**СОДЕРЖАНИЕ**

Contents

[Введение 7](#_Toc132386793)

[1 Анализ литературных источников, аналогов и постановка задачи к проектируемому программному средству 9](#_Toc132386794)

[1.1 Анализ литературных источников 9](#_Toc132386795)

[1.2 Анализ существующих аналогов 12](#_Toc132386796)

[1.3 Постановка задачи к проектируемому программному средству 18](#_Toc132386797)

[2 Разработка моделей и формирование требований к проектируемому программному средству 19](#_Toc132386798)

[2.1 Формирование требований к проектируемому программному средству 19](#_Toc132386799)

[2.2 Диаграмма вариантов использования 22](#_Toc132386800)

[2.3 Схема работы веб-сервиса 25](#_Toc132386801)

[2.4 Инфологическая модель базы данных 26](#_Toc132386802)

[3 Проектирование программного средства 28](#_Toc132386803)

[3.1 Проектирование языка 28](#_Toc132386804)

[3.2 Анализ конструкций языка 29](#_Toc132386805)

[3.3 Авторизация и аутентификация 36](#_Toc132386806)

[3.4 Сохранение запросов в базу данных 37](#_Toc132386807)

[3.5 Даталогическая модель базы данных 38](#_Toc132386808)

[3.6 Проектирование интерфейса пользователя клиентской части 40](#_Toc132386809)

[4 Реализация 45](#_Toc132386810)

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.

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. 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 в 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.

Разработка клиентской и серверной частей веб-сервиса также представляет собой интересный вызов, который позволит изучить принципы разработки клиент-серверных 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 программному средству

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.

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.

На практике, генераторы парсеров, такие как ANTLR, 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. 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. 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, 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, 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.

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.

**1.1.3** Анализ веб-сервиса

Веб-сервис – это программное обеспечение, которое позволяет различным приложениям взаимодействовать друг с другом через сеть, используя стандартные протоколы, например HTTP. 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.

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 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 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.

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 — это веб-сервис, который предоставляет возможность автоматически генерировать блок-схемы из исходного кода на собственном языке программирования. Данный язык представляет из себя 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.

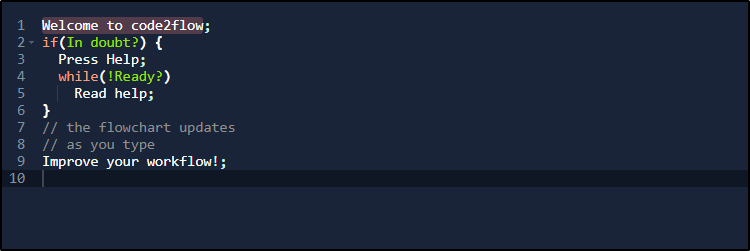


Рисунок 1.1 – Редактор кода в веб-сервисе «Code2Flow»

На рисунке 1.1 представлен пример кода, который описывается 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.

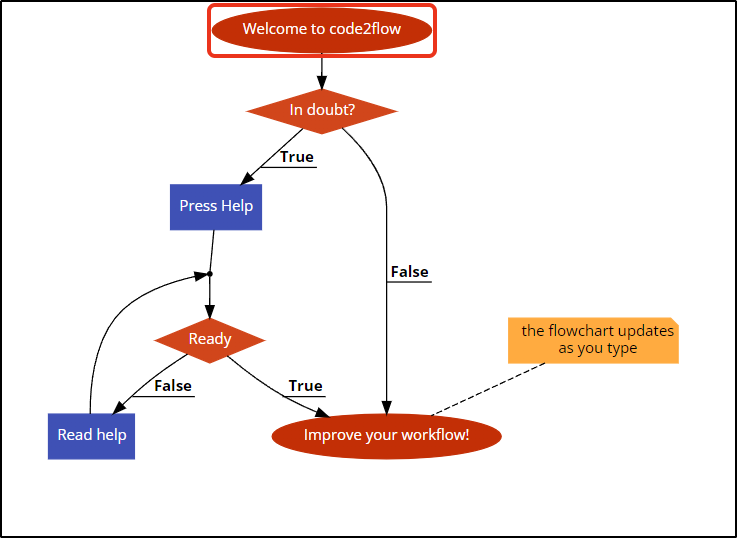


Рисунок 1.2 – Сгенерированная схема алгоритма в веб-сервисе Code2Flow

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

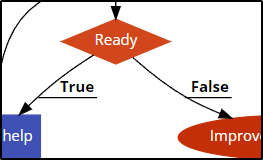


Рисунок 1.3 – Оператор инвертирования

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 из себя 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 начало и конца схемы.

**1.2.2** Веб-сервис «draw.io»

Сервис «draw.io» 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 время затратным. Также, ограниченные возможности кастомизации и оформления могут ограничивать пользовательский опыт.

