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

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Белоглазова Даниила Александровича

Тема работы:

ИССЛЕДОВАНИЕ УНИФИЦИРОВАННЫХ АРХИТЕКТУР И МЕХАНИЗМОВ РАСШИРЕНИЯ ЯДРА WEB-IDE ПРОЦЕСС-ОРИЕНТИРОВАННОГО ЯЗЫКА POST

«К защите допущена»
Заведующий кафедрой,
д. т. н., доцент
Зюбин В.Е./……..
«31» мая 2022 г.

Руководитель ВКР
к. т. н., доцент,
КафКТ ФИТ НГУ
Лях Т.В./……...
«31» мая 2022 г.

Соруководитель ВКР
к. т. н., доцент
КафКТ ФИТ НГУ
Розов А.С./……..
«31» мая 2022 г.

Новосибирск, 2022

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 компьютерных технологий

Направление подготовки 09.03.01 Информатика и вычислительная техника

Направленность (профиль): Программная инженерия и компьютерные науки

УТВЕРЖДАЮ

Зав. кафедрой Зюбин В. Е.

…….

(подпись)

«29» октября 2021 г.

ЗАДАНИЕ

НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студенту Белоглазову Даниилу Александровичу, группы 18205

Тема: Исследование унифицированных архитектур и механизмов расширения ядра Web-IDE процесс-ориентированного языка poST

утверждена распоряжением проректора по учебной работе от 29.10.2021 № 0297

Срок сдачи студентом готовой работы: «20» мая 2022 г.

Исходные данные (или цель работы): реализовать механизм расширения ядра poST IDE

Структурные части работы: анализ предметной области, составление требований, разработка архитектуры, реализация, экспериментальное исследование

Руководитель ВКР
к. т. н., доцент, ККТ ФИТ НГУ
Лях Т. В. /……..
«29» октября 2021 г.

Задание принял к исполнению
Белоглазов Д. А. /……..
«29» октября 2021 г.

Соруководитель ВКР
б/с, ст. преп., ККТ ФИТ НГУ
Розов А. С./……..
«29» октября 2021 г.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

1. IDE – integrated development environment, интегрированная среда разработки.
2. poST – process-oriented Structured Text, процесс-ориентированный язык программирования, разрабатываемый в Институте автоматики и электрометрии СО РАН.
3. API – Application Programming Interface, программный интерфейс приложения.
4. DSM – Domain Specific Module, узкоспециализированный модуль системы poST IDE.
5. JSON – JavaScript Object Notation, текстовый формат обмениваемых данных в стиле языка программирования JavaScript.
6. JAR – Java ARchive, архив с файлами программы для исполнения виртуальной машиной Java.

ВВЕДЕНИЕ

На сегодняшний день активно набирают обороты тенденции, сопряженные с явлением, получившим название «Индустрия 4.0» – внедрением киберфизических систем в производственные процессы.

В рамках развития этого тренда помимо прочего требуется наличие специфических языков для программирования систем промышленной автоматизации. В Институте автоматики и электрометрии СО РАН был разработан продолжающий свое развитие язык poST (process-oriented Structured Text) – процесс-ориентированное расширение языка Structured Text.

Поскольку использование интегрированных сред разработки (англ. «IDE» от «integrated development environment») значительно упрощает жизнь специалистам в области программирования, позволяя в одном и том же месте производить набор кода с использованием автоматического дополнения и синтаксической подсветки, сборку, отладку и прочие процессы, ИАиЭ СО РАН занимается разработкой соответствующего многофункционального инструмента для языка poST.

Перед разработчиками любого десктопного ПО возникает проблема потребности поддержки своего продукта. В то же время для пользователей появляется необходимость установки и настройки данной программы на своих машинах. Не являются исключением и IDE. Решением служит создание веб-версии продукта, Web-IDE, позволяющей производить разработку прямо в окне браузера без потребности совершать какие-либо действия на локальном компьютере. ИАиЭ СО РАН последовал по данному пути при разработке своей среды для языка poST.

Помимо этого, ИАиЭ разрабатывает poST IDE в виде расширяемой, что является распространенным свойством для сред разработки. Это значит, что программное обеспечение по умолчанию поставляется с самым базовым оснащением (в виде так называемого «ядра»), в то же время предоставляя возможность для расширения собственной функциональности путем подключения соответствующих модулей или плагинов.

В связи с процессом разработки расширяемой Web-IDE для языка poST Институт имеет ряд задач и проблем, в частности, создание и отлаживание механизма расширения ядра среды разработки, непосредственно позволяющего подключать к программе модули и собственно делающего данную IDE расширяемой.

Цель работы – исследование унифицированных архитектур и механизмов расширения ядра Web-IDE процесс-ориентированного языка poST.

Для ее достижения был поставлен следующий перечень задач:

1. анализ существующих подходов к реализации реконфигурируемых облачных сред разработки;
2. анализ специфики языков процесс-ориентированного программирования и особенностей разработки управляющих программ;
3. составление требований к архитектуре и механизмам расширения ядра Web-IDE для языка poST;
4. разработка архитектуры и механизмов расширения ядра poST Web-IDE;
5. реализация механизмов расширения ядра poST Web-IDE;
6. экспериментальное исследование реализованных механизмов на наборе тестовых модулей расширения.

Для Web-IDE языка poST была предложена микросервисная архитектура ее back-end-составляющей. Был разработан менеджер, оперирующий модулями IDE и выступающий в роли посредника между ними и ядром среды разработки. Также были переработаны существующие компоненты poST IDE для согласованной работы в рамках предложенных нововведений. Процесс настройки составляющих системы предложено производить посредством редактирования конфигурационных файлов. Разработка производилась с использованием технологий Java, Kotlin, Spring, Eclipse Theia.

Предложенные решения обеспечивают отказоустойчивость многомодульной системы среды разработки, а также заметно бóльшую простоту расширения ее функциональности под индивидуальные потребности пользователя.

Работа описана в трех главах. В первой главе приведен анализ существующих подходов к реализации архитектур реконфигурируемых приложений и подходы к реализации облачных приложений, предполагающих коворкинг; также приведен анализ специфики языков процесс-ориентированного программирования и особенностей разработки управляющих программ; обозначается список требований к архитектуре и механизмам расширения ядра Web-IDE для языка poST. Во второй главе описываются предлагаемые архитектура и механизмы расширения ядра poST Web-IDE. Третья глава посвящена вопросам реализации механизмов расширения ядра poST Web-IDE и результатам их экспериментального исследования.

ГЛАВА 1

1 Анализ существующих подходов в реализации реконфигурируемых облачных сред разработки

На сегодняшний день рынок предоставляет множество облачных программных продуктов, позволяющих вести разработку прямо в окне браузера с расширением функциональности среды путем подключение специализированных модулей. [1]

Далее приведены описания подходов к расширению самых заметных представителей ниши – облачных сред разработки Cloud9 и Eclipse Che-Theia.

1.1 Cloud9

AWS Cloud9 (C9) – это облачная IDE от компании Amazon, позволяющая производить операции с кодом (написание, запуск, отладка) с использованием браузера. C9 необходимые инструменты для динамических языков программирования, таких как C++, JavaScript, Python, Go, PHP и Ruby. [2]

Среда разработки полностью написана на JavaScript, и использует Node.js на серверной стороне.

Исходный код AWS открыт и доступен в GitHub-репозитории для того, чтобы любой пользователь мог разрабатывать дополнительные плагины к Cloud9, а также реализовывать собственные среды разработки на ее основе. [3] В помощь разработчикам в сети присутствует доступная документация. [4]

Блоками сборки в C9 являются пакеты (packages). Каждая составляющая функциональности IDE реализуется в виде пакета. Любой пакет состоит из одного или более плагинов.

Все плагины в C9 имеют доступ ко всем API, доступным базовым плагинам ядра. Таким образом, пользователь может расширять, реализовывать и пере-реализовывать функциональность среды разработки.

Пакет C9 содержит файл package.json в своём корне. Данный файл включает в себя метаданные о пакете: его версию, включенные в него плагины, а также их конфигурации.

При желании расширить или изменить поведение C9 пользователь добавляет в свой новый пакет один или более плагинов, каждый из которых имеет единственную обязанность и может зависеть от любых других доступных плагинов. Каждый плагин в пакете фиксируется в файле package.json.

Плагины предоставляют интерфейсы экосистеме C9. Интерфейсы одних плагинов могут использоваться другими. «Из коробки» доступен ряд интерфейсов плагинов ядра C9 (Plugin, fs, proc, net и прочие).

Под капотом C9 используется require.js (библиотека-загрузчик модулей для JavaScript). C9 загружает модули по AMD при помощи ‘define’.

Стоит дать пояснение, что такое AMD в данном контексте. AMD (Asynchronous Module Definition, асинхронное определение модуля) – это механизм определение модулей, при котором их загрузка, а также загрузка их зависимостей выполняются асинхронно. Он особенно хорошо подходит для среды браузера, где синхронная загрузка модулей вызывает проблемы с производительностью, удобством использования, отладкой и междоменным доступом. [5]

Жизненный цикл простого плагина имеет два события. Он начинается с загрузки (load) и завершается выгрузкой (unload). Каждое событие жизненного цикла запускается событием объекта плагина. Событие загрузки плагина запускается сразу после загрузки пакета. [6]

1.2 Eclipse Che-Theia

Eclipse Che – это серверное рабочее пространство, а также облачная среда разработки (Online IDE) от компании Eclipse Foundation. Включает в себя SDK (software development kit), позволяющий создавать плагины для различных языков программирования, фреймворков и инструментов. [7]

Имеет открытый исходный код, доступный в GitHub-репозитории. [8]

Eclipse Theia – это расширяемая среда для разработки полноценных многоязычных продуктов, подобных облачным и настольным IDE, с использованием современных веб-технологий.

Имеет открытый исходный код, доступный в GitHub-репозитории. [9]

Eclipse Che представляет собой стандартную Web-IDE для рабочих пространств, основанную на проекте Eclipse Theia. Это несколько иная вариация в сравнении с обычной Theia, имеющая отличие в виде дополнительной функциональности, заточенной под особенности Che. Данная версия Theia для Che получила название Che-Theia.

Исходный код Che-Theia открыт и доступен в GitHub-репозитории проекта. [10]

Плагин Che-Theia – это расширение среды разработки, изолированное от IDE. Плагины могут быть упакованы в файлы или контейнеры, для обеспечения их собственных зависимостей. [11]

Любой разработчик имеет возможность расширить IDE, поставляемую с Eclipse Che, создав плагин для Che-Theia. Плагины Che-Theia совместимы с любой другой IDE на основе Eclipse Theia.

