**Р Е Ф Е Р А Т**

ВЕБ-СЕРВИС ГЕНЕРАЦИИ СХЕМЫ АЛГОРИТМА ИЗ ИСХОДНОГО КОДА: дипломный проект / Бакыт М. – Минск: БГУИР, 2023, – п.з. – 87 с., чертежей (плакатов) – 7 л. формата А1.

Объектом проектирования является веб-сервис генерации схемы алгоритма из исходного кода.

Цель работы – разработка веб-сервиса генерации схемы алгоритма из исходного кода.

Разработка данного веб-сервиса обеспечит автоматизацию процесса построения схемы алгоритма из исходного кода.

Проведен анализ предметной области и аналогов, сделан обзор используемых технологий. Разработаны функциональные требования.

Рассмотрены программные средства и библиотеки для ведения разработки проекта и обоснован их выбор. Произведено проектирование веб-сервиса, построены диаграммы и схемы алгоритма.

Рассмотрено понятие тестирования, методы тестирования. Обоснован выбор используемого метода тестирования. Выполнено тестирование разработанного веб-сервиса.

Дипломный проект является завершенным, поставленная задача решена в полной мере. Планируется дальнейшее развитие веб-сервиса и расширение его функциональности. Проект выполнен самостоятельно, проведен анализ оригинальности в системе «Антиплагиат». Процент оригинальности составляет 98,24%. Цитирования обозначены ссылками на публикации, указанные в разделе «Список использованных источников».

**СОДЕРЖАНИЕ**

Введение 8

1 Анализ литературных источников, аналогов и постановка задачи к проектируемому веб-сервису 10

1.1 Анализ литературных источников 10

1.2 Анализ существующих аналогов 12

1.3 Постановка задачи к проектируемому веб-сервису 17

2 Разработка моделей и формирование требований к проектируемому веб-сервису 19

2.1 Формирование требований к проектируемому веб-сервису 19

2.2 Диаграмма вариантов использования 21

2.3 Схема работы веб-сервиса 24

2.4 Инфологическая модель базы данных 25

3 Проектирование веб-сервиса 27

3.1 Выбор архитектуры веб-сервиса 27

3.2 Проектирование языка 28

3.3 Анализ конструкций языка 29

3.4 Авторизация и аутентификация 36

3.5 Сохранение запросов в базу данных 36

3.6 Даталогическая модель базы данных 38

3.7 Проектирование интерфейса пользователя клиентской части 39

4 Создание веб-сервиса 44

4.1 Фреймворк Spring Security 44

4.2 Библиотека Liquibase 46

4.3 Создание веб-сайта 47

5 Тестирование веб-сервиса 48

6 Руководство по установке и использованию 52

6.1 Руководство по установке 52

6.2 Руководство по использованию 55

7 Экономическое обоснование разработки веб-сервиса генерации схемы алгоритма из исходного кода 62

7.1 Характеристика разработанного по индивидуальному заказу веб-

сервиса 62

7.2 Расчет затрат на разработку и цена веб-сервиса, созданного по индивидуальному заказу 62

7.3 Расчет результата от разработки веб-сервиса, созданного по индивидуальному заказу 65

7.4 Расчет показателей экономической эффективности разработки веб-сервиса, созданного по индивидуальному заказу 66

Заключение 67

Список использованных источников 68

Приложение А. Исходный код веб-сервиса 70

**ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ**

В настоящей пояснительной записке применяются следующие определения и сокращения.

*Блок-схема –* это графическое изображение последовательности операций, процессов или алгоритма, представленное в виде блоков, соединенных стрелками для демонстрации последовательности выполнения задач.

*Программа –* это последовательность инструкций, написанных на языке программирования, которые предназначены для выполнения определенной задачи на компьютере или другом устройстве.

*Программное средство –* это набор программных компонентов, предназначенных для решения определенных задач, автоматизации процессов и управления ресурсами компьютерной системы или другого устройства.

*Веб-сервис –* это программное обеспечение для обмена данными между приложениями и системами через Интернет, используя стандартизированные протоколы и форматы.

*Отладка –* это процесс идентификации и устранения ошибок в программном обеспечении, который может включать поиск и исправление проблем в исходном коде, тестирование и отслеживание ошибок во время выполнения программы, а также использование инструментов для анализа и устранения проблем.

*Лексер –* компонент программы, который анализирует исходный текст на языке программирования и разбивает его на лексемы, такие как идентификаторы, операторы, числа и строки, для дальнейшего использования в синтаксическом анализе.

*Парсер –* это компонент программы, который анализирует структуру входного текста на языке программирования, представленную в виде лексем, и строит древовидное представление этой структуры, используя правила грамматики языка, для проверки синтаксиса и семантики программы и дальнейшей обработки.

*Авторизация –* это процесс проверки подлинности пользователя и предоставления ему доступа к определенным ресурсам или функциям системы, которые защищены паролем, ключом доступа или другими методами идентификации.

*Аутентификация* – это процесс проверки подлинности идентификационных данных пользователя, таких как имя пользователя и пароль, ключ доступа или сертификат, для подтверждения того, что пользователь является тем, за кого себя выдает.

*Фреймворк* – это набор программных компонентов, инструментов и библиотек, которые предоставляют общую структуру и функциональность для разработки приложений или систем.

*Исключение* – это объект, который представляет ошибку, возникшую в процессе выполнения программы.

*Оператор* – это символ или ключевое слово в языке программирования, которое представляет собой определенную операцию, выполняемую над одним или несколькими операндами.

*Операнд* – это значение или объект, который используется в операции с одним или несколькими операторами в языке программирования.

*Стек программы* – это структура данных, которая используется в процессе выполнения программы для хранения временных данных и адресов возврата.

*Версионирование* – это процесс управления изменениями в программном обеспечении, включая контроль версий и управление релизами.

БД – база данных.

СУБД – система управления базами данных.

ЭВМ – электронная вычислительная машина.

ANTLR – another tool for language recognition.

UML – unified modeling language.

JWT – json web token.

JSON – javascript object notation.

AST – abstract syntax tree.

CORS – cross-origin resource sharing.

SPA – single-page application.

Введение

Современное программное обеспечение становится все более сложным, и понимание его работы может представлять трудность даже для опытных разработчиков. В связи с этим возникает потребность в инструментах, которые позволяют облегчить анализ и понимание кода. Один из таких инструментов – это визуальный подход к обучению программированию, который позволяет представлять сложные алгоритмы в графическом виде [1].

Однако, визуальный подход к программированию может стать еще более эффективным, если используется автоматическая генерация блок-схем на основе исходного кода. Такой подход может значительно упростить понимание и анализ кода, особенно в случае работы с большими проектами.

Для того чтобы разработать автоматический генератор блок-схем, необходимо решить несколько задач, связанных с лексическим и синтаксическим анализом кода. Важно также разработать алгоритм генерации блок-схем, который должен быть эффективным и точным, чтобы генерировать блок-схемы с минимальным числом ошибок.

Актуальность проблемы генерации блок-схем из исходного кода заключается в том, что это позволяет облегчить понимание кода, особенно в случае работы с большими проектами [2]. Также это может быть полезным для отладки и тестирования программного обеспечения. В современных условиях, когда количество программных продуктов растет с каждым днем, разработка инструментов, которые позволяют упростить анализ кода, становится все более актуальной.

Цель данного дипломного проекта заключается в разработке веб-сервиса, который будет автоматически генерировать блок-схемы из исходного кода программ. Генерация блок-схем из кода позволит облегчить понимание и анализ сложных алгоритмов, особенно в случае работы с большими проектами. Разработка веб-сервиса будет иметь большую практическую значимость, поскольку он может быть полезен для студентов, которые учатся программированию, и для опытных разработчиков, которые работают с большими программными проектами. Помимо этого, создание веб-сервиса может быть полезным для отладки и тестирования программного обеспечения. Общая идея проекта заключается в использовании визуального подхода к программированию, чтобы представлять сложные алгоритмы в графическом виде и тем самым облегчить их понимание и анализ.

Разработка данного дипломного проекта позволит получить более глубокие знания в области теории компиляторов и разработки клиент-серверных приложений. В процессе выполнения проекта будет решаться ряд задач, связанных с разработкой синтаксиса собственного языка программирования, алгоритма обхода синтаксического дерева, клиентской части веб-сервиса и тестирования полученного решения.

Разработка синтаксиса собственного языка программирования является одной из ключевых задач проекта, так как язык должен быть достаточно выразительным для описания различных алгоритмов и при этом легко понятным для пользователей. Для решения этой задачи необходимо изучить теорию компиляторов и научиться применять ее на практике.

Алгоритм обхода синтаксического дерева и построения блок-схемы также является важным компонентом проекта. Для его реализации необходимо будет изучить различные алгоритмы обхода деревьев и выбрать оптимальный вариант для данной задачи.

Помимо этого, клиентская и серверная части веб-сервиса должны соблюдать общепринятые стандарты по скорости работы и безопасности, правильная разработка которых позволит изучить принципы проектирования подобных приложений, их взаимодействия и оптимизации.

Таким образом, выполнение данного дипломного проекта позволит не только создать полезный инструмент для программистов и студентов, но и улучшить практические навыки в области теории компиляторов и разработки клиент-серверных приложений.

1 Анализ литературных источников, аналогов и постановка задачи к проектируемому веб-сервису

1.1 Анализ литературных источников

**1.1.1** Анализ алгоритмов разбора языка

Компиляция языка программирования – это процесс преобразования исходного кода на одном языке в эквивалентный код на другом языке или в машинный код. Она является важным шагом в создании программного обеспечения и позволяет перевести код, написанный на человеко-читаемом языке, в язык, который может быть исполнен компьютером. В контексте генерации схемы алгоритма, компиляция может быть рассмотрена как процесс преобразования исходного кода в графическое представление алгоритма в виде блок-схемы.

Процесс компиляции включает в себя несколько этапов: лексический анализ, синтаксический анализ, семантический анализ, оптимизацию и генерацию кода. Во время лексического анализа исходный код разбивается на токены, которые представляют отдельные элементы языка, такие как идентификаторы, числа и ключевые слова. Синтаксический анализ проверяет правильность расположения токенов и их соответствие грамматике языка. Семантический анализ проверяет, что код соответствует правилам языка и выполняет необходимые проверки на типы данных, области видимости переменных и другие аспекты, которые могут повлиять на правильность работы программы [3].

Существует два основных подхода к созданию синтаксических анализаторов для языков программирования: ручной и автоматизированный. В первом случае, разработчик самостоятельно пишет код парсера, который выполняет лексический и синтаксический анализ входного кода. Этот подход позволяет получить более точный контроль над процессом анализа и предоставляет возможность для оптимизации производительности. Однако, ручное написание парсера может быть очень трудоемким и затратным, особенно для языков с богатым и сложным синтаксисом.

Альтернативным подходом является автоматизированное создание парсера с помощью генераторов парсеров. Этот метод основан на грамматиках, описывающих синтаксис языка. Разработчик определяет грамматику языка в специальном формате, который может быть использован генератором парсера для автоматического создания соответствующего кода парсера [4].

На практике, генераторы парсеров часто используются для автоматического создания парсеров на основе грамматики языка, что значительно упрощает и ускоряет процесс разработки. Генераторы парсеров позволяют автоматически создавать парсеры для большинства языков программирования, обеспечивая высокую точность и скорость работы [5]. Использование генераторов парсеров позволяет разработчикам сосредоточиться на более важных задачах, таких как разработка лексического анализа и семантического анализа.

Для семантического анализа не существует общих подходов, так как его цель сильно зависит от конкретного языка программирования и контекста, в котором он используется [6]. В разрабатываемом веб-сервисе для генерации блок-схемы из исходного кода нет необходимости проводить полный семантический анализ, так как основной целью является только построение структуры алгоритма в виде графа. Для этого достаточно провести лексический и синтаксический анализ исходного кода, а также выполнить минимальный семантический анализ для определения типа блоков, таких как определение переменных, вызов функций и так далее.

**1.1.2** Анализ языков программирования

Одним из основных факторов при выборе языка программирования является его удобство в использовании. Языки, которые имеют простой и понятный синтаксис, а также богатые возможностями и инструментами, обычно считаются более привлекательными для программистов. Например, язык Python [7], благодаря своей читабельности и простоте в написании кода, популярен среди начинающих и опытных программистов.

Кроме удобства использования, важным аспектом является поддержка языка сообществом разработчиков и его популярность. Языки программирования с большой и активной пользовательской базой обычно имеют обширную документацию, богатую выборку инструментов и разнообразных библиотек. Например, язык Java, благодаря своей популярности [8], имеет обширную экосистему, которая включает в себя множество инструментов и библиотек.

Важным критерием при выборе языка программирования для создания веб-сервиса, позволяющего генерировать схему-алгоритма из исходного кода, является его легкость синтаксического и семантического разбора. Для достижения этой цели был проведен анализ существующих языков программирования, популярных в различных областях программирования. Было выяснено, что существуют языки с удобным и легким синтаксисом, который позволяет легко производить семантический и синтаксический анализ кода, но с другой стороны есть ряд правил, которых стоит избегать. Например, некоторые языки программирования используют многословные ключевые слова или сложные правила синтаксиса, как неоднозначности в определении границ выражений или использование разных символов для обозначения одного и того же действия, что затрудняет понимание и анализ кода. Кроме того, использование сложных конструкций, таких как макросы может усложнить семантический анализ, за счет изменения структуры кода на основе параметров [9].

С учетом этих факторов, при разработке нового языка для создания блок-схем, важно уделить внимание не только удобству синтаксиса, но и точности описания логических конструкций. Необходимо убедиться, что язык позволяет программисту легко выразить свои мысли и идеи в виде блок-схем, без необходимости тратить много времени на поиск нужных инструкций и ключевых слов, но при этом позволяет проводить генерацию схемы-алгоритма с наименьшим количеством проблем.

1.2 Анализ существующих аналогов

Произведение анализа существующих аналогов является важным шагом в разработке нового веб-сервиса генерации схемы алгоритма из исходного кода. Это поможет выявить особенности и проблемы имеющихся решений, а также определить возможности для улучшения и дальнейшего развития разрабатываемого веб-сервиса.

Существующие аналоги могут предоставлять различные функциональные возможности, иметь различные методы решения поставленных задач, и в то же время иметь некоторые ограничения и недостатки. Изучение конкурентов позволит определить лучшие практики и опыт, а также учесть ошибки и недочеты в разработке собственного веб-сервиса.

**1.2.1** Веб-сервис «Code2Flow»

Code2Flow [10] – это веб-сервис, который предоставляет возможность автоматически генерировать блок-схемы из исходного кода на собственном языке программирования. Данный язык представляет собой смесь естественного и машинного, в котором имеется возможность использовать привычные условные и циклические операторы, но при этом записывать выражения в виде обычных предложений. Пользователь может загрузить свой код и получить блок-схему, представляющую собой графическое представление алгоритма, что позволяет быстрее понять структуру и логику программы. Данный веб-сервис позиционирует свою основную функциональность, как инструмент для визуального отображения идей. На рисунке 1.1 представлен пример кода, который описывается в веб-сервисе Code2Flow.

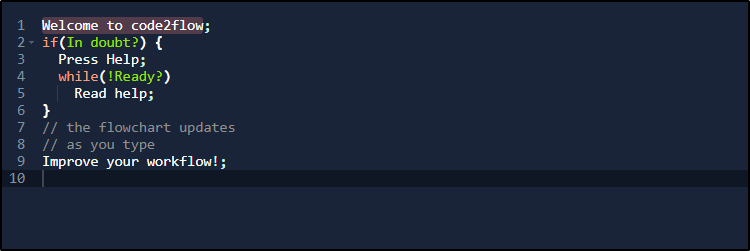


Рисунок 1.1 – Редактор кода в веб-сервисе «Code2Flow»

Как видно, в данном примере отсутствуют описание функций, а вместо вызова процедур или подпрограмм записаны предложения на английском языке. Также присутствуют условный и циклический операторы, которые выделены розовым цветом, а логическое условие в них – зеленым.

На рисунке 1.2 отображена схема алгоритма, которая была сгенерирована на основе данных, полученных из кода на рисунке 1.1.

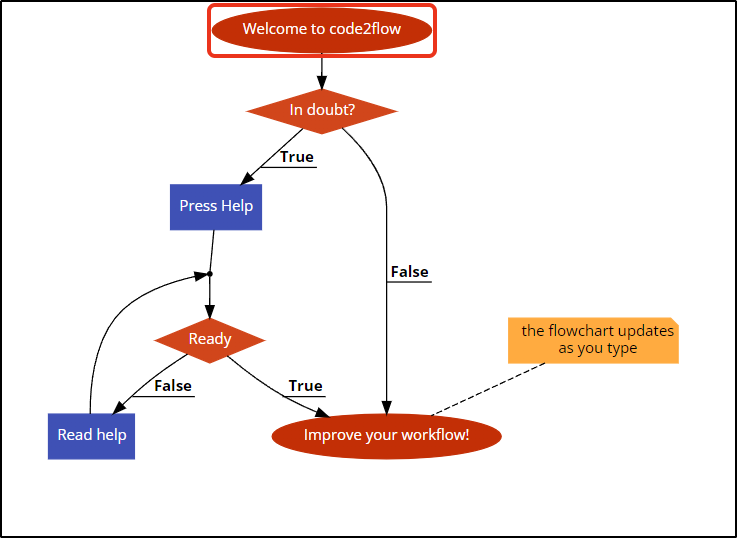


Рисунок 1.2 – Сгенерированная схема алгоритма в веб-сервисе Code2Flow

В данном примере необходимо обратить внимание на то, что первое и последнее предложение было отображено как начало и конец схемы, а комментарии отдельно выделены штриховой линией. Также видно, что логическое условие «Ready» было инвертировано, хотя в логике прописан оператор «!». На рисунке 1.3 показано, как выглядит данное условие без оператора инвертирования.

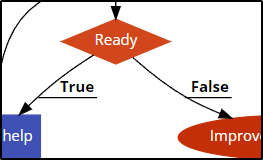


Рисунок 1.3 – Оператор инвертирования

Как видно из рисунка 1.3 сам оператор не был изменен, были изменены только направления стрелок «True» и «False», что может запутать пользователя.

Функционал данного веб-сервиса представляет собой интересную композицию генерации схемы алгоритма и визуального отображения идей. Благодаря такому синтаксису любой слушатель, абсолютно не понимающий принципов программирования, сможет сгенерировать блок-схему, которая будет понятна разработчикам. Помимо этого, можно также выделить следующие плюсы:

– удобный и понятный дизайн интерфейса редактора кода;

– генерация блок-схемы в реальном времени;

– отображение комментариев.

Необходимо учесть, что некоторые особенности сервиса Code2Flow, могут быть недостатками для разрабатываемого веб-сервиса, поскольку контексты их использования различны. Например, отсутствие блоков начала и конца цикла может быть удобнее для визуализации идей, но не подходит для визуализации конкретного алгоритма. Также можно отметить:

– отсутствие возможности редактировать схему алгоритма;

– изменение семантики написанного кода, как в примере с условием «Ready»;

– отсутствие возможности уточнения начала и конца схемы.

**1.2.2** Веб-сервис «draw.io»

Сервис «draw.io» [11] предоставляет возможность создавать и редактировать различные типы диаграмм, включая блок-схемы, диаграммы классов, диаграммы потоков данных и многие другие. Интерфейс приложения интуитивно понятен, что делает его простым в использовании. «Draw.io» позволяет сохранять диаграммы в различных форматах, таких как XML, PNG, JPEG, PDF, SVG, HTML, VSDX и многих других.

Одним из главных преимуществ сервиса draw.io является то, что он бесплатен и не требует установки на компьютер. Также в draw.io доступно большое количество встроенных форм, символов и элементов диаграмм, которые значительно упрощают и ускоряют процесс создания диаграммы. Однако, некоторые элементы и функции доступны только в платной версии приложения.

Однако, несмотря на все преимущества, «draw.io» также имеет некоторые недостатки. В частности, в нем отсутствует автоматическая генерация блок-схем из исходного кода, что делает процесс создания схем более трудоемким и времязатратным. Также, ограниченные возможности кастомизации и оформления могут ограничивать пользовательский опыт.

На рисунках 1.4 и 1.5 представлены примеры отдельных участков пользовательского интерфейса данного сервиса.

