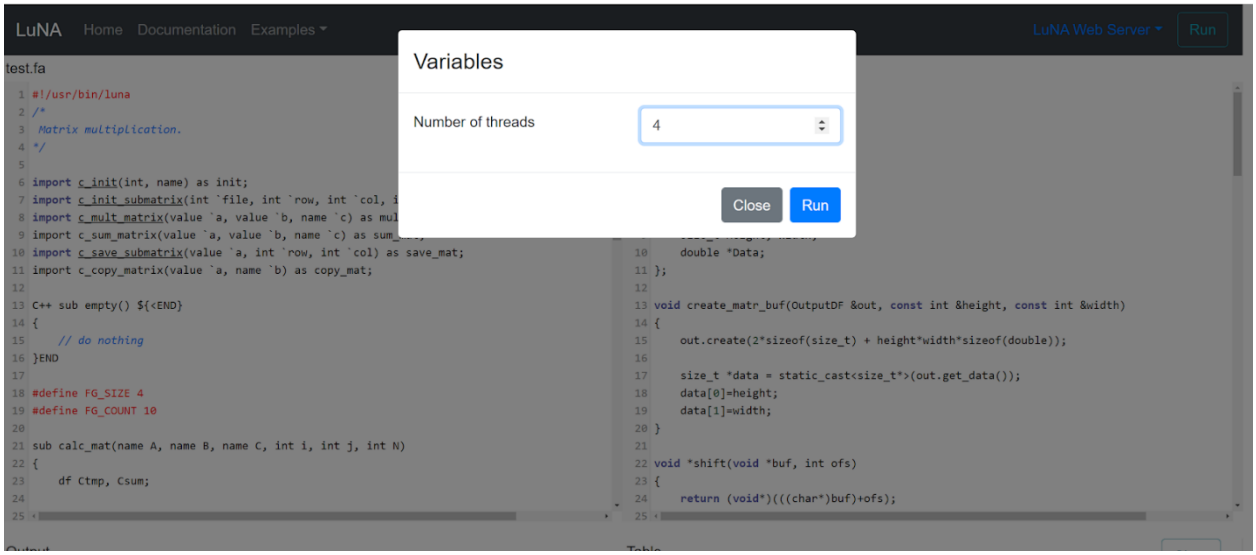

Рисунок 8 - Модальное окно с численным полем ввода для указания числа потоков

В состоянии приложения хранятся исходные коды на C++ и LuNA, выбранный сервер для запуска, вывод программы, а также в sessionStorage хранится массив результатов исполнения программ, который отображается на таблице.

Подсветка синтаксиса осуществляется с помощью библиотеки CodeMirror, отображение графика реализовано с использованием библиотеки D3, что удовлетворяет требованиям по отображению результатов времени и восприятию исходного кода.

Рисунок 9 - Отображение вывода программы и результатов времени исполнения в таблице

Исходный код выложен ~~на~~ в репозиторий: [https://gitlab.com/evgeny_alex/luna-cloud-ide]

2.2 Api Module

Api Module представляет собой подмножество HTTP запросов HPC Community Cloud, достаточных для запуска программы.

Состав HTTP запросов Api Module:

1. POST /api/projects - создание проекта. В теле запроса передаются параметры проекта, в теле ответа возвращается идентификатор проекта.
2. POST /api/fs - создание файлов исходных кодов. В теле запроса передается путь к файлу, тип и содержание файла, в ответ возвращается код о статусе операции (успех или неуспех).
3. POST /api/jobs - создание экземпляра программы с ее аргументами. В теле запроса передаются параметры запускаемой программы, на сервере создается экземпляр программы, возвращается идентификатор экземпляра программы.
4. POST /api/jobs/{id_job}/run - запуск программы. Происходит запуск программы и формируется результат исполнения.

5. POST /api/build/{path} – компиляция программы. Формируется файл “.ja” для дальнейшего использования в Web RTS.

При отправке программы с Front-end Application на вычислительную систему происходит следующий порядок выполнения запросов:

1. POST /api/projects - создание проекта
2. POST /api/fs - создание файлов исходных кодов
3. POST /api/jobs - создание экземпляра программы с ее аргументами
4. POST /api/jobs/{id_job}/run - запуск программы

2.3 LuNA Web Server

LuNA Web Server реализован на Java с использованием фреймворка Spring. Spring — это фреймворк состоящий из множества модулей для различных задач. Мы используем Spring Boot - набор уже настроенных модулей, которые работают в Spring. Эти модули упрощают конфигурацию и построение веб-приложения, написанного на Spring. С помощью Spring Boot можно реализовать HTTP API для взаимодействия с компонентом Api Module облачной IDE, что удовлетворит требованию к LuNA Web Server.

