МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
УРАЛЬСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ
имени первого Президента России Б. Н. Ельцина

ИНСТИТУТ ЕСТЕСТВЕННЫХ НАУК И МАТЕМАТИКИ

Кафедра вычислительной математики и компьютерных наук

РАЗРАБОТКА ВЕБ-ОРИЕНТИРОВАННОЙ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТКИ ДЛЯ GROOVY НА JAVASCRIPT

Направление подготовки 09.04.03 «Прикладная информатика»

Образовательная программа
«Прикладная информатика в аналитической экономике»

Зав. кафедрой:

_____

Магистерская диссертация

Лопес Рейнага
Луиса Кармело

_____

Нормоконтроллер:

_____

Научный руководитель:

_____

Екатеринбург

2019

Реферат

Лопес Рейнага Л. К. РАЗРАБОТКА ВЕБ-ОРИЕНТИРОВАННОЙ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТКИ ДЛЯ GROOVY НА JAVASCRIPT, выпускная квалификационная работа на соискание степени магистра наук по направлению 09.04.03 «Прикладная информатика»: стр. 57, рис. 9, табл. 0, библ. назв. 6

Ключевые слова: ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ, ТЕКСТОВЫЙ РЕДАКТОР, GROOVY, JAVASCRIPT, КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ.

Объект исследования — разработка веб интегрированной среды разработки для языка Groovy с использования Javascript.

Цель работы — проектирование и разработка веб-ориентированной интегрированной среды разработки для groovy на javascript с возможностью компиляции кода и загрузки сторонних библиотек.

ABSTRACT

Lopes Reynaga L.K DEVELOPMENT OF WEB IDE FOR GROOVY LANGUAGE, master’s dissertation on training course 09.04.03 "Applied Informatics": 57 pages, 3 images, 0 tables, 6 library names.

Keywords: INTEGRATED DEVELOPMENT ENVIRONMENT, TEXT EDITOR, GROOY, JAVASCRIPT, CLIENT-SERVER APPLICATION

Research subject – development of web-oriented integrated development environment for Groovy language with Javascript.

Aim of the work – to design and develop web-oriented integrated development environment with features such as code compilation and ability to use 3rd party libraries.

Содержание

Введение

С развитием сети Интернет всё больше прикладного ПО разрабатываются не как самостоятельные приложения, а как веб-ориентированные приложения, работающие в интернет браузерах. Даже несмотря на то, что интегрированные среды разработки — технически сложное ПО, функциональность которого на данный момент не была полностью воспроизведена в веб-приложениях, существует множество веб-ориентированных решений, реализующих какие-то отдельные их аспекты. Для различных языков программирования функционирует множество веб-сайтов, предоставляющие текстовый редактор и компилятор, а на языке JavaScript реализовано множество библиотек для расширения стандартных возможностей редактирования текста в интернет браузере, добавляющих такие важные возможности редакторов кода, как, к примеру, подсветка синтаксиса и автоматическое дополнение введённого текста.

Однако, существующие веб-ориентированные среды разработки зачастую не универсальны и как правило поддерживают наиболее популярные среди разработчиков языки программирования. Для языка Groovy выбор веб-ориентированных сред разработки достаточно мал и их функциональность ограничена. Анализу существующих решений в данной работе посвящён отдельный раздел.

Для решения этой проблемы в рамках данной работы было разработано собственное программное решение, обладающее следующими функциями: подсветка синтаксиса введённого кода, возможность загрузки сторонних библиотек и их использование, автоматическое дополнение текста, на основе введённых переменных, методов и загруженных java библиотек, компиляция введённого кода и ограничение используемых в коде инструкций на стороне сервера.

Для реализации был использован JavaScript для клиентской части среды разработки и http-сервера, а также язык Groovy для компиляции введённого кода и анализа загруженных java библиотек

1. ПОСТАНОВКА ЗАДАЧИ

Задача данной работы — проектирование и разработка веб-ориентированной интегрированной среды разработки для языка программирования Groovy с использованием JavaScript. Разрабатываемая среда разработки должна обладать следующими функциями:

а) – подсветка синтаксиса введённого кода

б) – возможность загрузки сторонних библиотек и их использование

в) – автоматическое дополнение текста, на основе введённых переменных, методов и загруженных java библиотек

д) – компиляция введённого кода

ж) – ограничение используемых в коде инструкций на стороне сервера

2. ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

На данный момент существует несколько веб-сайтов и JavaScript библиотек, реализующие некоторые аспекты разрабатываемой в рамках данной работы интегрированной среды разработки. В данном разделе будет проведён обзор таких решений и их сравнение с разрабатываемым решением.

Веб-сайты с функциональностью среды разработки Groovy включают 4 решения:

а) – https://groovyconsole.appspot.com/

б) – https://groovy-playground.appspot.com/

в) – https://www.tutorialspoint.com/execute_groovy_online.php

д) – https://www.jdoodle.com/execute-groovy-online

Среди JavaScript библиотек, реализующих текстовый редактор для веб-браузера, следующие 2 поддерживают язык Groovy:

а) – CodeMirror

б) – Ace

2.1 https://groovyconsole.appspot.com/

Предоставляет из себя окно ввода программы, блок с возможными действиями и окно вывода результата программы. Поддерживаемые действия:

а) – Execute script – исполняет написанный пользователем код программы

б) – New script – очищает окно с введённым кодом программы, давая возможность ввести новый

в) – Publish script – опубликовывает программный код в публичный доступ

д) – View recent scripts – позволят просмотреть недавно опубликованные другими пользователями скрипты

Данная среда разработки позволяет исполнять простые Groovy скрипты и публиковать в общий доступ. Доступна базовая подсветка синтаксиса, выделяются некоторые ключевые слова языка, а также строковые значения и численные значения. Отсутствует автоматическое дополнение введённого текста при наборе. Отсутствует возможность подключить стороннюю библиотеку, следовательно, отсутствует возможность автоматического дополнения введённого текста API библиотек. Решение является проектом с открытым исходным кодом, которые доступен на сервисе GitHub

2.2 https://groovy-playground.appspot.com/

Интерфейс предоставляет из себя окно ввода программы, блок с возможными действиями и окно вывода результата программы. Поддерживаемые действия:

а) – Execute – исполняет написанный пользователем код программы

б) – Create – создаёт gist на сервисе GitHub из введённого кода программы

в) – Load from Gist – позволяет загрузить gist из сервисе GitHub в окно ввода программного кода

д) – Clear editor – очищает окно с введённым кодом программы, давая возможность ввести новый

Данная среда разработки позволяет исполнять простые Groovy скрипты и публиковать как GitHub gist. Доступна базовая подсветка синтаксиса, выделяются ключевые слова, а также строковые и численные значения. Отсутствует автоматическое дополнение введённого текста при наборе. Отсутствует возможность подключить стороннюю библиотеку, следовательно, отсутствует возможность автоматического дополнения введённого текста API библиотек. Решение является проектом с открытым исходным кодом, которые доступен на сервисе GitHub

2.3 https://www.tutorialspoint.com/execute_groovy_online.php

Предоставляет из себя окно ввода программы, панель инструментов и окно вывода результата программы. Данная среда разработки является более функциональной в сравнении с предыдущими рассмотренными, предоставляя больший набор инструментов пользователю. Доступны функции работы с текстом: изменение размера шрифта, табуляции. Есть возможность изменять цветовую тему, выбирая из 11 предложенных опций. Есть возможность сохранять текущий проект, загружать ранее сохранённые проекты. Возможно встроить свой код программы на сторонний веб-сайт по генерируемой ссылке. Пользователю предоставляется возможность редактировать опции компилирования, с которыми будет запущен проект. Доступна базовая подсветка синтаксиса, выделяются ключевые слова, а также строковые, численные значения, названия классов, и специальные символы языка. Отсутствует автоматическое дополнение при наборе. Отсутствует возможность подключить стороннюю библиотеку, следовательно, отсутствует возможность автоматического дополнения введённого текста API библиотек.

