**Оглавление**

[Введение 7](#_Toc168279720)

[1. Теоретическое основание 8](#_Toc168279721)

[1.1 Актуальность разработки интегрированной среды разработки для Java 8](#_Toc168279722)

[1.2 Сравнительный анализ существующих решений и их недостатков 9](#_Toc168279723)

[1.3 Постановка целей и задач 14](#_Toc168279724)

[1.4 Концептуальное описание предметной области 16](#_Toc168279725)

[1.5 Методология исследования 18](#_Toc168279726)

[2 Средства реализации и разработки 19](#_Toc168279727)

[2.1 Язык программирования 20](#_Toc168279728)

[2.2 База данных 22](#_Toc168279729)

[2.3 Моделирование Бизнес-процессов 23](#_Toc168279730)

[2.3.1 Моделирование с использованием BPMN 24](#_Toc168279731)

[2.3.2 Определение требований 25](#_Toc168279732)

[2.3.3 Описание бизнес-процессов интегрированной среды разработки для Java 28](#_Toc168279733)

[2.3.4 Прикладное применение описания бизнес-процессов 29](#_Toc168279734)

[2.3.5 Основные инструменты описания бизнес-процессов 30](#_Toc168279735)

[2.3.6 Модель входов выходов 36](#_Toc168279736)

[2.4 Моделирование требований к интегрированной среде разработки 38](#_Toc168279737)

[2.4.1 Функциональные требования 38](#_Toc168279738)

[2.4.2 Нефункциональные требования 38](#_Toc168279739)

[2.5. Архитектурные паттерны и подходы к проектированию 39](#_Toc168279740)

[2.6. Технологии и инструменты для разработки среды 40](#_Toc168279741)

[3. Разработка технического задания 42](#_Toc168279742)

[3.1.1 Определение функциональных требований 42](#_Toc168279743)

[3.1.2 Определение нефункциональных требований 43](#_Toc168279744)

[3.2 Разработка архитектуры 50](#_Toc168279745)

[3.2.1 Выбор архитектурного стиля 50](#_Toc168279746)

[3.2.2 Моделирование компонентов системы 51](#_Toc168279747)

[3.3 Проектирование базы данных 54](#_Toc168279748)

[3.3.1 Разработка схемы базы данных 55](#_Toc168279749)

[3.3.2 Оптимизация запросов и структуры данных 60](#_Toc168279750)

[3.4 Разработка интерфейса 61](#_Toc168279751)

[3.5 Разработка логики работы 65](#_Toc168279752)

[3.5.1 Обработка пользовательских действий 65](#_Toc168279753)

[3.5.2 Управление данными 66](#_Toc168279754)

[3.5.3 Выполнение компиляции и тестирования 67](#_Toc168279755)

[3.5.4 Интеграция с внешними системами 68](#_Toc168279756)

[3.5.5 Интеграция с другими инструментами и сервисами 69](#_Toc168279757)

[3.5.6 Интеграция с системами контроля версий 70](#_Toc168279758)

[3.5.7 Интеграция с CI/CD инструментами 70](#_Toc168279759)

[3.5.8 Интеграция с облачными сервисами 71](#_Toc168279760)

[3.5.9 Интеграция с инструментами анализа и помощи в разработке 72](#_Toc168279761)

[3.5.10 Интеграция с системами управления проектами 72](#_Toc168279762)

[3.6 Тестирование и отладка 73](#_Toc168279763)

[3.6.1 Методы и подходы к тестированию 73](#_Toc168279764)

[3.6.2 Тестирование с использованием классов эквивалентности 74](#_Toc168279765)

[3.6.3 Тестирование с использованием базовых путей 74](#_Toc168279766)

[3.6.4 Тестирование критического пути 78](#_Toc168279767)

[3.6.5 Интеграционное тестирование 79](#_Toc168279768)

[3.6.6 Системное тестирование 81](#_Toc168279769)

[3.6.7 Приемочное тестирование 83](#_Toc168279770)

[4. Документация пользователя 84](#_Toc168279771)

[4.1 Введение 84](#_Toc168279772)

[4.2 Установка 84](#_Toc168279773)

[4.3 Настройка 84](#_Toc168279774)

[4.4 Основные функции 85](#_Toc168279775)

[Заключение 87](#_Toc168279776)

[Список использованных источников 89](#_Toc168279777)

[Техническая документация 91](#_Toc168279778)

[Исходный код 91](#_Toc168279779)

**РЕФЕРАТ**

Выпускная квалификационная работа на тему «Проектирование интегрированной среды разработки для Java».

Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В. Ф. Уткина», кафедра электронных вычислительных машин., Рязань, 2024 год.

Пояснительная записка к выпускной квалификационной работе объемом 83 страницы содержит 18 рисунков, 13 таблиц и список использованных источников из 18 наименований.

Актуальность данной работы заключается в том, что интегрированная среда разработки для Java должна соответствовать современным требованиям, чтобы эффективно поддерживать процесс разработки программного обеспечения. Для этого необходимо создать инструмент, который упростит работу разработчиков, оптимизирует различные аспекты программирования и, как результат, повысит продуктивность и качество разрабатываемых приложений.