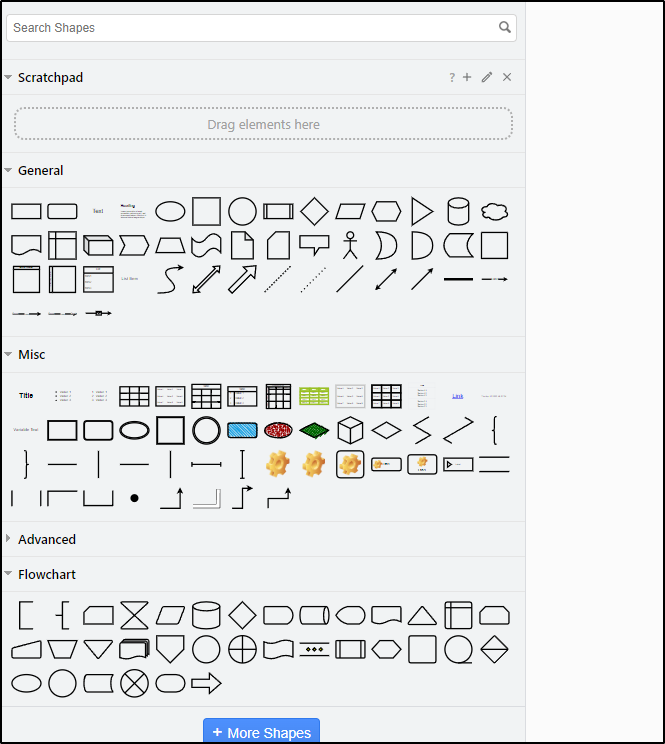


Рисунок 1.4 – Выбор форм в веб-сервисе «Draw.io»

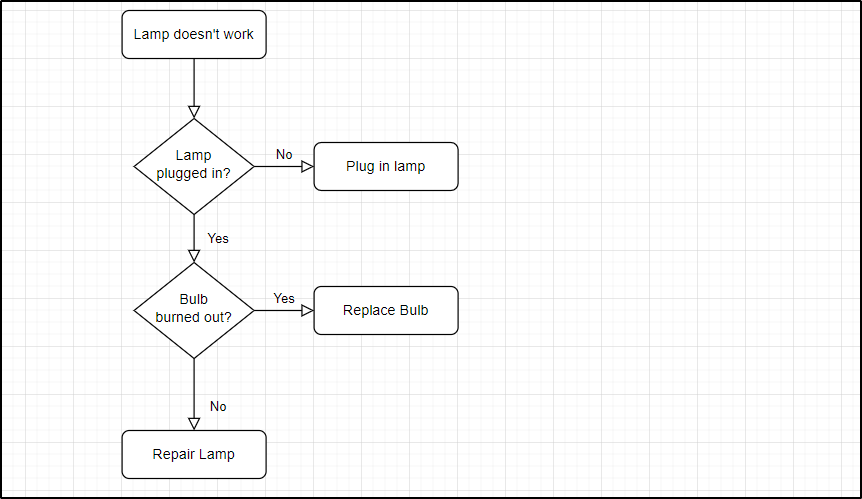


Рисунок 1.5 – Пример схемы алгоритма в сервисе «Draw.io»

Несмотря на ограниченные возможности настроек, как видно из рисунков 1.4 и 1.5, данный веб-сервис включает в себя большое количество встроенных форм, а также удобный редактор схем. Это позволяет пользователю создавать блок-схему любой сложности и размещать элементы таким образом, чтобы сделать ее максимально легкой в понимании. Также отдельном плюсом необходимо выделить возможность редактировать линии, соединяющие блоки, что не поддерживается большинством других аналогов. Однако отсутствие возможности генерировать схему алгоритма из исходного кода делает разработку в данном веб-сервисе долгим и однообразным процессом, что может привести к ошибкам, из-за человеческого фактора.

**1.2.3** Программное средство «Code Rocket»

Изначально, «Code Rocket» был разработан компанией «Rapid Quality Systems» в 2002 году для генерации диаграмм визуального моделирования исходного кода на языках C, C++ и Java. Программа имела возможности создавать диаграммы классов, диаграммы последовательностей, диаграммы состояний, диаграммы прецедентов и другие. «Code Rocket» поддерживал различные платформы, включая Windows, Linux и Mac OS X. Помимо этого «Code Rocket» был впоследствии перенесен на популярные редакторы кода, такие как «Visual Studio .NET» и «Eclipse» [12].

«Code Rocket» использовался для улучшения визуального моделирования процесса разработки программного обеспечения. В качестве основной функциональности программы были предложены автоматическая генерация диаграмм, расширенный поиск, просмотр проектов и многое другое. «Code Rocket» также предоставлял возможность экспортировать диаграммы в форматы, такие как PNG, JPEG, BMP и другие.

Несмотря на то, что «Code Rocket» был достаточно популярен в свое время, сейчас он считается устаревшим программным обеспечением, которое не поддерживается разработчиками и не обновляется уже более десяти лет. Кроме того, некоторые пользователи отмечают, что установка и настройка программы достаточно сложны, и использование «Code Rocket» требует определенной подготовки и знаний.

Тем не менее, программное обеспечение «Code Rocket» все еще может быть полезно для разработчиков, которые работают с языками C, C++ и Java, и которые заинтересованы в генерации диаграмм визуального моделирования исходного кода. Если у разработчика уже есть опыт работы с этой программой, то она может оказаться удобной для использования и сэкономить время на создание визуальной модели.

На рисунке 1.6 показан пример построения схемы алгоритма из исходного кода, написанного на языке Visual Basic.

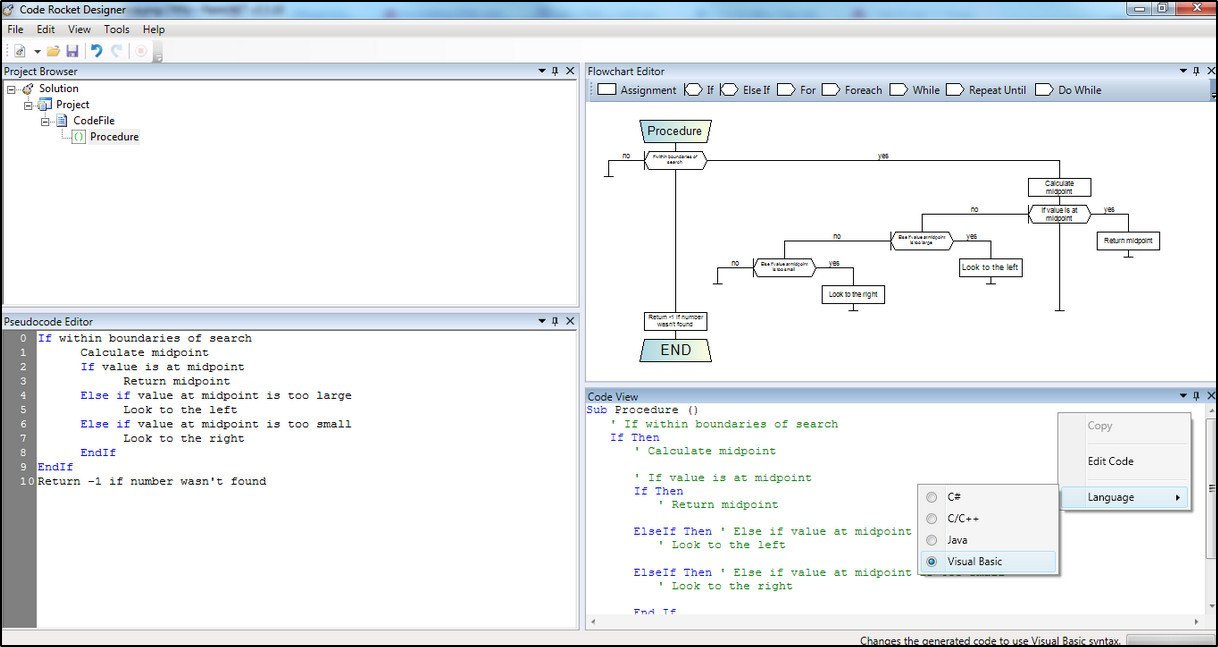


Рисунок 1.6 – Интерфейс пользователя программного средства «Code Rocket»

К достоинствам данного программного обеспечения можно отнести:

– Поддержка различных языков программирования, такие как C, C++, C#, Java, Python, Ruby и другие.

– Широкий спектр возможностей для создания диаграмм, включая блок-схемы, диаграммы классов и диаграммы последовательностей.

– Простой и интуитивно понятный интерфейс.

Однако программное средство также имеет и ряд серьезных недостатков, основными можно выделить:

– Необходимость установки на компьютер.

– Инструмент не поддерживается и не обновляется с 2013 года.

– Платный сервис.

– Необходимость знания конкретных языков программирования.

1.3 Постановка задачи к проектируемому веб-сервису

Целью данной работы является разработка веб-сервиса для автоматической генерации блок-схемы алгоритма на основе кода, написанного на специально разработанном для этой цели языке программирования.

Для реализации проекта необходимо разработать веб-сервис, который будет предоставлять возможность редактировать исходный код на разрабатываемом языке и отправлять его на обработку. Данный язык должен быть максимально удобен в использовании и легок в чтении, основанный на синтаксисе языка Си [13]. Он должен поддерживать основные типы данных, операторы и условные конструкции, необходимые для написания алгоритмов.Также необходимо учесть особенности, которые позволят сделать семантический анализ более легким, например, использование обязательных фигурных скобок.

Для обеспечения безопасности и эффективной работы веб-сервиса было принято решение использовать клиент-серверную архитектуру, где клиент будет отправлять запросы на сервер, а сервер будет обрабатывать эти запросы и возвращать результаты.

Для обеспечения корректности работы сервиса необходимо разработать анализатор кода, который будет проводить синтаксический и семантический анализ введенного кода на языке программирования. Он должен определять ошибки в коде и выводить соответствующие сообщения об ошибке пользователю.

Для достижения поставленной цели необходимо разработать:

– Язык программирования для написания алгоритмов и описания структур данных.

– Генератор блок-схем на основе введенного кода.

– Клиент-серверную архитектуру приложения.

– Анализатор кода для проверки корректности введенного кода.

– Интерфейс пользователя для взаимодействия с приложением.

Реализация поставленных задач позволит создать функциональный и удобный в использовании веб-сервис для генерации блок-схем алгоритмов на основе введенного кода, что будет полезно как для обучения программированию, так и для решения практических задач в профессиональной деятельности.

2 Разработка моделей и формирование требований к проектируемому веб-сервису

2.1 Формирование требований к проектируемому веб-сервису

Веб-сервис генерации схемы алгоритма из исходного кода, основанный на клиент-серверной архитектуре, должен обеспечивать следующий список общих для всех веб-сервисов функций:

– Авторизация и аутентификация пользователей.

– Обработка запросов и передача данных между клиентом и сервером.

– Управление базой данных.

– Определение и предоставление прав доступа для разных типов пользователей.

– Обработка исключительных ситуаций.

Помимо общего функционала, применяемого к любому виду подобных программных средств, веб-сервис генерации схемы алгоритма из исходного кода должен поддерживать следующий список требований:

– Генерация блок-схемы из исходного кода на разработанном языке.

– Сохранение сгенерированных схем в различных форматах изображений.

– Сохранение кода и сгенерированной схемы в базу данных.

– Просмотр списка сгенерированных блок-схем.

Входные данные для веб-сервиса должны быть представлены пользователем в виде исходного кода, в качестве его действий и ввода текстовой информации в редакторе кода клиентской части веб-сервиса.

Выходные данные должны быть представлены посредством обработки запроса от пользователя сервером, в виде компактного текстового ответа, полностью включающего в себя информацию о блоках данной схемы алгоритма, о всех соединениях, а также дополнительной текстовой информации для каждого элемента схемы. В связи с этим можно выделить дополнительные требования для пользовательского интерфейса данного веб-сервиса:

– Пользователь должен иметь возможность редактировать код напрямую в браузере.

– Синтаксис кода должен иметь подсветку, облегчающую читаемость кода и позволяющую быстрее определить ошибки.

– В случае возникновения ошибок клиентская часть должна отобразить номер строки и, по возможности, номер символа, в котором была произведена ошибка.

– Веб-сервис должен представлять схему алгоритма в готовом для чтения виде, без необходимости пользователю ее редактировать.

– В случае необходимости, пользователь должен иметь возможность перемещать блоки в схеме, при этом соответствующие соединения должны автоматически обновляться.

– Для повышения легкости воспроизведения участков кода, пользователь должен иметь возможность сгенерировать шаблон для всех операторов языка в коде.

– Интерфейс пользователя должен быть простым и интуитивно понятным.

Помимо формирования требований к проектируемому веб-сервису, также необходимо сформировать требования для разрабатываемого языка, которые позволят эффективно его использовать, для описания алгоритмов и быстро его обрабатывать, для генерации схемы алгоритма. Исходя из анализа литературы были выявлены следующие требования к новому разрабатываемому языку:

– Синтаксис языка должен быть описан средствами, поддерживаемыми генератором парсеров.

– Язык должен иметь СИ-подобный синтаксис, то есть должен быть основан на синтаксисе языка программирования СИ.

– Синтаксис языка должен быть легок в написании и понимании, что должно быть достигнуто посредством выполнения следующих требований:

а) определение прототипа функции ключевым словом «func»;

б) обязательное использование разделяющих знаков;

в) возможность использования пользовательских типов данных.

– Синтаксис языка не должен включать в себя правила, усложняющие последующий семантический анализ и генерацию схемы алгоритма, например:

а) наличие безусловных операторов перехода;

б) наличие макросов;

в) возможность объявления вложенных функций и т.д.

С точки зрения анализатора кода, помимо использования библиотеки для генерации парсеров, также должны быть написаны алгоритмы, позволяющие закончить синтаксический анализ, подготовить дерево разбора и произвести неполный семантический анализ.

Для простоты сопровождения кода и добавления дополнительной функциональности данные алгоритмы должны поддерживать любой возможный уровень вложенности кода посредством использования шаблона проектирования «Наблюдатель» и иметь возможность быть настроенным благодаря использованию конфигурационных классов и/или объектов.

Для того чтобы обеспечить правильную генерацию схемы алгоритма, семантический анализ должен определять:

– Вызов предопределенных подпрограмм, то есть определение тех функций, что были написаны и вызваны пользовательским кодом.

– Порядок вызовов операторов, для чего необходимо обрабатывать операторы break, continue и return.

– Тело оператора switch и его условий.

Для корректной работы серверной части веб-сервиса, ЭВМ, на которой должен функционировать веб-сервис, должна обладать следующими минимальными характеристиками:

– процессор Intel Core i5 или AMD Ryzen 4, с тактовой частотой не менее 3 ГГц;

– жесткий диск объемом 250 ГБ;

– оперативная память объемом 16 ГБ;

– сетевая карта Ethernet или Wi-Fi, с минимальной пропускной способностью 50 Мбит/с.

Также на этой ЭВМ должно быть установлено Java Runtime Environment версии 17.0.3 или выше и система управления базами данных PostgreSQL версии 14.7 или выше.

Клиентская часть веб-сервиса должна корректно функционировать на ЭВМ со следующими минимальными характеристиками:

– процессор Intel Core i3 или AMD Ryzen 3, с тактовой частотой не менее 1.5 ГГц;

– оперативная память объемом 4 ГБ;

– Функция выхода в сеть Интернет.

Для корректной работы клиентской части веб-сервиса также необходим один из следующих браузеров:

– Google Chrome 70;

– Opera 58;

– Mozilla Firefox 66;

– Apple Safari 12.0;

– Microsoft Edge 44.

2.2 Диаграмма вариантов использования

С целью описания функций веб-сервиса была выбрана диаграмма вариантов использования в спецификации языка графического описания UML.

UML – это унифицированный язык моделирования, система обозначений, которую можно применять для объектно-ориентированного анализа и проектирования. Его можно использовать для визуализации, спецификации, конструирования и документирования программных систем [14].

Диаграмма вариантов использования – это графическое представление функциональности системы, которая отображает актеров (пользователей) и взаимодействие между ними. В общем случае, диаграмма вариантов использования используется для описания требований к системе, и является одним из ключевых инструментов в области инженерии требований.

Исходя из выдвинутых требований, проектируемый веб-сервис предполагает поддержку системы ролей, т.е. разделение пользователей на различные роли, а, следовательно, и выдача различных прав нескольким категориям пользователей. В случае данного веб-сервиса разделение прав не вызывает никаких функциональных изменений для пользователей, однако это впоследствии может использоваться для добавления персонала и/или разграничения платного и бесплатного функционала.

Как видно на рисунке 2.1 разрабатываемый веб-сервис имеет два ключевых актера: Пользователь и Гость. В свою очередь Гость является Пользователем, который еще не был авторизован в программном средстве. Так как его запросы не могут быть аутентифицированы, его функциональность ограничена.

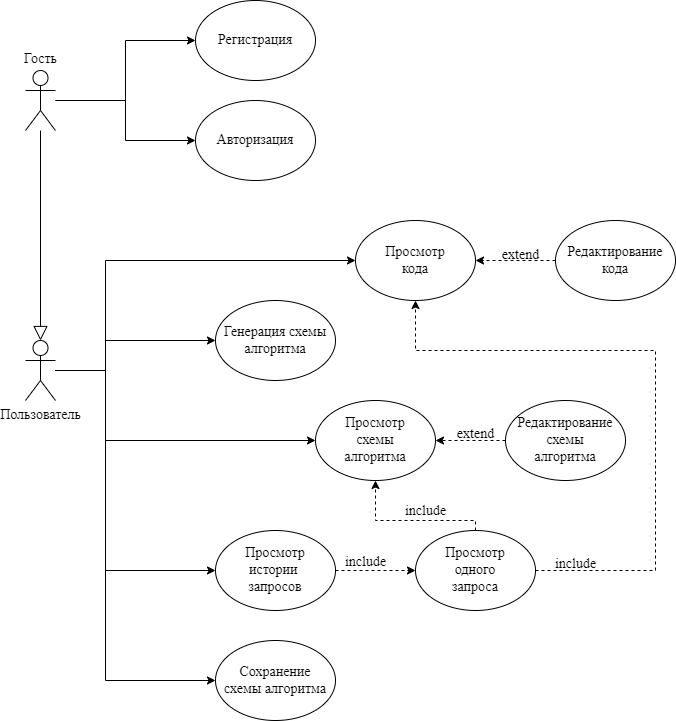


Рисунок 2.1 – Диаграмма использования

В случае данного веб-сервиса Гость имеет следующие функции:

– Регистрация. Под данной функцией подразумевается процесс ввода авторизационной информацией, которая еще не была сохранена в веб-сервисе. После регистрации пользователь получает доступ к генерации схемы алгоритма.

– Авторизация. Необходима для подтверждения подлинности пользователя. Процесс авторизации подразумевает, что пользователь уже проходил регистрацию.

После авторизации или регистрации Гость получает доступ к полной функциональности веб-сервиса. Являясь ключевым актером в разрабатываемом программном средстве, Пользователь имеет следующие функции:

– Просмотр кода. Данная функция подразумевает отображение кода с подсветкой ключевых слов языка. Отображение кода является неотъемлемой частью веб-сервиса, так как позволяет пользователю найти допущенные в коде ошибки, при этом не выходя из браузера. Данная функция также включает в себя возможность редактировать код.

– Редактирование кода. Дает пользователю возможность редактировать код в самом браузере, не используя для этой цели сторонние программы. Включает в себя возможность использования предопределенных операторов языка.

– Просмотр схемы алгоритма. Данная функция подразумевает отображение блок-схемы в отдельном компоненте клиентской части веб-сервиса. Она включает в себя возможность отдалять и приближать схему, а также автоматически отображает новую схему в готовом для просмотра виде.

– Редактирование схемы алгоритма. Дает пользователю возможность перемещать отдельные блоки схемы по своему усмотрению. Включает в себя автоматические определение расположения стрелок в схеме.

– Генерация схемы алгоритма. Является ключевой функциональностью проектируемого веб-сервиса. Позволяет пользователю сгенерировать схему из написанного им кода и отображает его в компоненте для просмотра.

– Сохранение схемы алгоритма. Данная функция подразумевает сохранение схемы в формате изображения. Позволяет пользователю выбрать расположение файла с изображением и сохранить схему, в одном из форматов хранения изображений.

– Просмотр истории запросов. Данная функция позволяет пользователю просмотреть все ранее сгенерированные им схемы алгоритмов, а также исходный код, из которой они были сгенерированы. Подразумевает под собой возможность просмотра времени сделанного запроса, а также сортировки по этому времени.

– Просмотр одного запроса. Позволяет пользователю вернуться в редактор кода и схемы алгоритма из истории, тем самым давая ему возможность заново просмотреть и отредактировать ранее сгенерированную блок-схему, а также исходный код, из которого она была сгенерирована.

2.3 Схема работы веб-сервиса

Схема работы веб-сервиса является важным инструментом проектирования и описания процессов, связанных с его функционированием. Она позволяет представить работу веб-сервиса в виде последовательности шагов и действий, которые выполняются для обработки запросов и предоставления пользователю необходимой информации.

На основании составленных требований и вариантов использования были определены ключевые шаги по обработке запроса от пользователя для генерации схемы-алгоритма. На рисунке 2.2 представлена общая схема этого процесса.

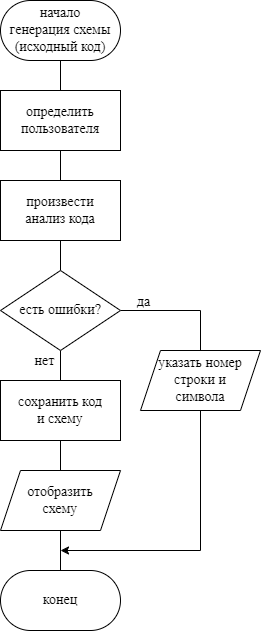


Рисунок 2.2 – Общая схема обработки запроса пользователя

Как видно на рисунке 2.2, процесс генерации схемы алгоритма из исходного кода можно разбить на 5 шагов. На первом шаге, после принятия запроса от пользователя производится аутентификация – определение личности пользователя. Данный шаг необходим для последующего сохранения запроса.