Плагины редактора Che-Theia позволяют добавлять в установку языки, отладчики и инструменты для поддержки рабочего процесса разработки. Плагины запускаются, когда редактор завершает загрузку. Если плагин Che-Theia не работает, основной редактор Che-Theia продолжает функционировать.

Плагины Che-Theia запускаются либо на стороне клиента, либо на стороне сервера. Схема концепции клиентского и серверного плагинов представлена на рисунке 1.

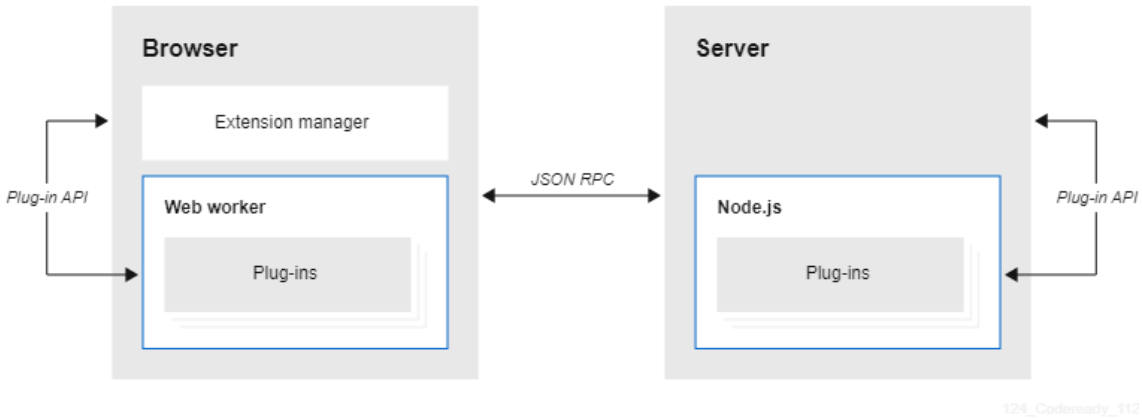

Рисунок 1 – Концепция клиентского и серверного плагинов

Один и тот же программный интерфейс подключаемого модуля Che-Theia доступен подключаемым модулям, работающим на стороне клиента (web-worker) или на стороне сервера (Node.js).

При разработке подключаемого модуля, который зависит от компонентов рабочих пространств Che (контейнеры, настройки, фабрики) или взаимодействует с ними, используются программные интерфейсы (Che API), встроенные в Che-Theia.

Подключаемый модуль Che-Theia включает в себя метаданные – информацию для реестра.

Метаданные подключаемого модуля Che-Theia определяются в файле meta.yaml. На эти файлы можно ссылаться в файле разработки, чтобы включить плагины Che-Theia в рабочую область.

Файл che-theia-plugins.yaml содержит список всех доступных плагинов Che-Theia в реестре.

Когда пользователь открывает рабочую область Che, запускается жизненный цикл для каждого подключаемого модуля Che-Theia. Шаги данного процесса представлены ниже:

1. Che-server проверяет наличие подключаемых модулей для запуска;
2. Che-server получает метаданные, распознает типы, сохраняет, что необходимо, в памяти;
3. Che-server подбирает посредника по типу плагина;
4. посредник производит установку и развертывание плагина.

Схема жизненного цикла плагина Che-Theia представлена на рисунке 2.

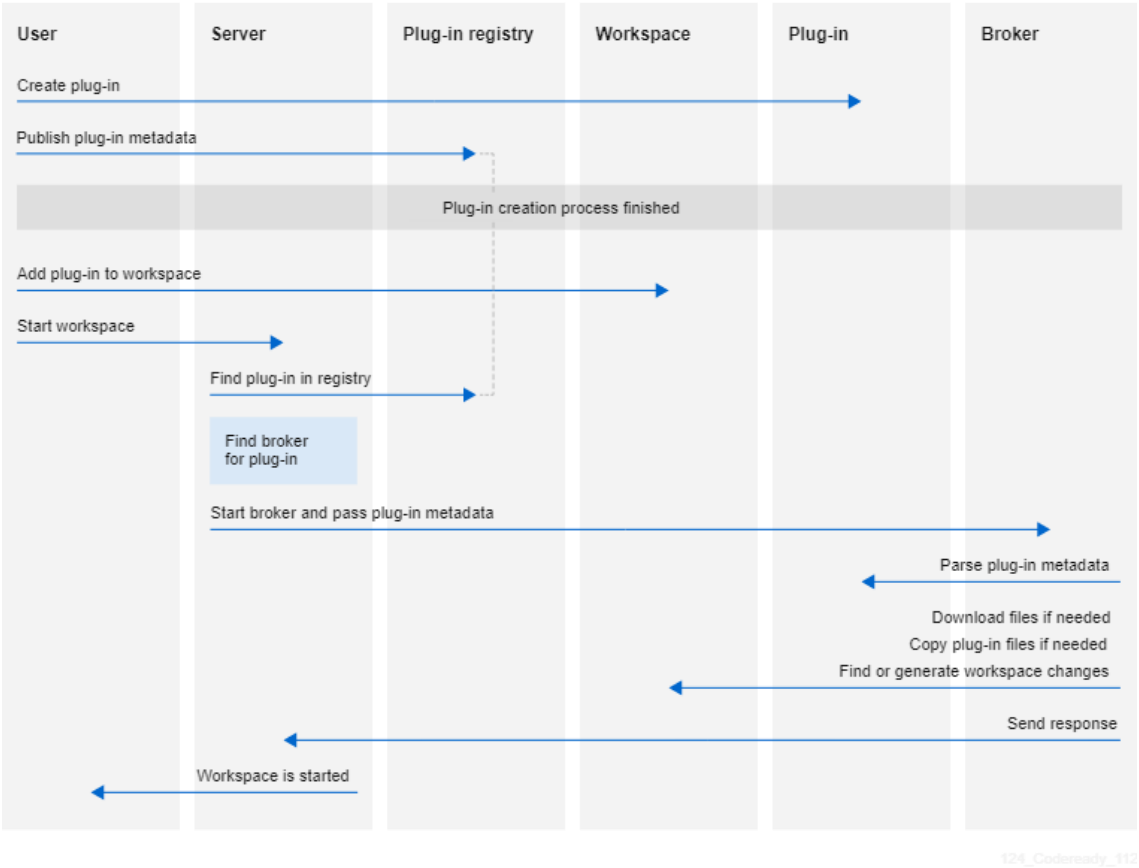

Рисунок 2 – Схема жизненного цикла плагина Che-Theia

Перед запуском рабочего пространства сервер Che запускает для него необходимые контейнеры:

1. посредник подключаемых модулей Che-Theia извлекает из файла .theia сведения о контейнерах, которые требуются конкретному подключаемому модулю;
2. посредник отправляет соответствующую информацию о контейнере на сервер Che;
3. посредник копирует подключаемый модуль Che-Theia, чтобы он был доступен для контейнера редактора Che-Theia;
4. Che-server запускает все контейнеры рабочей области;
5. Che-Theia запускается в своем контейнере и проверяет директорию для загрузки подключаемых модулей.

Пользовательский опыт:

1. Когда пользователь открывает вкладку браузера с Che-Theia, она запускает новый сеанс плагина с:

- web-worker для внешнего интерфейса,
- Node.js для back-end.

2. Che-Theia уведомляет все плагины Che-Theia о начале нового сеанса, вызывая функцию start для каждого запущенного плагина.
3. Сеанс подключаемого модуля Che-Theia запускается и взаимодействует с внутренним и внешним интерфейсом Che-Theia.
4. Когда пользователь закрывает вкладку браузера Che-Theia или сеанс завершается по истечении времени ожидания, Che-Theia уведомляет все плагины, вызывая stop для каждого запущенного плагина.

1.3 Вывод

Подходы в реализации описанных выше реконфигурируемых облачных сред разработки схожи, их можно свести к следующей формулировке:

- каждый модуль содержит метаданные о себе;
- в системе присутствуют метаданные о подключаемых модулях;
- механизм расширения подхватывает и использует эти метаданные, объединяя модули и ядро в единую согласованную систему.

2 Анализ специфики языков процесс-ориентированного программирования и особенностей разработки управляющих программ

2.1 Процесс-ориентированная парадигма

Процесс-ориентированная парадигма программирования применяется в области промышленной автоматизации и автоматизации научных исследований.

На текущий момент практически вся промышленная автоматизация реализуется на цифровых системах управления, в качестве базового элемента которых используются программируемые логические контроллеры (ПЛК). В их состав входит микропроцессорный модуль, функционирующий в соответствии с программным обеспечением (алгоритмом управления, АУ).

АУ контролирует значения входных и выходных сигналов, выполняет логические и арифметические операции, поддерживает требуемые технологией значения параметров, а также обеспечивает связь с датчиками и исполнительными устройствами.

АУ имеет специфику, отсутствующую либо не столь явно проявляющуюся в вычислительных задачах и пользовательских программах для персонального компьютера.

Прежде всего, это наличие внешнего мира, объекта управления (ОУ), с которым АУ производит постоянный обмен данными. Через различные датчики в АУ непрерывно поступает информация о текущем состоянии ОУ и происходящих событиях. На этот поток данных АУ должен реагировать через органы управления, пытаясь привести ОУ в требуемое состояние. Этот непрерывный характер обмена АУ с внешней средой делает неприменимой привычную схему вычисления дано-найти, теряют смысл понятия исходных данных и конечного результата.

Также в зависимости от событий на ОУ характер обработки входных данных может кардинальным образом меняться. Поэтому АУ должен быть событийно-управляемым (уметь перестраиваться).

К тому же реакция АУ должна соответствовать динамическим характеристикам и физической природе ОУ, а также физическим процессам, протекающим на нем. Таким образом, функционирование АУ предполагает большое число операций с временными интервалами: задержками, паузами, тайм-аутами.

И наконец, в АУ необходимо отражать существование множества физических процессов, независимо протекающих на ОУ. При наивной попытке организации монолитного управления становится необходимо рассматривать все возможные состояния системы целиком. Это влечет за собой так называемый комбинаторный взрыв сложности, то есть экспоненциальную зависимость размеров алгоритма от количества входных и выходных сигналов. Единственной возможностью практического решения проблемы в данном случае является обеспечение независимости описания и логического параллелизма его исполнения. [12]

Формулируя лаконично, концепция процесс-ориентированного подхода заключается в том, что программа описывается как совокупность взаимодействующих процессов.

2.2 Среда разработки

Интегрированная среда разработки, заточенная под некоторую парадигму, должна содержать блоки функциональности, удовлетворяющие потребности разработчиков в рамках данной специализации (для процесс-ориентированной парадигмы это блоки-генераторы и прочие).