Проект LuNA Web Server имеет следующую структуру:

- LunaBackApplication - главный класс точка входа в Spring Boot приложение.
- /controllers - директория, в которой хранится класс LunaController, обрабатывающий HTTP запросы с пользовательского ReactJS приложения.
- /dto - директория хранит классы DTO (Data Transfer Object) - объекты для передачи данных.
- /services - директория хранит класс LunaService, который реализует всю логику сервера. Сейчас сервер реализует небольшой функционал, поэтому достаточно одного класса сервиса.

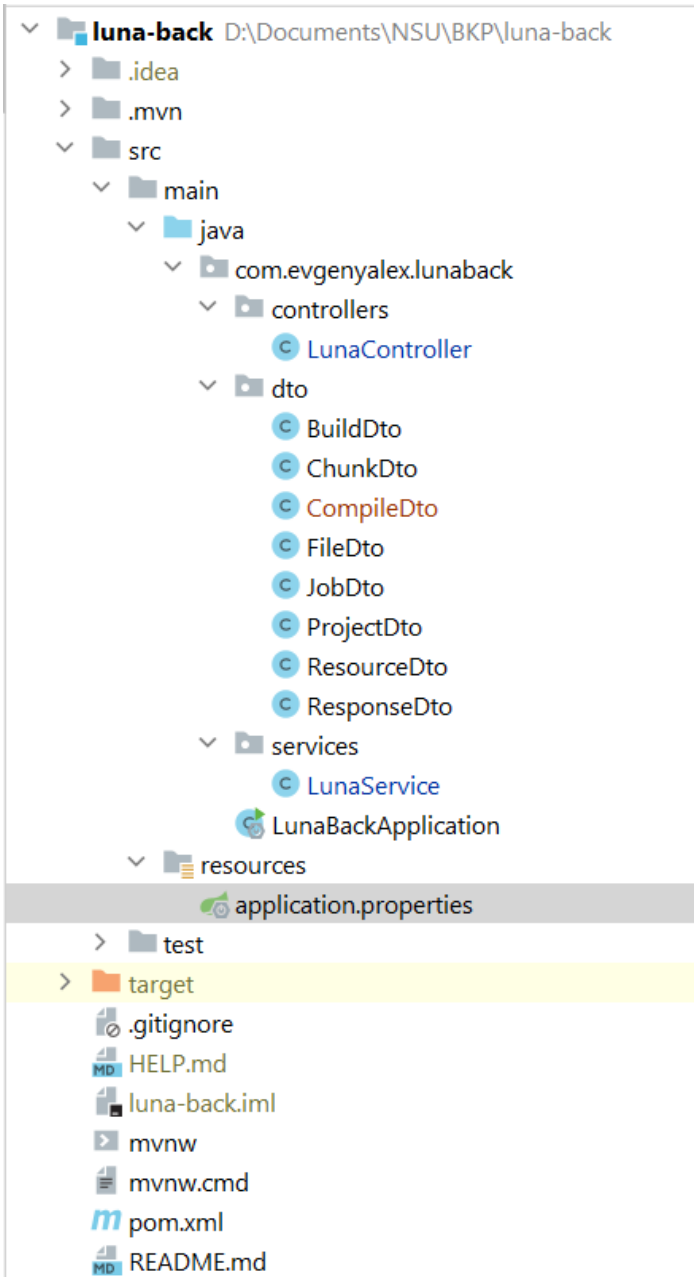

Рисунок 10 - Структура проекта LuNA Web Server

Контроллер LuNA Web Server обрабатывает запросы, представленные в предыдущем пункте при отправке программы. В сервисе через создание новых процессов выполняются команды по созданию папки проекта, файлов исходных кодов, а также создается скрипт, в котором устанавливаются необходимые аргументы запуска.

```
try (FileWriter writer = new FileWriter( fileName: "export.sh",  append: false)) {
    writer.write( str: "export CXX_FLAGS=-DNUM_THREADS=" + numThreads + "\n");
    writer.write( str: "luna -t " + jobMap.get(jobId).getExecutable_path());
    writer.flush();
} catch (IOException ex) {
    System.out.println(ex.getMessage());
}

try {
    Process proc1 = Runtime.getRuntime().exec( command: "chmod ugo+x export.sh");
    proc1.destroy();
    Process proc2 = Runtime.getRuntime().exec( command: "./export.sh");
```

Рисунок 11 - Создание скрипта для запуска программы и последующий его запуск.