Следующий шаг анализа исходного кода проводится в два этапа: сам анализ и определение ошибок. В случае возникновения исключительных ситуаций, по вине пользователя, разрабатываемый веб-сервис должен оповестить его о номере строки и символа, в котором, возможно, была допущена ошибка. В случае отсутствия ошибок данный запрос, вместе с исходным кодом и результатом необходимо сохранить. Далее веб-сервис должен отобразить блок-схему, для последующего просмотра и редактирования ее пользователем.

Таким образом, соответствуя разработанной схеме обработки запроса выполняются сразу несколько требований: происходит сама генерация блок-схемы, ее результат сохраняется для просмотра в истории и, в случае ошибок, пользователю об этом сообщается. При этом было принято решение о пропуске сохранения ошибочного результата, так как оно не является важным или необходимым для пользователя.

2.4 Инфологическая модель базы данных

Для хранения информации о запросах и самих пользователях необходима база данных. Несмотря на то, что в данном проекте для веб-сервиса не нужна сложная и большая база, разработка ее модели может предотвратить дальнейшие ошибки и облегчить процесс создания и использования ее в СУБД.

Инфологическая модель базы данных – это модель, которая отражает все данные, которые хранятся в базе данных, и связи между ними, но не содержит информацию о том, как эти данные будут физически храниться на жестком диске. Она описывает структуру данных в базе, а также отношения между этими данными.

Инфологическая модель базы данных не зависит от технологий и используемых инструментов, она описывает бизнес-сущности и их взаимосвязи. Она не определяет способ хранения данных, а только структуру и связи между ними. Она служит основой для дальнейшего проектирования и создания физической модели базы данных.

Инфологическая модель базы данных необходима для того чтобы правильно спроектировать структуру базы данных, учитывая бизнес-логику приложения и требования к хранению данных. Она позволяет определить, какие таблицы и поля будут использоваться в базе данных, какие связи между ними будут существовать, какие данные будут храниться в каждой таблице, и как эти данные будут связаны между собой. Она также позволяет увидеть возможные проблемы и несоответствия в структуре базы данных до ее создания, что позволяет избежать проблем в будущем.

Существуют несколько видов записи инфологической модели базы данных. Один из таких видов – это диаграмма сущность-связь. Она позволяет отобразить объекты базы данных и связи между ними. Еще один вид записи – это текстовое описание сущностей и связей, которое включает в себя описание атрибутов каждой сущности и ее свойств. Кроме того, существует графический способ – в UML-диаграммах используются блоки, линии и стрелки для представления сущностей, их атрибутов и связей между ними.

Текстовый вид записи инфологической модели является наиболее подходящим для простых баз данных по нескольким причинам. Во-первых, такой вид записи обладает высокой читаемостью и понятностью, что облегчает понимание структуры базы данных и облегчает совместную работу между разработчиками и аналитиками. Во-вторых, текстовый формат обеспечивает удобный способ хранения и редактирования инфологических моделей, что позволяет быстро вносить изменения в модель, если это необходимо. По этим причинам инфологическая модель включает в себя следующие объекты и свойства:

– Пользователь:

а) идентификатор;

б) имя;

в) хэш пароля;

г) список запросов;

д) роль.

– Роль:

а) идентификатор;

б) имя.

– Запрос:

а) идентификатор;

б) время обработки запроса;

в) исходный код;

г) сгенерированная схема алгоритма.

В данной модели необходимо обратить внимание на атрибут пользователя – хэш пароля. В целях безопасности хранение пароля в неизмененном виде не рекомендуется, так как может повлечь за собой утечку данных. Также для каждого объекта был присвоен уникальный идентификатор для последующих связей сущностей в базе данных.

Такая модель базы данных позволит хранить всю необходимую для веб-сервиса информацию в простом и унифицированном виде, что впоследствии облегчит работу с авторизацией и аутентификацией пользователей, а также отображение истории запросов.

3 Проектирование веб-сервиса

3.1 Выбор архитектуры веб-сервиса

Веб-сервис – это программное обеспечение, которое позволяет различным приложениям взаимодействовать друг с другом через сеть, используя стандартные протоколы, например HTTP [15]. Веб-сервис может быть реализован на разных языках программирования и может работать на разных операционных системах.

Клиент-серверная архитектура – это модель, которая разделяет приложение на две части: клиентскую и серверную. Клиентская часть представляет интерфейс пользователя, который взаимодействует с приложением, а серверная часть выполняет операции, обрабатывает запросы клиента и возвращает результаты [16].

Клиент-серверная архитектура является одной из наиболее популярных архитектур для создания веб-приложений и сервисов. Это связано с тем, что она предоставляет отличный уровень безопасности и масштабируемости. Каждая часть приложения выполняется в отдельном окружении, что предотвращает возможность несанкционированного доступа к чувствительной информации. Кроме того, клиент-серверная архитектура обеспечивает высокую производительность и гибкость в работе с данными.

Существуют различные языки программирования и фреймворки для реализации клиент-серверной архитектуры, некоторые из наиболее популярных языков включают JavaScript, Python, Java, Ruby, PHP и C#. Они используются в сочетании с различными фреймворками, такими как Node.js, Flask, Spring, Ruby on Rails, Laravel, ASP.NET и многими другими.

Spring Kotlin [17] предоставляет разработчикам множество возможностей для создания быстрых и эффективных веб-сервисов. Он основан на Java и позволяет использовать всю мощь Java-фреймворка Spring, который является одним из наиболее популярных в мире, а также генератор парсеров ANTLR. Кроме того, Kotlin предоставляет более простой и понятный синтаксис, что упрощает разработку и поддержку приложения.

Angular [18] обеспечивает высокую производительность, быструю загрузку страниц и хорошую масштабируемость, а также позволяет создавать модульные и переиспользуемые компоненты, что существенно упрощает разработку.

Таким образом, использование Spring Kotlin и Angular для разработки веб-сервиса генерации схемы алгоритма из исходного кода является оптимальным решением. Эти фреймворки обеспечивают высокую скорость разработки, хорошую масштабируемость и простоту в поддержке и сопровождении.

3.2 Проектирование языка

Исходя из выше определенных требований, первичный анализ исходного кода будет происходить посредством библиотеки ANTLR [19].

ANTLR – это генератор парсеров, который позволяет создавать лексический и синтаксический анализаторы для разных языков программирования. Он использует формальную грамматику, написанную на языке ANTLR, для генерации кода анализатора. ANTLR поддерживает множество языков программирования и платформ, включая Java, C++, Python, C#, JavaScript и др. Он также предоставляет инструменты для визуализации грамматик и отладки анализаторов. Результатом работы ANTLR является код, который можно использовать для анализа входных данных и генерации дерева синтаксического разбора.

Формальный язык – это набор символов и правил, которые определяют допустимые комбинации символов в этом языке. Формальные языки используются в теории формальных языков и автоматов, логике, математике и информатике. Они часто используются для описания синтаксиса языков программирования и различных форматов данных. Примерами формальных языков являются регулярные языки, контекстно-свободные языки, контекстно-зависимые языки и рекурсивно-перечислимые языки.

Грамматика – это набор правил, которые определяют, какие последовательности символов могут быть сформированы в формальном языке. В грамматике выделяются два типа символов: терминальные и нетерминальные. Терминальные символы – это конечные символы, которые могут быть напрямую записаны в тексте языка. Нетерминальные символы – это символы, которые описывают структуру языка, они используются для создания более сложных последовательностей символов. Нетерминальные символы могут быть заменены на другие последовательности символов с помощью правил грамматики. Формальная грамматика – это грамматика, описывающая формальный язык. В случае с синтаксисом языков программирования такие языки являются контекстно свободными [20].

В качестве основы грамматики для разрабатываемого языка была взята грамматика языка СИ, стандарта ANSI C11. Данный стандарт представляет из собой наиболее популярный стандарт для этого языка, включающий все знакомые конструкции и операторы. Однако данный синтаксис не подходит под определенные требования и большую часть правил необходимо изменить. Так, например, правило объявления функции изменится следующим образом:

(3.1)

где fdef – нетерминальный символ объявления функции;

FUNC – терминальный символ «func»;

id – нетерминальный символ идентификатора;

plist – нетерминальный символ списка параметров;

COLON – терминальный символ двоеточия;

type – нетерминальный символ типа данных;

comstat – нетерминальный символ многострочного блока кода.

В формуле (правиле) (3.1) присутствует символ вопросительного знака, что подразумевает вхождение группы правил в скобках может быть в количестве нуля или одного, то есть по данному правилу валидной комбинацией символов является как «func main(): void {…}», так и «func main() {…}». Помимо этого правила также необходимы изменения условных и циклических операторов, типов данных и так далее.

3.3 Анализ конструкций языка

Лексический и синтаксический анализ – это процессы, которые происходят во время компиляции или интерпретации программного кода. Лексический анализ – это процесс преобразования входного потока символов в последовательность лексем, которые представляют собой синтаксические единицы программы, такие как ключевые слова, операторы, числа, строки и идентификаторы переменных. На этом этапе, символы в программном коде анализируются и разбиваются на токены, которые затем будут использоваться для дальнейшей обработки.

Синтаксический анализ – это процесс анализа последовательности токенов, чтобы определить соответствует ли она грамматике языка программирования. На этом этапе, токены соединяются в синтаксические конструкции, такие как выражения, операторы и функции. В результате, появляется структурированное дерево, представляющее синтаксическую структуру программы. Если последовательность токенов не соответствует грамматике языка программирования, то синтаксический анализатор генерирует ошибку.

Выходными данными на этапе лексического анализа является список токенов, каждый из которых содержит информацию о типе лексемы и ее местоположении в программном коде. Выходными данными на этапе синтаксического анализа является дерево синтаксического разбора, которое представляет структуру программного кода в соответствии с грамматикой языка программирования. Следующим этапом анализа кода является семантический анализ

Семантический анализ – это процесс анализа и интерпретации значения конструкций языка программирования, с целью выявления смысла их использования в программе. Он выполняется после лексического и синтаксического анализа, и позволяет проверить совместимость типов данных, определить правильность использования переменных и функций, а также выполнить оптимизацию кода.

**3.3.1** Повышение уровня абстракции

Дерево синтаксического разбора является детальным представлением синтаксически корректного выражения в языке программирования. Оно содержит листовые элементы, которые представляют синтаксические конструкции, такие как скобки, запятые, точки с запятой и т.д., а также узлы, которые являются правилами языка, например, «список идентификаторов» или «выражение с точкой с запятой». Однако дерево разбора может содержать больше информации, чем нужно для последующей работы с выражением, поэтому принято проводить анализ конструкций в несколько этапов. На рисунке 3.1 показано, что при повышении уровня абстракции деревьев, понижается количество узловых элементов в этих деревьях.



Рисунок 3.1 – Зависимость уровня абстракции от количества узлов

Повышение уровня абстракции от дерева синтаксического разбора к блок-схеме в несколько этапов имеет несколько преимуществ. Во-первых, на более высоком уровне абстракции проще обрабатывать данные и производить необходимые операции. Во-вторых, блок-схема может содержать дополнительную информацию, которую не содержит дерево синтаксического разбора, такую как действия, которые нужно выполнить в рамках блока. Наконец, постепенное повышение уровня абстракции позволяет более четко выделить структуру алгоритма, что упрощает дальнейшую работу с ним.

Абстрактное синтаксическое дерево (AST – Abstract syntax tree) представляет собой дерево, которое описывает структуру синтаксически корректного выражения в языке программирования, отделяя его от деталей реализации. Каждый узел дерева представляет собой операцию, конструкцию или значение, а листьями являются терминалы или лексемы. AST используется в компиляторах и интерпретаторах для упрощения работы с выражениями и оптимизации кода.

Затем, используя шаблон проектирования посетитель, абстрактное синтаксическое дерево обходится и на каждом узле создается соответствующий блок-элемент блок-схемы. Каждый узел AST содержит информацию о соответствующей операции, а его потомки представляют аргументы и подоперации. Благодаря данной информации можно определить тип блока в схеме алгоритма, а также дополнительную информацию по нему, как, например, названия условий в операторе switch.

Следует отметить, что обход AST и сбор информации по схеме алгоритма уже можно относить к семантическому анализу кода. Обычно, при семантическом анализе, производится проверка смысла кода на его корректность. Так, на этом этапе, проверяются типы данных, их использование в операторах. Однако с точки зрения контекста данного веб-сервиса нет необходимости производить полный семантический анализ, так как подразумевается, что он будет использоваться в первую очередь для демонстраций алгоритмов.

**3.3.2** Шаблон программирования «Посетитель»

Так как языки программирования являются по своей сути рекурсивными, то есть могут содержать одинаковые конструкции, вложенные друг в друга, то и глубина дерева может быть сколь угодно большой. Для обработки таких структур данных необходимо использовать шаблон проектирования посетитель.

Шаблон проектирования посетитель используется для обхода элементов структуры, таких как деревья, и выполнения операций над каждым из элементов. Он позволяет добавлять новые операции к классам объектов без изменения этих классов. При обработке деревьев, посетитель позволяет выполнить операции над узлами дерева без привязки к конкретным классам узлов. Он проходит по каждому узлу дерева и вызывает соответствующий метод посетителя для выполнения операции [21]. Таким образом, шаблон проектирования посетитель обеспечивает гибкость и расширяемость при работе с деревьями и другими сложными структурами данных, при этом позволяет обходить каждый узел дерева и работать с ним как с отдельным элементом, без явной обработки вложенности кода. Для демонстрации работы с данным шаблоном можно использовать диаграмму классов.

Диаграмма классов – это один из видов UML диаграмм, который используется для визуализации структуры классов и их взаимосвязей в объектно-ориентированных программах. На диаграмме классы представляются в виде прямоугольников, а связи между классами – стрелками. Диаграмма классов помогает описать отношения между классами, полями и методами, а также определить интерфейсы и абстрактные классы.

На рисунке 3.2 показана структура классов, созданных для анализа дерева синтаксического разбора. В случае разбора абстрактного синтаксического дерева и дерева конструкций алгоритмов используется аналогичная структура классов с соответствующей имплементацией.

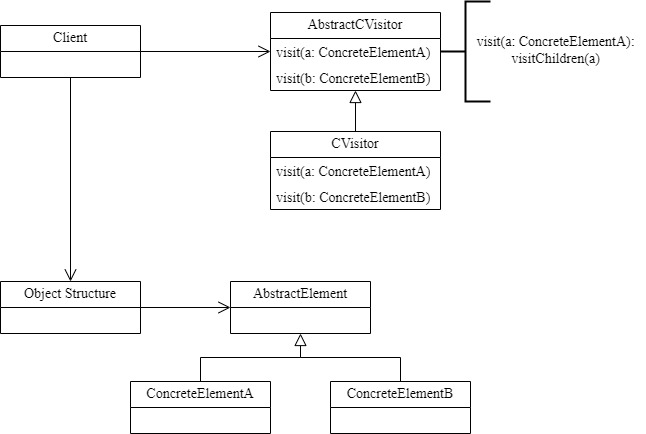


Рисунок 3.2 – Диаграмма классов шаблона «Посетитель»

На данном рисунке видно, что алгоритм шаблона проектирования посетитель можно условно разделить на две составляющие: имплементация самого посетителя и структура объектов. Структура объектов представляет собой узлы дерева, которые необходимо обработать. Так, в случае анализа дерева синтаксического разбора, структура объектов содержит в себе классы, содержащие в себе контекстную информацию каждого синтаксического символа: терминальный или нетерминальный, какие правила были включены, а какие нет, и так далее.

Абстрактный класс посетителя содержит в себе метод на каждый из типов, наследующих структуру объектов. Каждый такой метод содержит имплементацию, включающую в себя только вызов метода посетителя на все дочерние символы. Таким образом, если не унаследовать от абстрактного класса и не переопределять методы посетителя, обход дерева синтаксического разбора просто посетит каждый узел, но при этом не создаст никакие новые структуры данных.

Следует отметить, что при таком обходе происходит неявное построение абстрактного синтаксического дерева в стэке веб-сервиса. Для того чтобы эту структуру построить явно необходимо унаследовать абстрактный класс и переопределить те методы, обработка объекта которых позволит построить AST.

**3.3.3** Анализ деревьев разбора

Так как выходными данными дерева синтаксического разбора является абстрактное синтаксическое дерево, необходимо определить структуру объектов AST, для последующего применения шаблона посетитель:

– CNode. Представляет собой родительский абстрактный класс, который наследуют все остальные классы из структуры. Не содержит никаких свойств.

– ProgramNode. Представляет собой объект всей рабочей программы. Содержит список объявленных функций FunctionNode. Наследует класс Cnode.

– FunctionNode. Представляет собой одну объявленную функцию внутри программы. Содержит имя функции, список параметров, а также тело функции, которое является CompoundStatementNode. Наследует класс Cnode.

– StatementNode. Является логическим оператором языка. Не является конкретным оператором, но можно представить его как определение различных структур языка. Наследует класс Cnode.

– CompoundStatementNode. Является последовательностью вызовов логических единиц программы. Представляет собой список StatementNode. Наследует класс StatementNode, то есть подразумевая, что CompoundStatementNode может состоять из самого себя.

– DeclarationStatementNode. Представляет собой объявление переменной вместе с ее типом. Наследует класс StatementNode.

– IterationStatementNode. Представляет собой цикл внутри программы. Содержит информацию о типе цикла, его условии и также тела CompoundStatementNode. Наследует класс StatementNode.

– JumpStatementNode. Представляет собой оператор перехода. Содержит информацию о типе перехода, а также его условия. Наследует класс StatementNode.

– LabeledStatementNode. Представляет из себя именованный участок кода программы. Содержит информацию о типе именования, условии и тела участка CompoundStatementNode. Наследует класс StatementNode.

– SelectionStatementNode. Представляет собой условный оператор. Содержит информацию об его условии. Наследует класс StatementNode.

– SwitchStatementNode. Представляет из себя условный оператор switch. Содержит информацию о теле оператора, в виде списка LabeledStatementNode. Наследует класс SelectionStatementNode.

– IfStatementNode. Представляет собой условный оператор if. Содержит информацию о теле положительного условия и о теле отрицательного условия, в виде CompoundStatementNode. Наследует класс SelectionStatementNode.

– ExpressionStatementNode. Является логическим вызовом языка. Представляет собой законченную единицу вызова внутри программы, можно представить его как элемент кода, после которого должна ставиться точка с запятой. Наследует класс StatementNode.

– NoOpStatementNode. Представляет собой элемент-заглушку, в случае отсутствия вызовов внутри программы. Наследует класс ExpressionStatementNode.

– FunctionCallStatementNode. Представляет собой единый вызов функции внутри программы. Содержит информацию об имени вызываемой функции, а также переданных аргументах. Наследует класс ExpressionStatementNode.

– OtherExpressionStatementNode. Представляет собой все остальные возможные операции языка, например, сложение, вычитание, присваивание и так далее, которые не были явно объявлены структурой классов выше. Наследует класс ExpressionStatementNode.

– IterationStatementNodeType. Представляет собой тип цикла. Может принимать значения FOR, WHILE, DO\_WHILE.

– LabeledStatementNodeType. Представляет собой тип именования участка кода. Может принимать значения CASE, DEFAULT.

– JumpStatementNodeType. Представляет собой тип оператора перехода. Может принимать значения RETURN, BREAK, CONTINUE.

Перечисленная структура, состоящая из 19 классов, необходима для повышения уровня абстракции дерева синтаксического разбора, размер описания которого составляет 60 классов. После выполнения преобразования и заполнения всех свойств объектов AST, на выходе получается единый объект типа Cnode (его подкласса ProgramNode), который содержит полную структуру программы в виде дерева. Для дальнейшего преобразования его в схему алгоритма необходимо привести его к новой структуре, используя аналогичную имплементацию шаблона программирования посетитель, со следующей иерархией:

– CodeProgram. Представляет собой единую программу, содержащую список функций CodeFunction. Необходима только для объединения списка функций в одном объекте.

– CodeFunction. Представляет собой единую функцию внутри блок-схемы, ограниченную двумя прямоугольными блоками с закругленными сторонами. Содержит в себе название функции и список параметров, а также список вызовов внутри функции CodeStatement.

– CodeStatement. Интерфейс, каждая реализация которого представляет из себя один (например, в случае вызова предопределенной подпрограммы) или множество (например, в случае циклов и условных операторов) блоков схемы.

– CodeIteration. Представляет из себя участок схемы алгоритма, обрамленный началом и концом цикла. Содержит информацию об условии цикла, его типе и теле – список CodeStatement.

– CodeExpression. Представляет собой единый вызов внутри программы. Содержит тип вызова и тело.

– CodeIfSelection. Представляет собой участок схемы алгоритма с условным оператором – блок ромб. Содержит в себе информацию об условии оператора, теле положительного и теле отрицательного условия, в виде списка CodeStatement.

– CodeSwitchSelection. Представляет собой участок схемы алгоритма с условным оператором со множеством условий – блок ромб. Содержит в себе информацию об отношении названия условия перехода и тела в виде списка CodeStatement.

– CodeCall. Представляет собой вызов функции. Содержит имя вызываемой функции, а также список передаваемых аргументов.

– IterationType. Представляет собой вид цикла внутри схемы алгоритма. Может принимать значения предусловного и постусловного цикла.