В связи с тем, что процесс-ориентированная парадигма весьма молода, возможность максимально простого и унифицированного процесса разработки и включения в соответствующую IDE новых блоков функциональности (удобно спроектированная расширяемость) является крайне желательной.

3 Составление требований к архитектуре и механизмам расширения ядра Web-IDE для языка poST

Для пользователя: задействование в процессе работы с IDE необходимого модуля (плагина) «в несколько кликов».

Для разработчика модуля расширения: возможность разработки собственного узкоспециализированного блока функциональности с концентрацией исключительно на его непосредственной логике, без разрешения вопросов интеграции в систему.

Для администратора системы: возможность настройки взаимодействий компонентов лишь путем редактирования конфигурационных файлов без вмешательства в исходный код какой-либо составляющей.

ГЛАВА 2

1 Проектирование архитектуры и механизмов расширения

1.1 Общие принципы проектирования

Первый принцип – построение микросервисной архитектуры.

В отличие от монолитной, микросервисная архитектура предполагает, что приложение представляет собой набор небольших и слабосвязанных компонентов (микросервисов), которые можно разрабатывать, развертывать и поддерживать независимо друг от друга.

Такой подход в данном случае разработки IDE, которая задумывается как расширяемая, является предпочтительным по ряду причин:

- ядро и проблемные модули явно разделены между собой в виде микросервисов, разработчики каждого из которых имеют гораздо большую независимость друг от друга и руководствуются исключительно контрактом, роль которого играют принципы работы механизма расширения;
- микросервисная система является более отказоустойчивой, влияние на работу всего приложения выхода из строя отдельного сервиса (модуля) контролируется с большей легкостью;
- разработкой нового плагина может заниматься любое заинтересованное лицо, знающее, как обеспечить работу механизма расширения системы.

Второй принцип – выделение посредника между ядром и модулями.

Одним из самых заметных принципов как в объектно-ориентированной парадигме, так и программировании вообще является так называемый «single responsibility principle» – принцип единственной ответственности. Каждый компонент программы или программной системы должен иметь достаточно узкую область обязательств. В рамках концепции микросервисов этот принцип распространяется не только на отдельные ООП-классы (если разработка ведется на объектно-ориентированных языках программирования), но и на каждый микросервис в целом. Такой подход позволяет упростить процесс параллельной разработки и отладки каждого компонента системы разными людьми.

Поэтому правильным решением является выделение специального сервиса, отвечающего за учет модулей IDE и перенаправление запросов к ним от ядра среды разработки (выступающего в роли посредника).

Третий принцип – задел под использование конфигурационных файлов.

Сделать настройку свойств компонент системы можно значительно более простой и гибкой, если свести ее к редактированию специальных файлов конфигураций, на которые приложения будут опираться по мере своей работы.

Конфигурационные файлы следует задействовать:

1. внутри отдельно взятого модуля расширения – для редактирования его отличительных свойств;
2. внутри ядра – с целью указания доступных для подключения модулей.

1.2 Общее описание основных элементов системы по части модульности

Руководствуясь доводами в пользу обозначенного выше первого принципа проектирования, всю систему IDE следует конструировать в виде набора приложений-микросервисов.

И, как было указано в обосновании второго принципа, выделение отдельного приложения, выступающего посредником между модулями и ядром IDE, позволяет соблюсти принцип единственной ответственности и сделать процесс разработки проще, а схему работы приложения более явной.

Таким приложением предложено сделать DSM-manager, то есть менеджер модулей IDE.

Основные задачи DSM-менеджера:

1. запуск модулей;
2. остановка модулей;
3. учет активных модулей;
4. перенаправление модулям запросов от ядра.

На рисунке 3 представлена схема системы с учетом выделения в ней DSM-менеджера.

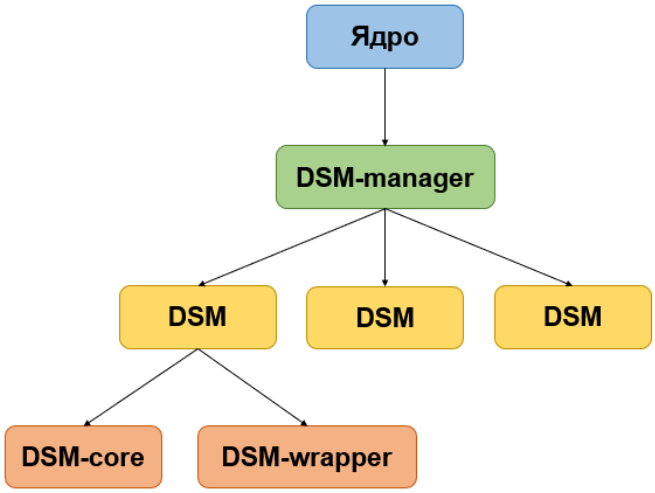

Рисунок 3 – Схема системы IDE с учетом выделения DSM-менеджера

Необходимо обозначить, что в рамках приведенной выше схемы представляет из себя каждый модуль (DSM).

Любой модуль системы состоит из двух компонентов:

1. корневой блок со специализированной логикой (DSM-core);
2. обертка над этим блоком (DSM-wrapper).

Два разных модуля отличаются друг от друга именно своими корневыми блоками.

Чтобы получить конечный модуль IDE, необходимо собрать приложение обертки вместе с некоторым корневым блоком DSM.

В конечном счете обертка оказывается посредником между инкапсулированной специальной логикой и внешним миром (менеджером модулей, ядром среды разработки).

Такое разделение позволяет достичь следующих преимуществ:

- разработчик нового DSM фокусируются исключительно на непосредственной логике своего модуля (DSM-core), не заботясь о том, какими средствами будет достигнута его интеграция в систему;
- DSM-wrapper является универсальным и одинаковым для всех модулей, что гарантирует единообразие (стабильный контракт), необходимое для коммуникаций с менеджером.

1.3 Технологии реализации

Взаимодействия ядро – DSM-manager и DSM-manager – DSM-wrapper предложено производить по сети.

Такое решение способствует расширяемости и снимает ограничение на размещение всех элементов системы в рамках одной физической машины.

Сетевое взаимодействие между элементами можно реализовать как:

- обмен единичными HTTP-запросами без удержания открытых соединений;
- обмен сообщениями в рамках удерживаемых соединений.

Несмотря на то, что второй подход позволил бы мгновенно ставить в известность менеджера об отключении от сети того или иного модуля, он является крайне накладным в перспективе присутствия большого множества модулей, работающих одновременно. Исходя из этого, для реализации предложен первый вариант.

Необходимо определиться также с языками программирования. Язык Java является крайне популярным средством для написания back-end-составляющих приложений. В сети для него можно найти качественную документацию и множество решений как типовых, так и неординарных задач.

В то же время Kotlin является куда более гибким при полной совместимости с Java в обе стороны и обладает приятными (по мнению автора) синтаксическими особенностями.

Но следует иметь в виду, что Kotlin значительно моложе и специалистов по работе с ним, как следствие, меньше. По этой причине писать на Kotlin всё сразу видится сомнительным в перспективе дальнейших доработок компонент системы, для которых в этом может возникнуть необходимость.

С целью достижения компромисса, принято решение поступить следующим образом:

- реализовать DSM-manager как приложение на Kotlin, поскольку, вероятней всего, кардинальных изменений в его логике в дальнейшем не последует;

- реализовать DSM-wrapper как приложение на Java, так как проектом, вполне возможно, будут заниматься и разработчики после.

Для реализации REST-приложений на Java/Kotlin существует фреймворк Spring. Он крайне распространен и относительно прост в применении, для него существует доступная документация, и, кроме того, он обладает внушительным сообществом пользователей.

Как итог, DSM-manager и DSM-wrapper предложено реализовать как Kotlin- и Java-приложения соответственно, сконструированные на базе фреймворка Spring и осуществляющие сетевое взаимодействие посредством обмена единичными HTTP-запросами.

1.4 Архитектура приложений

Архитектуры DSM-manager и DSM-wrapper выполнены в соответствии с концепцией “чистой архитектуры” Роберта Мартина. [13]

Каждое приложение имеет внутри себя 3 раздела (пакета). Эти разделы по смыслу представляют собой слои, наложенные один поверх другого (см. рис. 4):

1. domain – внутренний слой для доменных сущностей, отражающих назначение приложения;
2. application – промежуточный слой для сервисов, оперирующих доменными сущностями;
3. infrastructure – внешний слой, выступающий связующим звеном между окружающим миром и непосредственной логикой программы.

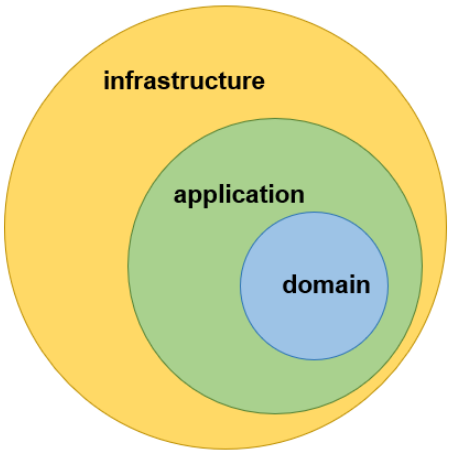

Рисунок 4 – Иллюстрация концепции чистой архитектуры

Такое разделение позволяет глубинной логике программы оставаться независимой от тонкостей реализации на внешнем уровне, которую становится возможно безболезненно изменять при необходимости (в нашем случае, если, например, в будущем будет решено осуществить переход на другой фреймворк).

1.5 Формат тел запросов и ответов в сетевом взаимодействии

Тела запросов и ответов сетевого взаимодействия предложено конструировать в формате JSON (JavaScript Object Notation). Такой формат позволяет передавать данные в человеко-читаемом виде с возможностью преобразовывать их в объекты Java/Kotlin при помощи специальных библиотек.

Для тел ответов предложена структура:

```
{
    “code”: …,
    “content”: …
}
```

Где code – код ответа (OK / ошибка), а content – вложенный JSON-объект, представляющий некоторое содержимое (например, список активных DSM).

Для начала предлагается реализовать два кода ошибки – это “OK” и “ERROR” для успешного и неуспешного завершения операции соответственно. В дальнейшем перечень кодов, разумеется, можно расширить для добавления конкретики, если в том возникнет потребность (если только лишь пояснения в разделе “content” станет недостаточно).

ГЛАВА 3

1 Реализация механизма расширения

1.1 Унификация в модулях IDE

В корневой библиотеке poST-IDE выделен интерфейс **IDsmExecutor** с единственным методом

String execute(LinkedHashMap<String, Object> request)

Аргумент метода – перечень параметров запроса. Возвращаемое значение – описание результата выполнения запроса.