После выполнения этого скрипта формируется ResponseDto с указанием времени исполнения, вывода программы и статуса выполнения. Затем ResponseDto отправляется ответом на Front-end Application, где извлекаются данные для обновления графика времени и отображения результата программы.

Исходный код выложен в репозиторий:

[https://gitlab.com/evgeny_alex/luna-web-server]

Глава 3. Описание работы с Web RTS, визуализатором и HPC Cloud

3.1 Взаимодействие с Web RTS

Для взаимодействия с Web RTS необходим файл “.ja” скомпилированной LuNA-программы, поэтому в классе LunaController LuNA Web Server реализован метод обрабатывающий POST запросы по адресу /build, он принимает исходный код LuNA-программы и возвращает “.ja” файл. В пользовательском интерфейсе реализована возможность выбора запуска LuNA-программы в правом верхнем углу.

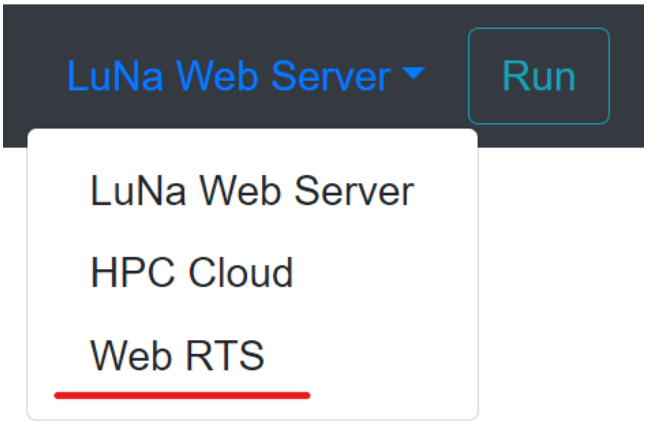

Рисунок 12 - Реализация выбора варианта запуска
на Front-end Application

В варианте с Web RTS после нажатия на Run будет скачиваться необходимый “.ja”, который уже можно использовать в Web RTS.

3.2 Визуализация программы

Архитектурно заложена возможность для визуализации профиля программы, тем самым, что существует возможность для передачи аргументов запуска, в которых можно задать запуск с получением визуализации профиля.

3.3 Взаимодействие с HPC Community Cloud

В Api Module реализовано подмножество HTTP API запросов HPC Community Cloud, таким образом, для взаимодействия с HPC Community Cloud достаточно указать его адрес в конфигурации модуля Front-end Application.

Заключение

В результате работы были составлены требования, разработана архитектура, реализован прототип облачной IDE, где можно запустить LuNA программу.

Результаты работы были представлены на 59-ой Международной научной студенческой конференции МНСК-2021 в виде устного доклада. Тезисы к докладу опубликованы в сборнике материалов конференции.

На защиту выносятся:

– Требования к облачной IDE для системы LuNA

– Архитектура облачной IDE

– Реализация модулей Front-end Application, Api Module, LuNA Web Server

В дальнейшем планируется интегрировать следующие инструменты:

– Web RTS

– HPC Community Cloud

– Визуализатор

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Александров Евгений Александрович _____

ФИО студента *Подпись студента*

« ____ »_____20 __г.

(заполняется от руки)