– ExpressionType. Представляет собой тип вызова внутри программы. Может принимать значения DECLARATION, RETURN, BREAK, CONTINUE, OTHER.

Данного представления программы уже достаточно для того чтобы за один проход можно было сгенерировать схему алгоритма, так как каждый отдельный элемент представляет собой отдельный блок или участок блоков внутри схемы. Однако такое представление является описанием графа, в котором связи между узлами представлены неявно в виде отношений между дочерними элементами. Такой вид записи графа хоть и является эффективным, с точки зрения обработки вершин, но не является эффективным с точки зрения хранения и передачи его по сети. В связи с этим необходимо еще раз обойти это дерево с помощью шаблона проектирования посетитель и построить следующий список объектов:

– Graph – содержит в себе всю необходимую информацию для сериализации и хранения представления графа как в памяти, так и в базе данных. Такое представление также подходит и для эффективной передачи по сети. Содержит следующие поля:

а) список GraphNode, который является списком узлов;

б) список смежности.

– GraphNode – является узлом графа, который содержит в себе информацию о каждом блоке внутри схемы. Содержит следующий набор полей:

а) идентификатор, необходимый для связи;

б) тип узла (форма блока);

в) текст узла.

В данном представлении информация о соединениях хранится с помощью списка смежности. Список смежности – это способ представления графа в виде списка, где каждый узел графа имеет свой список смежности, содержащий информацию о смежных с ним узлах. Каждый элемент списка представляет собой пару (v, w), где v – вершина, а w – вершина, смежная с v. Список смежности позволяет компактно хранить информацию о графе и быстро находить смежные вершины [22].

3.4 Авторизация и аутентификация

Аутентификация и авторизация являются ключевыми аспектами любого веб-приложения. Существует несколько типов алгоритмов, которые используются для этой цели, такие как Oauth, SAML, OpenID Connect и JSON Web Tokens (JWT). JWT – это стандарт авторизации, который использует токены для передачи информации между клиентом и сервером. Он основан на кодировании JSON-объектов, которые являются небольшими и легкими для чтения и передачи.

JWT состоит из трех частей: заголовок, полезная нагрузка и подпись. Заголовок содержит информацию о типе токена и используемом алгоритме шифрования. Полезная нагрузка содержит информацию о пользователе, а также дополнительные данные, которые могут быть переданы от сервера к клиенту. Подпись используется для проверки целостности токена и подлинности отправителя [23].

JWT имеет несколько преимуществ перед другими алгоритмами. Во-первых, токены могут быть легко переданы между клиентом и сервером, что делает их более эффективными чем сессии, которые требуют сохранения состояния на сервере. Во-вторых, токены могут быть шифрованы, что обеспечивает безопасность передачи информации. В-третьих, JWT можно легко интегрировать с различными языками программирования и фреймворками, что делает их удобными для использования в различных проектах.

JWT является наиболее подходящим выбором для данного проекта по нескольким причинам:

1) Обеспечивает безопасную аутентификацию и авторизацию пользователей.

2) Использует формат JSON, который обеспечивает простоту чтения и анализа данных.

3) Обеспечивает возможность передачи дополнительных пользовательских данных, которые могут быть полезны при реализации функциональности приложения.

4) Является легковесным и быстрым в использовании, что особенно важно для веб-сервисов с большой нагрузкой.

5) Большинство библиотек и фреймворков, в том числе Spring Boot, поддерживают работу с JWT, поэтому интеграция данного алгоритма не составит никаких проблем.

3.5 Сохранение запросов в базу данных

Шаблон проектирования «Посредник» представляет собой поведенческий паттерн, который используется для уменьшения связанности между объектами системы. Он предоставляет промежуточный объект-посредник, который управляет взаимодействием между объектами, скрывая сложность их коммуникации друг с другом.

В контексте разработки веб-сервисов, использование шаблона проектирования посредник может значительно повысить модульность и переиспользуемость кода. Разделение бизнес-логики и работы с базой данных на отдельные слои позволяет более эффективно управлять изменениями и расширять функциональность сервиса.

Применение прокси-сервиса в качестве посредника позволяет реализовать разделение кода, предоставляя чистый интерфейс для работы с базой данных. Это также позволяет более эффективно масштабировать приложение, путем добавления новых экземпляров сервисов без изменения бизнес-логики.

В контексте данного веб-сервиса роль прокси-сервиса играет сервис сохранения запросов. Разбивая логику сохранения исходного кода и сгенерированной схемы алгоритма от логики генерации этой схемы, повышается модульность и читаемость кода, а также облегчается последующее изменение сервиса генерации без необходимости менять код работы с базой данных.

На рисунке 3.3 отображена диаграмма последовательности. Диаграмма последовательности – это один из типов диаграмм, который показывает последовательность взаимодействия между объектами в рамках определенной задачи или сценария. Она состоит из вертикальных линий, называемых линиями жизни, и стрелок, которые показывают порядок выполнения операций. Данный тип диаграммы используется для визуализации процесса взаимодействия между объектами, участвующими в выполнении определенных задач или сценариев.

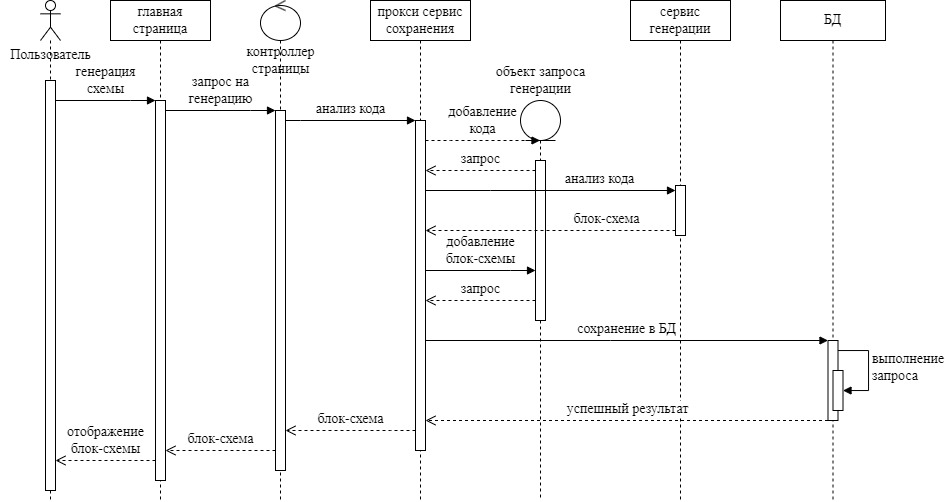


Рисунок 3.3 – Диаграмма последовательности обработки запроса

В данной диаграмме наглядно видно разделение ответственности сервисов сохранения и генерации блок-схемы. С одной стороны сервис генерации никак не взаимодействует с базой данных и даже не знает о ее существовании, с другой стороны сервис сохранения запросов перехватывает все вызовы, связанные с блок-схемой, и сохраняет их, для последующего отображения пользователю его истории.

3.6 Даталогическая модель базы данных

Даталогическая модель является одним из этапов проектирования базы данных и представляет собой описание структуры данных, их отношений и ограничений на уровне сущностей и атрибутов. Она описывает логическую организацию данных, которая может быть использована для создания реальной базы данных. В отличие от инфологической модели, которая описывает общие концепции и взаимосвязи между сущностями, даталогическая модель является более конкретной и детализированной, определяя типы данных, ограничения целостности и связи между таблицами.

Существует несколько форм записи даталогической модели базы данных, наиболее распространенными из которых являются ER-диаграммы и диаграммы классов UML. ER-диаграммы состоят из сущностей, атрибутов и связей между ними, а диаграммы классов UML описывают классы, атрибуты и отношения между классами. Также используются формы записи, основанные на текстовом описании структуры базы данных, такие как SQL-схемы. Каждый из этих подходов имеет свои преимущества и недостатки и может быть выбран в зависимости от конкретных потребностей проекта и предпочтений разработчика.

Использование UML для моделирования даталогической модели базы данных имеет ряд преимуществ. Во-первых, UML позволяет описать диаграмму классов, которая визуально отображает структуру базы данных и связи между таблицами. Во-вторых, UML обеспечивает возможность описания ограничений целостности данных и связей между таблицами, что делает модель более полной и точной. Наконец, UML имеет стандартную нотацию, что упрощает понимание модели другими разработчиками и программистами.

На рисунке 3.4 отображена схема даталогической модели базы данных веб-сервиса генерации схемы алгоритма из исходного кода.

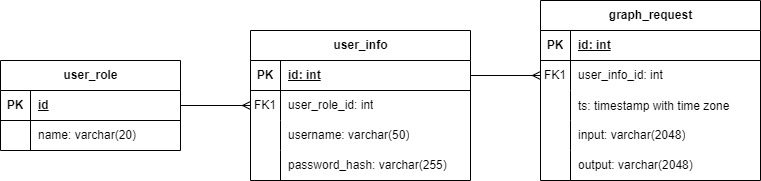


Рисунок 3.4 – Даталогическая модель

В данном случае каждому из сущностей присвоен уникальный искусственный ключ целочисленного типа. Данный выбор обоснован тем, что использование искусственного ключа, генерируемого самой системой, является более надежным и удобным способом идентификации записей в базе данных, чем использование естественного ключа, который основан на некотором естественном свойстве объекта, например, его названии или номере. Естественный ключ может быть неуникальным или изменяться со временем, что может привести к проблемам с целостностью данных. Искусственный ключ, напротив, гарантированно уникален и не изменяется в течение жизненного цикла объекта. Он также может быть проще в использовании для поиска, сортировки и связывания данных.

В таблице user\_role, которая представляет из себя роль пользователя в системе содержится только один атрибут – название роли. Было выбрано хранить роль в базе данных, так как, во-первых, это позволяет управлять ролями и разрешениями на уровне базы данных, что облегчает изменение прав доступа без необходимости изменения кода. Во-вторых, это повышает безопасность, так как база данных обычно имеет более высокий уровень защиты, чем код. В-третьих, это облегчает интеграцию с другими системами, так как данные о ролях пользователей могут быть легко переданы между различными приложениями и сервисами через базу данных. Кроме того, хранение ролей пользователей в базе данных улучшает масштабируемость, позволяя эффективно управлять правами доступа для большого количества пользователей и приложений.

Последнее, на что необходимо обратить внимание, это тип атрибута ts в таблице graph\_request. Данное поле хранит в себе время, в которое был произведен запрос на генерацию схемы алгоритма. Тип данных «timestamp with timezone» позволяет хранить временные данные вместе с информацией о часовом поясе [24]. Без учета часового пояса временная метка может быть некорректно интерпретирована.

3.7 Проектирование интерфейса пользователя клиентской части

Интерфейс пользователя – это то, через что пользователи взаимодействуют с программным продуктом или устройством, таким как приложение, веб-сайт или мобильное приложение. Он представляет собой совокупность элементов управления, таких как кнопки, поля ввода, списки, диалоговые окна и т.д., которые обеспечивают удобство и простоту использования программного продукта. Цель хорошего интерфейса пользователя заключается в том, чтобы облегчить выполнение задач пользователям и сделать использование программного продукта более эффективным и приятным.

Разработка интерфейса пользователя является важной частью процесса создания программного продукта. Это связано с тем, что интерфейс является главным способом взаимодействия пользователя с приложением. Пользователи оценивают продукт не только по его функциональности, но и по удобству использования. Поэтому важно разработать интерфейс, который будет интуитивно понятен и прост в использовании, чтобы пользователи могли максимально эффективно работать с приложением. Кроме того, хороший интерфейс может повысить удовлетворенность пользователей и стимулировать повторное использование продукта.

Помимо разработки основных страниц сайта веб-сервиса и их нормального поведения, также необходимо спроектировать обработку ошибочного поведения, как пользователя, так и веб-сервиса. Под этим подразумевается любое ошибочное состояние веб-сервиса: неправильно введенные авторизационные данные, ошибка в исходном коде и так далее. Также необходимо разработать и «пустое» состояние, то есть такое состояние, при котором еще не было произведено никакое действие. Разработка компонентов веб-сервиса в таком ключе не только улучшит внешний вид сайта, но также и повысит простоту его использования и сделает опыт работы приятнее.

На рисунках 3.5-3.8 не представлены конкретные решения по цветовой схеме и графическом наполнении сайта, а только его структурное описание. Тем не менее, такой подход значительно увеличит скорость последующей разработки страниц, так как он подразумевает разработку всех состояний веб-сервиса.

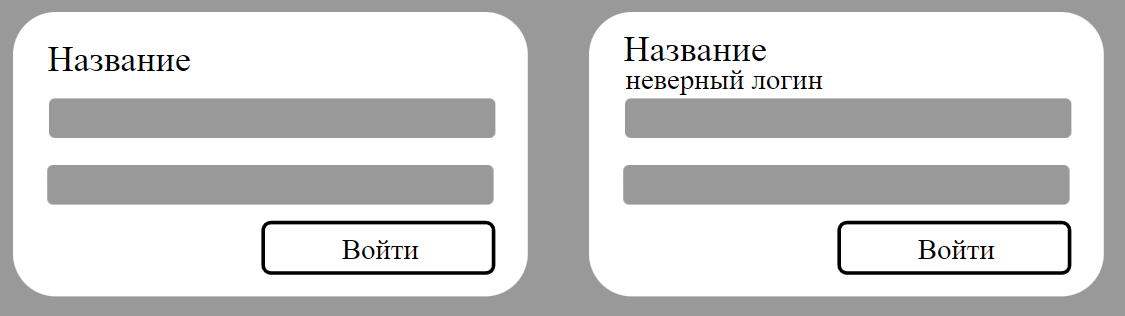


Рисунок 3.5 – Форма авторизации

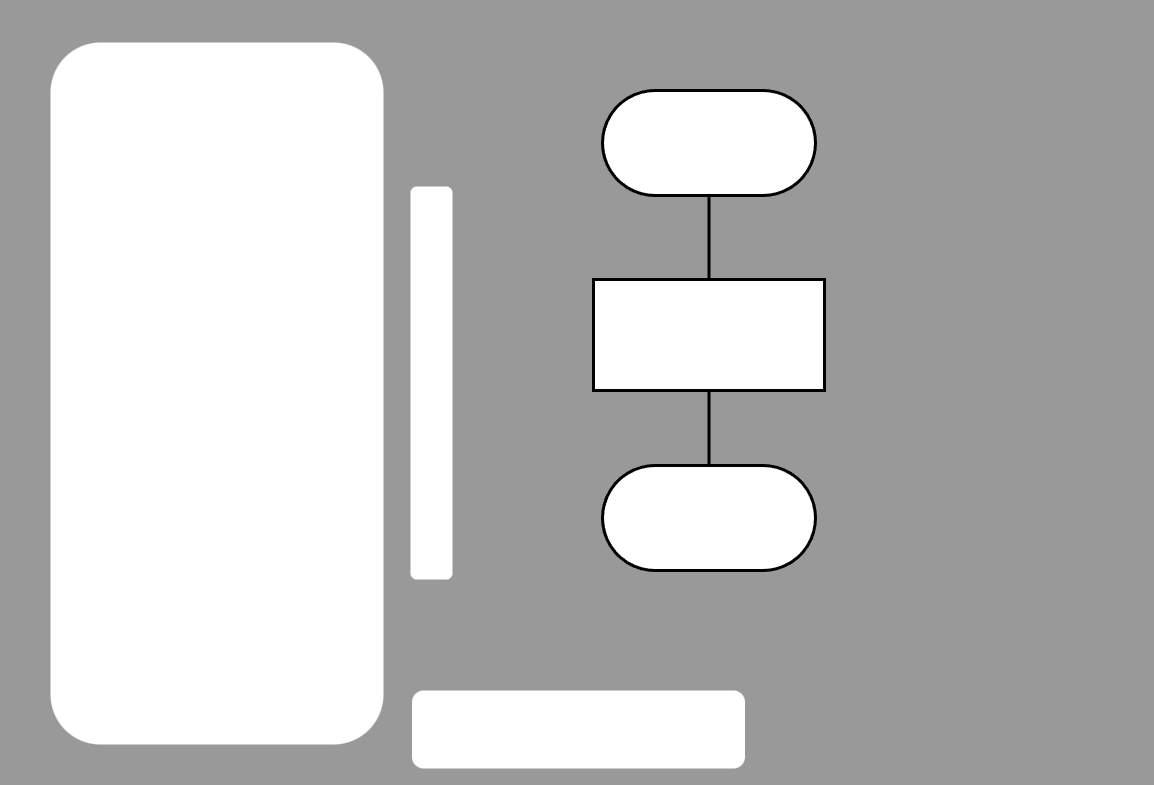


Рисунок 3.6 – Страница генерации схемы

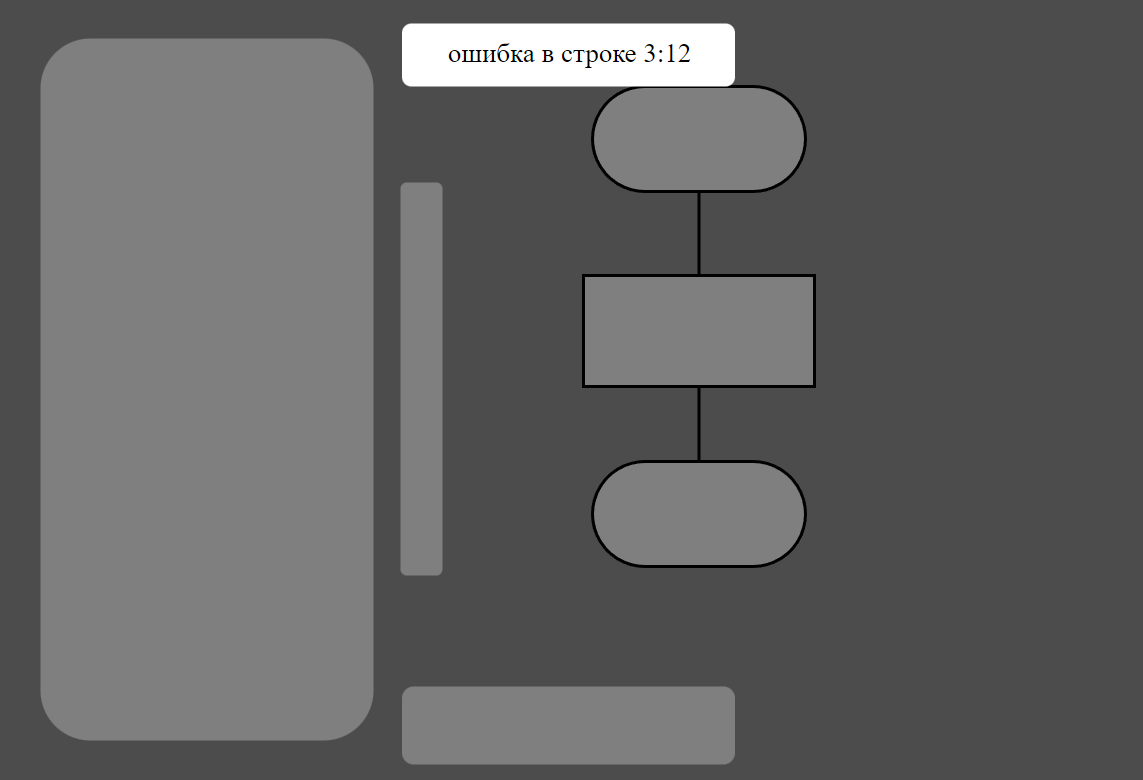


Рисунок 3.7 – Сообщение о синтаксической ошибке



Рисунок 3.8 – Страница истории

Рисунок 3.5 показывает, как должна выглядеть страница авторизации. Данная страница также является страницей регистрацией, в случае отсутствия записи о пользователе, создается новый. На рисунке 3.5 присутствуют:

– название веб-сервиса;

– поле для ввода имени пользователя;

– поле для ввода пароля;

– кнопка авторизации.

В случае возникновения ошибки авторизации необходимо отобразить сообщение.

На рисунке 3.6 видно, как должна выглядеть основная страница с редактором кода и схемы алгоритма. Данная страница позволяет пользователю изменять исходный код, а также отправлять его на генерацию блок-схемы. Рисунок 3.6 содержит:

– поле редактора кода с подсветкой синтаксиса;

– меню инструментов с кнопкой генерации и вставки операторов языка;

– поле редактора схемы алгоритма;

– меню выбора страницы.

В случае первоначальной загрузки страницы генерации схемы алгоритма, необходимо отобразить надпись, которая подсказывает пользователю, что необходимо инициировать процесс с помощью нажатия кнопки. В случае обнаружения синтаксических ошибок в коде пользователя, веб-сервису необходимо отобразить всплывающее сообщение с номером строки и символа, где была допущена ошибка, как это показано в пункте 1 рисунка 3.7.

На рисунке 3.8 отображена страница с историей запросов на генерацию схемы алгоритма, основным компонентом которого является таблица. У данной таблицы имеется возможность просмотра элементов в порядке их вызова, то есть отсортированному по времени. Данная таблица включает в себя следующие столбцы:

– время запроса;

– краткое содержание исходного кода;

– краткое описание содержания схемы алгоритма;

– просмотр запроса.