Объект исследования: процесс создания и использования интегрированной среды разработки для Java.

Цель выполнения выпускной квалификационной работы заключается в разработке интегрированной среды разработки, которая будет включать в себя мощные инструменты для написания, компиляции, тестирования и отладки кода, а также интеграцию с системами управления версиями и CI/CD.

При проведении исследования было выполнено изучение аналогов, описание исходной системы разработки, проектирование новой интегрированной среды, а также проведен анализ требований к системе.

В итоге была создана интегрированная среда разработки, которая выполняет следующие основные функции: написание и редактирование кода; компиляция и выполнение кода; управление проектами; интеграция с системами управления версиями; запуск и управление тестами; автоматизация сборки и развертывания приложений.

**Abstract**

Graduate Qualification Work on the topic "Design of an Integrated Development Environment for Java".

Federal State Budgetary Educational Institution of Higher Education "Ryazan State Radio Engineering University named after V. F. Utkin", Department of Electronic Computing Machines, Ryazan, 2024.

The explanatory note to the graduate qualification work, with a total volume of 83 pages, contains 18 figures, 13 tables, and a list of 18 sources.

The relevance of this work lies in the fact that an integrated development environment (IDE) for Java must meet modern requirements to effectively support the software development process. To achieve this, it is necessary to create a tool that simplifies the work of developers, optimizes various aspects of programming, and, as a result, increases the productivity and quality of the developed applications.

Object of research: The process of creating and using an integrated development environment for Java.

The goal of the graduate qualification work is to develop an integrated development environment that includes powerful tools for writing, compiling, testing, and debugging code, as well as integration with version control systems and CI/CD.

During the research, the following tasks were performed: study of analogs, description of the existing development system, design of the new integrated environment, and analysis of system requirements.

As a result, an integrated development environment was created that performs the following main functions: writing and editing code; compiling and executing code; project management; integration with version control systems; running and managing tests; automation of build and deployment processes.

**Список сокращений**

# Введение

В современном мире информационных технологий программирование является одной из наиболее динамично развивающихся областей. Язык программирования Java занимает особое место среди множества существующих языков, благодаря своей надежности, безопасности и возможности создания кроссплатформенных приложений. Интегрированные среды разработки (IDE) играют ключевую роль в процессе разработки программного обеспечения, предоставляя разработчикам инструменты для написания, тестирования и отладки кода.

На сегодняшний день существует несколько популярных IDE для Java, таких как IntelliJ IDEA, Eclipse и NetBeans, каждая из которых имеет свои сильные и слабые стороны. Однако развитие технологий и постоянно изменяющиеся требования нуждаются в создании новых решений, которые могут предложить улучшенные функциональные возможности, лучшую интеграцию с современными инструментами и повышение производительности разработчиков.

Цель данной работы состоит в разработке новой интегрированной среды разработки для Java, которая объединит в себе лучшие качества существующих решений и предложит инновационные возможности для улучшения процесса разработки программного обеспечения.

В ходе работы будут проведены анализ существующих решений, определение требований к новой среде, проектирование архитектуры и структуры системы, разработка и реализация основных компонентов, а также тестирование и оценка эффективности разработанного решения.

# 1. Теоретическое основание

## 1.1 Актуальность разработки интегрированной среды разработки для Java

Современная разработка программного обеспечения требует использования мощных и удобных инструментов, которые могут значительно повысить производительность труда разработчиков. Java является одним из самых популярных языков программирования, используемым для создания разнообразных приложений — от веб-сервисов до мобильных и корпоративных систем. Интегрированная среда разработки (IDE) для Java должна обеспечивать поддержку всех аспектов жизненного цикла разработки программного обеспечения, включая написание, компиляцию, тестирование и отладку кода.

Существующие на рынке решения, такие как IntelliJ IDEA, Eclipse и NetBeans, предлагают широкий спектр возможностей, но каждая из них имеет свои недостатки, которые могут мешать разработчикам в их повседневной работе. Например, IntelliJ IDEA известна своей функциональностью и расширяемостью, но она слишком ресурсоемкая и дорогая. Eclipse, будучи бесплатным и открытым инструментом, страдает от проблем с производительностью и стабильностью. NetBeans предлагает хорошую интеграцию с различными технологиями, но его интерфейс менее интуитивный и удобный по сравнению с конкурентами.

Разработка новой интегрированной среды для Java, которая будет учитывать современные требования и поддержку в русскоязычном сегменте, способна устранить существующие недостатки и предложить новые возможности, которые повысят продуктивность разработчиков и качество создаваемого программного обеспечения.

Одной из ключевых причин актуальности разработки новой IDE является необходимость импортозамещения. В условиях глобальной политической и экономической нестабильности есть ограничения доступа к зарубежным инструментам и технологиям. Создание отечественной интегрированной среды разработки позволит снизить зависимость от иностранных продуктов и обеспечить стабильное и безопасное функционирование разработческих процессов в отечественных компаниях.