Предложено требование: каждый разработчик DSM-core должен реализовать этот интерфейс таким образом, чтобы через него стало возможно запустить специализированную логику модуля. Необходимые для работы параметры нужно вычленять из ключей-значений аргумента request.

Таким образом, IDsmExecutor служит связующим звеном или контрактом между DSM-wrapper и DSM-core, дающим полную свободу для разработчика DSM-core реализовывать произвольную логику модуля с гарантией совместимости.

Соответствие ключей-значений запроса, предоставляемых модулю и ожидаемых им отдается на откуп инициирующей стороне – ядру poST-IDE. Например, если DSM-core для его работы необходим параметр “fileName”, ядро должно предоставить такой параметр в теле своего запроса на выполнение модуля (тогда он станет доступен в LinkedHashMap).

Произведены корректировки DSM-wrapper. DSM-wrapper преобразован в приложение на базе фреймворка Spring.

В контроллере REST-запросов реализованы следующие методы:

1. run – для запуска внутренней логики DSM-core (см. далее);
2. stop – для завершения работы модуля (остановки программы).

Как было обозначено в главе 2, одним из принципов реализации является динамическое конфигурирование свойств программ посредством извлечения информации из файлов свойств (properties-файлов).

Для DSM-wrapper создан помимо прочих properties-файлов (требуемых для Spring и библиотеки логирования log4j) файл dsm.properties, включающий в себя следующие атрибуты:

1. dsm.name – уникальное наименование модуля, под которым он будет известен в системе;
2. dsm.executorClassName – наименование класса из DSM-core (реализующего интерфейс IDsmExecutor, см. выше), объекту которого будет передано выполнение задачи в рамках специализации модуля;
3. manager.address – сетевой адрес по умолчанию для менеджера модулей.

При получении запроса на выполнение логики модуля, создавая методом Reflection API объект по имени dsm.executorClassName, реализующий интерфейс IDsmExecutor, DSM-wrapper обращается к нему на уровне интерфейса IDsmExecutor, передавая параметры полученного извне запроса.

Например, если в некотором DSM-core интерфейс IDsmExecutor реализуется классом StGenerator, значением dsm.executorClassName в DSM-wrapper этого модуля должно стать “<полное-имя-пакета>.StGenerator”.

На рисунке 5 представлена схема для изложенных выше действий:

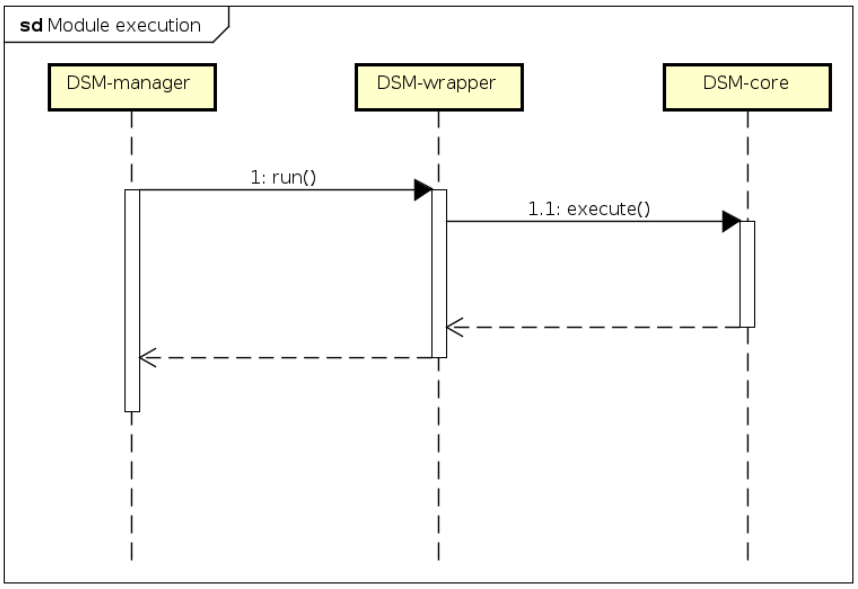

Рисунок 5 – Схема выполнения задачи модулем

В ходе своей работы DSM-wrapper создает записи в лог-файле. Параметры логирования возможно изменить посредством редактирования файла log4j.properties.

Создание нового модуля с учетом описанных выше нововведений сводится к следующему перечню действий:

1. разработать DSM-core, блок с узкоспециализированной функциональностью, таким образом, чтобы имелась возможность запустить выполнение его логики посредством вызова метода execute класса, реализующего интерфейс IDsmExecutor из библиотеки ядра;
2. создать Java-архив (jar) с DSM-core;
3. поместить Java-архив с DSM-core внутрь проекта DSM-wrapper;
4. актуализировать значения в properties-файлах проекта DSM-wrapper;
5. собрать DSM-wrapper в Java-архив.

Полученный в конечном итоге Java-архив представляет собой модуль poST-IDE. После запуска (java -jar) к нему становится возможно обращаться при помощи реализованных REST-методов (run и stop).

1.2 DSM-manager

Документация к программе менеджера модулей расширения IDE приведена в приложениях к работе («Руководство оператора» и «Описание программы» – Приложения В и Г соответственно).

Как обозначалось ранее, DSM-manager представляет собой Kotlin-приложение на базе фреймворка Spring.

В таблице 1 представлен перечень реализованных методов его REST-API с краткими описаниями.

Method + URL	**Описание**
GET /	“hello”-метод, позволяющий проверить, что менеджер запущен, и получить описание его API в виде HTML-страницы

POST /new-module	метод регистрации нового модуля в перечне активных
GET /alive-modules	метод получения актуального списка активных модулей
GET /available-modules	метод получения списка доступных модулей
POST /run/{moduleName}	метод выполнения логики модуля
GET /start-all	метод старта всех доступных модулей
GET /start/{moduleName}	метод старта конкретного модуля
GET /stop/{moduleName}	метод остановки конкретного модуля

Таблица 1 – API менеджера модулей

Более детальное описание этих методов приведено в следующих разделах.

Методы REST-контроллера занимаются лишь получением данных запросов с последующей их передачей внутренней сущности программы – объекту Manager, расположенному в слое application (см. п. 1.4 главы 2 об архитектуре). Под объектом в данном случае имеется в виду Kotlin-object - переосмысление шаблона “singleton” в Java, единственный объект своего класса.

Исполнение некоторых методов требует HTTP-взаимодействие с модулями. Однако элементы application-слоя не должны зависеть от конкретных деталей реализации логики. Поэтому такие методы принимают исполняемые выражения, что позволяет производить операции на абстрактном уровне. Такие выражения объединены в компоненты специального объекта ViaHttp.

Далее проиллюстрировано описанное выше на конкретном примере.

REST-контроллер менеджера имеет метод stopModule, выполняющий остановку модуля расширения по имени.

Остановка модуля в текущей реализации происходит путем вызова у него REST-метода /stop.

Объект ViaHttp имеет компоненту

stopModule: (Module) -> Result<String>

Это лямбда-выражение, принимающее на вход объект модуля и возвращающее результат с сообщением. Значение (описание) этого выражения соответствует вызову метода /stop у соответствующего модуля.

Объект Manager обладает методом со следующей сигнатурой:

fun stopModule(moduleName: String, stopModule: (Module) -> Result<String>): Result<String>

В нем вызывается выражение stopModule для модуля с именем moduleName.

В методе stopModule REST-контроллера происходит вызов

Manager.stopModule(moduleName, ViaHttp.stopModule)

Таким образом, Manager, находящийся в application-слое приложения не зависит от деталей реализации, сопряженных с HTTP-взаимодействиями, что в перспективе упростит процесс модернизации логики, если она потребуется.

Необходимо дать пояснение тому, каким образом менеджер ведет учет модулей. Внутри объекта Manager хранятся два списка: список активных модулей (их имена, адреса и порты) на текущий момент и список доступных модулей (имена их Java-архивов).

Первый список заполняется по мере регистрации модулей в системе.

Второй же заполняется данными из специального конфигурационного json-файла, имеющего следующую структуру:

```
{
    “directory”: “...”,
    “modules”: [
        “...”, …
    ]
}
```

Где:

- directory – путь к каталогу с Java-архивами модулей;
- modules – список наименований Java-архивов модулей.

Более подробные сведения о данном конфигурационном файле приведены в следующем разделе.

В ходе своей работы DSM-manager создает записи в лог-файле. Параметры логирования возможно изменить посредством редактирования файла log4j.properties.

1.3 Реализованные сценарии работы и интерфейсы сетевого взаимодействия

1.3.1 Старт менеджера

Запуск менеджера по умолчанию осуществляется посредством выполнения команды:

java -jar <dsm-manager-name>.jar

После запуска менеджер производит старт своего REST-сервера и ожидает входящих запросов.

Также реализована возможность запуска модулей одновременно со стартом менеджера. Для этого предложено использовать ключи: **-amj <available-modules.json-path>** и **-sam** (расшифровываются как “available modules json” и “start available modules” соответственно).

Первый позволяет явно указать путь к json-файлу с информацией о Java-архивах модулей (см. п. 1.2).

Без указания ключа -amj и значения для него информация об архивах извлекается из стандартного файла available-modules.json, расположенного в ресурсах проекта менеджера. Таким образом, в администрирование менеджером внесена дополнительная гибкость: становится возможно не указывать явно путь до json-файла с информацией о модулях, если проект менеджера уже и так собран с актуальным по содержанию файлом внутри.

Ключ -sam указывает на необходимость запуска Java-архивов модулей, приведенных в json-файле.

Извлечение информации об архивах и их запуск, если они требуются, осуществляются после старта сервера менеджера.

По умолчанию порт запуска сервера менеджера извлекается из конфигурационного файла application.properties, используемого фреймворком Spring.

Однако реализована возможность заменить значение порта из файла на некий конкретный или же автоматически подобранный (какой-либо свободный на данной машине). Для этого предлагается использовать ключи запуска -p <port> и -ap (от “auto-port”) соответственно.

1.3.2 Старт модуля

Запуск модуля осуществляется посредством выполнения команды

java -jar <dsm-wrapper-name>.jar

без дополнительных ключей или с ними.

Доступные дополнительные ключи:

- -name – явно указывает наименование модуля (взамен значения свойства dsm.name);
- -ma – явно указывает адрес менеджера модулей (взамен значения свойства manager.address).

После запуска модуль определяет свободный порт на текущей машине для дальнейшего старта собственного сервера.