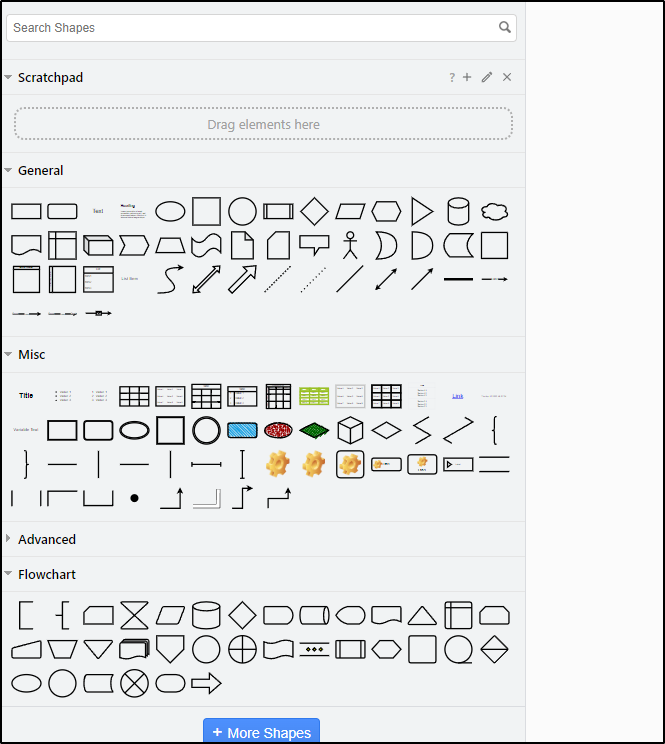


Рисунок 1.4 – Выбор форм в веб-сервисе «Draw.io»

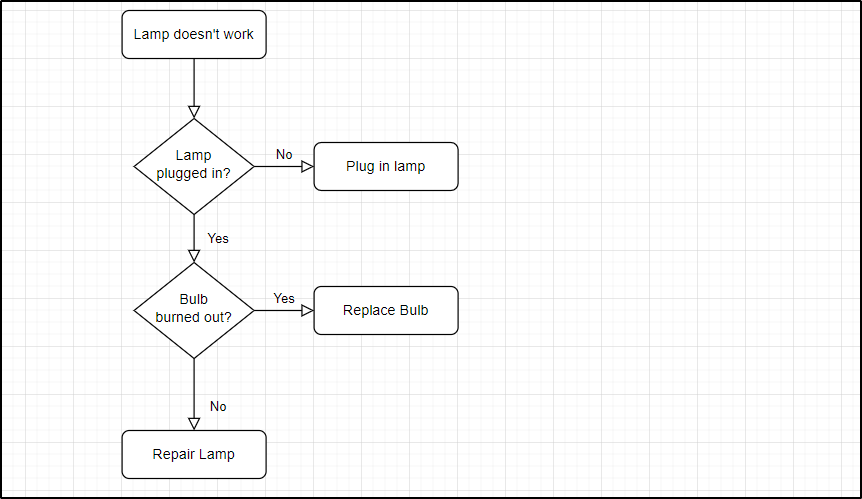


Рисунок 1.5 – Пример схемы алгоритма в сервисе «Draw.io»

Не смотря на ограниченные возможности кастомизации, 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 визуальной 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»

«Code Rocket» использовался для улучшения визуальной 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.

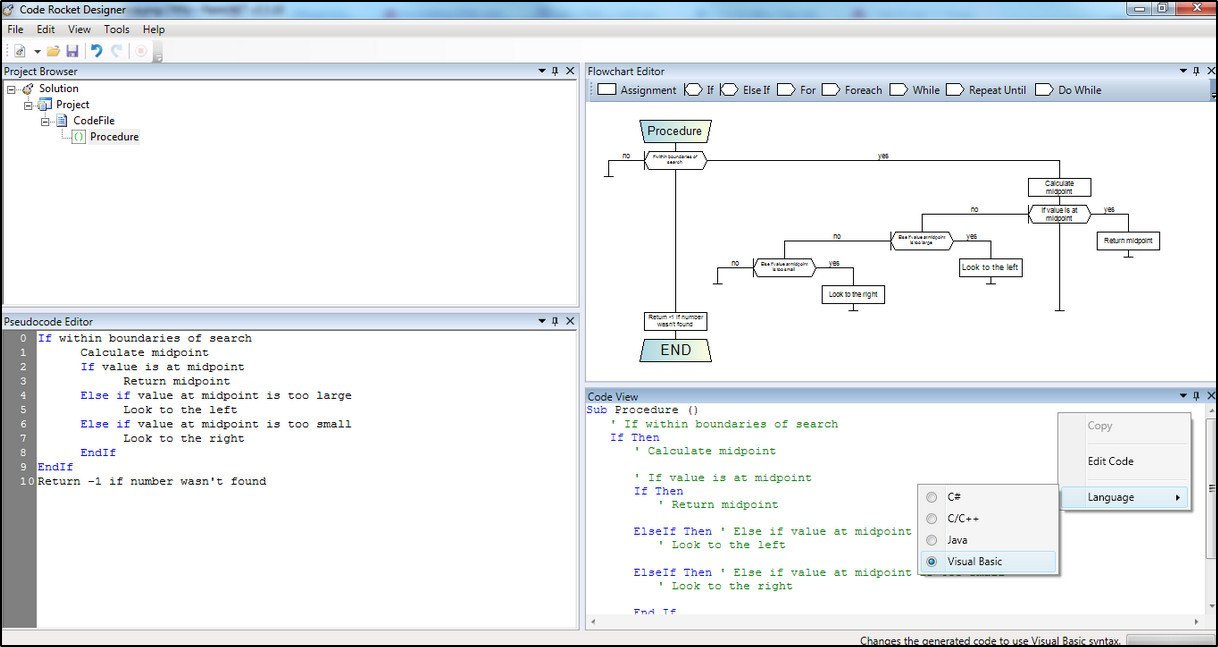


Рисунок 1.6 – Интерфейс пользователя программного средства «Code Rocket»

К плюсам данного программного обеспечения можно отнести:

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

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

– Простой и интуитивно понятный интерфейс.