2.4 https://www.jdoodle.com/execute-groovy-online

Интерфейс данной среды разработки состоит из блока ввода кода программы, блока с различными доступными действиями и блока вывода.

Возможны такие действия, как:

а) – Исполнение введённой программы

б) – Сохранение (доступно для зарегистрированных пользователей)

в) – Просмотр сохранённых проектов (доступно для зарегистрированных пользователей)

д) – Просмотр недавно выполненных пользователем программ

ж) – Возможность одновременной работы над одним проектом с другими пользователями в этом же окне браузера с использованием библиотеки Together.js

й) – Возможность встроить код проекта в сторонний веб сайт, получив ссылку на него

к) – Возможность открыть файл с кодом программы из компьютера пользователя и загрузить в текстовый редактор

л) – Возможность сохранить проект в файл с кодом программы на компьютер пользователя

м) – Возможность переключаться между двумя цветовыми темами: светлой и тёмной

н) – Возможность выбора версии используемой groovy sdk

п) – Возможность использования «интерактивного режима»

р) – Возможность задать аргументы командной строки

с) – Возможность задать стандартный ввод для программы

Доступна базовая подсветка синтаксиса, выделяются ключевые слова, а также строковые, численные значения, названия классов, и специальные символы языка. Отсутствует автоматическое дополнение при наборе. Отсутствует возможность подключить стороннюю библиотеку, следовательно, отсутствует возможность автоматического дополнения введённого текста API библиотек. Данная WEB IDE является частью проекта JDoodle, предоставляющего WEB IDE для множества языков.

2.5 CodeMirror и Ace

Данные JavaScript библиотеки были объединены в один раздел, так как обладают практически одинаковой функциональностью. Это редакторы кода, работающие в веб-браузере, которые возможно встроить в любой веб-сайт. Они разработаны для универсальной поддержки любого языка программирования с помощью отдельных от основной библиотеки модулей. Среди реализованных для них модулей также есть и модуль, поддерживающий Groovy.

Обе библиотеки не обладают возможностью компилировать и исполнять введённый код. Также они не обладают возможностью анализировать текст программы и предлагать варианты автоматического дополнения вводимого текста на основе введённого кода.

Однако, обе обладают широкими возможностями текстового редактора. Среди них такие, как

а) – подсветка синтаксиса

б) – переключение между различными цветовыми темами

в) – автоматическая вставка отступов

д) – возможность быстрого доступа к различным функциям редактора с помощью комбинаций клавиш

ж) – возможность сворачивания и разворачивания текста

и многие другие.

2.6 Итоги

Проанализировав доступные варианты веб интегрированных сред разработки и клиентских JavaScript библиотек, мы можем увидеть, что ни одно их существующих решений не обладает разрабатываемой в рамках этой работы функциональностью. Тогда как некоторые решения предоставляют подсветку введённого кода и исполнение его на стороне сервера, они не позволяют загружать и использовать сторонние библиотеки, автоматически дополнять текст введённого кода. Другие же решения обладают богатыми возможностями редактирования текста, но не предоставляют возможность исполнения кода.

3. АНАЛИЗ ПРОГРАММНЫХ ИНСТРУМЕНТОВ

Во время разработки веб-ориентированной интегрированной среды разработки для языка Groovy были использованы различные языки программирования, а также сторонние библиотеки. В данном разделе проводится анализ существующих инструментов разработки, а также описание выбранных языков и библиотек, используемых в различных компонентах решения.

3.1 Анализ и выбор инструментов

Так как решения является клиент-серверным, то есть состоит из двух самостоятельных частей — работающей в браузере и работающей на сервере, то и анализ доступных языков и инструментов будет разделён на две части.

Так как среда разработки должна быть веб-ориентированной, её клиентская часть должна работать в веб-браузере. Несмотря на то, что браузеры поддерживают язык JavaScript, этот язык не является единственным использующимся в разработке веб приложений. Существует множество языков, разработанных специально чтобы компилироваться в JavaScript, а также множество инструментов, которые позволяют компилировать программы, написанные на других языках, в JavaScript. К таким языкам относятся:

а) – TypeScript

б) – Elm

в) – CoffeScript

д) – Grooscript

ж) – Scala.js

й) – Emscripten

Среди доступных вариантов наиболее целесообразным представляется использование языков, специально разработанных для компиляции в JavaScript. Рассмотрим далее языки, указанные в первых трёх пунктах.

CoffeScript - самый старый из приведённых языков. Был популярен среди разработчиков клиентских частей веб приложений, но с развитием стандарта ECMAScript, был вытеснен обычным JavaScript. Основные отличия этого языка от TypeScript и Elm – то, что он не имеет статической типизации. И нуждается в Babel’е для компиляции.

Elm - функциональный язык программирования, созданный специально для разработки веб-приложений. Официальная документация заявляет, что одна из главных особенностей языка в том, что он позволят избежать runtime ошибок. Так же его особенностями можно считать то, что он имеет опциональную статическую типизацию, а также компилируется в JavaScript с помощью собственного компилятора

TypeScript - язык программирования, разработанный и поддерживающийся Microsoft. Является надмножеством языка JavaScript. Это значит, что любая программа на языке JavaScript является так же корректной программой на языке TypeScript. Так же, как Elm, этот язык имеет опциональную статическую типизацию и компилируется в JavaScript с помощью собственного компилятора

Тогда как использование TypeScript или Elm могут существенно облегчить некоторые аспекты разработки, к примеру, благодаря наличию статической типизации, само по себе добавление инструмента – языка, компилируемого в JavaScript, увеличит сложность самого процесса разработки, поэтому в рамках этого проекта будет использован только JavaScript

Современная экосистема языка JavaScript предоставляет выбор из многих инструментов для разработки клиентской части приложения. Разделим их на следующие категории

а) – Инструменты управления состоянием приложения

1) Flux
2) Redux
3) Mobx
4) Многие другие

б) – Инструменты для отображения интерфейса

1) React
2) Vue
3) Многие другие

в) – Фреймворки, реализующие обе задачи

1) Angular
2) Ext.js

Сравним между собой инструменты из каждой категории, чтобы понять, нужны ли они в данной разработке.

Flux и Redux достаточно похожи друг на друга – оба реализуют предложенную Facebook flux архитектуру, альтернативу MVC архитектуре. Особенность архитектуры заключается в однонаправленном потоке данных, то есть для изменения store’а, который играет роль модели, используются action’ы, проходящие через его dispatcher и изменяющие состояние с помощью reducer’ов – чистых функций. Различие между Flux и Redux же

заключается в том, что с использованием Flux принято разделять состояние приложения на несколько store’ов, тогда как в Redux существует только один store, описывающий всё состояние приложения.

Mobx, согласно документации, не является контейнером для хранения состояния приложения, хотя может использовать в качестве такового. Mobx позволяет реактивно управлять состоянием и не привязывает разработчика к какой-то определённой архитектуре.

Так как по постановке задачи проекта все вычисления будут производиться на стороне сервера, клиентская часть не предполагает наличия каких-либо сложных данных и манипуляций с ними. Это позволяет обойтись в данном проекте без использования данных инструментов.

React и Vue – инструменты, созданные для задачи отображения интерфейса, тогда как Angular и Ext.js совмещают в себе инструменты так и для управления его состоянием. В отличие от React и Vue, Angular и Ext.js так же вынуждают разработчика использовать определённую архитектуру при написании приложения.

Данный проект не подразумевает использования сложных интерфейсов, поэтому использование отдельных инструментов для этого представляется нецелесообразным.

Для реализации серверной части приложения существует большое количество различных языков и инструментов. Но так как разрабатывается интегрированная среда разработки для языка для платформы Java, имеет смысл упомянуть именно эти языками. Среди языков для платформы Java можно выделить

а) – Java

б) – Groovy

в) – Scala

д) – Kotlin

Но несмотря на достоинства платформы Java, в разработке данного приложения выбор был остановлен именно на платформе Node.js и языке JavaScript. Обоснование этого решения будет дано в пункте 3.3 настоящего раздела.