Затем модуль осуществляет подключение к менеджеру с целью регистрации. Если адрес менеджера не был передан явно (посредством использования ключа -ma), его значение извлекается из файла свойств (manager.address).

Если подключение к менеджеру прошло успешно, модуль выполняет старт собственного REST-сервера.

Подробнее о процессе регистрации модуля см. п. 1.3.3.

Запуск модуля может быть произведен как вручную, так и посредством вызова специального REST-метода менеджера модулей.

Во втором случае происходит запуск Java-архива с предварительным анализом файла available-modules.json, в ходе которого выясняется:

1. доступен ли такой модуль в принципе;

2. если доступен, то в какой директории располагается его Java-архив.

Помимо метода запуска конкретного модуля по его имени, реализован также метод запуска всех доступных модулей. Действия при этом выполняются аналогичные, как и при использовании ключа -sam старта менеджера.

В таблице 2 представлен интерфейс сетевого запроса запуска доступного модуля.

HTTP method: GET	
URL: /start/{moduleName}	
Request body	Response body

-	{
	“code”: “OK” / “ERROR”,
	“content”: {
	“result or error message”
	}
	}

Таблица 2 – API /start

В таблице 3 представлен интерфейс сетевого запроса запуска всех доступных модулей.

HTTP method: GET	
URL: /start-all	
Request body	Response body

-	{
	“code”: “OK” / “ERROR”,
	“content”: {
	“result or error message”
	}

	}

Таблица 3 – API /start-all

1.3.3 Регистрация модуля

На рисунке 6 представлена схема взаимодействия компонент системы при регистрации только что запущенного модуля.

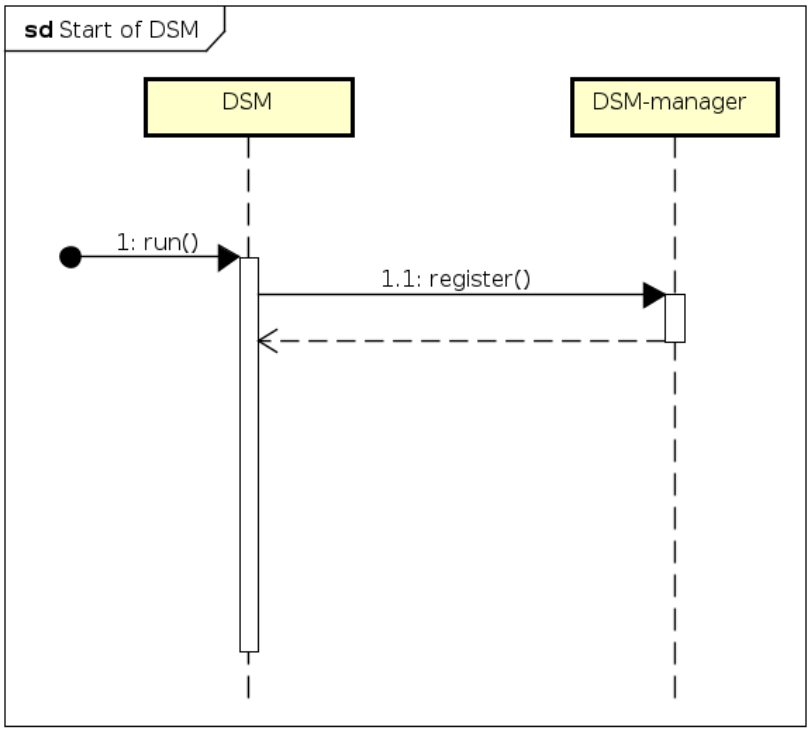

Рисунок 6 – Схема старта модуля и его дальнейшей регистрации

После запуска модуль осуществляет подключение к менеджеру по URL /new-module с целью регистрации.

DSM-manager принимает запрос от модуля и регистрирует его, добавляя запись в список активных модулей.

В таблице 4 представлен интерфейс сетевого запроса регистрации нового модуля.

HTTP method: POST	
URL: /new-module	
Request body	Response body

{
“name”: “...”,
“port”: …
}
	-

Таблица 4 – API /new-module

Адрес регистрируемого модуля (host) в теле запроса не передается, он извлекается из данных о запросе.

При использовании опций -am и -sam для запуска менеджера, все описанные выше операции происходят автоматически – модули запускаются с ключом -ma для указания реального адреса менеджера. Таким образом, реализована возможность подъёма менеджера со всеми модулями исполнением единственной команды.

1.3.4 Получение списка активных модулей

На рисунке 7 представлена схема взаимодействия компонент при запросе актуального списка активных модулей.

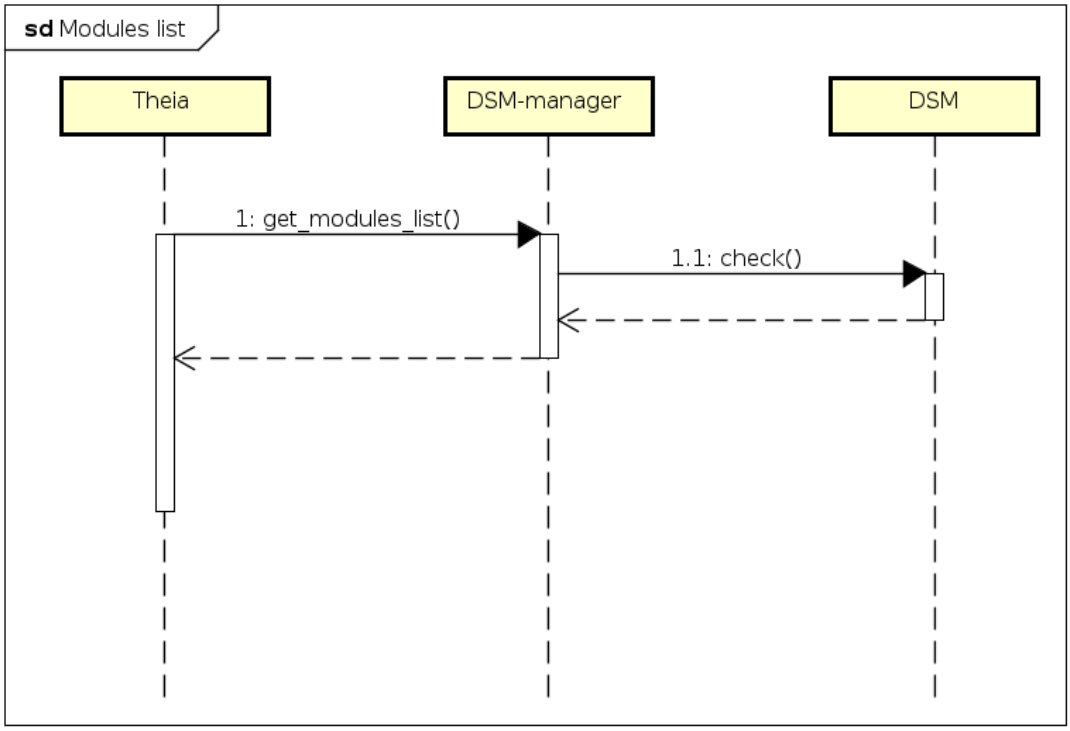

Рисунок 7 – Схема запроса актуального списка активных модулей

DSM-manager получает от ядра IDE запрос актуального списка активных модулей. После этого для каждого модуля из текущего списка менеджер

выполняет запрос на корневой REST без тел запроса и ответа с целью выявить, находится ли тот по-прежнему в сети. Удалив из своего списка модули, которые оказались неактивны, менеджер возвращает его в качестве ответа на запрос.

В таблице 5 представлен интерфейс сетевого запроса получения актуального списка активных модулей.

<table>
<tr><td colspan="2">HTTP method: GET</td></tr>
<tr><td colspan="2">URL: /alive-modules</td></tr>
<tr><td>Request body</td><td>Response body</td></tr>
<tr><td>-</td><td>{
“code”: “OK”,
“content”: {
“modules”: [
{
“name”: “...”,
“host”: “...”,
“port”: …
},
…
]
}
}</td></tr>
</table>

Таблица 5 – API /alive-modules

1.3.5 Получение списка доступных модулей

На рисунке 8 представлена схема взаимодействия компонент при запросе списка доступных для запуска модулей.

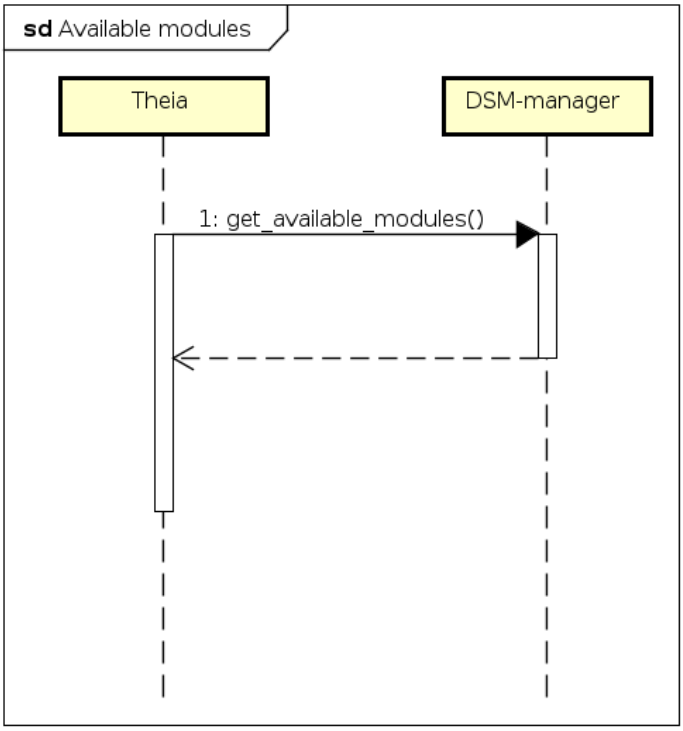

Рисунок 8 – Схема запроса списка доступных модулей

DSM-manager получает от ядра IDE запрос списка доступных модулей. Если соответствующей информации у него нет, он предварительно извлекает ее из соответствующего конфигурационного файла по умолчанию (available-modules.json в директории ресурсов проекта).

В таблице 6 представлен интерфейс сетевого запроса получения списка доступных модулей.

HTTP method: GET	
URL: /available-modules	
Request body	Response body

-	{
	“code”: “OK”,
	“content”: {
	“availableModules”: {
	“directory”: “...”,
	“modulesJarNames”: [

	“...”, …] } } }

Таблица 6 – API /available-modules

1.3.6 Запуск модуля на исполнение

На рисунке 9 представлена схема взаимодействия компонент при запросе на исполнение DSM.