Список использованных источников и литературы

1. S.Kireev, V.Malyshkin Fragmentation of Numerical Algorithms for Parallel Subroutines Library // The Journal of Supercomputing. Vol. 57. Number 2. 2011. pp. 161-171.
2. Victor Malyshkin, V. Fragmented Programming of Library Parallel Numerical Subroitines – In the Proceedings of the 7-th Int. conference on New Trends in Software Methodologies, Tools and Techniques, IOS Press, Vol. 193, pp. 425-430. 28-30 September, 2007, Dubai
3. Gorodnichev M, Malyshkin V, Medvedev Y. 2013. HPC Community cloud: эффективная организация работы научно-образовательных суперкомпьютерных центров. Научный вестник Новосибирского государственного технического университета. 3 (52):5.
4. Vaycel S, Gorodnichev M. 2013. HPC Community Cloud: разработка инструментария для повышения уровня взаимодействия пользователей с объединенными HPC-системами. Седьмая Сибирская конференция по параллельным и высокопроизводительным вычислениям. Программа и тезисы докладов. Томск, 12-14 ноября 2013 г., Изд-во ТГУ, стр. 82-84.
5. Malyshkin V.E., Перепелкин ВА. 2011. LuNA Fragmented Programming System, Main Functions and Peculiarities of Run-Time Subsystem. Parallel Computing Technologies. LNCS 6873:53-61.
6. Welcome to Python.org [Электронный ресурс] URL: https://www.python.org/ (дата обращения 10.02.2021)
7. The Go Playground [Электронный ресурс] URL: https://play.golang.org/ (дата обращения 15.03.2021)
8. React – A JavaScript library for building user interfaces [Электронный ресурс] URL: https://reactjs.org/ (дата обращения 10.02.2021)
9. Java Documentation [Электронный ресурс] URL: https://docs.oracle.com/en/java/ (дата обращения 25.02.2021)
10. Java Spring: установка, архитектура MVC Framework [Электронный ресурс] URL: https://javahelp.online/osnovy/java-spring (дата обращения 25.02.2021)

11. Spring | Home [Электронный ресурс] URL: https://spring.io/ (дата обращения 26.02.2021)
12. D3.js – Data-Driven Documents [Электронный ресурс] URL: https://d3js.org/ (дата обращения 10.03.2021)
13. Александров Е. А. Разработка прототипа облачной IDE для системы фрагментированного программирования LuNA / Е. А. Александров // Информационные технологии: Материалы 59-й Междунар. науч. студ. конф. 12–23 апреля 2021 г. / Новосиб. гос. ун-т. — Новосибирск: ИПЦ НГУ, 2021. — С. 110.

ПРИЛОЖЕНИЕ А

1. POST /api/projects - создание проекта

```
/api/projects:
post:
    is: [ secured, Unauthorized, BadRequest, InternalServerError]
    description: Create new project
    body:
      application/json:
        schema: |
          {
            "$schema": "http://json-schema.org/draft-03/schema",
            "type": "object",
            "required": true,
            "properties": {
              "type": {
                "type": "string",
                "enum": ["app", "framework", "model"],
                "required": true
              },
              "name": {
                "type": "string",
                "required": true
              },
              "template": {
                "type": "string",
                "required": false
              },
              "cluster_name": {
                "type": "string",
                "required": false
              },
              "make_configuration": {
                "type": "string",
                "required": false
              }
            }
          }
        example: |
          {
            "type": "app",
            "name": "pic_test",
            "template": "mpicxx",
            "cluster_profile": "ssdproj-nks-g6",
            "make_configuration": "release"
          }
    responses:
      201:
        description: Project was successfully created
        body:
          application/json:
            schema: |
              {
                "$schema": "http://json-schema.org/draft-03/schema",
                "type": "object",
                "required": true,
                "properties": {
                  "project_id": {
                    "type": "integer",
                    "required": true
                  },
                  "make_configuration_id": {
```

```
                "type": "integer",
                "required": true
              }
            }
          }
        example: |
          {
            "project_id": 23,
            "make_configuration_id": 15
          }
```

2. POST /api/fs - создание файлов исходных кодов

```
/api/fs:
  post:
    is: [ secured, Unauthorized, BadRequest, InternalServerError ]
    description: Create file or directory
    body:
      application/json:
        schema: |
          {
            "$schema": "http://json-schema.org/draft-03/schema",
            "type": "object",
            "required": true,
            "properties": {
              "path": {
                "type": "string",
                "required": true
              },
              "type": {
                "type": "string",
                "enum": [
                  "file",
                  "catalog"
                ],
                "required": true
              },
              "body": {
                "type": "string",
                "required": false
              }
            }
          }
        example: |
          {
            "path": "/jobs/job1/bin/exec.exe",
            "type": "file",
            "body": "echo 'Hello world'"
          }
    responses:
      201:
        description: File or directory was successfully created
```