3.2 Front-end инструменты

Так как разрабатываемая среда разработки является веб-интегрированной, она должна поддерживать работу в интернет-браузерах. Существует множество веб-браузеров, поддерживающих различные стандарты и технологии, но все они, за редким исключением работают с языком программирования JavaScript. В следствии этого клиентская часть решения была разработана с применением именно этого языка.

Несмотря на то, что большинство браузеров поддерживает базовые функции JavaScript, стандарт этого языка ежегодно дополняется множеством новых возможностей, которые разработчики многих браузеров могут не реализовывать по различным причинам. Одна из таких возможностей – модульная система, введённая в стандарте EcmaScript 6.

Модульная система позволяет разбивать код на множество файлов, в отличие от подхода, при котором весь код приложения находится в одном единственном файле. В данный момент модульная система ES6 не поддерживается некоторыми мобильными браузерами, а также браузером Internet Explorer.

Чтобы решить проблему поддержки некоторых стандартов браузерами, для Front-end разработки используется множество инструментов. Программы транспиляторы могут переводить код, написанный на одной версии языка JavaScript на код, написанный на другой версии. Программы-бандлеры могут собрать код, разделённый по разным файлам и объединить в код, собранный в одном файле. Один из таких бандлеров, поддерживающий работу вместе с трайнспайлером – Webpack.

Webpack – статический бандлер модулей для современных JavaScript приложений. Он строит внутренний граф зависимостей модулей и генерирует один или несколько итоговых JavaScript файлов. Webpack поддерживает работу без файла конфигурации, но при этом имеет возможности для очень гибкой конфигурации. Также он предоставляет возможности по минификации JavaScript кода – то есть его сжатия различными способами, такими, как удаление не использующегося кода или сокращение названий методов и переменных и удалений лишних символов пунктуации.

Для примера рассмотрим использующуюся в данном решении структуру модулей. Здесь приведены некоторые из них

а) – index.js

б) – compiler.js

в) – uploader.js

д) – line-numbers.js

ж) – compiler-output.js

Как видно, каждый модуль отвечает за определённый элемент интерфейса или его функцию. Так как они используют модульную систему

EcmaScrip 6, их работа не гарантируется в некоторых браузерах. Поэтому с помощью бандлера Webpack мы можем объединить их все в один файл – index.js, который и будет исполняться в браузере.

Кроме модульности исходного кода проекта, существует ещё одна проблема — его размер. Вместе с использованными библиотеками собранный Webpack код имеет размер около 600КБ, что является большим размером для относительно простой функциональности. Минификатор, встроенный в Webpack, может решить и эту проблему. Он сокращает объём кода, отправляемого клиенту, всего до 30КБ, то есть в 20 раз.

В самом Front-end приложении была использована библиотека Rx.js. Эта библиотека облегчает работу с асинхронным кодом, позволяя использовать реактивное программирование вместо callback'ов. Она была использована для осуществления AJAX запросов и обработки сайд эффектов и ответа на запрос.

Пример таких сайд эффектов – визуальные изменения в интерфейсе в момент отправки запроса или ответа на него. В данной интегрированной среде разработки осуществляются запросы на сервер, чтобы исполнить код или загрузить и сканировать jar библиотеку. В момент отправки запроса и до момента получения ответа интерфейс должен показать пользователю, что идёт исполнение неких операций, которые могу занять неопределённое время. Для этого пользователю показывается анимация загрузки. Управление этой анимацией и является сайд эффектом AJAX запросов.

Чтобы организовать работу с зависимостями Front-end приложения, был использован менеджер пакетов npm. Он позволяет легко получать необходимые зависимости из репозитория npmjs.com, а также настраивать скрипты для облегчения таких задач, как сборка исходных кодов приложения

в готовый JavaScript файл или пересборка их во время разработки при изменении исходного кода.

Итак, основными используемыми инструментами для Front-end части среды разработки можно выделить

а) – JavaScript

б) – Webpack

в) – Rx.js

г) – npm

3.3 Back-end инструменты

Back-end — то есть серверная часть приложения решает более широкий круг задач, чем Front-end. В зону ответственности Back-end приложения входят:

а) – http сервер

б) – реализация API

в) – возможность сохранения загруженных jar файлов для дальнейшего использования

г) – анализ загруженны jar файлов для предоставления клиентской части информации для автоматического дополнения введённого текста

д) – компиляция и исполнение принятого от клиентской части кода

е) – запрет на исполнение определённых инструкций

В качестве языка реализации Back-end были выбраны 2 языка – JavaScript и Groovy.

Несмотря на то, что JavaScript был разработан как язык для работы в интернет-браузерах, его развитие привело к тому, что была разработана платформа node.js. Node.js – это событийно ориентированная асинхронная платформа для исполнения JavaScript.

Node.js был разработан специально для того, чтобы обеспечить разработку сетевых приложений. В связи с этим в стандартной поставке включены инструменты для лёгкой работы с сетью, в частности, Node.js позволяет запустить http сервер с использованием всего одной стандартной функции.

За годы развития, Node.js стал широко используемым инструментов и в среде Front-end разработки, так как именно с помощью этой платформы работают менеджер пакетов npm и бандлер Webpack. Это позволяет объединить Front-end и Back-end в однородную среду, используя одни и те же инструменты и на клиентской и на серверной части.

Тот факт, что Node.js позволяет использовать не только одни и те же инструменты, но один и тот же язык и для клиентской и для серверной части также позволяет проектировать и разрабатывать так называемые изоморфные приложения. Такие приложения проще разрабатывать, так как нет необходимости связывать клиентское и серверное приложение с помощью какого-либо третьего инструмента.

Для таких функций, как, сканирование jar файлов, исполнение кода, а также запрет на исполнение определённых инструкций, использованы скрипты, написанные на языке Groovy.

Так как Groovy является языком для платформы Java, он имеет доступ к большому количеству различных инструментов, поставляемых с Java Development Kit. Одна из таких возможностей — анализ jar файлов.

Одна из важных особенностей Groovy — это то, что он может использоваться как скриптовый язык, то с его использованием можно выполнять простые инструкции и при этом исходный код на groovy не обязательно компилировать, в его стандартную поставку входит возможность исполнения .groovy файлов.

Для выполнения функции компилятора присланного с клиентской части кода, в библиотеке Groovy поставляется класс GroovyShell, который и выполняет переданные ему скрипты. GroovyShell поддерживает гибкую конфигурацию, что позволяет реализовать другую функцию разрабатываемой интегрированной среды разработки — конфигурируемый запрет на выполнение каких-либо инструкций. Использование класса стандартной библиотеки SecureASTCustomizer даёт возможность контролировать операции, выполняемы GroovyShell.

Итак, среди возможных инструментов были выбраны платформа Node.js для создания http-сервера, а также обеспечения работы инструментов разработки клиентской части, таких как бандлер Webpack. Node.js позволил использовать менеджер пакетов npm как для разработки Front-end, так и для Back-end части интегрированной среды разработки. Для решения проблем написания асинхронного кода клиентской части был использован Rx.js, который позволил использовать техники реактивного программирования для отправки AJAX запросов. Для реализации функции компилятора Groovy

кода, написанного на клиенте, а также для анализа jar файлов для расширения возможностей автодополнения и для реализации конфигурируемой системы запретов на вызов определённых операций из компилируемого кода, были использованы скрипты на языке Groovy и возможности его стандартной библиотеки.

4. ЯЗЫК ПРОГРАММИРОВАНИЯ GROOVY

Apache Groovy — объектно-ориентированный язык программирования для платформы Java с совместимым с Java синтаксисом. Данный язык программирования поддерживает как статическую, так и динамическую типизацию. Он может быть использован и как язык программирования и как сценарный язык.

4.1 Краткая история Groovy

Впервые о разработке Groovy заговори Джеймс Стрэчен в 2003 году. Groovy был отправлен jsp как JSR 241 и принят в 2004 году. В период с 2004 по 2006 было выпущено несколько версий языка. Версия, названная 1.0 была выпущена 2 января 2007 года. Далее в ноябре 2007 года была выпущена версия 1.5.