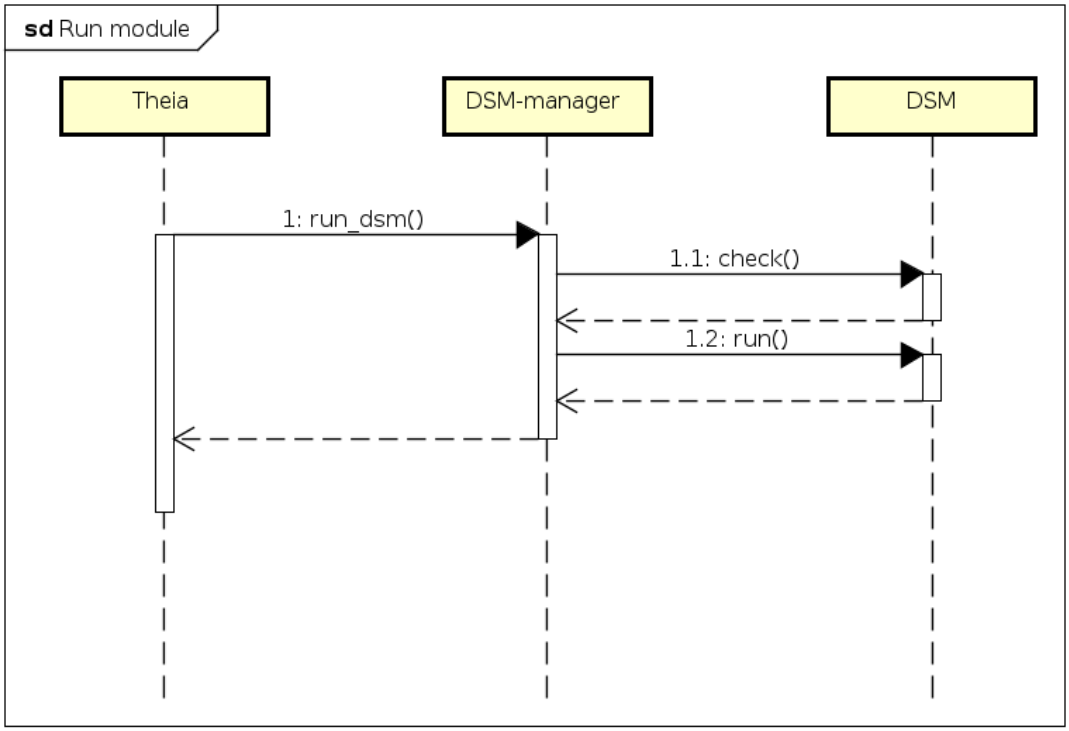

Рисунок 9 – Схема запуска модуля на исполнение

DSM-manager получает от ядра IDE запрос на исполнение определенного модуля. После проверки, что модуль все еще активен, менеджер перенаправляет ему пришедший от ядра запрос.

DSM-wrapper получает от менеджера запрос на исполнение, после чего запускает внутреннюю логику DSM-core с пришедшими из запроса параметрами.

В таблице 7 представлен интерфейс сетевого запроса на запуск определенного модуля на примере модуля генерации poST-to-ST.

HTTP method: POST	
URL: /run/{moduleName}	
Request body	Response body

{
“id”: “...”,
“root”: “...”,
“fileName”: “...”,
“ast”: “...”
}
	{
	“code”: “OK” / “ERROR”,
	“content”: {
	“result or error message”
	}
	}

Таблица 7 – API /run/{moduleName}

1.3.7 Остановка модуля

На рисунке 10 представлена схема взаимодействия компонент при запросе на остановку DSM.

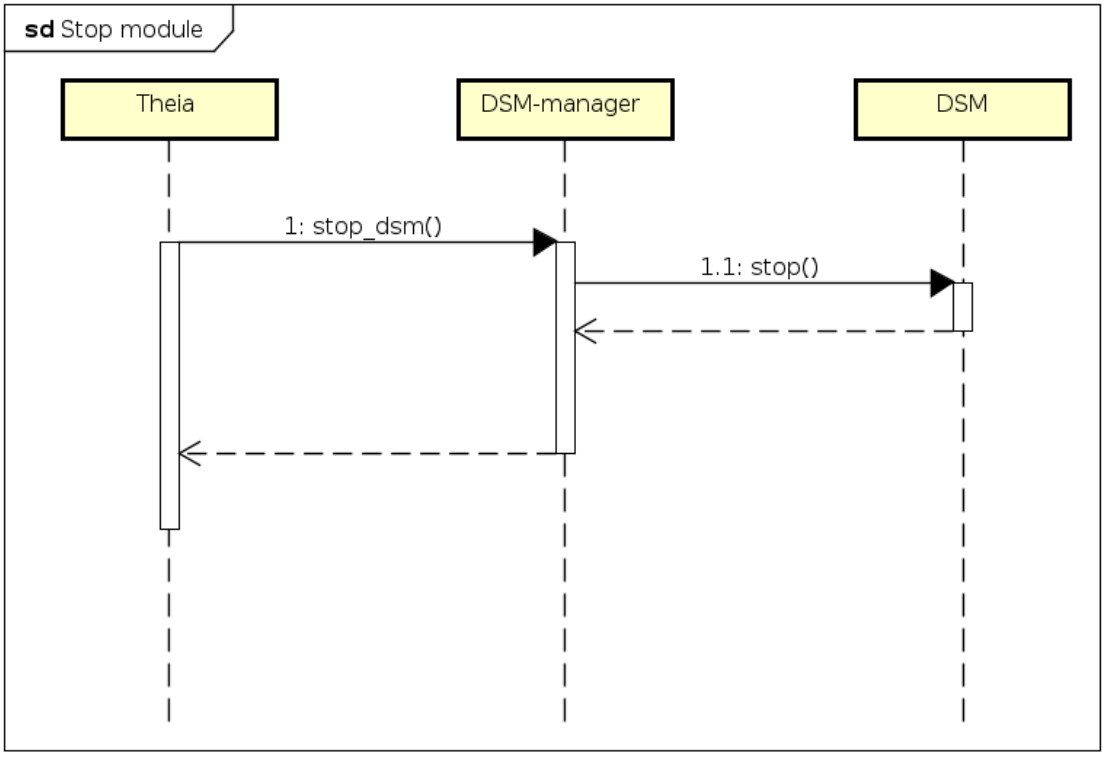

Рисунок 10 – Схема остановки модуля

DSM-manager получает от ядра IDE запрос на остановку определенного модуля. После проверки, что модуль все еще активен, менеджер перенаправляет ему пришедший от ядра запрос.

DSM-wrapper получает от менеджера запрос на остановку, после чего завершает свою работу.

В таблице 8 представлен интерфейс сетевого запроса на остановку определенного модуля.

<table>
<tr><td colspan="2">HTTP method: GET</td></tr>
<tr><td colspan="2">URL: /stop/{moduleName}</td></tr>
<tr><td>Request body</td><td>Response body</td></tr>
<tr><td>-</td><td>{
“code”: “OK” / “ERROR”,
“content”: {
“result or error message”
}
}</td></tr>
</table>

Таблица 8 – API /stop/{moduleName}

1.4 Корректировки ядра IDE

Произведена актуализация ядра poST-IDE. Выделена отдельная директория dsm-management для DSM-менеджера и конфигурационных файлов available-modules.json, manager.properties.

После старта среды происходит автоматический запуск DSM-менеджера с ключами -amj, -sam и -p на порту, указанном в файле manager.properties, что влечет за собой также подъем всех доступных модулей. Значением ключа -amj является путь до созданного в директории dsm-management файла available-modules.json.

Администратору для работы системы требуется лишь актуализировать содержимое файлов manager.properties (указать порт запуска менеджера) и

available-modules.json (указать корректные путь до директории с модулями и имена их Java-архивов).

В ходе работы менеджера и модулей создаются записи в лог-файлах в директории browser-app/log проекта poST IDE.

2 Экспериментальное исследование

Произведено экспериментальное исследование работоспособности механизма расширения на примере тестового модуля генератора poST-to-ST.

В DSM-core генератора класс STGenerator был сделан реализующим интерфейс IDsmExecutor из корневой библиотеки poST IDE.

Далее DSM-wrapper был собран вместе с Java-архивом обновленного ядра генератора и актуализированным файлом dsm.properties.

Затем были произведены:

- загрузка полученного архива модуля в директорию dsm-management/dsms poST IDE;
- актуализация конфигурационных файлов available-modules.json и manager.properties.

После старта poST IDE менеджер модулей вместе с модулем генератора poST-to-ST, как и требовалось, запустились автоматически. Содержимое сгенерированных ими лог-файлов представлено в Приложении А.

Результаты отработки запросов на получение от менеджера актуального списка активных модулей и списка доступных модулей соответствовали ожидаемым и приведены в Приложении Б.

ЗАКЛЮЧЕНИЕ

В ходе данной работы были проанализированы существующие подходы к реализации реконфигурируемых облачных сред разработки, а также специфика языков процесс-ориентированного программирования и особенности разработки управляющих программ. Составлены требования к архитектуре и механизмам расширения ядра Web-IDE для языка poST. Предложена микросервисная архитектура ее back-end-составляющей. Разработан менеджер, оперирующий модулями IDE и выступающий в роли посредника между ними и ядром среды разработки. Переработаны существующие компоненты IDE для согласованной работы в рамках предложенных нововведений. Предоставлена возможность производить настройку составляющих системы посредством редактирования конфигурационных файлов. Проведено экспериментальное исследование разработанных механизмов расширения ядра poST Web-IDE.

Исходный код разработанных, доработанных и переработанных компонентов poST-IDE располагается в GitHub-репозиториях. [14][15][16][17][18]

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».

Белоглазов Даниил Александрович _____

ФИО студента *Подпись студента*

« ____ »_____20 __г.

(заполняется от руки)

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1. Лащевски Т., Арора К., Фарр Э. Облачные архитектуры. Разработка устойчивых и экономичных облачных приложений. / Спб.: Питер, 2022. 320 с.

2. Сведения об AWS Cloud9 [Электронный ресурс] // Amazon Web Services [сайт]. URL: https://aws.amazon.com/ru/cloud9/details/ (дата обращения: 14.11.2021).

3. Cloud9 Core - Part of the Cloud9 SDK for Plugin Development [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/c9/core (дата обращения: 14.11.2021).

4. Getting started with the Cloud9 SDK [Электронный ресурс] // Cloud9 SDK [сайт]. URL: https://cloud9-sdk.readme.io/docs (дата обращения: 14.11.2021).

5. Houses the Asynchronous Module Definition API [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/amdjs/amdjs-api (дата обращения: 16.11.2021).