3. POST /api/jobs - создание экземпляра программы с ее аргументами

```
/api/jobs:
post:
       is: [ secured, Unauthorized, BadRequest, InternalServerError ]
    description: Create job
    body:
      application/json:
        schema: |
          {
            "$schema": "http://json-schema.org/draft-03/schema",
```

```
"type": "object",
"required": true,
"properties": {
  "name": {
    "type": "string",
    "required": true
  },
  "type": {
    "type": "string",
    "required": true
  },
  "cluster_profile_id": {
    "type": "integer",
    "required": true
  },
  "state": {
    "type": "string",
    "required": true,
    "enum": [
      "edited",
      "queued",
      "run"
    ]
  },
  "chunks": {
    "type": "array",
    "required": true,
    "items": {
      "count": {
        "type": "integer",
        "required": true
      },
      "resources": {
        "type": "array",
        "required": true,
        "items": {
          "id": {
            "type": "integer",
            "required": true
          },
          "value": {
            "type": "string",
            "required": true
          }
        }
      }
    }
  },
  "place": {
    "type": "string",
    "required": false
  },
  "walltime": {
    "type": "string",
    "required": true
  },
  "env_vars": {
    "type": "string",
    "required": false
  },
  "args": {
    "type": "string",
    "required": false
```

```
                },
                "queue_name": {
                  "type": "string",
                  "required": true
                }
              }
            }
          example: |
            {
              "name": "jobname",
              "type": "OpenPBS",
              "cluster_profile_id" : 1,
              "state": "run",
              "chunks": [
                {
                  "count": 1,
                  "resources": [
                    {
                      "name": "ncpus",
                      "value": "4"
                    },
                    {
                      "name": "mpiprocs",
                      "value": "4"
                    }
                  ]
                }
              ],
              "walltime": "00:10:00",
              "executable_path": "apps/testapp/builds/nks-
g6.sscc.ru/release/",
              "args": "--normal-run",
              "env_vars": "VAR1=value1;VAR2=value2",
              "queue_name": "G7_q"
            }
      responses:
        201:
          description: Job was successfully created
          headers:
            Location:
              example: "jobs/123"
```

4. POST /api/jobs/{id_job}/run - запуск программы

```
/api/jobs/{id_job}/run:
      post:
        is: [ secured, Unauthorized, BadRequest, InternalServerError ]
        description: Run Job
        responses:
          204:
             description: _run_ command successfully queued
```

5. POST /api/build/{path} – компиляция программы

```
/api/build/{path}:
      post:
        is: [ secured, Unauthorized, BadRequest, InternalServerError ]
        responses:
          201:
             description: Program was successfully compiled
             body: {
                     "type": "file",
                     "json": "content"
         }
```

ПРИЛОЖЕНИЕ Б

Облачная IDE для системы LuNA
Руководство программиста

Новосибирск, 2021

Содержание

Аннотация

В данном программном документе приведено руководство программиста для облачной IDE для системы LuNA. Исходным языком программы является для модуля Front-end Application – JavaScript, для LuNA Web Server – Java. Средство разработки – среда разработки Intellij IDEA от компании JetBrains.

Оформление программного документа «Руководство программиста» произведено по требованиям ГОСТ 19.504-79 «ЕСПД. Руководство программиста» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

1. Назначение и условия применения программы

1.1 Назначение программы

Облачная IDE предназначена для онлайн разработки LuNA программ, с последующим запуском на различных вычислительных системах.

1.2 Функции, выполняемые программой

Набор функций состоит из списка: возможность писать исходный код программы, компилировать и запускать программу, возможность видеть результат исполнения программы, получать сообщения об ошибке компиляции/выполнения программы, скачивать исходный код, просматривать и запускать тестовые примеры, иметь доступ к документации, видеть визуализацию фрагментов программы, отображать результаты производительности.

1.3 Условия, необходимые для выполнения программы

Наличие исходного кода программы, доступность LuNA Web Server и доступность вычислительной системы.

2. Характеристика программы

Облачная IDE представляет собой сайт, где можно писать исходный код LuNA программы и атомарные фрагменты кода на C++, за счет Api Module обладает гибкостью выбора вычислительной системы. Реализовано хранение результатов времени исполнения программы в sessionStorage, которые отображаются в диаграмме.

3. Обращение к программе

Веб-приложение облачной IDE будет развернуто на сервере, затем по IP-адресу в браузере будет возможность посетить сайт.

4. Входные и выходные данные

4.1 Организация используемой входной информации

На рисунке 1 представлены блоки ввода исходного кода LuNA программы и атомарных процедур на C++.

Рисунок 1 - Скриншот, иллюстрирующий расположение блоков ввода исходного кода.

4.2 Организация используемой выходной информации

На рисунке 2 представлены блоки вывода результата программы и таблица результатов времени исполнения.

Рисунок 2 - Отображение вывода программы и результатов времени исполнения в таблице

5. Сообщения

Программист получает ответы HTTP запросов, производимых при запуске программы. Подробное описание представлено в приложении А.