Groovy выиграл JAX 2007 Innovation award в 2007 году, а в 2008 Grails — фреймворк для веб-разработки для Groovy выиграл JAX 2008 Innovation award. В ноябре 2008 года SpringSource приобрели компании Groovy и Grails. В августе 2009 года SpringSource была приобретена VMware. В июле 2012 версия 2.0 Groovy была выпущена. Кроме прочих новых возможностей, она добавляла статическую компиляцию и статическую проверку типов.

На данный момент существую две версии Groovy: стабильная 2.5.7 и бета-версия 3.0.0.

4.2 Особенности Groovy

Большинство корректных java файлов так же являются корректными groovy файлами. Несмотря на то, что эти два языка очень похожи, Groovy код может быть более компактным, поскольку он может опускать некоторые элементы, которые нужны Java.

Возможности Groovy, не встречающиеся в Java включают в себя динамическую типизация с ключевым словом def, перегрузку операторов, поддержка синтаксисических конструкций для списков и ассоциативных массивов, регулярных выражений и интерполяции строк, дополнительные вспомогательные методы, оператор безопасной навигации ?. для избегания null pointer.

Начиная со второй версии, Groovy также поддерживает модульность, что позволило уменьшить размер Groovy библиотеки, проверку типов и статическую компиляцию, а также multicatch blocks.

Groovy предоставляет поддержку для языков разметки, таких как xml и html.

В отличие от Java, исходные коды Groovy могут быть выполнены без компиляции, как скрипт, если они имеют код вне определений классов, если это класс с методом main или если он Runnable либо GroovyTestCase. Groovy скрипт полностью парсится и компилируется внутри стандартных инструментов поставки Groovy и полученный артефакт не сохраняется.

Groovy реализует GroovyBeans — Groovy версию JavaBeans. В Groovy геттеры и сеттеры свойств генерируются неявно.

Groovy поддерживает расширение прототипов, что позволяет, к примеру, добавлять новые методы к стандартным классам, таким как Number, String и другие. Изменения прототипов, вызванные Groovy, однако, не видны из Java кода.

Groovy является объектно-ориентированным языком программирования, однако он предоставляет и возможности для функционального программирования, такие как замыкание и каррирование.

Groovy предоставляет доступ к абстрактному синтаксическому дереву, полученному во время компиляции для того, чтобы разработчики могли модифицировать его.

5. СХЕМА ВЗАИМОДЕЙСТВИЯ КОМПОНЕНТОВ

Данная интегрированная среда разработки является клиент-серверным приложением, соответственно её можно разделить на Front-end и Back-end. Но и их, в свою очередь, можно разделить на множество компонент каждый из которых выполняет свою задачу.

Ниже приведена схема компонент Front-end части приложения

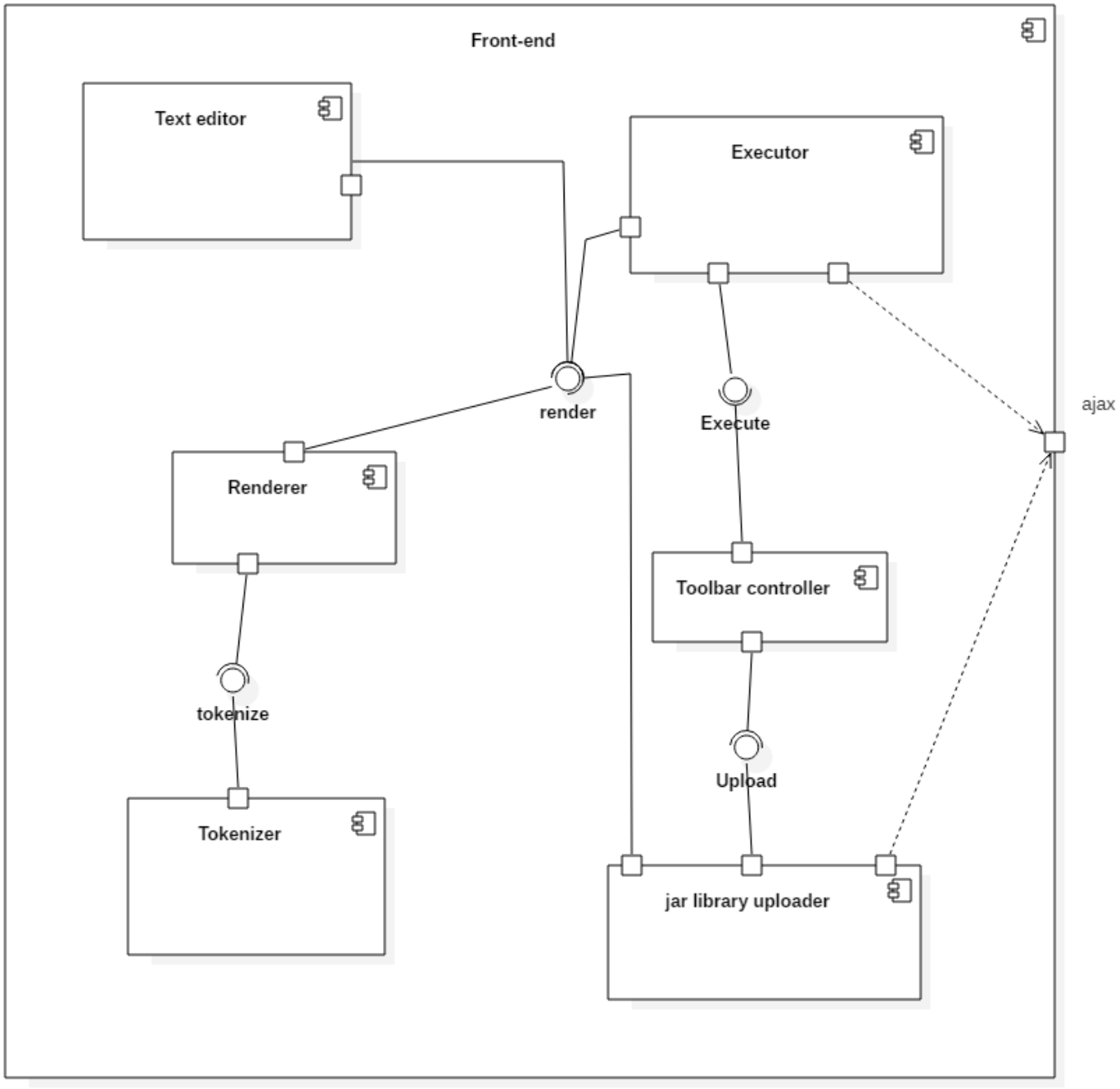

Рисунок 1 - компоненты Front-end

На рисунке отображена схема взаимодействия компонент. Среди них можно выделить Text editor – компонента, отвечающая за редактирование кода, Tokenizer – отвечающий за токенизацию текста, введённого в редактор кода, Renderer – отвечающий за отображение текста в формате HTML и вывод его в текстовый редактор, Toolbar controller, отвечающий дающий пользователю возможность загрузить jar библиотеку или скомпилировать код, с помощью jar library uploader и Executor соответственно.

Далее рассмотрим схему Backend части приложения.