Существующие IDE часто не предоставляют полного набора документации и технической поддержки на русском языке, что затрудняет работу разработчиков, не владеющих английским языком на достаточном уровне. Новая интегрированная среда разработки для Java будет включать полную русскоязычную документацию, обучающие материалы и техническую поддержку, что существенно облегчит использование инструмента и повысит его привлекательность для русскоязычных разработчиков.

Разработка новой IDE также актуальна для образовательного сектора. Существует необходимость в качественном и доступном инструменте для обучения программированию на языке Java в школах и университетах. Предоставление бесплатной или льготной лицензии для образовательных учреждений позволит повысить уровень подготовки студентов и школьников, сделать процесс обучения более эффективным и интересным. Новая среда разработки будет включать учебные материалы и задания, специально разработанные для образовательных целей, что поможет преподавателям более эффективно обучать программированию и прививать навыки, необходимые в реальной профессиональной деятельности.

Таким образом, создание новой интегрированной среды разработки для Java, ориентированной на российский рынок и образовательные учреждения, имеет высокую актуальность и значительные преимущества, которые могут повысить продуктивность разработчиков, улучшить качество создаваемого программного обеспечения и способствовать развитию программирования в России.

## 1.2 Сравнительный анализ существующих решений и их недостатков

Сравнительный анализ IntelliJ IDEA, Eclipse, NetBeans и Visual Studio Code нужен для выбора наилучшей интегрированной среды разработки (IDE) и оптимизации некоторых бизнес-процессов разработки программного обеспечения.

Эффективная система для анализа и автоматизации деятельности разработчиков становится ключевым фактором в успешном управлении проектами и повышении продуктивности. Данные IDE имеют разные функциональные возможности и стоимость использования, поэтому сравнительный анализ поможет определить наиболее подходящую систему для работы. Сравнительный анализ позволит оценить преимущества и недостатки каждой системы, а также соотнести их с потребностями бизнеса.

1. Функциональность

IntelliJ IDEA – мощная интегрированная среда разработки, обеспечивающая поддержку широкого спектра языков программирования и технологий, включая интеллектуальное автозавершение кода, глубокую интеграцию с системами контроля версий, мощные инструменты рефакторинга и встроенные средства тестирования.

Eclipse – бесплатная и открытая IDE, поддерживающая множество языков и платформ, с обширным набором плагинов, но имеющая проблемы с производительностью и стабильностью.

NetBeans – предоставляет хорошие возможности для разработки Java-приложений, поддерживает множество языков и технологий, но интерфейс менее интуитивный по сравнению с конкурентами.

Visual Studio Code – легкая и настраиваемая IDE с множеством расширений, поддерживающая широкий спектр языков программирования и инструментов, но требующая настройки для достижения максимальной функциональности.

1. Интерфейс

IntelliJ IDEA – интуитивно-понятный и удобный интерфейс, пользователи могут легко освоить систему и начать работать.

Eclipse – интерфейс довольно сложный, особенно для новых пользователей, но предоставляет много настроек и возможностей.

NetBeans – интерфейс неудобный и не интуитивно-понятный, и может казаться устаревшим по сравнению с конкурентами.

Visual Studio Code – интерфейс легкий и простой, что способствует быстрому освоению системы, но для максимальной эффективности требуется установка дополнительных расширений.

1. Интеграция

IntelliJ IDEA – легко интегрируется с большинством современных инструментов и сервисов, включая системы контроля версий, базы данных, серверы приложений и облачные сервисы.

Eclipse – обладает широкой поддержкой плагинов для интеграции с различными инструментами и сервисами, но интеграция сложная и требует дополнительных настроек.

NetBeans – плохая интеграция с множеством технологий и количество доступных плагинов и расширений меньше по сравнению с Eclipse.

Visual Studio Code – поддерживает огромное количество расширений, позволяющих интегрировать практически любые инструменты и сервисы, но может требовать дополнительной настройки.

1. Многопользовательская среда

IntelliJ IDEA – поддержка командной работы через интеграцию с системами контроля версий и инструментами совместной разработки.

Eclipse – поддерживает многопользовательский режим через интеграцию с системами контроля версий и плагинами для совместной разработки.

NetBeans – поддержка командной работы через интеграцию с системами контроля версий.

Visual Studio Code – поддержка командной работы через расширения для совместного редактирования и интеграцию с системами контроля версий.

1. Возможность работы в облачной среде

IntelliJ IDEA – поддерживает работу в облачных средах через интеграцию с облачными сервисами и инструментами.

Eclipse – поддержка работы в облаке через плагин Cloud IDE и другие облачные сервисы.

NetBeans – поддержка работы в облаке через интеграцию с облачными сервисами.

Visual Studio Code – активно поддерживает работу в облаке через расширения и интеграцию с облачными сервисами.

1. Аналитика данных

IntelliJ IDEA – предоставляет встроенные инструменты для анализа кода, покрытия тестами и метрик производительности.

Eclipse – множество плагинов для аналитики кода и покрытия тестами, но требует дополнительных настроек.

NetBeans – поддерживает аналитические инструменты для анализа кода и покрытия тестами, но возможности ограничены по сравнению с IntelliJ IDEA.