6. Create a package [Электронный ресурс] // Cloud9 SDK, 2021. URL: https://cloud9-sdk.readme.io/docs/create-a-package (Дата обращения: 14.11.2021).

7. Introduction to Eclipse Che [Электронный ресурс] // Eclipse Che Documentation [сайт]. URL: https://www.eclipse.org/che/docs/stable/overview/introduction-to-eclipse-che/ (дата обращения: 29.11.2021).

8. The Kubernetes-Native IDE for Developer Teams [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/eclipse/che (дата обращения: 29.11.2021).

9. Eclipse Theia is a cloud & desktop IDE framework implemented in TypeScript [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/eclipse-theia/theia (дата обращения: 29.11.2021).

10. Eclipse Che-Theia [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/eclipse-che/che-theia (дата обращения: 29.11.2021).

11. What is a Che-Theia plug-in [Электронный ресурс] // Eclipse Che Documentation [сайт]. URL: https://www.eclipse.org/che/docs/che-7/end-user-guide/what-is-a-che-theia-plug-in/ (дата обращения: 29.11.2021).

12. Зюбин В. Е. Процесс-ориентированное программирование: Учеб. пособие / Новосибирск. Новосиб. гос. ун-т. 2011. 194 с.

13. Мартин Р. Чистая архитектура. Искусство разработки программного обеспечения. / Спб.: Питер, 2018. 352 с.

14. Доработанное ядро poST-IDE [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/Zheltog/post-ide (дата обращения: 29.04.2022).

15. Доработанная корневая библиотека poST [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/Zheltog/post-ide-core (дата обращения: 29.04.2022).

16. Доработанный DSM-core модуля генератора poST-to-ST [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/Zheltog/post-to-st (дата обращения: 29.04.2022).

17. Переработанный DSM-wrapper для модулей poST-IDE [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/Zheltog/post-ide-dsm-wrapper (дата обращения: 29.04.2022).

18. DSM-manager (менеджер модулей) poST-IDE [Электронный ресурс] // GitHub [сайт]. URL: https://github.com/Zheltog/post-ide-dsm-manager (дата обращения: 29.04.2022).

ПРИЛОЖЕНИЕ А

Содержимое лог-файлов после экспериментального запуска

dsm-manager.log

2022-04-27 22:45:18 INFO App: use -help to see available running configurations

2022-04-27 22:45:18 INFO App: running manager on port 8383

2022-04-27 22:45:21 INFO Manager: starting available modules

2022-04-27 22:45:21 INFO Manager: getting available modules

2022-04-27 22:45:21 INFO Manager: available modules: AvailableModules(directory=/home/belog/post/post-ide/dsm-management/build/dsms/, modulesJarNames=[post2st])

2022-04-27 22:45:21 INFO Manager: starting module: post2st

2022-04-27 22:45:21 INFO Manager: starting /home/belog/post/post-ide/dsm-management/build/dsms/post2st.jar with -ma http://127.0.1.1:8383

2022-04-27 22:45:22 INFO Controller: request for /new-module

2022-04-27 22:45:22 INFO Manager: registered new module: name = post-to-st, host = 127.0.0.1, port = 35975

post-to-st.log

2022-04-27 22:45:22 INFO App: registering ourselves

2022-04-27 22:45:22 INFO App: connecting to manager by url: http://127.0.1.1:8383

ПРИЛОЖЕНИЕ Б

Результаты отработки запросов

На рисунке 11 представлен результат отработки запроса на получение актуального списка активных модулей.

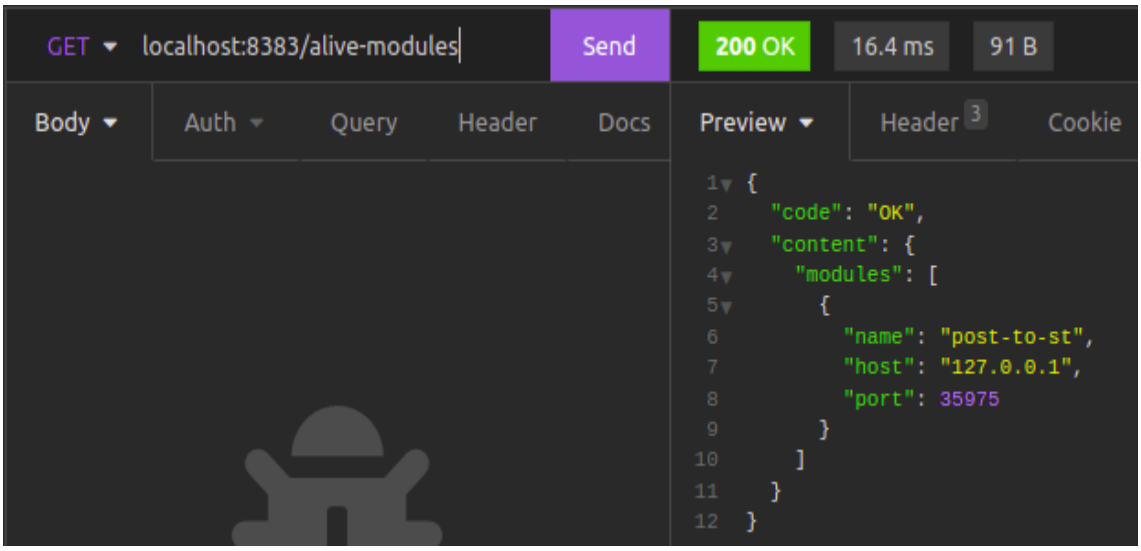

Рисунок 11 – Получение актуального списка активных модулей

На рисунке 12 представлен результат отработки запроса на получение списка доступных модулей.

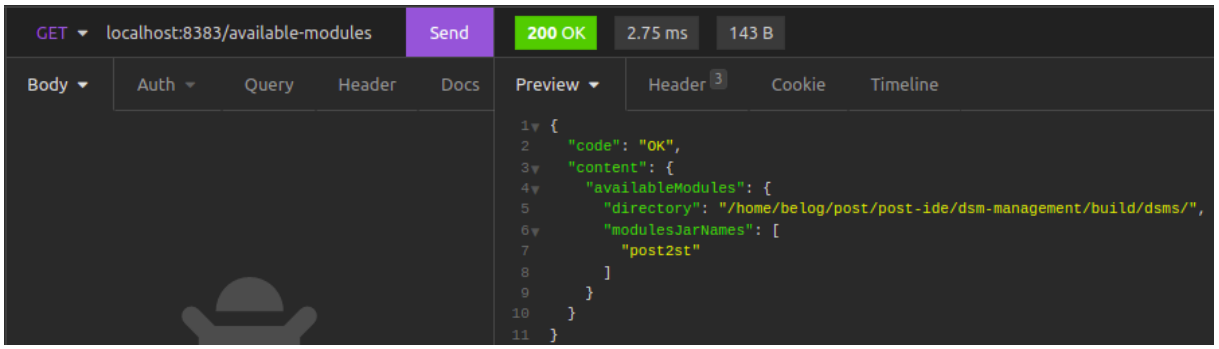

Рисунок 12 – Получение списка доступных модулей

ПРИЛОЖЕНИЕ В

Менеджер модулей poST IDE

Руководство оператора

Листов 9

Новосибирск, 2022

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном программном документе приведено руководство оператора по применению и эксплуатации программной системы, представленной в виде менеджера модулей (DSM-manager) интегрированной среды разработки для процесс-ориентированного языка poST.

В данном документе, в разделе «Назначение программы» указаны сведения о назначении программы и перечислены ее функции. В разделе «Условия выполнения программы» перечислены условия, являющиеся необходимыми для выполнения программы. Раздел «Выполнение программы» содержит последовательность действий оператора, которые необходимы для загрузки, запуска, выполнения и завершения программы. В разделе «Сообщения оператору» приведено описание текстовых сообщений и описаны действия оператора при их возникновении.

Оформление программного документа «Руководство оператора» произведено по требованиям ГОСТ 19.505-79 «ЕСПД. Руководство оператора» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

1 Назначение программы

Программная система предназначена для управления модулями (DSM) интегрированной среды разработки для процесс-ориентированного языка poST.

Функции:

- регистрация нового модуля в перечне активных;
- получение актуального списка активных модулей;
- получение списка доступных модулей;
- выполнение логики модуля;
- старт всех доступных модулей;
- старт конкретного модуля;
- остановка конкретного модуля.

2 Условия выполнения программы

2.1 Минимальный состав аппаратных средств

Для работы программного средства требуется персональный компьютер, включающий в себя:

- процессор, совместимый с архитектурой x86 с тактовой частотой 2 ГГц или выше;
- оперативную память объемом от 128 Мб;
- свободное пространство на диске от 1 Гб;
- устройства ввода: клавиатура, мышь;
- устройство вывода: монитор;
- операционную систему, совместимую с JVM (виртуальной машиной Java).

2.2 Минимальный состав программных средств

Для работы программного средства требуется наличие на персональном компьютере установленной JVM. Минимальная версия Java: 8. Также необходима установленная интегрированная среда разработки для языка poST (poST IDE).

2.3 Требования к оператору

Конечный оператор (администратор poST IDE) программы должен обладать практическими навыками работы с терминалом, с вызовом методов REST-API, а также с программами на языке Java.

3 Выполнение программы

3.1 Загрузка и запуск программы

Менеджер модулей и его конфигурационные файлы по умолчанию располагаются в подкаталоге dsm-management/build каталога poST IDE.

Файлы имеют следующие наименования:

- Java-архив DSM-менеджера – “manager.jar”;
- конфигурационный файл с перечнем доступных модулей – “available-modules.json”;
- конфигурационный файл со свойствами DSM-менеджера – “manager.properties”.

По умолчанию менеджер модулей автоматически запускается на порту, указанном в файле его свойств, после старта ядра IDE для языка poST. После старта менеджера происходит запуск всех модулей, упомянутых в соответствующем конфигурационном файле.

Администратору требуется выполнить следующие действия:

1. актуализировать порт запуска менеджера в файле manager.properties, если это необходимо;
2. актуализировать содержимое конфигурационного файла available-modules.json с перечнем доступных модулей (которые будут запущены сразу после старта IDE) – указать директорию расположения Java-архивов DSM и их наименования.

В случае, если менеджер модулей перестал работать по какой-то причине, администратор должен либо перезапустить IDE, либо выполнить ручной запуск менеджера. Ручной запуск осуществляется выполнением команды:

java -jar <наименование Java-архива менеджера> [ключи запуска]

Доступные ключи запуска:

- -amj <имя конфигурационного файла> – явное указание конфигурационного json-файла с перечнем доступных модулей;
- -sam – автоматический запуск всех доступных модулей, упомянутых в конфигурационном файле;

- -p <порт> – явное указание порта запуска менеджера;
- -ap – автоматический подбор порта запуска менеджера.