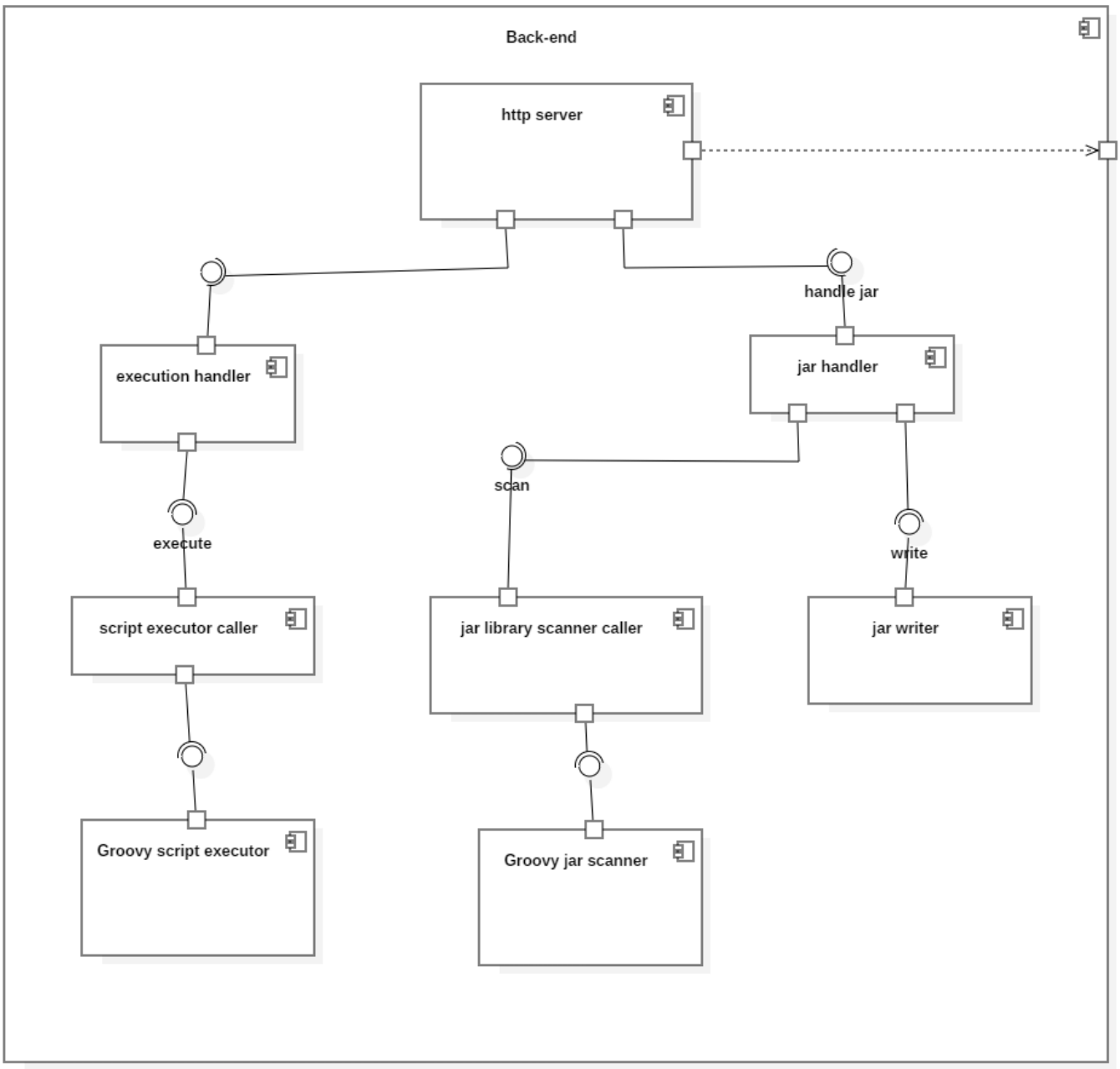

Рисунок 2 - схема компонентов Back-end

На рисунке 2 видно, что в Back-end части приложения можно выделить такие части, как

а) – http-server – принимает и отвечает на http-запросы клиентской части

б) –execution handler – принимает запрос на выполнение кода программы и формирует ответ

в) – script executor caller – компонент, запускающий Groovy скрипт для выполнения полученного от клиента кода

г) – Groovy script executor – Groovy скрипт, компилирующий и выполняющий присланный с клиентской части код

д) – jar handler – принимает загруженный jar файл с библиотекой и формирует ответ

е) – jar writer – записывает во временное хранилище принятый jar файл

ж) – jar library scanner caller – компонент, запускающий Groovy скрипт для анализа присланного файла

з) – Groovy jar scanner – Groovy скрипт, анализирующий присланный jar файл

Как видно из рисунка 1 и рисунка 2, Front-end и Back-end взаимодействуют друг с другом с помощью протокола http. За формирование ajax запросов в клиентской части среды разработки отвечают компоненты Executor и jar library uploader, а за принятие запроса и ответ на него в Back-end части отвечает http-server.

На схемах не показаны такие компоненты, как сама Node.js платформа, хранилище jar файлов, т.к. оно представляет из себя просто область на HDD/SSD, command line shell, связывающий между собой приложение на Node.js и Groovy скрипты, а также веб-браузер, но их использование в

данной интегрированной среде разработки, было изложено в других разделах.

6. ИНТЕРФЕЙС СРЕДЫ РАЗРАБОТКИ

Для удобства использования данной средой разработки, был реализован графический интерфейс пользователя. Так как она является веб-ориентированной, интерфейс доступен в веб-браузере при переходе по адресу http сервера. Рассмотрим интерфейс подробнее на приведённом ниже рисунке

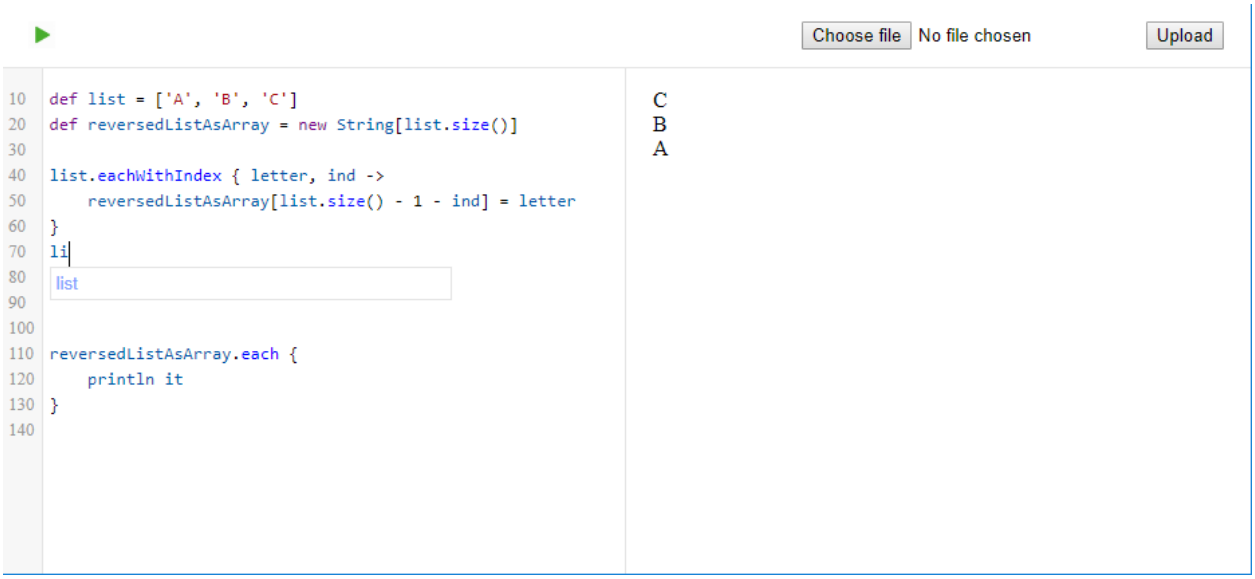

Рисунок 3- интерфейс клиентской части приложения

На рисунке 3 видно, что условно интерфейс можно разделить на 3 части:

а) – панель инструментов, находящая в верхней части интерфейса и содержащая кнопку запуска кода и кнопку загрузки jar библиотек на серверную часть приложения

б) – рабочая область, содержащая текстовый редактор для ввода кода программы

в) – область для вывода результата работы программы

Рабочая область, в свою очередь, разделяется на панель, отображающую номер строки кода в формате n*10, где n – порядковый номер строки, а также текстовый редактор. Как видно из рисунка 3, текстовый редактор выделяет различные составные части кода, такие как ключевые слова, методы, строки и переменные, различными цветами. На рисунке также отображено окно, предоставляющее подсказки для автоматического дополнения набранного текста. На примере при наборе символов li предлагается автоматически дополнить это до названия переменной – list.