Однако оно также имеет и ряд серьезных недостатков, основными которые можно выделить:

– Необходимость установки на компьютер.

– Последняя версия была выпущена в 2013 году, что говорит о том, что инструмент уже не поддерживается и не обновляется.

– Платный сервис.

– Необходимость знания конкретных языков программирования.

1.3 Постановка задачи к проектируемому программному средству

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 программное средство, которое 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. 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 необходимо использовать клиент-серверную архитектуру, где клиент будет отправлять запросы на сервер, а сервер будет обрабатывать эти запросы и возвращать результаты. Для реализации серверной части необходимо использовать фреймворк Spring Kotlin, а для клиентской части – фреймворк Angular.

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 выполнить следующие задачи:

– Разработать язык программирования для написания алгоритмов и описания структур данных.

– Разработать генератор блок-схем на основе введенного кода.

– Разработать клиент-серверную архитектуру приложения.

– Разработать анализатор кода для проверки корректности введенного кода.

– Разработать 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 программному средству

2.1 Формирование требований к проектируемому программному средству

Веб-сервис генерации схемы алгоритма из исходного кода, являясь основанная 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:

– Возможность генерации блок-схемы из исходного кода на разработанном языке.

– Возможность сохранения сгенерированных схем в различных форматах изображений.

– Возможность сохранения кода и сгенерированной схемы в базу данных.

– Возможность просмотра 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 программному средству, 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:

а) Синтаксис языка должен быть описан средствами, поддерживаемыми генератором парсеров ANTLR.

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

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

1) определение прототипа функции ключевым словом «func»;

2) обязательное использование разделяющих знаков.

3) возможность использования пользовательских типов данных.

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

1) наличие безусловных операторов перехода;

2) наличие макросов;

3) возможность объявления вложенных функций и т.д.

С точки зрения анализатора кода, помимо использования библиотеки для генерации парсеров ANTLR, также должны быть написаны алгоритмы, позволяющие закончить синтаксический анализ, подготовить дерево разбора для семантического анализа 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 сконфигурированы благодаря использованию конфигурационных классов и/или объектов.

Для того, чтобы обеспечить правильную генерацию схемы алгоритма, семантический анализ должен включать в себя:

а) Определение вызова предопределенных подпрограмм, то есть определение тех функций, что были написаны и вызваны пользовательским кодом.

б) Правильное определение порядка вызовов, для чего необходимо обрабатывать операторы break, continue и return.

в) Определение тела оператора switch и его условий.

Для корректной работы серверной части веб-сервиса, ЭВМ, на которой должно функционировать программное средство, должна обладать следующими минимальными характеристиками:

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

– Жёсткий 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 программного средства 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.

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.

В рамках диаграммы вариантов использования нарисованы овалы, называемые use case (вариант использования), которые представляют функциональные возможности системы. Пользователи обозначаются в виде фигур человека или пиктограммы. Взаимодействие между актерами и use case показываются линиями.

Исходя из выдвинутых требований, проектируемое программное средство 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.

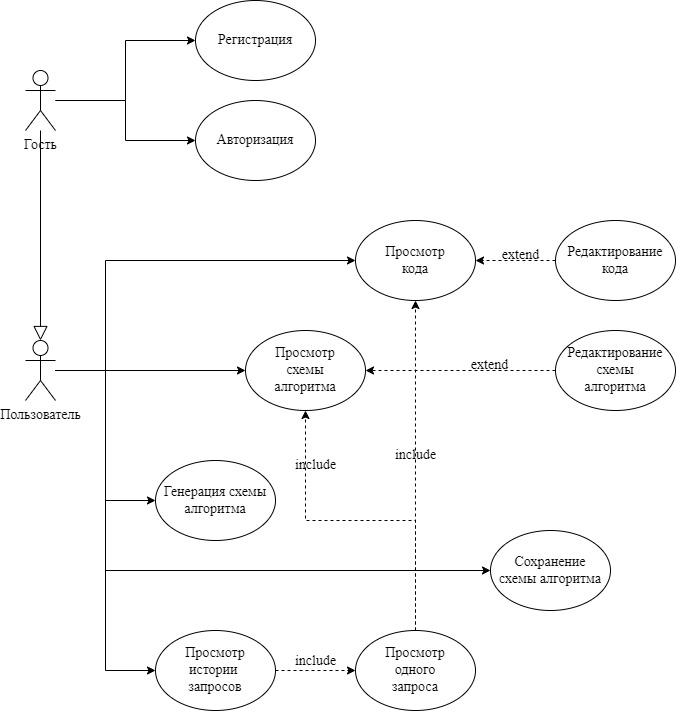


Рисунок 2.1 – Диаграмма использования

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. 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 обзорная схема этого процесса.