В случае нажатия на кнопку в пункте 5 необходимо открыть страницу генерации схемы, отображенную на рисунке 3.6, предварительно заполнив редактор кода исходным кодом из запроса, а редактор блок-схемы – схемой алгоритма из запроса.

4 Создание веб-сервиса

4.1 Фреймворк Spring Security

Spring Security [25] – это фреймворк для обеспечения безопасности приложений на платформе Java. Он предоставляет множество инструментов и функций для аутентификации и авторизации пользователей в приложении. Spring Security обеспечивает защиту от множества видов атак, таких как атаки на сессии, подделку запросов, XSS и другие.

Spring Security используется для обеспечения безопасности приложений на платформе Java. Он позволяет разработчикам легко добавлять функции безопасности к своим приложениям, не тратя много времени на разработку своих собственных решений. Spring Security предоставляет мощную конфигурационную модель, которая позволяет определять правила безопасности для различных ресурсов приложения, включая URL, методы и аннотации. Кроме того, Spring Security имеет множество плагинов и расширений, которые облегчают интеграцию с другими фреймворками и библиотеками.

В данном дипломном проекте уместно использовать Spring Security для работы с авторизацией и аутентификацией по нескольким причинам. Во-первых, Spring Security предоставляет множество готовых решений для обеспечения безопасности приложения, включая механизмы аутентификации, авторизации, защиту от атак и т.д. Это позволяет значительно ускорить разработку и обеспечить высокую надежность системы безопасности.

Во-вторых, Spring Security интегрируется с другими фреймворками Spring, такими как Spring MVC и Spring Boot, что обеспечивает удобную работу с различными компонентами приложения и позволяет легко настраивать параметры безопасности. Кроме того, Spring Security легко расширяем и настраиваем, что позволяет адаптировать его под конкретные требования проекта.

Основным понятием в этом фреймворке являются фильтры. Фильтры в Spring Security позволяют выполнять определенные операции при обработке запросов, прошедших через систему безопасности. Они могут применяться для авторизации и аутентификации пользователей, а также для выполнения других операций, например, логирования или обработки ошибок. Фильтры в Spring Security работают в цепочке, где каждый фильтр выполняет свою операцию, а затем передает управление следующему фильтру. Это позволяет гибко настраивать процесс обработки запросов и применять различные операции в зависимости от нужд приложения. На рисунке 4.1 показана цепочка фильтров, через которую проходит запрос до того, как обработается сервисом.

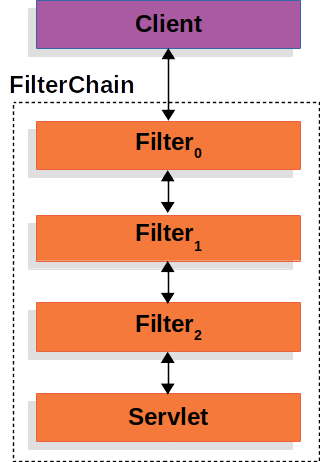


Рисунок 4.1 – Цепочка фильтров

Помимо этого, в веб-сервисе также был настроен CORS с использованием Spring Security. CORS – это механизм безопасности, который контролирует доступ к ресурсам на другом домене или порте веб-страницы. Он используется для предотвращения атак, связанных с межсайтовым скриптингом (XSS). Браузеры применяют политики CORS, чтобы определить, разрешить ли веб-странице запрашивать ресурсы с других источников и разрешать ли веб-серверу отвечать на эти запросы. Если политики CORS не настроены правильно, браузер может заблокировать запрос к ресурсам на других доменах или портах.

Ниже представлен участок кода, с помощью которого настраиваются заголовки CORS при помощи использования фильтров:

*// Данный фильтр обрабатывает ответ в самую последнюю очередь,*

*// чтобы проставить все необходимые заголовки*

@Component

**class** CorsFilter : OncePerRequestFilter() {

*// Обработка запроса и ответа внутри фильтра*

**override** **fun** doFilterInternal(

request: HttpServletRequest,

response: HttpServletResponse,

filterChain: FilterChain

) {

*// установление заголовка разрешенного имени источника (сайта)*

response.setHeader(“Access-Control-Allow-Origin”, FRONTEND\_URL)

*// установление заголовка разрешенных методов*

response.setHeader(“Access-Control-Allow-Methods”,

“GET, POST, OPTIONS”)

*// установление заголовка разрешенных заголовков*

response.setHeader(“Access-Control-Allow-Headers”,

“Authorization, Content-Type”)

*// вызов всех оставшихся фильтров*

filterChain.doFilter(request, response)

}

}

Благодаря данной конфигурации браузер может определить, является ли запрос на сервер авторизованным, и, в случае несовпадения значения каких-либо заголовков, отклонить ответ от запроса, во избежание утечки защищенной информации пользователя.

4.2 Библиотека Liquibase

Liquibase [26] – это инструмент для управления схемой базы данных, который позволяет создавать и применять изменения в структуре базы данных. Он представляет собой Java-библиотеку, которая может быть интегрирована в проекты Java и некоторые другие языки программирования. Liquibase использует XML-файлы для описания изменений, которые должны быть применены к базе данных, и генерирует SQL-скрипты для автоматического применения изменений.

Liquibase позволяет автоматически обновлять базу данных при изменении структуры или добавлении новых функций, и предоставляет механизм отката изменений в случае ошибок. Он также позволяет контролировать версии базы данных, чтобы облегчить управление изменениями и согласованность схемы между разными средами. Liquibase может использоваться в различных сценариях, от небольших проектов до крупных приложений, и является распространенным инструментом в сообществе разработчиков.

В данном дипломном проекте использование Liquibase имеет несколько преимуществ. Во-первых, Liquibase позволяет удобно управлять и обновлять схему базы данных. Он позволяет создавать и изменять структуру базы данных в виде «миграций», которые можно версионировать и применять в автоматическом режиме. Это снижает вероятность ошибок при обновлении базы данных, упрощает сопровождение проекта и повышает надежность при работе с данными.

Во-вторых, Liquibase позволяет использовать множество различных типов баз данных и SQL-диалектов, что делает его универсальным решением для работы с базами данных. Это особенно важно в случае, когда проект использует несколько баз данных разных типов, так как Liquibase позволяет управлять ими из единого места и использовать единый подход к управлению схемой баз данных. Несмотря на то, что в данном веб-сервисе используется только одна СУБД – PostgreSQL, использование этой библиотеки обеспечит безошибочную работу БД в дипломном проекте.

Ниже представлен фрагмент конфигурационного файла, настраивающий таблицу с информацией пользователя, а также первичный и внешний ключи:

- createTable: # команда создания таблицы в БД

tableName: user\_info # название таблицы

columns:

- column:

name: id # имя поля

type: int # тип поля

autoIncrement: true

constraints:

nullable: false

primaryKey: true # установление первичного ключа

# остальные поля и таблицы ...

- addForeignKeyConstraint: # команда создания внешнего ключа

constraintName: fk\_graph\_user # название внешнего ключа

baseTableName: graph\_request

baseColumnNames: user\_info\_id

referencedTableName: user\_info # имя таблицы

referencedColumnNames: id # имя первичного ключа

4.3 Создание веб-сайта

SPA – это веб-приложение, которое работает внутри одной страницы без перезагрузки страницы при каждом запросе. В SPA вся страница загружается единожды, а затем все действия пользователя, взаимодействующие с сервером, происходят без перезагрузки страницы, изменяя только часть содержимого страницы.

Одним из главных преимуществ SPA является быстродействие и более плавный пользовательский интерфейс, так как пользовательский опыт более похож на приложение, работающее на компьютере. Кроме того, SPA позволяет сократить объем передаваемых данных между клиентом и сервером, так как не требуется загрузка каждой новой страницы.

В данном дипломном проекте использование SPA вполне уместно, так как проект является веб-сервисом с динамическим содержимым и требует быстрого отклика. SPA также позволяет легко реализовывать асинхронные запросы к серверу без перезагрузки страницы и обновления только нужных данных на странице.

Наиболее значимой и динамической частью веб-сервиса является страница с генераций схемы алгоритма. Пользователь может загрузить исходный код на страницу и нажать кнопку для генерации схемы алгоритма. С помощью SPA будет отправлен запрос на сервер, который будет анализировать код и возвращать результаты в формате JSON. SPA затем будет использовать эти результаты для построения схемы алгоритма, которая будет отображаться на странице в удобном виде для пользователя. В таком формате работы генерация происходит незаметно для пользователя, так как передача данных небольшого размера не занимает большого количества времени, а отсутствие перезагрузки дает возможность браузеру отрисовывать не все части страницы в начале работы, а лишь делать это по необходимости и с ограниченным списком видимых элементов.

5 Тестирование веб-сервиса

Тестирование – это процесс проверки программного обеспечения на соответствие требованиям и ожиданиям пользователей. Цель тестирования – обнаружить дефекты в программном обеспечении до его выпуска в эксплуатацию.

Существуют различные виды тестирования, которые могут варьироваться в зависимости от цели и характеристик тестируемого продукта. Например, функциональное тестирование проверяет, соответствует ли программа функциональным требованиям, а тестирование производительности оценивает скорость работы программного обеспечения в различных условиях использования.

Тестирование необходимо для обеспечения качества программного обеспечения, уменьшения рисков для бизнеса и улучшения пользовательского опыта. Тестирование позволяет выявить дефекты и ошибки в программном обеспечении, которые могут привести к сбоям или ошибкам при использовании продукта, а также позволяет убедиться в соответствии программного обеспечения требованиям и ожиданиям пользователей.

Ручное тестирование – это процесс проверки функциональности и качества программного продукта с помощью ручных действий тестировщиков. В отличие от автоматизированного тестирования, где тесты запускаются с помощью компьютерных программ, ручное тестирование выполняется вручную, что позволяет тестировщикам проверить продукт на наличие ошибок и проблем, которые могут быть упущены в автоматическом режиме.

Ручное тестирование может включать в себя различные виды тестирования, такие как функциональное тестирование, тестирование на соответствие требованиям, тестирование на устойчивость к нагрузкам, тестирование на безопасность и т.д. Ручное тестирование может быть выполнено как на уровне отдельных функций или модулей, так и на уровне всей системы в целом.

В таблице 5.1 приведены сценарии ручного тестирования, которые были использованы для проверки пользовательского интерфейса, а также для проверки правильности работы бизнес-логики.

Таблица 5.1 – Набор тестов для проверки веб-сервиса

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название теста | Шаги | Ожидаемый результат | Действительный результат |
| 1 | Регистрация | 1. Перейти на основную страницу веб-сервиса  2. Ввести корректные и уникальные данные  3. Нажать кнопку «Login» | 1. В БД появится новая информация о пользователе  2. Основная страница сменится на страницу генерации схемы | 1. Создана новая сущность пользователя  2. Страница автоматически сменилась на редактор кода  Тест пройден успешно |
| 2 | Авторизация | 1. Перейти на основную страницу веб-сервиса  2. Ввести корректные данные пользователя  3. Нажать кнопку «Login» | Основная страница сменится на страницу генерации схемы | Страница автоматически сменилась на редактор кода  Тест пройден успешно |
| 3 | Валидация форм авторизации | 1. Перейти на основную страницу веб-сервиса  2. Ввести некорректные данные  3. Нажать кнопку «Login» | На форме авторизации отобразится сообщение об ошибке валидации | На форме авторизации отобразилось сообщение с указанием ошибки валидации  Тест пройден успешно |
| 4 | Генерация схемы алгоритма | 1. Перейти на страницу с редактором кода  2. Ввести корректный код  3. Нажать зеленую кнопку | В редакторе схемы отобразится схема алгоритма для написанного кода | В редакторе схемы отобразилась корректная схема алгоритма  Тест пройден успешно |
| 5 | Валидация кода | 1. Перейти на страницу с редактором кода | Отобразится сообщение с указанием | Отобразилось сообщение с указанием |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название теста | Шаги | Ожидаемый результат | Действительный результат |
|  |  | 2. Ввести некорректный код  3. Нажать зеленую кнопку | номера строки и символа с некорректным символом | номера строки и символам с некорректным кодом  Тест пройден успешно |
| 6 | Изменение схемы | 1. Повторить шаги из теста 4  2. Редактировать схему с помощью нажатия и перемещения блоков | Блоки в схеме алгоритма перемещаются с мышкой, стрелки автоматически обновляются | Блоки перемещаются вместе с указателем мыши, стрелки рисуются корректно  Тест пройден успешно |
| 7 | Сохранение схемы в изображение | 1. Повторить шаги из теста 4  2. Нажать кнопку с иконкой «Export» | Изображение со схемой алгоритма сохраниться на устройство | Изображение в формате PNG сохранилось на устройство  Тест пройден успешно |
| 8 | Просмотр истории | 1. Несколько раз повторить шаги из теста 4  2. Перейти на страницу истории | В таблице отобразится то количество запросов, которые были произведены в первом шаге данного теста | В таблице отобразилось последние 5 запросов, а также в отдельном пункте отобразилось общее количество запросов  Тест пройден успешно |
| 9 | Просмотр одного элемента истории | 1. Повторить шаги из теста 8 | Страница переключится на страницу c | Страница с редактором кода отобразила код |

Продолжение таблицы 5.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Название теста | Шаги | Ожидаемый результат | Действительный результат |
|  |  | 2. Нажать кнопку «View» рядом с одним из запросов | редактором кода, а редактор кода и схемы заполнятся данными из запроса | и схему алгоритма из запроса  Тест пройден успешно |
| 10 | Выход из аккаунта | 1. Перейти на любую страницу веб-сервиса  2. Нажать красную кнопку с иконкой креста | Отобразится страница авторизации | Отобразилась страница авторизации, при попытке перейти на другую страницу веб-сервиса автоматически открывается страница авторизации  Тест пройден успешно |

Согласно таблице 5.1 в результате тестирования было установлено, что все тесты были пройдены успешно, веб-сервис функционирует без ошибок и соответствует всем требованиям, предъявляемым к подобным веб-приложениям. Все реализованные функции работают корректно, а интерфейс приложения интуитивно-понятен и удобен для использования.

Таким образом, можно заключить, что веб-сервис выполнен в полном соответствии с поставленными задачами и требованиями.

6 Руководство по установке и использованию

6.1 Руководство по установке

Данный веб-сервис предоставляется в виде репозитория. Для полноценной работы веб-сервиса на целевой ЭВМ, помимо самих исполняемых файлов, также необходима установка Java и NodeJS, а также СУБД PostgreSQL.

**6.1.1** Установка Java 17

Для установки Java 17 необходимо выполнить следующие пункты:

1) Необходимо загрузить установочный файл Java 17 с официального сайта Oracle, перейдя по ссылке на страницу загрузки: https://www.oracle.com/java/technologies/javase-downloads.html. Для этого необходимо нажать на последовательность кнопок, приведенную на рисунке 6.1.

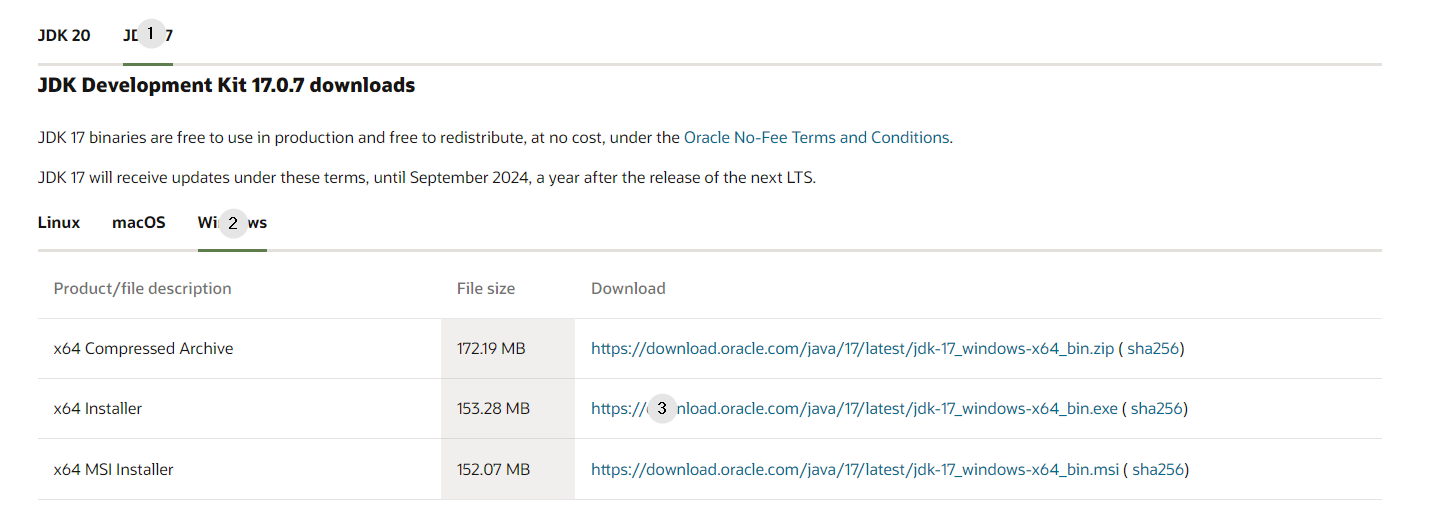


Рисунок 6.1 – Загрузка Java 17

2) Перейти в папку загрузок на ЭВМ, где сохранен загруженный файл установки.

3) Открыть загруженный файл установки, дважды щелкнув на нем мышкой.

4) Необходимо выбрать язык установки, нажав на соответствующую кнопку.

5) Прочитать и принять условия лицензионного соглашения. Для того чтобы прочитать условия лицензионного соглашения, необходимо нажать на ссылку «License Agreement».

6) Выбрать место установки на ЭВМ. Для выбора места установки необходимо нажать на кнопку «Change».

7) Дождаться завершения установки.

8) Закрыть окно установки.

После выполнения этих действий на ЭВМ будет установлена рабочая версия Java 17.

**6.1.2** Установка NodeJS 18

Для установки NodeJS 18 необходимо выполнить следующие пункты:

1) Необходимо загрузить установочный файл Node.js 18.14.0 с официального сайта Node.js, перейдя по ссылке на страницу загрузки: https://nodejs.org/en/download/. Для этого необходимо нажать на последовательность кнопок, приведенную на рисунке 6.2.



Рисунок 6.2 – Загрузка NodeJS 18

2) Перейти в папку загрузок на ЭВМ, где сохранен загруженный файл установки.

3) Открыть загруженный файл установки, дважды щелкнув на нем мышкой.

4) Необходимо выбрать язык установки, нажав на соответствующую кнопку.

5) Прочитать и принять условия лицензионного соглашения. Для того чтобы прочитать условия лицензионного соглашения, необходимо нажать на ссылку «License Agreement».

6) Выбрать место установки на ЭВМ. Для выбора места установки необходимо нажать на кнопку «Change».

7) Дождаться завершения установки.

8) Закрыть окно установки.

После выполнений этих действий на ЭВМ будет установлена рабочая версия NodeJS.

**6.1.3** Установка PostgreSQL

Для установки СУБД PostgreSQL необходимо выполнить следующие пункты:

1) Необходимо скачать установочный файл PostgreSQL для Windows с официального сайта. Для этого можно перейти по ссылке https://www.postgresql.org/download/windows/ и выбрать нужную версию.

2) Необходимо открыть установочный файл и нажать на кнопку «Next» для начала установки.

3) Следующим шагом необходимо выбрать каталог, в который будет установлен PostgreSQL. Рекомендуется использовать каталог по умолчанию.

4) После выбора компонентов следует указать порт, который будет использоваться для подключения к PostgreSQL. Рекомендуется оставить значение по умолчанию – 5432.

5) Далее необходимо указать пароль для пользователя postgres, который будет использоваться для доступа к базе данных. Рекомендуется выбрать сложный и безопасный пароль.

6) После указания пароля следует выбрать тип установки. Рекомендуется выбрать тип «Complete», который устанавливает все компоненты.

7) Далее следует нажать на кнопку «Next» и дождаться окончания установки.

8) После установки необходимо запустить PostgreSQL. Для этого можно найти в меню «Пуск» пункт «PostgreSQL» и выбрать «pgAdmin».

9) После входа в pgAdmin нужно выбрать «Servers» в левой панели и выбрать сервер PostgreSQL, который был установлен.

10) Далее нужно выбрать «Databases» и нажать на кнопку «Create» в правой части экрана.

11) В появившемся диалоговом окне необходимо ввести название базы данных «ctog\_db».

12) После указания всех необходимых параметров необходимо нажать на кнопку «Save» для создания базы данных.

После выполнения данных действий на ЭВМ будет установлена и настроена рабочая СУБД с необходимой базой данных.

**6.1.4** Загрузка веб-сервиса

Для загрузки веб-сервиса необходимо выполнить пункты:

1) Необходимо открыть веб-браузер и перейти на страницу https://gtihub.com/madopew/ctog.

2) Необходимо найти кнопку «Code» и нажать на нее.

3) В раскрывшемся меню нужно выбрать «Download ZIP», чтобы скачать весь репозиторий в виде ZIP-архива.

4) Далее необходимо выбрать путь для сохранения архива на ЭВМ. Для выбора пути следует нажать на кнопку «Save».

5) Необходимо распаковать архив в нужную директорию на ЭВМ.