3.2 Выполнение программы

Предоставлена возможность производить запуск и остановку модулей в процессе работы DSM-менеджера в ручном режиме посредством вызова REST-методов API менеджера.

Для того, чтобы получить список доступных модулей, считанный из конфигурационного файла, необходимо выполнить REST-запрос без тела:

GET manager-host:manager-port/available-modules

Для того, чтобы получить актуальный список запущенных модулей, необходимо выполнить REST-запрос без тела:

GET manager-host:manager-port/alive-modules

Для того, чтобы запустить все доступные модули, упомянутые в конфигурационном файле, необходимо выполнить REST-запрос без тела:

GET manager-host:manager-port/start-all

Для запуска конкретного доступного модуля, упомянутого в конфигурационном файле, необходимо выполнить REST-запрос без тела:

GET manager-host:manager-port/start/{имя_модуля}

Для того, чтобы остановить конкретный активный модуль, необходимо выполнить REST-запрос без тела:

GET manager-host:manager-port/stop/{имя_модуля}

3.3 Завершение работы программы

Для завершения работы программного средства оператору необходимо выполнить остановку poST-IDE.

4 Сообщения оператору

В ходе работы программной системы производятся записи в лог-файлы в подкаталоге browser-app/log каталога poST IDE. Для DSM-менеджера и каждого запущенного DSM создаются собственные лог-файлы. Удаление содержимого какого-либо лог-файла или удаление какого-либо лог-файла как такового не влияет на работу системы – при необходимости записи новых сообщений недостающие файлы создаются автоматически.

Содержимое лог-файлов является человеко-читаемым. Каждое сообщение сопровождается временной меткой и комментарием на английском языке, позволяющим понять, какое конкретно действие выполнялось программной компонентой, а также прочитать описание ошибок в случае, если они возникли в ходе работы.

Исходя из содержания сообщений об ошибках, администратору требуется выполнить соответствующие действия для их устранения.

5 Лист регистрации изменений

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа	Подпись	Дата
Номер изм.	измененных	замененных	новых	аннулированных					

Таблица 9 – Лист регистрации изменений в программном документе «Руководство оператора»

ПРИЛОЖЕНИЕ Г

Менеджер модулей poST IDE

Описание программы

Листов 13

Новосибирск, 2022

СОДЕРЖАНИЕ

АННОТАЦИЯ

В данном программном документе приведено описание программного средства, представленной в виде менеджера модулей (DSM-manager) интегрированной среды разработки для процесс-ориентированного языка poST.

Оформление программного документа «Описание программы» произведено по требованиям ГОСТ 19.402-78 «ЕСПД. Описание программы» и ГОСТ 19.105-78 «Единая система программной документации (ЕСПД). Общие требования к программным документам (с Изменением N 1)».

1 Общие сведения

1.1 Обозначение и наименование программы

Наименование программы – poST IDE DSM-manager.

1.2 Программное обеспечение, необходимое для функционирования программы

- операционная система, совместимая с JVM (виртуальной машиной Java);
- JVM (минимальная версия Java: 8);
- интегрированная среда разработки для языка poST (poST IDE).

1.3 Языки программирования

Программное средство написано на языке Kotlin (версия 1.5.10) с использованием фреймворка Spring (версия Spring Boot: 2.4.5).

2 Функциональное назначение

2.1 Назначение программы

Программное средство предназначено для управления модулями (DSM) интегрированной среды разработки для процесс-ориентированного языка poST (poST IDE).

Программа решает следующие задачи:

- регистрация нового модуля в перечне активных;
- получение актуального списка активных модулей;
- получение списка доступных модулей;
- выполнение логики модуля;
- старт всех доступных модулей;
- старт конкретного модуля;
- остановка конкретного модуля.

2.2 Сведения о функциональных ограничениях на применение

Модуль IDE (DSM) должен включать специальную «компоненту-обёртку» (DSM-wrapper), удовлетворяющую контрактам (имеющую специфические ключи запуска и методы REST API). В противном случае DSM не сможет быть интегрирован в систему компонент poST IDE.

3 Описание логической структуры

3.1 Структура программы

Программное средство DSM-manager представляет собой kotlin-приложение на базе фреймворка Spring, являющееся REST-сервером.

Реализованные методы REST-API представлены в таблице 10.

Method + URL	**Описание**
GET /	“hello”-метод, позволяющий проверить, что менеджер запущен, и получить описание его API в виде HTML-страницы
POST /new-module	метод регистрации нового модуля в перечне активных
GET /alive-modules	метод получения актуального списка активных модулей
GET /available-modules	метод получения списка доступных модулей
POST /run/{moduleName}	метод выполнения логики модуля
GET /start-all	метод старта всех доступных модулей
GET /start/{moduleName}	метод старта конкретного модуля
GET /stop/{moduleName}	метод остановки конкретного модуля

Таблица 10 – DSM-manager API

Программа имеет несколько конфигурационных файлов, описание которых приведено в таблице 11.

Наименование файла	**Описание содержимого**
application.properties	конфигурационный файл фреймворка Spring

available-modules.json	стандартный конфигурационный файл с перечнем доступных модулей
log4j.properties	конфигурационный файл библиотеки логирования log4j

Таблица 11 – конфигурационные файлы DSM-менеджера

Программа имеет несколько доступных ключей запуска, их описание приведено в таблице 12.

Ключ запуска	**Описание**
-help	вывод информации о ключах запуска
-amj	явное указание конфигурационного файла с перечнем доступных модулей
-sam	автоматический запуск доступных модулей, перечисленных в конфигурационном файле, сразу после старта менеджера
-p	явное указание порта запуска сервера менеджера
-ap	автоматический подбор порта запуска сервера менеджера

Таблица 12 – ключи запуска DSM-менеджера

3.2 Алгоритм программы

После запуска менеджер производит старт своего REST-сервера и ожидает входящих запросов.

Если при запуске был указан ключ -amj, происходит считывание информации о доступных модулях из конфигурационного файла.

Если при запуске был указан ключ -sam, происходит запуск всех перечисленных в файле доступных модулей.

Если при запуске был указан ключ –p или -ap, REST-сервер менеджера стартует на приведенном или автоматически подобранном порту. По умолчанию значение порта считывается из конфигурационного файла application.properties.

В зависимости от того, какой метод REST-API сервера менеджера был вызван, программа выполняет соответствующие действия:

- /: возвращает описание программного интерфейса;

- /new-module: сохраняет информацию о регистрируемом модуле из содержимого запроса (адрес, порт, наименование);
- /alive-modules: производит запрос на каждый зарегистрированный модуль с целью отсеять неактивные, после чего возвращает обновлённый актуальный список активных модулей;
- /available-modules: возвращает информацию о доступных модулях, считанную из конфигурационного файла;
- /run/{module-name}: совершает запрос POST module-host:module-port/run с пришедшим телом запроса, URL адресата формируется по параметрам модуля, зарегистрированного под соответствующим наименованием;
- /start-all: совершает запуск Java-архивов доступных модулей, упомянутых в конфигурационном файле;
- /start/{module-name}: совершает запуск Java-архива конкретного доступного модуля из числа упомянутых в конфигурационном файле;
- /stop/{module-name}: совершает запрос GET module-host:module-port/stop, URL адресата формируется по параметрам модуля, зарегистрированного под соответствующим наименованием.

Выполнение операций сопровождается сохранением записей в лог-файл.

3.3 Связи между составными частями программы

Связи между основной процедурой (методом “main” main-класса) и функциями программы выполняются в виде стандартных вызовов подпрограмм (методов объектов).

3.4 Связи программы с другими программами

Программное средства работает в связке с ядром интегрированной среды разработки для языка poST и ее модулями (DSM).

4 Используемые технические средства

Программное средство эксплуатируется на персональном компьютере, на котором должны быть установлены: операционная система, совместимая с JVM (виртуальной машиной Java), JVM (Java 8+), а также poST IDE.

Режим работы – в формате консольного приложения, с взаимодействием с оператором / ядром poST IDE.

Входные и выходные данные находятся в телах REST-запросов.

5 Вызов и загрузка

Менеджер модулей и его конфигурационные файлы по умолчанию располагаются в подкаталоге dsm-management/build каталога poST IDE.

Файлы имеют следующие наименования:

- Java-архив DSM-менеджера – “manager.jar”;
- конфигурационный файл с перечнем доступных модулей – “available-modules.json”;
- конфигурационный файл со свойствами DSM-менеджера – “manager.properties”.

По умолчанию менеджер модулей автоматически запускается на порту, указанном в файле его свойств, после старта ядра IDE для языка poST. После старта менеджера происходит запуск всех модулей, упомянутых в соответствующем конфигурационном файле.

Администратору требуется выполнить следующие действия:

- актуализировать порт запуска менеджера в файле manager.properties, если это необходимо;
- актуализировать содержимое конфигурационного файла available-modules.json с перечнем доступных модулей (которые будут запущены сразу после старта IDE) – указать директорию расположения Java-архивов DSM и их наименования.

В случае, если менеджер модулей перестал работать по какой-то причине, администратор должен либо перезапустить IDE, либо выполнить ручной запуск менеджера. Ручной запуск осуществляется выполнением команды:

java -jar <наименование Java-архива менеджера> [ключи запуска]

Доступные ключи запуска:

- -amj <имя конфигурационного файла> – явное указание конфигурационного json-файла с перечнем доступных модулей;
- -sam – автоматический запуск всех доступных модулей, упомянутых в конфигурационном файле;
- -p <порт> – явное указание порта запуска менеджера;

- -ap – автоматический подбор порта запуска менеджера.

6 Входные данные

В качестве входных данных некоторых методов REST-API (new-module, run) сервер менеджера ожидает получить запросы с телами формата JSON (JavaScript Object Notation).

7 Выходные данные

В качестве выходных данных всех методов REST-API сервера менеджера, кроме метода new-module, передаются запросы с телами в формате JSON со следующей структурой:

{

“code”: …,

“content”: …

}

Где code – код ответа (OK / ошибка), а content – вложенный JSON-объект, представляющий некоторое содержимое (например, список активных DSM).

8 Лист регистрации изменений

Лист регистрации изменений									
Номера листов (страниц)					Всего листов (страниц) в документе	№ документа	Входящий № сопроводительного документа	Подпись	Дата
Номер изм.	измененных	замененных	новых	аннулированных					

Таблица 13 – Лист регистрации изменений в программном документе «Описание программы»