7. РЕАЛИЗАЦИЯ СРЕДЫ РАЗРАБОТКИ

Как было сказано в предыдущих разделах, данная среда разработки была разработана по архитектуре клиент-серверного приложения. Хотя клиентская и серверная часть связаны друг с другом, они будут рассмотрены по отдельности.

7.1 Реализация API серверной части

API серверной части включает в себя 3 точки входа, находящиеся по адресам

а) – «/»

б) – «/api/execute»

в) – «/api/upload»

Роутер принимает GET запрос по адресу в пункте а) и отправляет в ответ HTML страницу, сформированную из handlebars шаблона, содержащую разметку, а также ссылки на стили или JavaScript, необходимые для работы клиентской части приложения.

Для адреса в пункте б) принимаются только POST запросы. Это API используется для приёма кода с клиентской части, его исполнения и ответа с результатом. Запрос должен иметь Content-type: application/json и его тело должно иметь следующий формат:

text: string

includeJars: string[]

Здесь поле text содержит код программы, представленный одной строкой, а поле includeJars список названий jar файлов, которые следует подключить при выполнении скрипта. Далее текст команды и список jar файлов передаётся модулю приложения, отвечающему за выполнение программы. Этот модуль будет описан подробнее ниже. Ответ этого модуля обрабатывается и передаётся клиентской части. Клиентской части отправляется ответ формата

result: String;

exception: String;

line: Number;

Поле result содержит в себе результат выполнения программы в текстовом виде. Оно может быть равным null, тогда в ответе будут заполнены остальные поля: exception – текст ошибки, которая произошла во время выполнения программы и line – номер строки, в которой произошла ошибка. Не все ошибки будут иметь конкретный номер строки и в этих случаях он будет равен -1.

Адрес в пункте в) также принимает только POST запросы. Запрос должен содержать в себе один jar файл и иметь Content-type: multipart/form-data. Далее файл передаётся в модуль, который сохраняет файл в хранилище.

Каждому клиенту в момент обращения к http серверу присваивается уникальный id, который, хранится в виде cookie в браузере. Каждому такому уникальному id соответствует свой раздел в хринилище jar файлов. Модуль сохранения jar файлов при каждом запуске серверной части приложения очищает хранилище. Далее, когда модуль API обращается к модулю сохранения, по cookie из запроса определяется, в какой именно раздел хранилища записывать jar библиотеку.

После ответа от модуля хранилища об удачной записи, управление передаётся модулю, отвечающему за сканирование jar файлов. Подробнее его работа будет описана ниже. Ответ от этого модуля обрабатывается и отсылается в формате

filename: String

scan: Array

filename – имя файла, загруженного с клиентской части, с которым он был сохранён в хранилище jar файлов. В дальнейшем клиентской части понадобится это имя, чтобы включать его в запросы для компилирования кода.

scan – информация, полученная при анализе jar файла. Подробный формат этого поля будет описан в следующем подразделе.

7.2 Реализация исполнения программы и анализа jar файлов

Модуль исполнения программы состоит из двух частей

а) – JavaScript код, запускающий Groovy скрипт

б) – Groovy скрипт, компилирующий программу

Листинг кода из пункта а) приведён в приложении А. Из него можно увидеть, что для того, чтобы запустить Groovy скрипт, Node.js запускает shell подпроцесс, который в свою очередь запускает Groovy скрипт и передаёт ему аргументом текст программы, присланный с клиентской части.

Если запрос, присланный клиентским приложением, содержал в себе названия jar файлов, groovy передаётся аргумент «–cp» с полными путями до этих файлов. Это добавляет jar библиотеки в classpath, чтобы к ним можно было получить доступ во время исполнения присланного с клиентского приложения кода.

Можно заметить, что вместо стандартного метода Node.js exec(), который мог бы сразу запустить groovy, используется метод spawn(), который запускает PowerShell, который уже в свою очередь запускает groovy. Это вызвано особенностями работы утилиты cmd операционной системы windows, которую использует Node.js при вызове exec(). При попытке передать из cmd аргументы groovy, содержащие символ перевода строки «\n», программа выполняется только до первого перевода строки. Эта проблема отсутствует в PowerShell.

В приложении Б приведена часть кода файла scriptRunner.groovy, который отвечает за исполнение присланного клиентом скрипта. В листинге можно увидеть, как настраивается SecureASTCustomizer, который позволяет ограничивать исполнение некоторых операций в GroovyShell. Здесь полю receiversBlackList выставляется значение ["java.lang.System"], означающее, что в момент любого обращения к методам класса System исполнение кода, присланного клиентом, будет прервано с соответствующим исключением.

Далее в зависимости от успешности или не успешности исполнения программы, заполняется объект, описывающий результат исполнения программы. Этот объект выводится в формате JSON и, как показано в приложении А в листинге, в строках с 28 по 31, передаётся модулю API.

Стоит отметить, что в приложении Б в строках 49 и 50 создаются объекты класса StringWriter и PrintWriter, который, в свою очередь, присваивается свойству out объекта класса Script в строке 54. Это вызвано тем, что стандартный вывод исполняемого скрипта по умолчанию тот же, что и у самой программы, запускающей GroovyShell скрипт. То есть если в коде, присланном с клиентского приложения встретится операция print или println, она будет выведена в PowerShell. В нашем же случае необходимо перехватывать вывод GroovyShell, чтобы в дальнейшем сформировать из

него объект с результатами выполнения скрипта. Для этой цели и используется PrintWriter. Но существуют способы обойти PrintWriter и выводить результаты скрипта в PowerShell, к примеру, использование System.out.println. Для таких случаев в конфигурации SecureASTCustomizer запрещёно обращение к методам System.

Также, как и модуль исполнения программы, модуль анализа jar файлов состоит из 2 частей

а) – JavaScript код, запускающий Groovy скрипт

б) – Groovy скрипт, анализирующий jar файл

Листинг кода из пункта а) приводиться не будет в силу его схожести с кодом из приложения А. Он также запускает PowerShell для запуска Groovy скрипта и передаёт адрес анализируемого jar файла в classpath и как аргумент Groovy скрипта.

Приложение В включает в себя листинг кода Groovy скрипта jarScanner.groovy. Из листинга видно, что для анализа jar файла используются стандартные средстка Java Development Kit. Объект класса JarInputStream предоставляет доступ к пакетам и классам, найденным в jar файле. Далее мы получаем информацию о всех найденные классах с помощью статического метода forName() класса Class. Из найденного класса мы получаем информацию о публичных методах и полях, которые нужны будут клиентской части приложения для автоматического дополнения.

Описание каждого класс формирует объект ClassDescription, поля которого включают:

а) – fullName – полное имя класса вместе с именем пакета. Пример: org.test.CustomClass

б) – name – имя класса

в) – packageName - имя пакета, содержащего класс

д) – methods– массив названий публичных методов клсса

ж) – fields – массив названий публичных полей класса

Каждый найденный класс затем добавляется в массив result и скрипт возвращает объект в формате JSON как результат выполнение.

7.3 Реализация клиентской части

Введённый в клиентском приложении код разбивает на токены токенизатор. Токенизатор принимает на вход текст, разбивает его на строки и по-очереди обрабатывает их. Токенайзер не находит синтаксические ошибки в введённом коде.

Любые пробелы в строке игнорируются. В начале токенизатор проверяет, не является ли следующая подстрока строкой с точки зрения языка Groovy, то есть не начинается ли она с кавычек. Если подстрока начинается с кавычек, необходимо проверить, нет ли в данном случае интерполяции строк.

Если подстрока не начинается с кавычек, токенизатор проверяет, не начинается ли строка со специальных символов, регулярным выражением /[\[\]{}\(\),;\:\.]/. Если первый символ удовлетворяет выражению, это означает, что этот символ является символом пунктуации. В зависимости от того, какой именно это символ, он может означать начало нового контекста или конец предыдущего.

Если символ не удовлетворяет этому регулярному выражению, токенизатор проверяет, не является ли символ цифрой. Если символ является числом, дальнейшая подстрока, содержащая цифры, будет отмечена как число.