**6.1.5** Установка серверной части веб-сервиса

Для установки и запуска серверной части необходимо выполнить следующие пункты:

1) Необходимо перейти в каталог с распакованным проектом.

2) Далее необходимо открыть файл application.properties по пути backend\src\main\resources.

3) Следующим шагом необходимо установить значения имени пользователя и пароля из пункта 6.1.3 в поля spring.datasource.username и spring.datasource.password, соответственно.

4) Далее необходимо открыть командную строку и перейти в backend.

5) После перехода в нужный каталог нужно выполнить команду «mvn clean install» для сборки проекта и создания исполняемого файла.

6) После сборки проекта можно запустить его, выполнив команду «java -jar target/<название исполняемого файла>.jar», где <название исполняемого файла> – это имя файла, который был создан при сборке проекта.

В случае успешного запуска приложения необходимо перейти к установке клиентской части веб-сервиса.

**6.1.6** Установка клиентской части веб-сервиса

Для установки и запуска клиентской части веб-сервиса необходимо выполнить следующие пункты:

1) Необходимо перейти в каталог с распакованным проектом.

2) Далее необходимо командую строку и перейти в каталог frontend.

3) После перехода в нужный каталог нужно выполнить команду «npm install», для установки всех библиотек.

4) После установки всех библиотек необходимо запустить сервер разработки при помощи команды «npm start».

В случае успешного запуска приложение необходимо открыть веб-браузер и перейти по адресу «http://localhost:4200», на котором будет отображена основная страница веб-сервиса.

6.2 Руководство по использованию

После успешной установки и запуска веб-сервиса в веб-браузере будет открыта страница авторизации проекта.

**6.2.1** Регистрация и авторизация

На рисунке 6.3 представлена форма регистрации и авторизации, которая будет отображена на основной странице веб-сервиса.

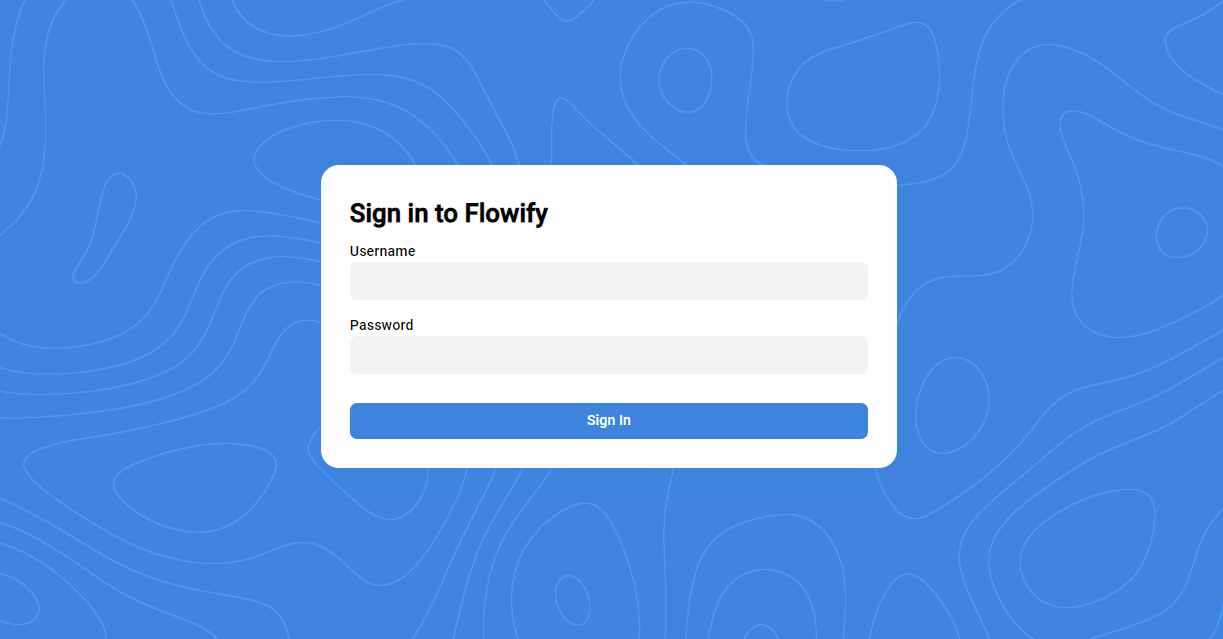


Рисунок 6.3 – Форма регистрации и авторизации

В данном веб-сервисе регистрация и авторизация выполнены на одной форме. В случае отсутствия информации об аккаунте при первичной авторизации в БД будет создана новая запись пользователя. Для этого необходимо ввести соответствующие значения в поля username и password. На рисунке 6.4 представлена заполненная форма.

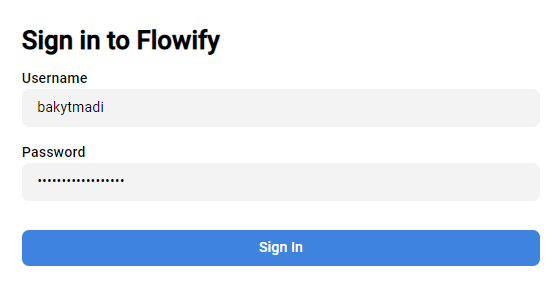


Рисунок 6.4 – Пример заполнения формы

После заполнения формы регистрации и авторизации необходимо нажать кнопку «Sign in» для продолжения работы. В случае возникновения ошибки, информация по ней будет представлена над полем username. Рисунок 6.5 показывает пример ошибки валидации в случае ввода некорректного имени пользователя.

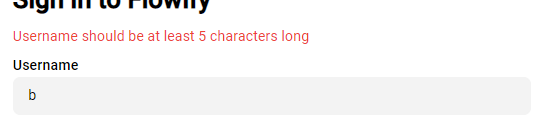


Рисунок 6.5 – Пример ошибки валидации

В таком случае необходимо исправить соответствующие значения полей и повторить процесс авторизации. После успешной авторизации откроется страница с редактором кода.

**6.2.2** Генерация схемы алгоритма

На рисунке 6.6 представлена страница с редактором кода, которая открывается сразу после процесса авторизации.

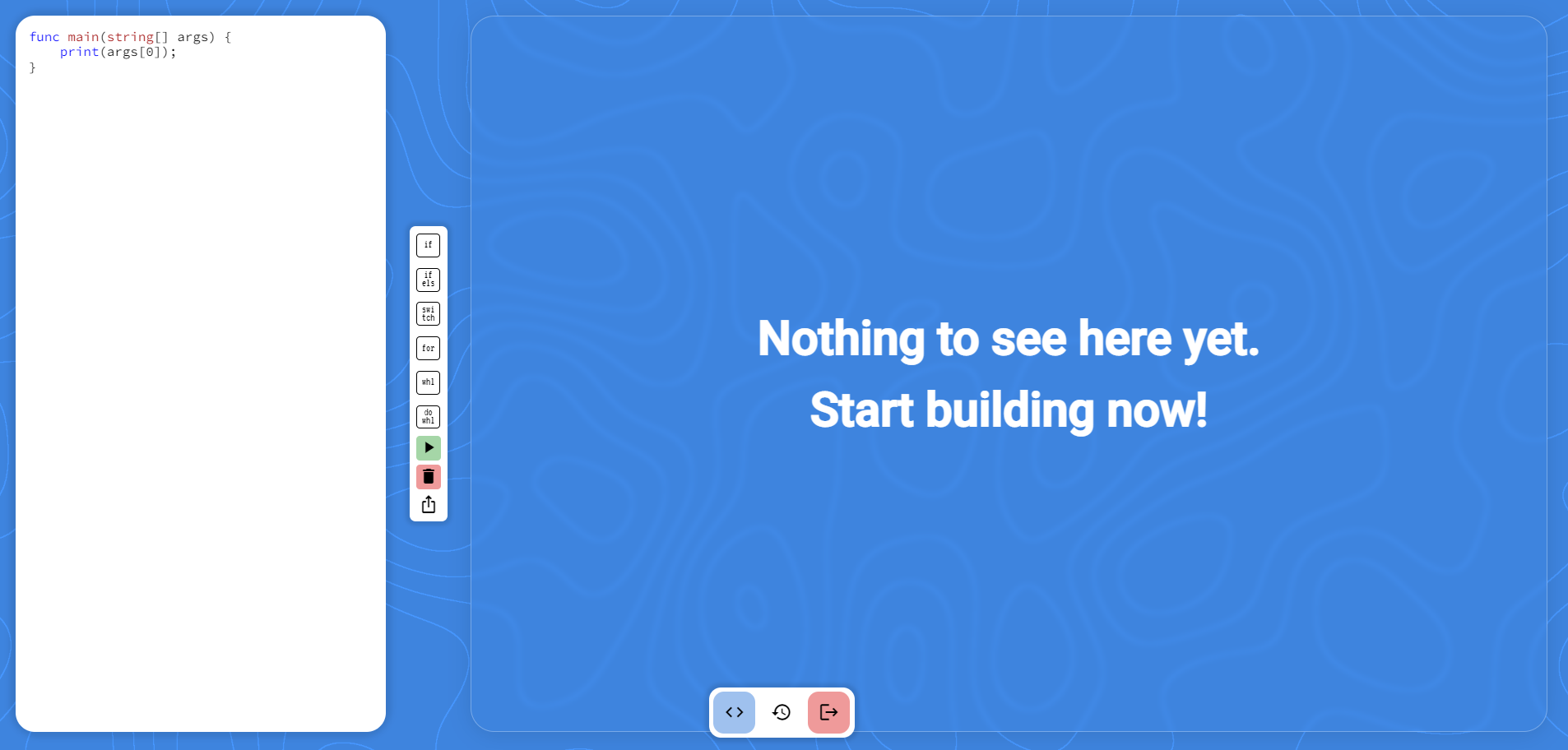


Рисунок 6.6 – Страница с редактором кода

На данной странице присутствуют следующие элементы:

– редактор кода;

– меню инструментов;

– редактор схемы алгоритма;

– меню страниц.

Для генерации схемы алгоритма необходимо ввести исходный код. Для этого нужно изменить значение редактора кода посредством ввода или вставки того фрагмента, для которого необходимо сгенерировать схему алгоритма. На рисунке 6.7 представлен пример кода, описывающий рекурсивный алгоритм нахождения значения факториала числа.

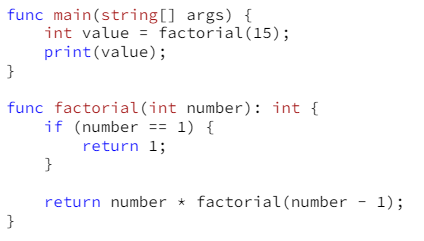


Рисунок 6.7 – Пример кода

Далее для генерации схемы, а также для удобства работы с редактором кода необходимо воспользоваться меню инструментов. На рисунке 6.8 представлено меню инструментов страницы.



Рисунок 6.8 – Меню инструментов

В меню инструментов присутствуют 9 кнопок:

1) «if»;

2) «if else»;

3) «switch»;

4) «for»;

5) «while»;

6) «do while»;

7) «build»;

8) «clear»;

9) «export».

Первые шесть кнопок являются кнопками вставки операторов языка. Для удобства пользования редактором кода, а также ускорения написания исходного кода, можно воспользоваться этими кнопками. По нажатию они описывают соответствующий их названиям оператор языка в месте расположения курсора в редакторе кода.

Кнопка «build» производит генерацию схемы алгоритма.

Кнопка «clear» очищает содержимое редакторов кода и схемы алгоритма.

Кнопка «export» сохраняет результат генерации в виде изображения.

Для генерации схемы алгоритма необходимо нажать зеленую кнопку. На рисунке 6.9 представлена схема, сгенерированная из примера на рисунке 6.7.

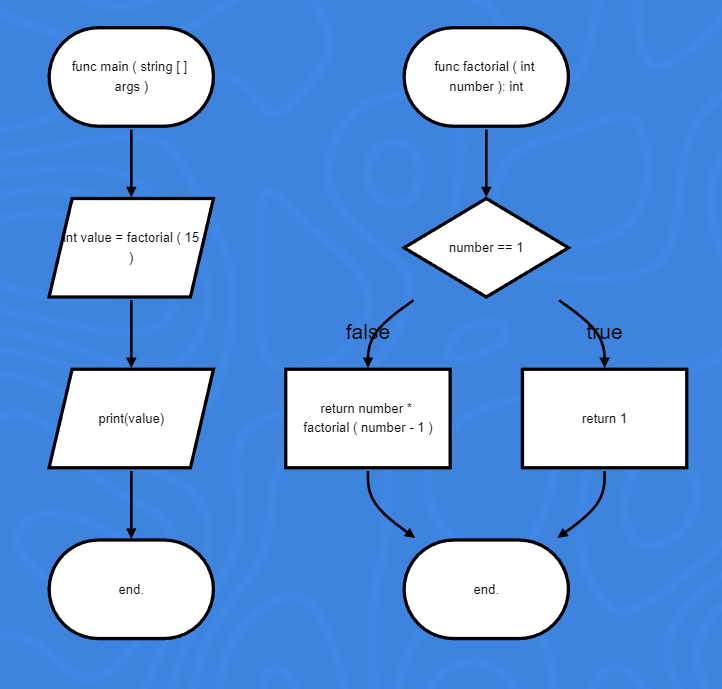


Рисунок 6.9 – Пример схемы алгоритма

Для редактирования блоков схемы необходимо нажать и удерживать левой кнопкой мыши на желаемый блок и переместить его. Структура схемы обновится автоматически.

Для изменения масштаба схемы необходимо воспользоваться колесом прокрутки мыши.

**6.2.3** Просмотр истории

Для просмотра истории генерации схем алгоритмов в меню страниц необходимо нажать на центральную кнопку. На рисунке 6.10 представлено меню страниц после перехода на страницу истории.

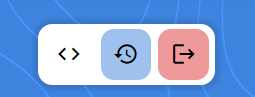


Рисунок 6.10 – Меню страниц

После перехода на страницу истории на экране отобразится таблица со всеми выполненными запросами. На рисунке 6.11 показана данная таблица с двумя генерациями.



Рисунок 6.11 – Таблица истории запросов

Данная таблица содержит следующие колонки:

– время выполнения запроса;

– исходный код;

– краткая информация о схеме;

– переход к редактору.

Помимо этого, в таблице присутствует информация о номере запроса и их количестве, а также кнопки переключения страниц.

Колонка времени показывает время выполнения и обработки запроса генерации.

Колонка исходного кода показывает сокращенный код, из которого была сгенерирована схема.

Колонка краткой информации о схеме показывает количество функций и общее количество блоков, содержащихся в схеме.

Колонка действий содержит кнопку просмотра элемента истории.

В случае, при котором общее количество запросов превышает 5 элементов, кнопки переключения страниц таблицы становятся активными. При помощи них можно просмотреть более старые запросы.

При нажатии кнопки действия будет выполнен переход со страницы истории на страницу с редактором кода, а все необходимые данные будут заполнены соответствующей информацией об исходном коде и схеме алгоритма. Благодаря этой функции можно перейти к редактированию старого запроса и изменить его вид.

**6.2.4** Выход из аккаунта

Для выхода из аккаунта необходимо нажать красную кнопку в меню страниц. После произведения этого действия отобразится основная страница с формой регистрации и авторизации, а доступ к редактору кода истории станет невозможным. Данные по запросам сохраняются вместе с аккаунтом в БД, однако необходимо помнить, что в случае выхода из аккаунта по время редактирования исходного кода значение редактора кода и схемы алгоритма не сохранятся и весь прогресс будет утерян.

7 Экономическое обоснование разработки веб-сервиса генерации схемы алгоритма из исходного кода

7.1 Характеристика разработанного по индивидуальному заказу веб-сервиса

Целью разработки данного веб-сервиса является создание инструмента для визуализации алгоритмов, который способствует облегчению процесса обучения и повышению эффективности работы программистов. Веб-сервис может использоваться в различных областях, где требуется быстрое и точное создание схем алгоритмов.

Цель разработки веб-сервиса достигается благодаря спроектированному языку программирования, способствующий не только улучшению качества кода, а также облегчения его читаемости и анализа. Построенная схема-алгоритма позволяет визуально продемонстрировать алгоритм, тем самым улучшая процесс обучения, особенно для людей, не имевших опыт программирования на каких-либо существующих языках программирования.

Заказчиком выступает организация, выполнившая индивидуальный заказ на разработку веб-сервиса.

7.2 Расчет затрат на разработку и цена веб-сервиса, созданного по индивидуальному заказу

Разработку веб-сервиса будет осуществлять команда из 4 человек: руководитель проекта, главный программист, дизайнер и тестировщик. Расчет основной заработной платы участников команды разработчиков осуществляется по формуле 7.1:

(7.1)

где *n* – количество исполнителей;

Кпр – коэффициент премий (1,65);

Зч*i* – часовая заработная плата *i*-го исполнителя, р.;

*ti* – трудоемкость работ, выполняемых *i*-м исполнителем, ч.

Данные по заработной плате команды разработчиков предоставлены организацией на 23 марта 2023 г.

Таблица 7.1 ‒ Расчет затрат на заработную плату команды разработчиков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Категория  исполнителя | Месячный оклад, р. | Часовой оклад, р. | Трудоем-кость  работ, ч | Итого, р. |
| Руководитель проекта | 3200 | 19,05 | 50 | 952,38 |
| Главный программист | 2800 | 16,67 | 168 | 2800 |
| Дизайнер | 2600 | 15,48 | 64 | 990,48 |
| Тестировщик | 1400 | 8,33 | 112 | 933,33 |
| Итого | | | | 5676,19 |
| Премия (65%) | | | | 3689,52 |
| Всего затрат на основную заработную плату исполнителей | | | | 9365,71 |

Дополнительная заработная плата (Зд) определяется по формуле:

(7.2)

где Зо – основная заработная плата, руб.;

Нд – норматив дополнительной заработной платы (15%).

Воспользовавшись формулой 7.2, получим затраты на дополнительную заработную плату:

Отчисления на социальные нужды (в фонд социальной защиты населения и на обязательное страхование) определяются в соответствии с действующим законодательством по формуле:

(7.3)

где Нсоц – норматив отчислений от фонда оплаты труда (34,6%).

Прочие расходы определяются по следующей формуле:

(7.4)

где Нпр – норматив прочих расходов (35%).

Тогда, если подставить полученные ранее значения в формулу 7.4, получим:

Общая сумма затрат на разработку вычисляется по формуле:

(7.5)

Следовательно, подставив значения в формулу 7.5, общая сумма затрат равна:

Плановая прибыль, включаемая в цену веб-сервиса, определяется по следующей формуле:

(7.6)

где Рв.с – рентабельность затрат на разработку веб-сервиса (30%).

Итого получается:

Отпускная цена веб-сервиса высчитывается, как:

(7.7)

Подставив значения в формулу 7.7, отпускная цена веб-серсива получается:

Налог на добавленную стоимость при таком подходе к отпускному ценообразованию можно посчитать по формуле:

(7.8)

где Ндс – ставка налога на добавленную стоимость, определяется действующим законодательством и равняется 20%.

Тогда, согласно формуле 7.8, налог на добавленную стоимость будет равен:

Таблица 7.2 ‒ Расчет затрат на разработку веб-сервиса

|  |  |
| --- | --- |
| Наименование статьи затрат | Сумма, р. |
| Основная заработная плата исполнителей | 9365,71 |
| Дополнительная заработная плата исполнителей | 1404,86 |
| Отчисления на социальные нужды | 3726,62 |
| Прочие расходы | 3278 |
| Общая сумма затрат на разработку | 17775,19 |
| Плановая прибыль, включаемая в цену веб-сервиса | 5332,56 |
| Отпускная цена веб-сервиса | 23107,75 |

7.3 Расчет результата от разработки веб-сервиса, созданного по индивидуальному заказу

Экономический эффект от разработки веб-сервиса по индивидуальному заказу представляет собой прирост чистой прибыли, полученной от разработки и реализации заказчику. Веб-сервис будет реализовываться по отпускной цене, следовательно, прирост чистой прибыли от разработки определяется по следующей формуле:

(7.9)

где Нп – ставка налога на прибыль (определяется законодательством, и равняется 20%).

Таким образом прирост чистой прибыли, согласно формуле 7.9, равняется:

7.4 Расчет показателей экономической эффективности разработки веб-сервиса, созданного по индивидуальному заказу

Экономическая эффективность данного веб-сервиса рассчитывается с помощью формулы простой нормы прибыли (рентабельности инвестиций), по формуле:

(7.10)

Исходя из формулы 7.10, рентабельность инвестиций получается:

По результатам расчетов затраты на разработку составили 17775,19 р., прирост чистой прибыли от разработки веб-сервиса – 4266,09 р., при этом рентабельность разработки равна 24%.

Заключение

Основной целью данного дипломного проекта являлась разработка качественного и надежного веб-сервиса для генерации схемы алгоритма.

В процессе анализа был проведен обзор существующих литературных источников, связанных с тематикой дипломного проекта. Были проанализированы научные статьи, книги, электронные ресурсы и другие материалы, которые позволили установить актуальность и значимость исследуемой проблемы, а также проведен анализ существующих аналогов и определены их достоинства и недостатки. На основе результатов анализа источников и аналогов была сформулирована и поставлена задача по проектированию веб-сервиса.