Visual Studio Code – множество расширений для аналитики кода, но требует настройки для достижения максимальной функциональности.

1. Безопасность данных

IntelliJ IDEA – обеспечивает надежную защиту данных, включая шифрование и защиту от несанкционированного доступа.

Eclipse – поддерживает безопасность данных через настройки и плагины.

NetBeans – обеспечивает базовую защиту данных.

Visual Studio Code – обеспечивает безопасность данных через настройки и расширения.

1. Стоимость

IntelliJ IDEA – платная, с бесплатной версией Community Edition, но для полного набора функций требуется лицензия.

Eclipse – бесплатная и открытая IDE.

NetBeans – бесплатная и открытая IDE.

Visual Studio Code – бесплатная IDE с множеством бесплатных расширений.

1. Надежность

IntelliJ IDEA – высоконадежная система с регулярными обновлениями и поддержкой.

Eclipse – надежная, но могут возникать проблемы с производительностью и стабильностью.

NetBeans – надежная и стабильная, но с ограниченными обновлениями.

Visual Studio Code – надежная с регулярными обновлениями.

1. Поддержка

IntelliJ IDEA – обеспечивает высокий уровень поддержки клиентов и решения проблем через документацию, форумы и техническую поддержку.

Eclipse – поддержка через сообщество и документацию, но официальная поддержка ограничена.

NetBeans – поддержка через сообщество и документацию.

Visual Studio Code – поддержка через сообщество, документацию и официальные каналы.

Таблица 1 – таблица сравнительно анализа

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Метрика | IntelliJ IDEA | Eclipse | NetBeans | Visual Studio Code |
| Функциональность | 10 | 8 | 7 | 8 |
| Интерфейс | 9 | 6 | 7 | 8 |
| Интеграция | 9 | 8 | 7 | 9 |
| Многопользовательская среда | 9 | 8 | 7 | 9 |
| Возможность работы в облачной среде | 9 | 7 | 7 | 9 |
| Аналитика данных | 9 | 7 | 7 | 8 |
| Безопасность данных | 9 | 7 | 7 | 8 |
| Стоимость | 4 | 10 | 10 | 10 |
| Надежность | 9 | 7 | 7 | 9 |
| Поддержка | 9 | 6 | 7 | 8 |

## 1.3 Постановка целей и задач

Целью данной работы является создание интегрированной среды разработки для Java, которая объединит лучшие качества существующих решений и предложит новые инновационные возможности. Для достижения этой цели необходимо решить следующие задачи:

* Провести анализ существующих интегрированных сред разработки и их возможностей.
  + Описание функциональных возможностей каждой среды;
  + Сравнение преимуществ и недостатков;
* Определить требования к новой среде разработки.
  + Интервью с профессиональными разработчиками для сбора их требований и пожеланий;
  + Анализ полученных данных для уточнения требований к системе;
* Спроектировать архитектуру и структуру системы.
  + Оценка различных архитектурных стилей (MVC, многослойная, микро-сервисная, модульная);
  + Обоснование выбранного подхода;
* Определение основных модулей и их функций;
  + Моделирование взаимодействий между модулями;
  + Выбор технологий для фронтенда и бэкенда;
  + Описание инструментов и фреймворков, которые будут использованы;
  + Создание диаграмм компонентов, классов и последовательностей;
  + Документирование архитектуры системы;
* Разработать и реализовать основные компоненты среды.
  + Разработка интерфейса редактора;
  + Реализация функций подсветки синтаксиса, автозавершения и рефакторинга;
  + Интеграция с компилятором Java;
  + Интеграция с системой управления версиями;
  + Внедрение поддержки Git;
  + Разработка интерфейса для управления репозиториями и ветками;
  + Реализация инструментов для тестирования;
  + Интеграция с JUnit и другими фреймворками для тестирования;
  + Разработка системы плагинов;
  + Создание API для разработки плагинов;
  + Реализация базовых плагинов для расширения функциональности;
* Провести тестирование и оценку эффективности разработанного решения.
  + Тестирование всех компонентов системы;
  + Сбор и анализ результатов тестирования;
  + Исправление ошибок и повторное тестирование;
  + Оценка производительности и эффективности;
  + Сравнение производительности новой среды с существующими решениями;
  + Оценка удобства использования и продуктивности разработчиков;

## 1.4 Концептуальное описание предметной области

Интегрированная среда разработки (IDE) представляет собой комплексное программное обеспечение, предназначенное для поддержки всех этапов разработки программного обеспечения. Основные компоненты IDE включают редактор исходного кода, компилятор, отладчик, систему управления версиями, инструменты для тестирования и деплоя.

Редактор исходного кода обеспечивает написание и редактирование кода с функциями подсветки синтаксиса, автозавершения, навигации по коду, рефакторинга и встроенной документации. В редактор могут быть интегрированы инструменты для анализа качества кода и обнаружения ошибок на этапе написания.

Компилятор преобразует исходный код на языке высокого уровня в машинный код, исполняемый на целевой платформе. Для языка Java это означает компиляцию в байт-код, который исполняется виртуальной машиной Java (JVM).