В случае если символ является слэшом «/», токенизатор проверяет следующие за ним сиволы. Если это символ «*» или «/», то он либо обрабатывает следующий строки как комментарий, либо сразу помечает всю текущую строку как комментарий. В противном случае следующая подстрока токенизируется как string.

Если символ равен «-» и сразу за ним следует символ «>», токенизация начинается заново со следующей подстроки.

Если символ удовлетворяет регулярному выражению /[+\-*&%=<>!?|\/~]/, то следующая за ним подстрока, удовлетворяющая регулярному выражению /[+\-*&%=<>|~]/, считается оператором.

Если первый символ не удовлетворил ни одному описанному выше условию, указатель текущего символа двигается по подстроке, пока он удовлетворяет регулярному выражению /[\w\$_]/.

Если следующий текущий символ равен «@», указатель снова двигается, пока следующий символ удовлетворяет регулярному выражению /[\w\$_]/ и вся подстрока отмечается как meta.

Если же до этой подстроки находился символ «.», подстрока помечается как property.

Если следующий символ равен символу «:», то подстрока также помечается как property.

Если ни одно из условий не было удовлетворено, текущая подстрока сравнивается с ключевыми словами языка Groovy. Если подстрока равна одному из слов "null true false this", подстрока помечается как atom.

Если подстрока равна одному из ключевых слов "abstract as assert boolean break byte case catch char class const continue def default do double else enum extends final finally float for goto if implements import in instanceof int interface long native new package private protected public return short static strictfp super switch synchronized threadsafe throw throws trait transient try void volatile while", подстрока помечается как keyword.

Если ни одно из условий не было удовлетворено, подстрока помечается как variable.

Во время токенизации на каждый найденный токен вызывается колбэк. В этом колбэке клиентское приложение выполняет 3 функции:

а) - заполняет массив строк, где каждая строка – это массив токенов, вместе с подстрокой, индексом начального символа в строке и пометкой, которую вернул токенайзер.

б) – заполняет Set’ы переменных и свойств, которые затем будут использованы в автодополнении печатаемого текста

в) – заполняет массив строк в виде простого текста, как они были введены в текстовый редактор

Затем результат работы токенайзера получает модуль, отвечающий за отображение текстового редактора.

Текстовый редактор не может быть реализован стандартными средствами браузера, такими как textarea или input, потому что в связи с необходимостью подсветки введённого кода, необходимы большие

возможности стилизации текста, а стандартные элементы для ввода текста не предоставляют таких возможностей.

Вместо того, чтобы просто выводить текст в стандартные элементы для редактирования текста браузера, модуль отображения текстового редактора выводит каждую строку, обёрнутую в специальный HTML тег pre, который сохраняет форматирование текста дочерних элементов. Каждый токен выводится внутрь элемента pre соответствующей строки и обёрнут в тег span, которому присвоен класс, соответствующий пометке, которую присвоил токену токенайзер. Этот класс используется, чтобы с помощью css стилизовать различные токены текста для подсветки синтаксиса языка.

Такой подход позволяет стилизовывать выводимый в окно текстового редактора код, но у него есть минус: это окно не является «текстовым редактором», это просто текст, с которым пользователь никак не может взаимодействовать.

Следовательно, все функции текстового редактора, должны быть реализованы самостоятельно. В данном текстовом редакторе реализованы такие базовые функции, как: отображение и позиционирование курсора, позиционирование курсора с помощью клавиатуры, добавление и удаление символов.

Зачастую интегрированные среды разработки предоставляют такую функцию, как автоматическое дополнение набираемого текста. Когда пользователь начинает набирать название переменной или метода, появляется список с возможными названиями, из которых можно выбрать нужное значение, чтобы мгновенно дополнить строку до этого значения. Такая функция реализована и в клиентской части данной интегрированной среды разработки.

Модуль автоматического дополнения связан с модулем текстового редактора и попытка вычислить и показать подсказки происходит при каждом изменении введённого текста.

Чтобы вычислить подсказки, модуль получает несколько Set’ов, сформированные после работы токенайзера, а также загрузчика jar файлов на серверную часть:

а) – доступные из текущего кода свойства

б) – доступные из текущего кода переменные

в) – доступные из загруженных jar файлов переменные

д) – доступные из загруженных jar файлов свойства

Далее модуль по номеру текущей строки получает массив токенов текущей строки и по позиции курсора вычисляет, какой токен в данный момент редактируется текстовым редактором. Используя текстовое значение текущего токена, на основании информации из Set’ов переменных и свойств, модуль вычисляет похожие строки и показывает их в виде выпадающего списка.

Когда отображён список подсказок, в него можно перейти из поля текстового редактора с помощью клавиатуры или мыши и выбрать нужную подсказку.

В клиентском приложении также реализованы модули для исполнения набранного кода и загрузки jar файла.

Модуль загрузки состоит из простой функции отправки ajax запроса с текстом программы и названием подключённых jar файлов, а также отображения – одной кнопки. Кроме этого, модуль осуществляет исполнение набранного кода программы по нажатию клавиши «F9».

Подробное описание API запроса исполнения кода программы и форматы запроса и ответа приведены в разделе, посвящённом реализации API.

После получения ответа модуль проверяет его и если в ответе содержится поле result, то модуль выводит его в поле вывода результатов работы программы. Если в ответе содержится поле exception, модуль выводит его содержимое в то же поле, но изменяет класс его DOM node, чтобы предоставить возможность различной стилизации результата работы программы и ошибок, возникнувших в процессе исполнения кода программы.

Модуль загрузки jar файла также, как и модуль исполнение кода, состоит из визуального представления: input’a с типом file, который позволяет пользователю выбрать файл со своего компьютера. Input также имеет атрибут accept=".jar", чтобы облегчить фильтрацию файлов в проводнике. Рядом с input’ом в представлении находится кнопка, вызывающая саму функцию загрузки файла.

Функция загрузки отправляет один ajax запрос на серверную часть приложения. Описание API и формата запроса дано в соответствующем разделе.

Когда модуль получает ответ, он проверяет тело ответа на наличие поля exception. Если оно присутствует, то во время обработки jar серверной частью произошла некая ошибка и эта ошибка выводится пользователю. Если в ответе присутствует поле scan, то из этого поля получается информация о пакетах, классах, их методах и переменных, которые содержались в загруженном jar файле. Каждый класс обрабатывается и в текст, программы вставляются инструкции импорта классов и перезапускается обновление отображение текстового редактора. Информация

о методах и полях класса же заполняет соответствующие Set’ы, которые в дальнейшем используются модулем автоматического дополнение текста кода программы.

8. ВОЗМОЖНОСТИ ДАЛЬНЕЙШЕГО РАЗВИТИЯ РЕШЕНИЯ

У данной интегрированной среды разработки существует множество возможностей дальнейшего развития. Хотя некоторые её функции в текущий момент и отсутствуют у её аналогов, некоторые их функции, в свою очередь отсутствуют и у данного решения.

Начиная с клиентской части приложения, во-первых, стоит отметить возможность развития и добавление новых функций к текстовому редактору. На текущей стадии развития текстовый редактор является очень простым, поддерживающим только базовые функции, такие как вставка текста, удаление, ориентация по тексту с помощью клавиатуры. В продвинутых редакторах же присутствуют такие возможности, как использование нескольких курсоров, скрытие и раскрытие блоков текста, возможность использования нескольких вкладок и многие другие.

Далее возможно расширить общую функциональность среды разработки. Позволять пользователю загружать скрипт со своего компьютера, сохранять его состояние в local storage браузера. Добавить возможность просматривать загруженные им ранее jar файлы и подключать/отключать их из интерфейса.

Кроме новой функциональности, также возможно оптимизировать быстродействие клиентской части. К примеру, в данный момент при изменении текста происходит полная перерисовка отображения текстового редактора, тогда как более производительным подходом было бы обновлять только строки, в которых были сделаны изменения.