В последствии постановки задачи были сформированы функциональные требования к веб-сервису, которые должны были быть учтены при его разработке, а также разработаны модели веб-сервиса, включающие в себя диаграммы вариантов использования, схему функционирования веб-сервиса и инфологическую модель базы данных.

Согласно функциональным требованиям, был проведен процесс проектирования веб-сервиса, в процессе которого были разработаны архитектура веб-сервиса и его функциональных компонентов, даталогическая модель базы данных и интерфейс пользователя.

В процессе создания, на основе разработанных ранее моделей, был реализован веб-сервис. Для этого были использованы различные программные инструменты, библиотеки, фреймворки и технологии для создания функционального веб-сервиса, покрывающего весь набор разработанных требований.

По окончанию реализации веб-сервиса, были сформулированы и проведены тестовые сценарии, которые включали проверку на соответствие требованиям и корректность работы функциональных возможностей.

На завершающем этапе было составлено подробное руководство по установке и использованию веб-сервиса, которое позволяет быстро и надежно привести его к рабочему состоянию и освоить работу с программным продуктом.

Помимо этого, также была рассмотрена экономическая сторона разработки веб-сервиса. Были произведены расчеты различных экономических показателей разработки программного продукта, а также целесообразность его реализации.

Необходимо отметить, что данный веб-сервис имеет большой потенциал для дальнейшего развития. Помимо добавления дополнительной функциональности к редактированию схемы алгоритма, можно реализовать генерацию схемы в соответствии с различными стандартами предприятий, позволяющую использовать веб-сервис не только в демонстрационных целях, а также для генерации схем к дипломным проектам.

Список использованных источников

[1] Kramarski, B., Michalsky, P. Towards a Visual Learning Environment for Learning Algorithms // Journal of Educational Technology & Society. – 2017. – С. 146-160.

[2] Цветкова О. А., Баранов А. А. Проблемы обучения программированию: анализ, поиск решений // Информатизация образования и науки – 2017 – С. 84-92

[3] Aho, A. V., Sethi, R., Ullman, J. D. Compilers: principles, techniques, and tools. – 1-е изд. – Addison-Wesley, 1986. – 1009 с.

[4] Grune, D., Jacobs, C. J. H. Parsing Techniques: A Practical Guide. – Springer, 2008. – 648 с.

[5] Susan L., Robert M., Michael A. Automatic generation of efficient   
parsers. – ACM Press, 1971. – 268 с.

[6] Semantics in Programming Languages by Glynn Winskel [Электронный ресурс]. – Режим доступа: https://www.cs.cmu.edu/~fp/courses/15312-f04/handouts/winskel-book-ch4.pdf. – Дата доступа: 01.05.2023.

[7] Python [Электронный ресурс]. – Режим доступа: https://docs.python.org/3/. – Дата доступа: 01.05.2023.

[8] The Top Programming Languages 2022 by TIOBE Software [Электронный ресурс]. – Режим доступа: https://www.tiobe.com/tiobe-index/.

[9] Stefik, A., Siebert, S. Syntax Design for Reliable Language Processing. – Morgan & Claypool Publishers, 2012. – 174 с.

[10] Code2Flow [Электронный ресурс]. – Режим доступа: https://code2flow.com/. – Дата доступа: 01.05.2023.

[11] Draw.io [Электронный ресурс]. – Режим доступа: https://draw.io/. – Дата доступа: 01.05.2023.

[12] Parkes, Stephen, Craig R. Ramsay, Alan Spark. Code Rocket: Improving Detailed Design Support in Mainstream Software Development // International Conference on Computer and Management. – 2011: – С. 1-4.

[13] C [Электронный ресурс]. – Режим доступа: https://devdocs.io/c/ – Дата доступа: 01.05.2023.

[14] Booch, G., Rumbaugh, J., Jacobson, I. The Unified Modeling Language User Guide. – Addison-Wesley Professional, 1998 – 300 с.

[15] What is a web service by IBM [Электронный ресурс]. – Режим доступа: https://www.ibm.com/cloud/learn/web-services. – Дата доступа: 01.05.2023.

[16] Client-Server Architecture [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/client-server-architecture/. – Дата доступа: 01.05.2023.

[17] Spring [Электронный ресурс]. – Режим доступа: https://docs.spring.io/. – Дата доступа: 01.05.2023.

[18] Angular [Электронный ресурс]. – Режим доступа:   
https://angular.io/. – Дата доступа: 01.05.2023.

[19] Antlr [Электронный ресурс]. – Режим доступа:   
https://www.antlr.org/. – Дата доступа: 01.05.2023.

[20] Jeffrey D., John E. Formal Languages and Automata Theory. –   
3-е изд. – Addison-Weasley, 2006. – 550 с.

[21] Gamma E., Helm R., Johnson R., Vlissides J. Design Patterns: Elements of Reusable Object-Oriented Software – Addison-Wesley, 1994 – 395 с.

[22] Graph Data Structures [Электронный ресурс]. – Режим доступа: https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/. – Дата доступа: 01.05.2023.

[23] JSON Web Tokens [Электронный ресурс]. – Режим доступа: https://auth0.com/docs/tokens/json-web-tokens. – Дата доступа: 01.05.2023.

[24] Data Types [Электронный ресурс]. – Режим доступа: https://www.postgresql.org/docs/current/datatype.html. – Дата доступа: 01.05.2023.

[25] Spring Security [Электронный ресурс]. – Режим доступа: https://docs.spring.io/spring-security. – Дата доступа: 01.05.2023.

[26] Liquibase [Электронный ресурс]. – Режим доступа: https://docs.liquibase.com/home.html. – Дата доступа: 01.05.2023.

[27] Тестирование программного обеспечения: учеб. пособие / С. С. Куликов [и др.]. – Минск: БГУИР, 2019. – 276 с.: ил.

[28] Куликов, С. С. Тестирование программного обеспечения. Базовый курс / С. С. Куликов. – EPAM Systems, RD Dep., 2017. – http://svyatoslav.biz/software\_testing\_book/. – Дата доступа: 01.05.2023.

[29] Экономическое обоснование проекта по разработке программного обеспечения: Методическое пособие для студентов всех специальностей БГУИР. / Горовой В.Г. и др. – Минск: БГУИР, 2018 г. – 12 с.

[30] Методические указания по дипломному проектированию для студентов специальности 1-40 01 01 «Программное обеспечение информационных технологий» (ПОИТ). / Бранцевич П.Ю., Бахтизин В.В. – Минск: БГУИР, 2018. – 82 с.

[31] Глухова Л. А., Фадеева Е. П., Фадеева Е. Е. Основы алгоритмизации и программирования: Лаб. практикум для студ. спец. I-40 01 01 «Программное обеспечение информационных технологий» дневной формы обучения. В 4 ч. Ч.3. – Минск: БГУИР, 2007. – 51 с.

[32] Куликов, С. C. Работа с MySQL, MS SQL Server и Oracle в примерах: практ. пособие. / С. С. Куликов. – Минск: БОФФ, 2016. – 556 с.

Приложение А

**Исходный код веб-сервиса**

@ControllerAdvice

class RestExceptionConfig {

@ExceptionHandler(MethodArgumentNotValidException::class)

fun handlerExceptionResolver(ex: MethodArgumentNotValidException, response: HttpServletResponse) {

response.handleException(ValidationException(ex))

}

}

@Configuration

class SecurityConfig {

@Bean

fun passwordEncoder(): PasswordEncoder = BCryptPasswordEncoder()

@Bean

fun filterChain(

http: HttpSecurity,

authenticationProvider: AuthenticationProvider,

jwtFilter: JwtFilter,

exceptionFilter: ExceptionFilter,

corsFilter: CorsFilter

): SecurityFilterChain {

return http.csrf().disable()

.authorizeHttpRequests()

.antMatchers(HttpMethod.OPTIONS).permitAll()

.antMatchers("$API\_AUTH/login").permitAll()

.antMatchers("$API\_AUTH/me").authenticated()

.antMatchers("$API\_GRAPH/\*\*").hasAnyAuthority(ROLE\_DEFAULT, ROLE\_ADMIN)

.antMatchers("$API\_ADMIN/\*\*").hasAuthority(ROLE\_ADMIN)

.anyRequest().authenticated()

.and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

.and()

.authenticationProvider(authenticationProvider)

.addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter::class.java)

.addFilterBefore(exceptionFilter, WebAsyncManagerIntegrationFilter::class.java)

.addFilterBefore(corsFilter, ExceptionFilter::class.java)

.exceptionHandling()

.authenticationEntryPoint { \_, \_, \_ -> throw UnauthenticatedException() }

.accessDeniedHandler { \_, \_, \_ -> throw ForbiddenException() }

.and()

.build()

}

@Bean

fun authenticationProvider(

userDetailsService: UserDetailsService,

passwordEncoder: PasswordEncoder

): AuthenticationProvider {

val provider = DaoAuthenticationProvider()

provider.setUserDetailsService(userDetailsService)

provider.setPasswordEncoder(passwordEncoder)

return provider

}

@Bean

fun authenticationManager(config: AuthenticationConfiguration): AuthenticationManager {

return config.authenticationManager

}

@Bean

fun jwtSecretKey(@Value("\${custom.security.key.secret}") secretKey: String): Key {

return Keys.hmacShaKeyFor(Decoders.BASE64.decode(secretKey))

}

@Bean

fun jwtParser(key: Key): JwtParser =

Jwts.parserBuilder().setSigningKey(key).build()

}

const val API\_AUTH = "/api/v1/auth"

const val API\_GRAPH = "/api/v1/graph"

const val API\_ADMIN = "/api/v1/admin"

const val ROLE\_ADMIN = "ADMIN"

const val ROLE\_DEFAULT = "DEFAULT"

const val FRONTEND\_URL = "http://localhost:4200"

@RestController

@RequestMapping(API\_ADMIN)

class AdminController(

private val userService: UserService

) {

@GetMapping("/users")

fun getAllUsers() = userService.getAll()

}

@RestController

@RequestMapping(API\_AUTH)

class AuthController(

private val authService: AuthService

) {

@PostMapping("/login")

fun login(@Valid @RequestBody request: AuthDto): LoginResponseDto {

return authService.login(request)

}

@GetMapping("/me")

fun me() {

}

}

@RestController

@RequestMapping(API\_GRAPH)

class GraphController(

private val graphService: GraphService

) {

@PostMapping

fun getGraph(@RequestBody(required = false) source: String?): List<GraphDto> {

val user = SecurityContextHolder.getContext().authentication.principal as UserDetails

return graphService.build(user, source).map { it.toDto() }

}

@PostMapping("/filter")

fun filterRequests(

@PageableDefault(

sort = ["ts"],

direction = Sort.Direction.DESC

) pageRequest: Pageable

): Page<GraphRequestDto> {

val user = SecurityContextHolder.getContext().authentication.principal as UserDetails

return graphService.filter(user, pageRequest).map { it.toDto() }

}

}

@SpringBootApplication

class CtogApplication

fun main(args: Array<String>) {

runApplication<CtogApplication>(\*args)

}

data class GraphDto(

val nodes: List<GraphNodeDto>,

val edges: Map<Int, Map<Int, String?>>

)

data class GraphNodeDto(

val type: NodeType,

val text: String

)

data class GraphRequestDto(

val ts: Instant,

val input: String,

val output: List<GraphDto>

)

class AuthDto {

@NotNull

@Size(min = 5, message = "Username should be at least 5 characters long")

@Pattern(regexp = "^[a-z]+\$", message = "Username should contain only lowercase english letters")

lateinit var username: String

@NotNull

@NotEmpty

@Size(min = 7, message = "Password should be at least 7 characters long")

lateinit var password: String

}

data class LoginResponseDto(

val token: String

)

class UserDetailsDto(

private val username: String,

private val password: String,

role: String

) : UserDetails {

private val authorities = listOf(SimpleGrantedAuthority(role))

constructor(user: UserInfo) : this(

user.username,

user.passwordHash,

user.userRole.name

)

override fun getAuthorities() = authorities

override fun getPassword() = password

override fun getUsername() = username

override fun isAccountNonExpired() = true

override fun isAccountNonLocked() = true

override fun isCredentialsNonExpired() = true

override fun isEnabled() = true

}

data class HttpErrorDto(

val message: String,

val httpStatusCode: Int

)

class BadRequestException(msg: String) : CtogException(msg, 400)

open class CtogException(

override val message: String,

val httpCode: Int

): Exception()

class ForbiddenException : CtogException("You do not have permission to access this resource", 403)

class NotFoundException(msg: String) : CtogException(msg, 404)

class ParserException(message: String): CtogException(message, 400)

class ServerException : CtogException("Something went wrong", 500)

class UnauthenticatedException : CtogException("Check your credentials", 401)

class ValidationException(ex: MethodArgumentNotValidException): CtogException(

ex.allErrors.first().defaultMessage ?: "Provided parameters do not pass validation", 400

)

private val logger = LoggerFactory.getLogger("ServletExtension")

fun HttpServletResponse.handleException(e: Exception) {

logger.warn("Handling response exception", e)

if (e is CtogException) {

this.handleBusinessException(e)

} else {

this.handleBusinessException(ServerException())

}

}

private fun HttpServletResponse.handleBusinessException(e: CtogException) {

val error = e.toHttpError()

this.contentType = MediaType.APPLICATION\_JSON\_VALUE

this.status = error.httpStatusCode

jacksonObjectMapper().writeValue(this.outputStream, error)

this.outputStream.flush()

}

class GraphBuilder(

private val config: GraphConfiguration = GraphConfiguration.DEFAULT

) {

fun build(program: CodeProgram): List<Graph> {

val functionNames = program.functions.map { it.name }

val isLocal = { name: String ->

functionNames.any { it == name }

}

return program.functions.map { BuildGraphVisitor(config, isLocal).build(it) }

}

private class BuildGraphVisitor(

val config: GraphConfiguration,

val isLocal: (String) -> Boolean

) {

var cycleDepth = 1

val nodes = mutableListOf<GraphNode>()

val edges = mutableMapOf<Int, MutableMap<Int, String?>>()

var functionContext: GraphNode? = null

val breakContext: ArrayDeque<GraphNode> = ArrayDeque()

val continueContext: ArrayDeque<GraphNode> = ArrayDeque()

fun build(function: CodeFunction): Graph {

visitCodeFunction(function)

return Graph(nodes, edges)

}

fun addEdge(from: GraphNode, to: GraphNode, label: String?) {

if (!edges.containsKey(from.id)) edges[from.id] = mutableMapOf()

edges[from.id]!![to.id] = label

}

fun visitCodeFunction(function: CodeFunction) {

val startNode = GraphNode(type = NodeType.START\_END, text = function.definition)

val endNode = GraphNode(type = NodeType.START\_END, text = config.endKeyword)

functionContext = endNode

nodes.add(startNode)

addEdge(startNode, visitStatements(function.statements, endNode), null)

nodes.add(endNode)

}

fun visitStatements(statements: List<CodeStatement>, last: GraphNode): GraphNode {

if (statements.isEmpty()) return last

if (statements.size == 1) return visitStatement(statements.first(), last)

return visitStatement(statements.first(), visitStatements(statements.drop(1), last))

}

fun visitStatement(statement: CodeStatement, last: GraphNode): GraphNode = when (statement) {

is CodeCall -> visitCodeCall(statement, last)

is CodeExpression -> visitCodeExpression(statement, last)

is CodeIfSelection -> visitCodeIfSelection(statement, last)

is CodeIteration -> visitCodeIteration(statement, last)

is CodeSwitchSelection -> visitCodeSwitchSelection(statement, last)

else -> throw IllegalArgumentException("Unsupported statement type: ${statement.javaClass.name}")

}

fun visitCodeCall(statement: CodeCall, last: GraphNode): GraphNode {

val type = if (isLocal(statement.functionName)) NodeType.LOCAL\_ACTION

else if (config.isInputFunction(statement.functionName)) NodeType.INPUT

else if (config.isOutputFunction(statement.functionName)) NodeType.OUTPUT

else NodeType.ACTION

val node = GraphNode(type = type, text = statement.toString())

nodes.add(node)

addEdge(node, last, null)

return node

}

fun visitCodeExpression(statement: CodeExpression, last: GraphNode): GraphNode {

val type = when (statement.type) {

ExpressionType.DECLARATION -> NodeType.INPUT

else -> NodeType.ACTION

}

val node = GraphNode(type = type, text = statement.body)

nodes.add(node)

when (statement.type) {

ExpressionType.RETURN -> {

addEdge(node, functionContext!!, null)

}

ExpressionType.BREAK -> {

if (breakContext.isNotEmpty()) {

addEdge(node, breakContext.last(), null)

} else {

addEdge(node, last, null)

}

}

ExpressionType.CONTINUE -> {

if (continueContext.isNotEmpty()) {

addEdge(node, continueContext.last(), null)

} else {

addEdge(node, last, null)

}

}

else -> {

addEdge(node, last, null)

}

}

return node

}

fun visitCodeIfSelection(statement: CodeIfSelection, last: GraphNode): GraphNode {

val node = GraphNode(type = NodeType.CONDITION, text = statement.condition)

nodes.add(node)

addEdge(node, visitStatements(statement.ifBody, last), config.trueKeyword)

addEdge(node, visitStatements(statement.elseBody, last), config.falseKeyword)

return node

}

fun visitCodeIteration(statement: CodeIteration, last: GraphNode): GraphNode {

val depthText = "${config.depthKeyword}${cycleDepth++}"

val startNode = if (statement.type == IterationType.PRE\_CONDITION) {

GraphNode(type = NodeType.CYCLE\_START, text = "$depthText\n${statement.condition}")

} else {

GraphNode(type = NodeType.CYCLE\_START, text = depthText)

}

val endNode = if (statement.type == IterationType.PRE\_CONDITION) {

GraphNode(type = NodeType.CYCLE\_END, text = depthText)

} else {

GraphNode(type = NodeType.CYCLE\_END, text = "$depthText\n${statement.condition}")

}

breakContext.addLast(last)

continueContext.addLast(endNode)

nodes.add(startNode)

addEdge(startNode, visitStatements(statement.body, endNode), null)

addEdge(endNode, last, null)

cycleDepth--

nodes.add(endNode)

breakContext.removeLast()

continueContext.removeLast()

return startNode

}

fun visitCodeSwitchSelection(statement: CodeSwitchSelection, last: GraphNode): GraphNode {

val startNode = GraphNode(type = NodeType.CONDITION, text = statement.condition)

nodes.add(startNode)

statement.cases.forEach { (case, body) ->

addEdge(startNode, visitStatements(body, last), case)

}

return startNode

}

}

}

fun Graph.toDto(): GraphDto {

val nodes = this.nodes.map { it.toDto() }

val edges = mutableMapOf<Int, MutableMap<Int, String?>>()

this.edges.forEach { (fromId, toList) ->

val fromNode = this.nodes.single { it.id == fromId }

val fromIndex = this.nodes.indexOf(fromNode)

edges[fromIndex] = mutableMapOf()

toList.forEach { (toId, text) ->

val toNode = this.nodes.single { it.id == toId }

val toIndex = this.nodes.indexOf(toNode)

edges[fromIndex]!![toIndex] = text

}

}

return GraphDto(nodes, edges)

}

fun GraphNode.toDto(): GraphNodeDto =

GraphNodeDto(this.type, this.text)

fun GraphRequest.toDto() = GraphRequestDto(

ts,

input,

jacksonObjectMapper().readValue(output, object : TypeReference<List<Graph>>() {}).map { it.toDto() }

)

fun CtogException.toHttpError() = HttpErrorDto(this.message, this.httpCode)

data class Graph(

val nodes: List<GraphNode>,

val edges: Map<Int, Map<Int, String?>>

)

data class GraphConfiguration(

val endKeyword: String,

val trueKeyword: String,

val falseKeyword: String,

val depthKeyword: String,

val inputFunctions: List<String>,

val outputFunctions: List<String>

) {

fun isInputFunction(function: String) = inputFunctions.contains(function)

fun isOutputFunction(function: String) = outputFunctions.contains(function)

companion object {

val DEFAULT = GraphConfiguration(

endKeyword = "end.",

trueKeyword = "true",

falseKeyword = "false",

depthKeyword = "A",

inputFunctions = listOf("input", "read"),

outputFunctions = listOf("output", "print")

)

}

}

data class GraphNode(

val id: Int = Random.nextInt(),

val type: NodeType,

val text: String

)

@Entity

class GraphRequest {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

var id = 0

@Column(nullable = false)

lateinit var input: String

@Column(nullable = false)

lateinit var output: String

@Column(nullable = false)

var ts: Instant = Instant.now()

@ManyToOne

@JoinColumn(name = "user\_info\_id", nullable = false)

lateinit var userInfo: UserInfo

}

enum class NodeType {

START\_END,

CYCLE\_START,

CYCLE\_END,

ACTION,

LOCAL\_ACTION,

CONDITION,

INPUT,

OUTPUT

}

@Entity

class UserInfo {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

var id = 0

@Column(nullable = false)

lateinit var username: String

@Column(nullable = false)

lateinit var passwordHash: String

@ManyToOne

@JoinColumn(name = "user\_role\_id", nullable = false, insertable = false, updatable = false)

@LazyCollection(LazyCollectionOption.FALSE)

lateinit var userRole: UserRole

@OneToMany(mappedBy = "userInfo", cascade = [CascadeType.ALL])

lateinit var requests: MutableList<GraphRequest>

}

@Entity

class UserRole {

@Id

private var id = 0

@Column(nullable = false)

lateinit var name: String

@OneToMany(mappedBy = "userRole")

private lateinit var users: Set<UserInfo>

}

internal class BuildCodeVisitor {

fun visitProgramNode(node: ProgramNode): CodeProgram {

return CodeProgram(

functions = node.functions.map { visitFunctionNode(it) }

)

}

private fun visitFunctionNode(node: FunctionNode): CodeFunction {

return CodeFunction(

name = node.name!!,

definition = node.definition!!,

statements = node.body!!.statements.flatMap { visitStatementNode(it) }

)

}

private fun visitStatementNode(node: StatementNode): List<CodeStatement> = when (node) {

is CompoundStatementNode -> node.statements.flatMap { visitStatementNode(it) }

is DeclarationStatementNode -> listOf(visitDeclarationStatementNode(node))

is FunctionCallStatementNode -> listOf(visitFunctionCallStatementNode(node))

is IfStatementNode -> listOf(visitIfStatementNode(node))

is IterationStatementNode -> listOf(visitIterationStatementNode(node))

is JumpStatementNode -> listOf(visitJumpStatementNode(node))

is LabeledStatementNode -> listOf()

is NoOpStatementNode -> listOf()

is OtherExpressionStatementNode -> listOf(visitOtherExpressionStatementNode(node))

is SwitchStatementNode -> listOf(visitSwitchStatementNode(node))

else -> throw IllegalArgumentException("Cannot visit $node")

}

private fun visitDeclarationStatementNode(node: DeclarationStatementNode) =

CodeExpression(ExpressionType.DECLARATION, node.name!!)

private fun visitFunctionCallStatementNode(node: FunctionCallStatementNode) =

CodeCall(node.name!!, node.arguments ?: "")

private fun visitIfStatementNode(node: IfStatementNode) =

CodeIfSelection(

node.condition!!,

visitStatementNode(node.ifBody!!),

if (node.elseBody != null) visitStatementNode(node.elseBody!!) else listOf()

)

private fun visitIterationStatementNode(node: IterationStatementNode): CodeIteration {

val type = when (node.type!!) {

IterationStatementNodeType.FOR -> IterationType.PRE\_CONDITION

IterationStatementNodeType.WHILE -> IterationType.PRE\_CONDITION

IterationStatementNodeType.DO\_WHILE -> IterationType.POST\_CONDITION

}

return CodeIteration(type, node.condition!!, visitStatementNode(node.body!!))