Отладчик – это инструмент, позволяющий пошагово выполнять код, устанавливать точки останова, отслеживать значения переменных и состояние памяти.

Интеграция с системами, такими как Git, позволяет отслеживать изменения в коде, работать с ветками разработки, объединять изменения и разрешать конфликты. Это критично для командной разработки.

Поддержка юнит-тестирования, интеграционного тестирования и тестирования производительности. Интеграция с фреймворками, такими как JUnit и TestNG, обеспечивает автоматизацию и удобство в написании и запуске тестов.

Автоматизация процесса развертывания приложений на различные среды (тестирование, производство). Это включает интеграцию с контейнерами, облачными сервисами и другими платформами.

Особенностью интегрированной среды разработки для Java является необходимость поддержки специфических функций и библиотек, характерных для этого языка программирования. Это включает:

Поддержка стандартных библиотек Java: коллекции, потоки ввода/вывода, многопоточность, сетевые библиотеки и многое другое.

Интеграция с популярными фреймворками: Spring, Hibernate, JavaFX, Apache Maven и Gradle для управления зависимостями и сборки проектов.

Обеспечение совместимости с последними версиями языка и библиотек, включая нововведения в синтаксисе и функциональности.

Обеспечение совместимости с различными платформами и технологиями.

Java известна своей платформонезависимостью благодаря принципу "пиши один раз, запускай везде" (Write Once, Run Anywhere). IDE должна поддерживать разработку приложений для различных платформ:

Десктопные приложения с использованием JavaFX и Swing.

Веб-приложения с использованием JSP, Servlets, Spring MVC и других технологий.

Поддержка инструментов для разработки Android-приложений, интеграция с Android SDK.

Поддержка Java EE, интеграция с серверами приложений, такими как Apache Tomcat, JBoss и другие.

## 1.5 Методология исследования

Для выполнения поставленных задач в данной работе будет использован комплексный подход, включающий следующие методы:

* Анализ существующих решений:
  + Исследование и сравнение популярных интегрированных сред разработки для Java с целью выявления их сильных и слабых сторон;
* Проектирование системы:
  + Разработка архитектуры и структуры новой интегрированной среды, включая моделирование основных компонентов и их взаимодействий;
* Реализация компонентов:
  + Написание кода и создание функциональных модулей, обеспечивающих работу среды;
* Тестирование:
  + Проверка работоспособности и производительности системы, выявление и устранение ошибок;
* Оценка эффективности:
  + Сравнение новой среды с существующими решениями, анализ результатов и формулировка выводов;

# 2 Средства реализации и разработки

Разработка интегрированной среды разработки (IDE) требует тщательного выбора инструментов и технологий. Основные компоненты любой IDE включают язык программирования, систему управления версиями, инструменты для тестирования и базы данных. В данной работе основной упор делается на язык программирования Java, однако необходимо также рассмотреть выбор других компонентов, необходимых для успешной реализации проекта.

Выбор Java в качестве основного языка программирования для разрабатываемой IDE обусловлен несколькими ключевыми факторами. Java славится своей портативностью, объектно-ориентированной природой и широким выбором библиотек и фреймворков. Эти характеристики делают данный язык программирования идеальным выбором для разработки разнообразных приложений, от веб-сервисов до мобильных и корпоративных систем. Кроме того, Java обладает мощной экосистемой инструментов и библиотек, которые облегчают разработку и тестирование кода.

При сравнении с другими популярными языками программирования, такими как C++ и C#, Java имеет ряд преимуществ, которые сыграли решающую роль в выборе именно этого языка для проекта. C++ известен своей высокой производительностью и мощными возможностями управления памятью, что делает его отличным выбором для разработки системного и встроенного программного обеспечения. Однако, сложность синтаксиса и управление памятью вручную в C++ могут привести к ошибкам и усложнить процесс разработки, особенно в крупных проектах. С другой стороны, C# является современным и мощным языком программирования, созданным компанией Microsoft. C# предлагает отличную интеграцию с платформой .NET и хорошо подходит для разработки Windows-приложений. Однако, несмотря на свои преимущества, C# не является платформонезависимым языком, что ограничивает его использование в кроссплатформенных проектах.

Java, в отличие от C++ и C#, предлагает оптимальное сочетание производительности, простоты в использовании и кроссплатформенной совместимости. Принцип "Write Once, Run Anywhere" (WORA) позволяет приложениям, написанным на Java, выполняться на любой платформе, где установлена виртуальная машина Java (JVM). Это делает Java идеальным выбором для разработки кроссплатформенных приложений. Кроме того, объектно-ориентированная природа Java способствует созданию модульного и легко расширяемого кода, что особенно важно для разработки сложных программных систем, таких как IDE. Богатая экосистема библиотек и фреймворков, таких как Spring, Hibernate и Apache Struts, упрощает разработку различных типов приложений, от веб-сервисов до корпоративных систем. Наличие множества инструментов и сред разработки, таких как IntelliJ IDEA, Eclipse и NetBeans, которые поддерживают Java, обеспечивает разработчикам удобные средства для написания, компиляции, отладки и тестирования кода.