На стороне серверной части среды разработки также возможны различные доработки, в главную очередь доработки производительности. Хотя у использования платформы Node.js есть свои плюсы, одним из способов оптимизации производительности будет перевод всего серверного приложения на язык Groovy, что позволит запускать компиляцию программного кода и анализ jar файлов напрямую из приложения.

9. КРОССПЛАТФОРМЕННОСТЬ И КРОССБРАУЗЕРНОСТЬ

Так как данная интегрированная среда разработки является веб-ориентированной, важной её возможностью является работа в различных веб-браузерах.

Данная среда разработки полностью функционирует в браузерах, основанных на Chromium, Firefox, а с использованием Babel для транспиляции кода клиентской части, может работать и в браузере Edge.

Серверная же часть разработана с использованием платформ Java и Node.js, которые работают на различных операционных системах, следовательно и серверное приложение является кроссплатформенным.

ЗАКЛЮЧЕНИЕ

В ходе данной работы, была разработана веб-ориентированная интегрированная среда разработки для языка Groovy. Приложение построено по клиент-серверной архитектуре.

Среда разработки позволяет редактировать код, написанный на языке Groovy, предоставляет подсветку синтаксиса, возможность автоматического дополнения текста, загрузки сторонних jar библиотек и исполнения написанного кода, а также возможность ограничивать на стороне сервера исполняемы в написанном коде операции.

Было найдено и проанализировано 4 аналога, функциональность которых не отвечает требованиям. Также было проанализировано 2 текстовых редактора, работающих в браузере. Текстовые редакторы хоть и имели богатые возможности по работе с кодом на различных языках, в том числе Groovy, не обладали возможностями полноценной интегрированной среды разработки.

Клиентская и серверная часть среда разработки написаны с использование JavaScript с использование платформы Node.js. Также часть функций серверного приложения выполняют скрипты, написанные на Groovy и запускаемые основным серверным приложением.

Были подробно разобраны компоненты как клиентской, так и серверной части.

Полученное решение является кроссплатформенным и кроссбраузерным.

Разработанная интегрированная среда разработки обладает широкими возможностями дальнейшего развития, как в клиентской, так и в серверной части приложения. Возможно не только добавление новых функций, но и оптимизация работы существующего кода.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ И ЛИТЕРАТУРЫ

1 Dierk K. Groovy in Action MEAP Edition, 2011. — 340c

2 The Apache Groovy programming language [Электронный ресурс] // URL: http://groovy-lang.org (дата обновления: 23.05.2019)

3 Overview (Groovy 2.5.7) [Электронный ресурс] // URL: http://docs.groovy-lang.org/latest/html/api/index.html (дата обновления: 24.05.2019)

4 Overview (Java Platform SE 7) [Электронный ресурс] // URL: https://docs.oracle.com/javase/7/docs/api/overview-summary.html (дата обновления: 24.05.2019)

5 CodeMirror [Электронный ресурс] // URL: https://codemirror.net (дата обновления: 21.05.2019)

6 Ace - The High Performance Code Editor for the Web [Электронный ресурс] // URL: https://ace.c9.io (дата обновления: 21.05.2019)

ПРИЛОЖЕНИЕ А

```
const {exec, spawn} = require('child_process');
const path = require('path');


const getJarFullPath = (req, jarName) => {
    return path.join(process.cwd(), 'jar-storage', req.cookies['g-session'], jarName);

};

const buildScriptParameters = (req, script, jars, scriptRunnerPath) => {
    if (jars && Array.isArray(jars) && jars.length > 0) {
        const jarsPaths = jars.map(jar => getJarFullPath(req, jar));
        return [`groovy -cp "${jarsPaths.join(';')}" ${scriptRunnerPath} "${script}"`]
    } else {
        return [`groovy ${scriptRunnerPath} "${script}"`]
    }
};

const executeScript = (req, script, jars) => {
    return new Promise(
        (res, rej) => {
            const scriptRunner = spawn('powershell.exe', buildScriptParameters(req, script, jars, path.join(__dirname, 'scriptRunner.groovy')));

            let timeoutId;
            let isDataSent = false;


            scriptRunner.stdout.on('data', (data) => {
                res(data.toString());
                isDataSent = true;
            });

            scriptRunner.stderr.on('data', (data) => {
                res(data.toString());
                isDataSent = true;
            });

            scriptRunner.on('exit', () => {
                clearTimeout(timeoutId);
                !isDataSent && rej();
            });

            scriptRunner.on('error', () => {
                rej();
            });

            timeoutId = setTimeout(
                () => {
                    spawn("taskkill", ["/pid", scriptRunner.pid, '/f', '/t']);
                    rej();
```

Рисунок А.1 Код executeGroovy.js, часть 1 из 2

```
                },
                7000
            )
        }
    )
};

module.exports = {
    executeScript
};
```

Рисунок А.2 Код executeGroovy.js, часть 2 из 2

ПРИЛОЖЕНИЕ Б

```groovy
def astCustomizer = new SecureASTCustomizer()
astCustomizer.with {
    receiversBlackList = ["java.lang.System"]
}

def conf = new CompilerConfiguration()
conf.addCompilationCustomizers(astCustomizer)

GroovyShell shell = new GroovyShell(new Binding(), conf)

def result = new ResultBuilder()

if (args.length === 0) {
    result.addException('No argument supplied', 0)
    println JsonOutput.toJson(result)
    return
}

Script script

def str = args[0]

def stringWriter = new StringWriter()
def writer = new PrintWriter(stringWriter)

try {
    script = shell.parse(str, SCRIPT_NAME)
    script.setProperty("out", writer)
    script.run()
    result.setResult stringWriter.toString()
} catch(Exception ex) {

    String output = ex.getMessage()
    if(output == null || output.isEmpty()) {
        output = String.format("%s has been thrown without a message.", ex.getClass().getName())
    }

    Matcher matcher = COMPILATION_LINE_PATTERN.matcher(ex.getMessage())
    int line = -1
    if(matcher.find()) {
        line = Integer.valueOf(matcher.group( group: 1))
    }

    result.addException(output, line)
}

println JsonOutput.toJson(result)

```

Рисунок Б.1 код scriptRunner.groovy

ПРИЛОЖЕНИЕ В

```groovy
import java.util.jar.JarEntry
import java.util.jar.JarInputStream
import groovy.json.JsonOutput


class ClassDescription {
    String fullName
    String name
    String packageName
    ArrayList<String> methods = new ArrayList<>()
    ArrayList<String> fields = new ArrayList<>()
}

class Result {
    ArrayList<ClassDescription> result = new ArrayList<>()
    String exception
}

def result = new Result()

if (args.length < 1) {
    result.exception = 'No jar path supplied'
    println JsonOutput.toJson(result)
    return
}

def jarPath = args[0]

this.getClass().classLoader.rootLoader.addURL(new File(jarPath).toURL())

try {
    File f = new File(jarPath)
    FileInputStream fis = new FileInputStream(f)
    JarInputStream jis = new JarInputStream(fis)
    JarEntry next = jis.getNextJarEntry()

    while (next != null) {
        final String name = next.getName()

        if (name.endsWith('.class')) {

            def clsDescr = new ClassDescription()

            clsDescr.fullName = name.replaceAll( regex: '/', replacement: '.').substring(0, name.lastIndexOf( str: "."))
            clsDescr.name = clsDescr.fullName.substring(clsDescr.fullName.lastIndexOf( str: ".") + 1)
            clsDescr.packageName = clsDescr.fullName.substring(0, clsDescr.fullName.lastIndexOf( str: "."))

            def cls = Class.forName(clsDescr.fullName)
            methods = cls.getDeclaredMethods0( publicOnly: true)
            fields = cls.getDeclaredFields0( publicOnly: true)

            methods.each {
                clsDescr.methods.add(it.getName())
            }
            fields.each {
                clsDescr.fields.add(it.getName())
            }

            result.result.add(clsDescr)
        }

        next = jis.getNextJarEntry()
    }
    jis.close()
    fis.close()
} catch (Exception e) {
    result.exception = e.getMessage()
}

println JsonOutput.toJson(result)
```

Рисунок В.1 – код jarScanner.groovy