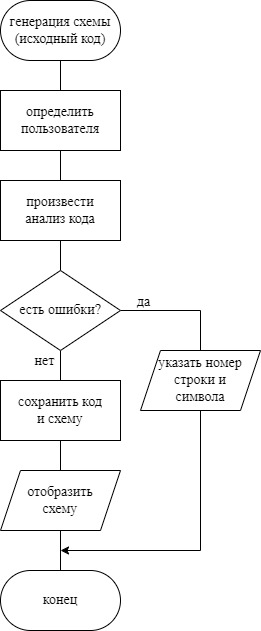


Рисунок 2.2 – Обзорная схема обработки запроса пользователя

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. Данных 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, разрабатываемое программное средство должно 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 пользователей необходима база данных. Несмотря на то, что в данном проекте для веб-сервиса не нужна сложная и большая база, разработки 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:

а) Пользователь:

1) идентификатор;

2) имя;

3) хэш пароля;

4) список запросов;

5) роль.

б) Роль:

1) идентификатор;

2) имя.

в) Запрос:

1) идентификатор;

2) время обработки запроса;

3) исходный код;

4) сгенерированная схема алгоритма.

В данной модели необходимо обратить внимание на атрибут пользователя – хэш пароля. В целях безопасности хранение пароля в неизменённом 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 программного средства

3.1 Проектирование языка

Исходя из выше определённых требований, первичный анализ исходного кода будет происходить посредством библиотеки ANTLR.

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 языке. В случае с синтаксисом языков программирования такие языки являются контекстно свободными.

В качестве основой грамматики для разрабатываемого языка была взята грамматика языка СИ, стандарта ANSI C11. Данный стандарт представляет из себя 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 и «func main(): void {…}», так и «func main() {…}». Помимо этого правила также необходимы изменения условных и циклических операторов, типов данных и так далее.

3.2 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.

**3.2.1** 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 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, абстрактно 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.

**3.2.2** Шаблон программирования «Посетитель»

Так как языки программирования являются по своей сути рекурсивными, то есть могут содержать одинаковые конструкции вложенный 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. 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 шаблонов 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 классов 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.

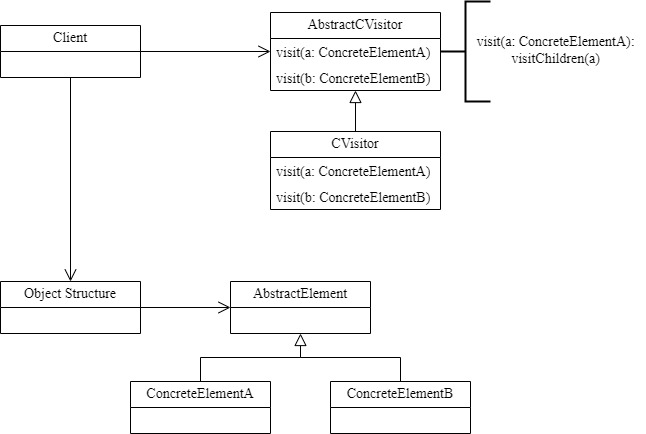


Рисунок 3.2 – Диаграмма классов шаблона «Посетитель»

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

На данном рисунке видно, что алгоритм шаблона проектирования посетитель можно условно разделить на две составляющие: имплементация самого посетителя и структура объектов. Структура объектов представляет из себя 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 и какие нет, и так далее.

Абстрактный класс посетителя содержит в себе метод на каждый из типов, наследующих структуру объектов. Каждый такой метод содержит имплементацию, включающий 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 абстрактно-синтаксического дерева в стэке программного средства. Для того, чтобы эту структуру построить явно необходимо унаследовать абстрактный класс и переопределить те методы, обработка объекта которых позволит построить AST.

**3.2.3** Анализ деревьев разбора

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

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

– ProgramNode. Представляет из себя объект всей рабочей программы. Содержит список объявленных функций FunctionNode. Наследует класс CNode.

– FunctionNode. Представляет из себя 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 себе как определение различных структур языка. Наследует класс CNode.

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

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

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

– JumpStatementNode. Представляет из себя 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 из себя 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 из себя 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 из себя законченную единицу вызова внутри программы, можно представить его как элемент кода, после которого должна ставиться точка с запятой. Наследует класс StatementNode.

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

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

– OtherExpressionStatementNode. Представляет из себя 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 из себя тип цикла. Может принимать значения FOR, WHILE, DO\_WHILE.

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

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

Не смотря на достаточно большой размер структуры объектов – 19 классов, такое количество необходимо для повышения уровня абстракции от дерева синтаксического разбора, так как его размер равен 60 классам. 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 из себя единую программу, содержащую список функций CodeFunction. Необходима только для объединения списка функций в одном объекте, не имеет никакого значения с точки зрения схемы-алгоритма.

– CodeFunction. Представляет из себя единую функции 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 предопределённой 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. Предствляет из себя единый вызов внутри программы. Содержит тип вызова и тело.

– CodeIfSelection. Представляет из себя 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 из себя 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 из себя вызов функции. Содержит имя вызываемой функции, а также список передаваемых аргументов.