## 2.1 Язык программирования

Java была разработана Джеймсом Гослингом в компании Sun Microsystems и впервые представлена публике в 1995 году. Основной целью создания Java было предоставление языка программирования, который бы позволял создавать платформонезависимые приложения, т.е. программы, которые могли бы работать на любых устройствах и операционных системах без необходимости изменения кода. Это стало возможным благодаря использованию виртуальной машины Java (JVM), которая обеспечивает выполнение байт-кода Java на любой платформе, для которой разработана JVM. Начало проекта Java относится к 1991 году, когда он был первоначально назван Oak. Официальный выпуск Java 1.0 состоялся в 1995 году.

В 1999 году была введена Java 2 (J2SE 1.2) с улучшенными библиотеками и инструментами. В 2006 году Sun Microsystems выпустила Java под лицензией GPL, сделав её открытым исходным кодом. В 2010 году Oracle приобрела Sun Microsystems и продолжила развитие Java. В 2014 году был выпущен Java 8 с поддержкой лямбда-выражений и потоков. В 2021 году вышла Java 16 с множеством новых функций и улучшений.

Использование Java в качестве основного языка программирования для разрабатываемой интегрированной среды разработки (IDE) обусловлено следующими преимуществами. Благодаря принципу "Write Once, Run Anywhere" (WORA), приложения, написанные на Java, могут быть запущены на любой платформе, где установлена JVM. Это делает Java идеальным выбором для разработки кроссплатформенных приложений. Java поддерживает объектно-ориентированную парадигму программирования, что способствует созданию модульного и легко расширяемого кода. Это особенно важно для разработки сложных программных систем, таких как IDE. Java имеет богатую экосистему библиотек и фреймворков, таких как Spring, Hibernate, Apache Struts, которые упрощают разработку различных типов приложений, от веб-сервисов до корпоративных систем. Существует множество инструментов и сред разработки, таких как IntelliJ IDEA, Eclipse и NetBeans, которые поддерживают Java и обеспечивают разработчикам удобные средства для написания, компиляции, отладки и тестирования кода. Java имеет одно из крупнейших сообществ разработчиков в мире. Это обеспечивает доступ к огромному количеству ресурсов, документации, учебных материалов и открытого исходного кода, что значительно упрощает процесс обучения и решения возникающих проблем. Java обеспечивает высокую производительность благодаря современным оптимизациям компилятора и JVM. Встроенные механизмы безопасности и строгая система типов позволяют создавать надежные и защищенные приложения.

Для разрабатываемой IDE Java была выбрана в качестве основного языка программирования благодаря её портативности, объектно-ориентированной природе и богатой экосистеме библиотек и инструментов. Эти характеристики делают Java идеальным выбором для разработки IDE, которая будет поддерживать разнообразные приложения, от веб-сервисов до мобильных и корпоративных систем. Кроме того, мощная экосистема Java и её сообщество облегчают разработку и тестирование кода, что является значительным преимуществом для реализации данного проекта.

## База данных

Для хранения данных, необходимых для работы IDE, был выбран PostgreSQL. PostgreSQL, также известная как Postgres, является одной из самых мощных и широко используемых систем управления базами данных с открытым исходным кодом. Проект PostgreSQL был начат в 1986 году как POSTGRES, в Калифорнийском университете в Беркли под руководством профессора Майкла Стоунбрейкера. Первоначальная версия была выпущена в 1989 году, и проект продолжал развиваться в течение 1990-х годов. В 1996 году система получила свое современное название — PostgreSQL, отражающее ее поддержку SQL (Structured Query Language).

Основной целью создания PostgreSQL было создание реляционной базы данных, поддерживающей расширяемость и соответствие транзакционным требованиям. PostgreSQL стала известна своей продвинутой функциональностью, надежностью и гибкостью, что сделало ее популярной как среди разработчиков, так и среди крупных предприятий. В настоящее время проект поддерживается активным сообществом разработчиков и пользователей по всему миру, что обеспечивает его постоянное развитие и улучшение. Этот выбор обусловлен несколькими причинами:

* Производительность:
  + PostgreSQL известен своей высокой производительностью и эффективной обработкой запросов, что особенно важно для работы с большими объемами данных.
* Масштабируемость:
  + PostgreSQL предоставляет мощные возможности для масштабирования, что позволит системе эффективно работать с увеличением количества данных и пользователей.
* Надежность:
  + PostgreSQL обеспечивает высокий уровень надежности и целостности данных благодаря поддержке ACID-транзакций и эффективным механизмам резервного копирования.
* Сообщество и поддержка:
  + PostgreSQL имеет большое сообщество пользователей и разработчиков, что обеспечивает доступ к обширной документации и поддержке.

## 2.3 Моделирование Бизнес-процессов

Моделирование бизнес-процессов – это процесс создания графического представления бизнес-процесса с целью анализа, оптимизации и улучшения деятельности компании. Моделирование бизнес-процессов позволяет выявить узкие места в производственных и управленческих процессах, улучшить производительность и качество работы организации, повысить эффективность бизнес-процессов.