}

private fun visitJumpStatementNode(node: JumpStatementNode) =

CodeExpression(

when (node.type) {

JumpStatementNodeType.RETURN -> ExpressionType.RETURN

JumpStatementNodeType.BREAK -> ExpressionType.BREAK

JumpStatementNodeType.CONTINUE -> ExpressionType.CONTINUE

else -> ExpressionType.OTHER

},

node.toReadableString()

)

private fun visitOtherExpressionStatementNode(node: OtherExpressionStatementNode) =

CodeExpression(type = ExpressionType.OTHER, body = node.body!!)

private fun visitSwitchStatementNode(node: SwitchStatementNode) =

CodeSwitchSelection(

node.condition!!,

node.body.associate { labeledStatement ->

val label = labeledStatement.label ?: "default"

val statements = labeledStatement.body!!.statements.flatMap { visitStatementNode(it) }

label to statements

}

)

private fun JumpStatementNode.toReadableString(): String {

return if (label != null) "$type $label" else type.toString()

}

}

data class CodeCall(

val functionName: String,

val argumentList: String

) : CodeStatement {

override fun toString(): String {

return "$functionName($argumentList)"

}

}

data class CodeExpression(

val type: ExpressionType,

val body: String

) : CodeStatement

data class CodeFunction(

val name: String,

val definition: String,

val statements: List<CodeStatement>

)

data class CodeIfSelection(

val condition: String,

val ifBody: List<CodeStatement>,

val elseBody: List<CodeStatement>

) : CodeStatement

data class CodeIteration(

val type: IterationType,

val condition: String,

val body: List<CodeStatement>

) : CodeStatement

data class CodeProgram(

val functions: List<CodeFunction>

)

interface CodeStatement {

}

data class CodeSwitchSelection(

val condition: String,

val cases: Map<String, List<CodeStatement>>

) : CodeStatement

enum class ExpressionType {

DECLARATION,

RETURN,

BREAK,

CONTINUE,

OTHER

}

enum class IterationType {

PRE\_CONDITION,

POST\_CONDITION,

}

internal class BuildAstVisitor(

private val tokens: List<String>

) : CBaseVisitor<CNode?>() {

override fun visitCompilationUnit(ctx: CParser.CompilationUnitContext): ProgramNode {

return if (ctx.translationUnit() != null) {

visitTranslationUnit(ctx.translationUnit())

} else {

ProgramNode()

}

}

override fun visitTranslationUnit(ctx: CParser.TranslationUnitContext): ProgramNode {

return ProgramNode().apply {

functions = ctx.functionDefinition()

.map { visitFunctionDefinition(it) }

.toMutableList()

}

}

override fun visitFunctionDefinition(ctx: CParser.FunctionDefinitionContext): FunctionNode {

val functionDefinition = buildString {

append(ctx.Function().text)

append(' ')

append(ctx.declarator().fullText)

if (ctx.declarationSpecifiers() != null) {

append(ctx.Colon().text)

append(' ')

append(ctx.declarationSpecifiers().fullText)

}

}

return FunctionNode().apply {

name = ctx.declarator().directDeclarator().directDeclarator().Identifier().text

definition = functionDefinition

body = visitCompoundStatement(ctx.compoundStatement())

}

}

override fun visitCompoundStatement(ctx: CParser.CompoundStatementContext): CompoundStatementNode {

return if (ctx.blockItemList() != null) {

visitBlockItemList(ctx.blockItemList())

} else {

CompoundStatementNode()

}

}

override fun visitBlockItemList(ctx: CParser.BlockItemListContext): CompoundStatementNode {

return CompoundStatementNode().apply {

statements = ctx.blockItem()

.mapNotNull { visitBlockItem(it) }

.toMutableList()

}

}

override fun visitBlockItem(ctx: CParser.BlockItemContext): StatementNode {

return super.visitBlockItem(ctx) as StatementNode

}

override fun visitStatement(ctx: CParser.StatementContext): StatementNode {

return super.visitStatement(ctx) as StatementNode

}

override fun visitDeclaration(ctx: CParser.DeclarationContext): DeclarationStatementNode {

return DeclarationStatementNode().apply {

name = buildString {

append(ctx.declarationSpecifiers().fullText)

if (ctx.initDeclaratorList() != null) {

append(' ')

append(ctx.initDeclaratorList().fullText)

}

}

}

}

override fun visitLabeledStatement(ctx: CParser.LabeledStatementContext): LabeledStatementNode {

return LabeledStatementNode().apply {

body = visitCompoundStatement(ctx.compoundStatement())

if (ctx.Case() != null) {

type = LabeledStatementNodeType.CASE

label = ctx.constantExpression().fullText

} else if (ctx.Default() != null) {

type = LabeledStatementNodeType.DEFAULT

}

}

}

override fun visitSelectionStatement(ctx: CParser.SelectionStatementContext): SelectionStatementNode {

return if (ctx.If() != null) {

IfStatementNode().apply {

condition = ctx.expression().fullText

ifBody = visitCompoundStatement(ctx.compoundStatement(0))

if (ctx.compoundStatement(1) != null) elseBody = visitCompoundStatement(ctx.compoundStatement(1))

}

} else {

SwitchStatementNode().apply {

condition = ctx.expression().fullText

body = ctx.labeledStatement()

.map { visitLabeledStatement(it) }

.toMutableList()

}

}

}

override fun visitIterationStatement(ctx: CParser.IterationStatementContext): IterationStatementNode {

val iterationType = if (ctx.Do() != null) IterationStatementNodeType.DO\_WHILE

else if (ctx.For() != null) IterationStatementNodeType.FOR

else IterationStatementNodeType.WHILE

return IterationStatementNode().apply {

type = iterationType

condition = if (ctx.forCondition() != null) ctx.forCondition().fullText else ctx.expression().fullText

body = visitCompoundStatement(ctx.compoundStatement())

}

}

override fun visitJumpStatement(ctx: CParser.JumpStatementContext): JumpStatementNode {

return JumpStatementNode().apply {

if (ctx.Continue() != null) {

type = JumpStatementNodeType.CONTINUE

} else if (ctx.Break() != null) {

type = JumpStatementNodeType.BREAK

} else {

type = JumpStatementNodeType.RETURN

if (ctx.expression() != null) label = ctx.expression().fullText

}

}

}

override fun visitExpressionStatement(ctx: CParser.ExpressionStatementContext): ExpressionStatementNode {

return if (ctx.expression() == null) {

NoOpStatementNode()

} else {

visitExpression(ctx.expression())

}

}

override fun visitExpression(ctx: CParser.ExpressionContext): ExpressionStatementNode {

return if (ctx.assignmentExpression().size > 1) {

OtherExpressionStatementNode().apply {

body = ctx.fullText

}

} else {

visitAssignmentExpression(ctx.assignmentExpression(0))

}

}

override fun visitAssignmentExpression(ctx: CParser.AssignmentExpressionContext): ExpressionStatementNode {

return if (ctx.conditionalExpression() != null) {

visitConditionalExpression(ctx.conditionalExpression()) as FunctionCallStatementNode?

?: OtherExpressionStatementNode().apply {

body = ctx.fullText

}

} else {

OtherExpressionStatementNode().apply {

body = ctx.fullText

}

}

}

override fun visitPostfixExpression(ctx: CParser.PostfixExpressionContext): FunctionCallStatementNode? {

val isFunctionCall = ctx.children.filterIsInstance<TerminalNode>().singleOrNull { it.text == "(" } != null

return if (isFunctionCall) {

FunctionCallStatementNode().apply {

name = ctx.primaryExpression().fullText

if (ctx.argumentExpressionList(0) != null) arguments = ctx.argumentExpressionList(0).fullText

}

} else {

null

}

}

private val ParserRuleContext.fullText: String

get() = tokens.subList(start.tokenIndex, stop.tokenIndex + 1).joinToString(" ")

}

internal open class CNode {

}

internal class CompoundStatementNode: StatementNode() {

var statements = mutableListOf<StatementNode>()

}

internal class DeclarationStatementNode: StatementNode() {

var name: String? = null

}

internal open class ExpressionStatementNode: StatementNode() {

}

internal class FunctionCallStatementNode: ExpressionStatementNode() {

var name: String? = null

var arguments: String? = null

}

internal class FunctionNode: CNode() {

var name: String? = null

var definition: String? = null

var body: CompoundStatementNode? = null

}

internal class IfStatementNode : SelectionStatementNode() {

var ifBody: CompoundStatementNode? = null

var elseBody: CompoundStatementNode? = null

}

internal class IterationStatementNode: StatementNode() {

var type: IterationStatementNodeType? = null

var condition: String? = null

var body: CompoundStatementNode? = null

}

enum class IterationStatementNodeType {

FOR,

WHILE,

DO\_WHILE

}

internal class JumpStatementNode: StatementNode() {

var type: JumpStatementNodeType? = null

var label: String? = null

}

internal enum class JumpStatementNodeType {

CONTINUE,

BREAK,

RETURN;

override fun toString(): String = when (this) {

CONTINUE -> "continue"

BREAK -> "break"

RETURN -> "return"

}

}

internal class LabeledStatementNode: StatementNode() {

var type: LabeledStatementNodeType? = null

var label: String? = null

var body: CompoundStatementNode? = null

}

enum class LabeledStatementNodeType {

CASE,

DEFAULT

}

internal class NoOpStatementNode: ExpressionStatementNode() {

}

internal class OtherExpressionStatementNode: ExpressionStatementNode() {

var body: String? = null

}

internal class ProgramNode: CNode() {

var functions = mutableListOf<FunctionNode>()

}

internal open class SelectionStatementNode: StatementNode() {

var condition: String? = null

}

internal open class StatementNode: CNode() {

}

internal class SwitchStatementNode : SelectionStatementNode() {

var body = mutableListOf<LabeledStatementNode>()

}

interface GraphRepository : JpaRepository<GraphRequest, Int>, JpaSpecificationExecutor<GraphRequest> {

fun findAllByUserInfo(userInfo: UserInfo, pageable: Pageable): Page<GraphRequest>

}

interface UserRepository: JpaRepository<UserInfo, Int>, JpaSpecificationExecutor<UserInfo> {

fun findByUsername(username: String): UserInfo?

}

@Service

@Transactional

class GraphService(

private val userService: UserService,

private val graphRepository: GraphRepository

) {

fun build(user: UserDetails, input: String?): List<Graph> {

val userInfo = userService.findByUsername(user.username)

val result = build(input)

val request = GraphRequest().apply {

this.input = input ?: ""

this.output = jacksonObjectMapper().writeValueAsString(result)

this.userInfo = userInfo

}

userInfo.requests.add(request)

userService.save(userInfo)

return result

}

fun filter(user: UserDetails, pageRequest: Pageable): Page<GraphRequest> {

val userInfo = userService.findByUsername(user.username)

return graphRepository.findAllByUserInfo(userInfo, pageRequest)

}

private fun build(input: String?): List<Graph> {

if (input == null) return emptyList()

val lexer = CLexer(CharStreams.fromString(input))

val tokens = lexer.allTokens.map { it.text }

lexer.reset()

val parser = CParser(CommonTokenStream(lexer)).apply {

addErrorListener(object : BaseErrorListener() {

override fun syntaxError(

recognizer: Recognizer<\*, \*>?,

offendingSymbol: Any?,

line: Int,

charPositionInLine: Int,

msg: String?,

e: RecognitionException?

) {

throw ParserException("Syntax error at line $line position $charPositionInLine")

}

})

}

val tree = parser.compilationUnit()

val astVisitor = BuildAstVisitor(tokens)

val programNode = astVisitor.visitCompilationUnit(tree)

val codeVisitor = BuildCodeVisitor()

val codeProgram = codeVisitor.visitProgramNode(programNode)

val graphBuilder = GraphBuilder()

return graphBuilder.build(codeProgram)

}

}

@Component

@Transactional

class UserDetailsService(

private val userService: UserService

) : UserDetailsService {

override fun loadUserByUsername(username: String): UserDetails {

return UserDetailsDto(userService.findByUsername(username))

}

}

@Service

@Transactional

class UserService(

private val repo: UserRepository

) {

fun save(user: UserInfo): UserInfo {

return repo.save(user)

}

fun getByUsername(username: String) = repo.findByUsername(username)

fun findByUsername(username: String) = getByUsername(username)

?: throw NotFoundException("User $username not found")

fun getAll(): List<UserInfo> {

return repo.findAll()

}

}

@Service

@Transactional

class AuthService(

private val userService: UserService,

private val jwtService: JwtService,

private val passwordEncoder: PasswordEncoder,

private val authenticationManager: AuthenticationManager,

private val entityManager: EntityManager

) {

fun login(request: AuthDto): LoginResponseDto {

val user = userService.getByUsername(request.username) ?: return register(request)

authenticationManager.authenticate(

UsernamePasswordAuthenticationToken(request.username, request.password)

)

return LoginResponseDto(jwtService.generateToken(UserDetailsDto(user)))

}

private fun register(request: AuthDto): LoginResponseDto {

val user = userService.save(

UserInfo().apply {

username = request.username

passwordHash = passwordEncoder.encode(request.password)

}

)

entityManager.refresh(user)

return LoginResponseDto(jwtService.generateToken(UserDetailsDto(user)))

}

}

@Component

class CorsFilter : OncePerRequestFilter() {

override fun doFilterInternal(

request: HttpServletRequest,

response: HttpServletResponse,

filterChain: FilterChain

) {

response.setHeader("Access-Control-Allow-Origin", FRONTEND\_URL)

response.setHeader("Access-Control-Allow-Methods", "GET, POST, OPTIONS")

response.setHeader("Access-Control-Allow-Headers", "Authorization, Content-Type")

filterChain.doFilter(request, response)

}

}

@Component

class ExceptionFilter : OncePerRequestFilter() {

override fun doFilterInternal(

request: HttpServletRequest,

response: HttpServletResponse,

filterChain: FilterChain

) {

try {

filterChain.doFilter(request, response)

} catch (e: NestedServletException) {

response.handleException(e.cause as Exception)

} catch (e: Exception) {

response.handleException(e)

}

}

}

@Component

class JwtFilter(

private val userDetailsService: UserDetailsService,

private val jwtService: JwtService

) : OncePerRequestFilter() {

override fun doFilterInternal(

request: HttpServletRequest,

response: HttpServletResponse,

filterChain: FilterChain

) {

val authHeader = request.getHeader("Authorization")

if (authHeader == null || !authHeader.startsWith("Bearer ")) {

filterChain.doFilter(request, response)

return

}

val jwt = authHeader.substring(7)

if (SecurityContextHolder.getContext().authentication == null) {

val username = jwtService.extractUsername(jwt)

val userDetails = this.userDetailsService.loadUserByUsername(username)

if (!jwtService.isTokenExpired(jwt)) {

val authToken = UsernamePasswordAuthenticationToken(userDetails, null, userDetails.authorities)

authToken.details = WebAuthenticationDetailsSource().buildDetails(request)

SecurityContextHolder.getContext().authentication = authToken

}

}

filterChain.doFilter(request, response)

}

}

@Service

class JwtService(

private val jwtParser: JwtParser,

private val jwtKey: Key,

@Value("\${custom.security.expire.millis}")

private val expireTimeMillis: Long

) {

fun generateToken(userDetails: UserDetails): String {

val currentTime = System.currentTimeMillis()

return Jwts

.builder()

.setClaims(mapOf("role" to userDetails.authorities.first().authority.lowercase()))

.setSubject(userDetails.username)

.setIssuedAt(Date(currentTime))

.setExpiration(Date(currentTime + expireTimeMillis))

.signWith(jwtKey, SignatureAlgorithm.HS256)

.compact()

}

fun extractUsername(jwt: String): String {

return extractClaim(jwt, Claims::getSubject)

}

fun isTokenExpired(token: String): Boolean {

return extractExpiration(token).before(Date())

}

private fun extractExpiration(jwt: String): Date {

return extractClaim(jwt, Claims::getExpiration)

}

private fun <T> extractClaim(jwt: String, resolver: (Claims) -> T): T {

val claims = extractAllClaims(jwt)

return resolver(claims)

}

private fun extractAllClaims(jwt: String): Claims {

return try {

jwtParser.parseClaimsJws(jwt).body

} catch (e: Exception) {

throw UnauthenticatedException()

}

}

}

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *Обозначение* | | | | | *Наименование* | | | | *Дополнитель-ные сведения* | | |
|  | | | | | Текстовые документы | | | |  | | |
|  | | | | |  | | | |  | | |
| БГУИР ДП 1–40 01 01 01 012 ПЗ | | | | | Пояснительная записка | | | | 87 с. | | |
|  | | | | | Отзыв руководителя | | | |  | | |
|  | | | | | Рецензия | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | | Графические документы | | | |  | | |
|  | | | | |  | | | |  | | |
| ГУИР.951006-01 СА | | | | | Генерация схемы алгоритма. | | | | Формат А1 | | |
|  | | | | | Схема алгоритма | | | |  | | |
| ГУИР.951006-02 СА | | | | | Взаимодействие с пользователем. | | | | Формат А1 | | |
|  | | | | | Схема алгоритма | | | |  | | |
| ГУИР.951006-03 СА | | | | | Обработка операторов перехода. | | | | Формат А1 | | |
|  | | | | | Схема алгоритма | | | |  | | |
| ГУИР.951006-01 ПЛ | | | | | Диаграмма использования. | | | | Формат А1 | | |
|  | | | | | Плакат | | | |  | | |
| ГУИР.951006-02 ПЛ | | | | | Диаграмма классов AST. | | | | Формат А1 | | |
|  | | | | | Плакат | | | |  | | |
| ГУИР.951006-03 ПЛ | | | | | Пример работы веб-сервиса. Оператор for. | | | | Формат А1 | | |
|  | | | | | Плакат | | | |  | | |
| ГУИР.951006-04 ПЛ | | | | | Пример работы веб-сервиса. Оператор switch. | | | | Формат А1 | | |
|  | | | | | Плакат | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  | | | | |  | | | |  | | |
|  |  |  |  |  | БГУИР ДП 1-40 01 01 01 012 Д1 | | | | | | |
|  |  |  |  |  |
|  |  |  |  |  |
| Изм | Лист | № докум. | Подп. | Дата | Веб-сервис генерации  схемы алгоритма  из исходного кода.  Ведомость дипломного  проекта | Лит. | | | Лист | | Листов |
| Разработал | | Бакыт М. |  |  | Т |  |  | | 87 | 87 |
| Проверил | | Петровская В. |  |  | Кафедра ПОИТ  гр. 951006 | | | | | |
| Т.контроль | |  |  |  |
| Н.контрол | | Данилова Г. |  |  |
| Утвердил | |  |  |  |