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

– ExpressionType. Представляет из себя 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 из себя 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:

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

1) список GraphNode, который является списком узлов;

2) список смежности.

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

1) идентификатор, необходимый для связи;

2) тип узла (форма блока);

3) 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.

3.3 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.

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. Во-первых, он обеспечивает безопасную аутентификацию и авторизацию пользователей. Во-вторых, JWT использует формат JSON, который обеспечивает простоту чтения и распаршивания данных. В-третьих, JWT обеспечивает возможность передачи дополнительных пользовательских данных, которые могут быть полезны при реализации функциональности приложения. В-четвертых, JWT является легковесным и быстрым в использовании, что особенно важно для веб-сервисов с большой нагрузкой. Наконец большинство библиотек и фреймворков, в том числе Spring Boot, поддерживают работу с JWT, поэтому интеграция данного алгоритма не составит никаких проблем.

3.4 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.

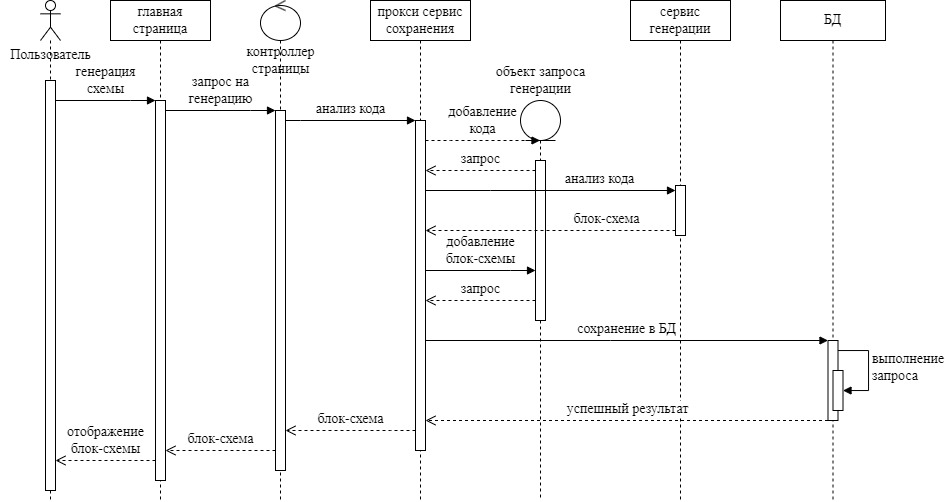


Рисунок 3.3 – Диаграмма последовательности обработки запроса

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.

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.

3.5 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.

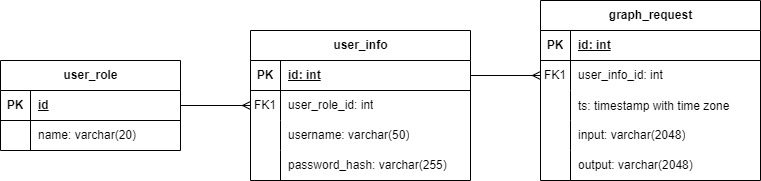


Рисунок 3.4 – Даталогическая модель

На рисунке 3.4 отображена схема даталогической модели базы данных веб-сервиса генерации схемы алгоритма из исходного кода. 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 улучшает 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. Без учета часового пояса, временная метка может быть некорректно интерпретирована.

3.6 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 поведение, как пользователя, так и программного средства. Под этим подразумевается любое ошибочное состояние веб-сервиса: неправильно введённые 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.

В данном разделе не будут 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 программного средства.