Существует несколько типов моделирования бизнес-процессов:

1. Функциональное моделирование – это создание графического представления структуры компании и описания функций, выполняемых каждым отделом. Цель функционального моделирования – узнать, как каждый отдел компании взаимодействует с другими отделами и какие задачи решает.

2. Процессное моделирование – это создание графического представления бизнес-процессов компании с целью оптимизации и улучшения их работы. Процессное моделирование позволяет выявить узкие места в производственных и управленческих процессах, увеличить эффективность работы компании и повысить качество продукции.

3. Моделирование данных – это создание графического представления информационной структуры компании. Моделирование данных позволяет выявить проблемы с хранением, обработкой и передачей информации, а также улучшить ее качество и защиту.

4. Моделирование требований – это создание графического представления требований к программным системам компании. Моделирование требований позволяет выявить проблемы с функциональностью программных систем и способствует созданию качественных и эффективных решений.

## 2.3.1 Моделирование с использованием BPMN

BPMN (Business Process Model and Notation)– это набор символов и правил, которые используются для описания бизнес-процессов. BPMN позволяет моделировать, анализировать и осуществлять мониторинг бизнес-процессов в организации в графическом виде, что облегчает взаимопонимание процессов между пользователями, управленцами и программистами.

Методология позволяет описать процессы бизнеса на разных уровнях детализации, в том числе и на уровне отдельных действий и задач. Символы и знаки на диаграмме BPMN обозначают типы процессов и связи между ними, участников процесса, временные ограничения, роли и ответственности и т.д.

BPMN часто используется в проектах разработки информационных систем и в исследовательских проектах, связанных с оптимизацией бизнес-процессов. В рамках данной работы BPMN может быть использован следующим образом:

* для выявления узких мест и возможных улучшений в бизнес-процессах;
* для разработки функциональных требований к ИС, определяя основные функциональные блоки, взаимодействия между ними и возможные ограничения;
* для описания сценариев использования в ИС;
* для оценки эффективности ИС как инструмента оптимизации бизнес-процессов.

Таким образом, использование BPMN в рамках проектирования информационной системы позволит разработать более точное и полное представление о бизнес-процессах, определить требования к ИС и разработать эффективную ИС, которая будет управлять бизнес-процессами.

## 2.3.2 Определение требований

Бизнес-требования – это спецификации того, какая функциональность нужна для информационной системы (ИС), чтобы она могла отвечать на потребности бизнес-процессов. Цель выявления бизнес-требований состоит в том, чтобы точно определить, какие потребности организации должна удовлетворять ИС. Это позволяет убедиться, что система будет способна решать предъявляемые к ней требования и несколькими привлекательными свойствами, такими как:

1. Более точное соответствие потребностям бизнеса: при определении бизнес-требований система разрабатывается с учетом всех потребностей бизнес-процессов, что обеспечивает максимально эффективную работу.

2. Оптимизация затрат: при использовании информационной системы, на которой учтены бизнес-требования, возможно сократить издержки и повысить эффективность работы.

3. Увеличение контроля за процессом: ИС, разработанная с учетом бизнес-требований, позволяет более точно контролировать работу в определенных областях бизнеса и процессов.

Основные виды бизнес-требований:

1. Функциональные требования – это спецификация функций, которые должна обеспечивать ИС для выполнения конкретных задач. Такие требования включают набор функций, необходимых для обеспечения определенной работой ИС.

Функциональные требования описывают, какую функциональность должна обеспечивать система, чтобы удовлетворить нужды пользователей. Они включают в себя подробные описания действий, которые должна выполнять система, чтобы обеспечить требуемую функциональность.

Функциональные требования к ИС нужны для того, чтобы убедиться, что система будет работать эффективно, соответствовать требованиям пользователей и достигать поставленных целей. Они характеризуют желательные результаты, которые должна достигать система. Без функциональных требований невозможно оценить эффективность ИС и сравнить ее с другими решениями.

Цели выявления функциональных требований к ИС:

* определение потребностей и требований пользователей;
* спецификация необходимых функциональных возможностей ИС;
* установление стандартов выполнения задач системой;
* обеспечение соответствия ИС требованиям стандартов и правил;
* предоставление основы для оценки согласованности системы с требованиями пользователей;
* осуществление контроля за выполнением функций системы и ее возможностей;
* предоставление основы для управления внесением изменений в программное обеспечение;
* создание стандартизованных документов, необходимых для разработки, тестирования ИС и их эксплуатации.

Таким образом, функциональные требования к ИС являются ключевым элементом при разработке, тестировании, реализации и использовании системы, позволяя ее эффективно и успешно функционировать в различных условиях.

2. Нефункциональные требования (НФТ) – это описание требований к качеству и другим свойствам системы, которые необходимы для того, чтобы ИС работала правильно в определенных условиях. НФТ не описывают функциональности системы, а описывают общечеловеческие характеристики, связанные с ее работой. Они описываются как ограничения и обеспечивают качество функциональных требований.

Цель выявления нефункциональных требований заключается в создании пригодной для использования системы, способной соответствовать ожиданиям пользователей и требованиям рынка.