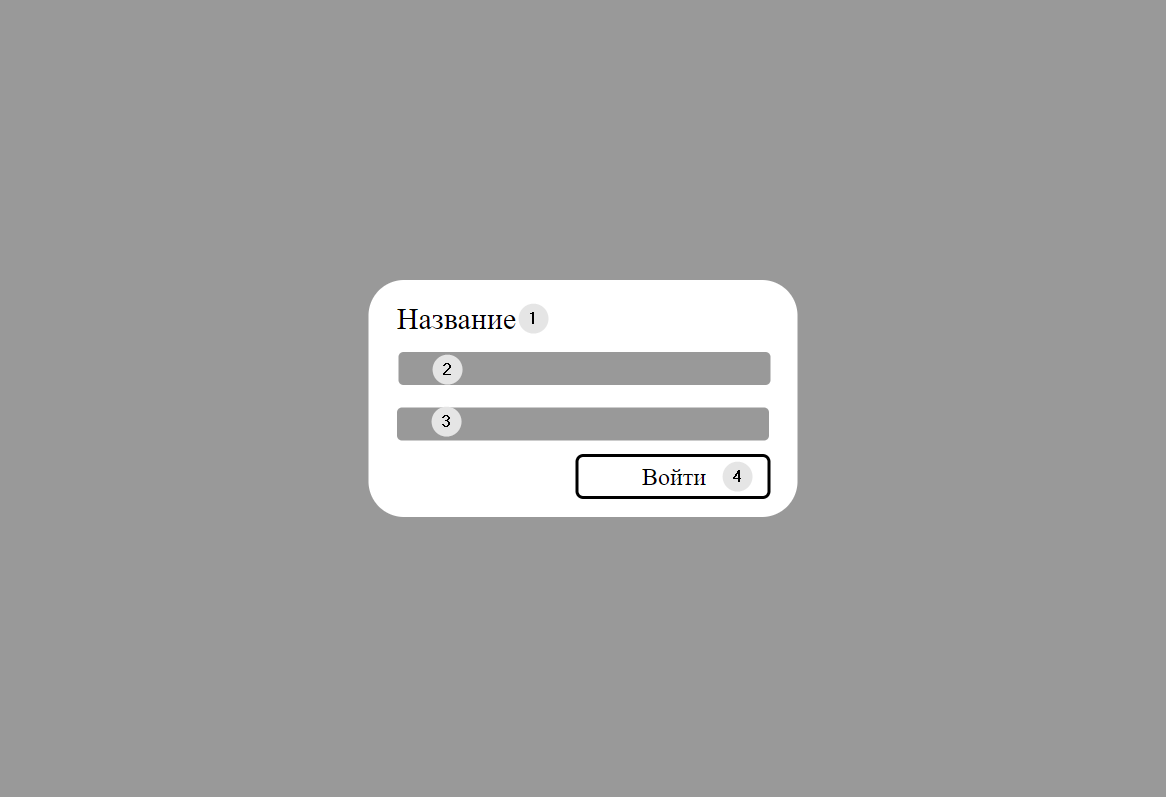


Рисунок 3.5 – Страница авторизации



Рисунок 3.6 – Сообщение об авторизационной ошибке

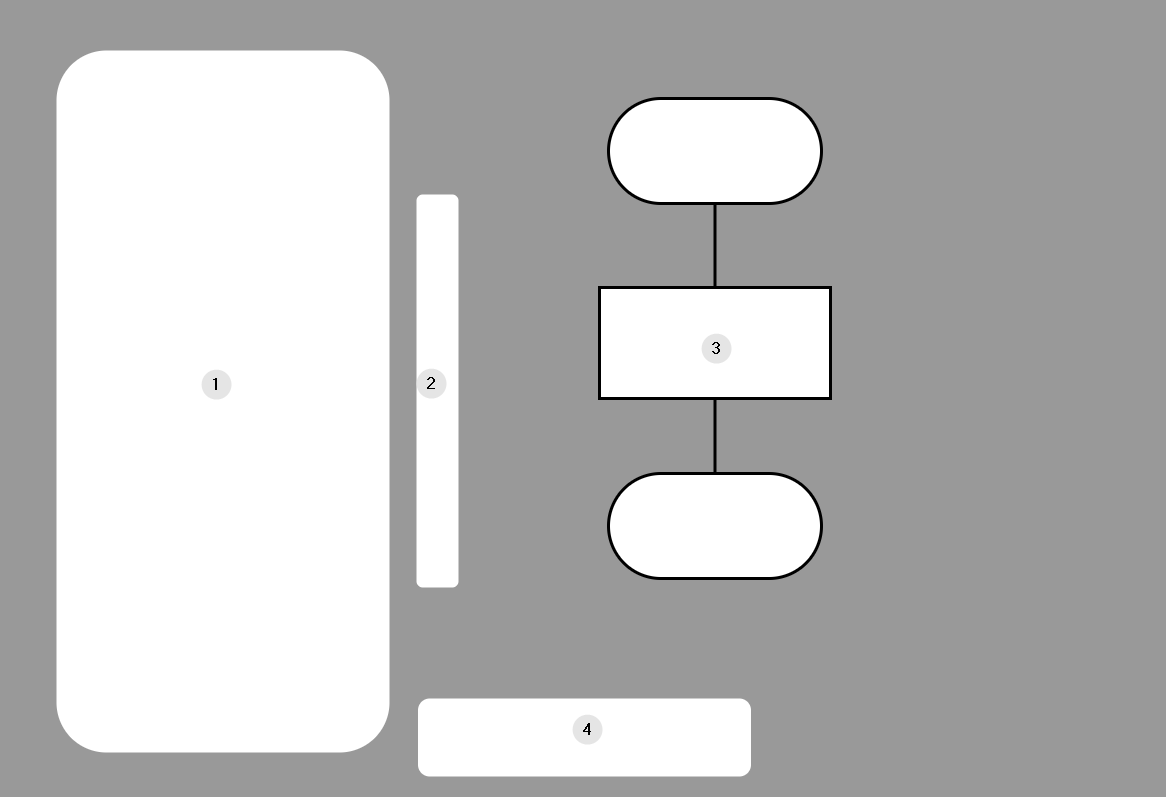


Рисунок 3.7 – Страница генерации схемы

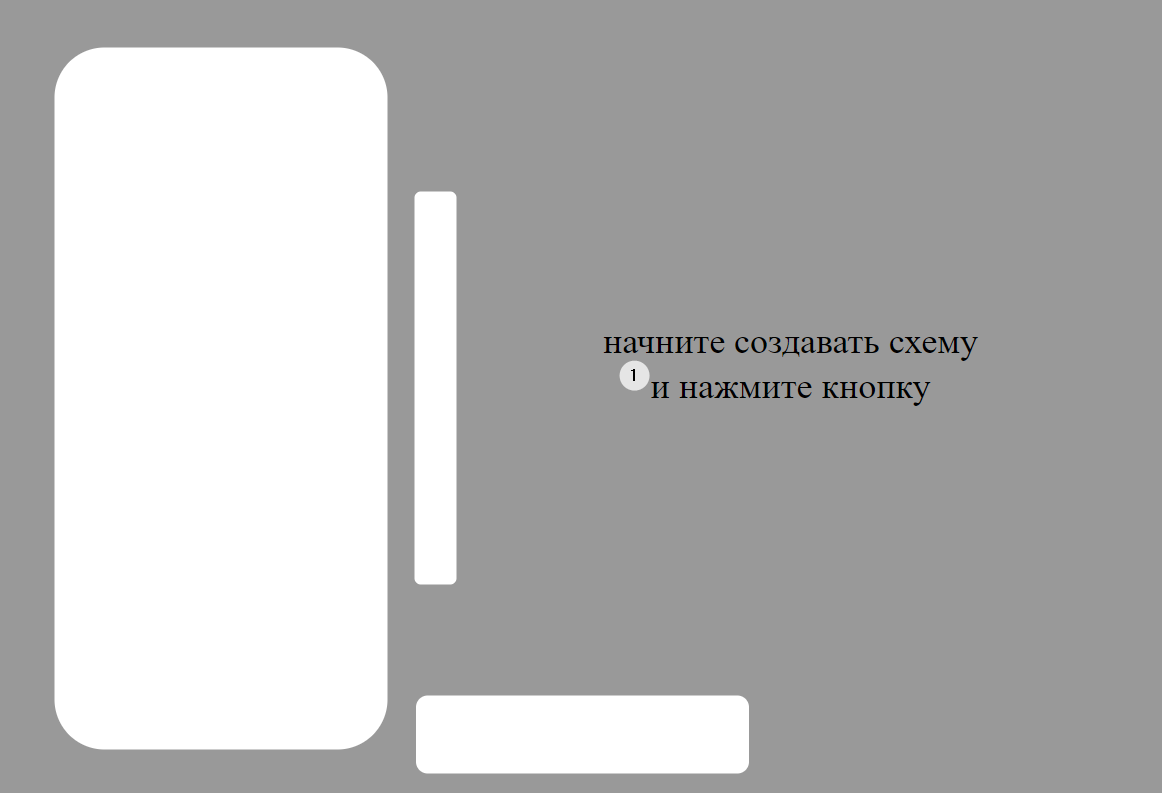


Рисунок 3.8 – Сообщение об отсутствии схемы

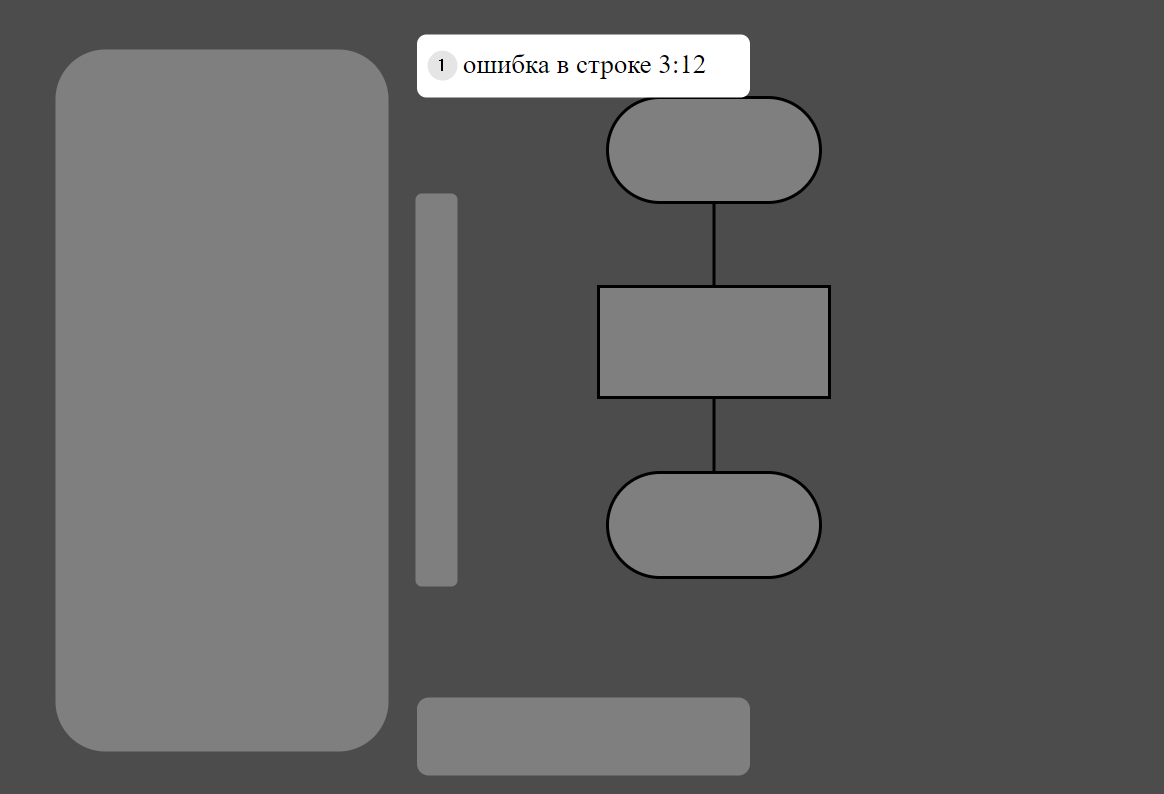


Рисунок 3.9 – Сообщение о синтаксической ошибке

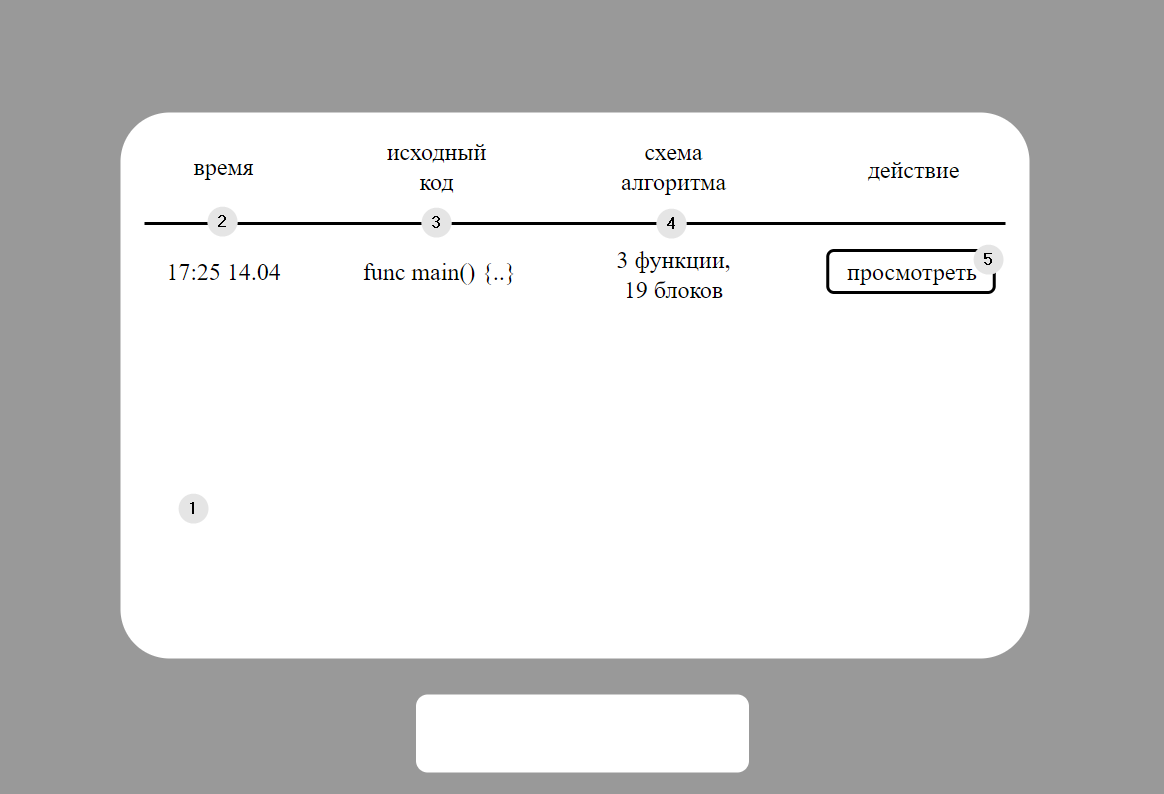


Рисунок 3.10 – Страница истории

Рисунки 3.5 и 3.6 показывают, как должна выглядеть страница авторизации. Данная страница по совместительству также является страницей регистрацией, в случае отсутствия записи о пользователе, создается новый. На рисунке 3.5 присутствуют:

1) название веб-сервиса;

2) поле для ввода имени пользователя;

3) поле для ввода пароля;

4) кнопка авторизации.

В случае возникновения ошибки авторизации необходимо отобразить сообщение, как показано в пункте 1 рисунка 3.6.

Рисунки 3.7, 3.8 и 3.9 показывают, 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 3.7 содержит:

1) поле редактора кода с подсветкой синтаксиса;

2) меню инструментов с кнопкой генерации и вставки операторов языка;

3) поле редактора схемы алгоритма;

4) меню выбора страницы.

В случае первоначально загрузки страницы генерации схемы алгоритма, необходимо отобразить надпись, подсказывающая пользователю, что необходимо инициировать процесс с помощью нажатия кнопки, как это отображено в пункте 1 рисунка 3.8. 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 3.9.

На рисунке 3.10 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. Помимо этого, данная страница включает в себя:

1) компонент таблицы;

2) столбец со временем запроса;

3) столбец с кратким содержанием исходного кода;

4) столбец с кратким описание содержания схемы алгоритма;

5) столбец с кнопкой просмотра запроса.

В случае нажатия на кнопку в пункте 5 необходимо открыть страницу генерации схемы, отображенную на рисунке 3.7, предварительно заполнив редактор кода исходным кодом из запроса, а редактор блок-схемы – схемой алгоритма из запроса.

4 Реализация