Примеры нефункциональных требований к информационной системе:

* надежность;
* производительность;
* масштабируемость;
* безопасность;
* удобство использования;
* совместимость;
* поддержка.

Знание НФT важно при проектировании и создании системы, так как правильное выявление этих требований гарантирует качество и удобство ее использования, а также повышает уровень удовлетворенности пользователей.

3. Технические требования – это требования к аппаратному и программному обеспечению, которое будет использоваться в информационной системе. Данные требования описывают технические характеристики системы, необходимые для ее корректной работы и соответствия ее функциональным требованиям.

Цель технических требований в рамках разработки ИС заключается в том, чтобы определить требования к материальной базе системы, которые могут влиять на ее производительность, надежность и безопасность. Они также могут служить для определения соответствующих технологий и стандартов, необходимых для использования ИС.

Примеры технических требований к ИС:

* минимальные и рекомендуемые характеристики компьютерной станции, на которой будет использоваться ИС, например, объем оперативной памяти, процессор и т. д.;
* версии программных продуктов, включая операционную систему, браузер и т. д., которые поддерживаются ИС;
* требования к безопасности, такие как использование шифрования для защиты данных, установка антивирусного программного обеспечения и т. д.;
* требования к сетевым настройкам, например, настройки протокола безопасности, которые должны быть поддержаны в ИС;
* требования к производительности, например, доступность ИС для ряда пользователей и количество одновременных сеансов.

Знание технических требований позволяет разработчикам ИС создавать системы, которые соответствуют критериям эффективности, производительности и безопасности. Они также могут помочь в минимизации рисков, связанных с несоответствием системных требований, что обеспечивает максимальный уровень качества ИС, которые могут быть разработаны и реализованы.

## 2.3.3 Описание бизнес-процессов интегрированной среды разработки для Java

Описание бизнес-процессов интегрированной среды разработки (IDE) для Java необходимо для оптимизации работы и улучшения качества разработки программного обеспечения. Это позволяет разработчикам и организациям более четко понимать, какие именно операции выполняются в процессе разработки, и какие результаты могут быть получены.

Конкретные цели описания бизнес-процессов в IDE для Java могут меняться в зависимости от требований компании, однако наиболее распространенные цели приведены ниже:

1. Оптимизация процессов и улучшение качества работы. Описание процессов позволяет идентифицировать узкие места, которые могут замедлять и затруднять выполнение задач, что позволяет оптимизировать работу IDE и улучшить качество предоставляемых инструментов. Это включает улучшение производительности редактора кода, ускорение компиляции, а также оптимизацию работы отладчика и системы управления версиями.

2. Снижение затрат. Описание процессов может помочь выявить возможности для снижения затрат на временные ресурсы, вычислительные мощности, трудовые ресурсы и т.д. Это особенно важно в условиях импортозамещения, когда сокращение зависимости от дорогих зарубежных инструментов становится приоритетной задачей.

3. Улучшение коммуникации. Описание процессов может помочь улучшить коммуникацию между разработчиками, менеджерами проектов и другими заинтересованными сторонами. Это может снизить количество ошибок и недоразумений, улучшить документирование кода и управление проектами. Важным аспектом является наличие русскоязычной документации и технической поддержки.

4. Контроль качества. Описание бизнес-процессов позволяет контролировать качество разработки и соответствие инструментов заявленным стандартам. Это включает в себя обеспечение надежности компилятора, отладчика, тестировочных инструментов и других модулей IDE.

## 2.3.4 Прикладное применение описания бизнес-процессов

Прикладное применение описания бизнес-процессов может быть различным: от повышения эффективности работы IDE до улучшения уровня обслуживания пользователей и увеличения продуктивности разработки. Благодаря описанию процессов IDE может снизить свои расходы и повысить эффективность работы, что в свою очередь позитивно сказывается на репутации и доходах компании. Разработка новой IDE должна учитывать следующие аспекты:

1. Импортозамещение. Разработка отечественного продукта, который может заменить зарубежные решения, снизить зависимость от иностранных технологий и обеспечить стабильное функционирование в условиях ограничений.

2. Русскоязычная поддержка. Полная русскоязычная документация, обучающие материалы и техническая поддержка, что облегчает использование инструмента русскоязычными разработчиками и образовательными учреждениями.

3. Использование в образовательных целях. Предоставление бесплатной или льготной лицензии для образовательных учреждений, что позволит повысить уровень подготовки студентов и школьников. Это включает создание учебных материалов и заданий, специально разработанных для образовательных целей, что поможет преподавателям более эффективно обучать программированию.

## 2.3.5 Основные инструменты описания бизнес-процессов

В качестве основных инструментов описания бизнес-процессов были выбраны методологии ARIS и BPMN. Эти методологии позволяют детально описать и визуализировать все этапы разработки, взаимодействие между различными модулями и компонентами системы, а также обеспечить прозрачность и управляемость процесса разработки.

ARIS (Architecture of Integrated Information Systems — это методология позволяет описать архитектуру и процессы разработки, определить ключевые компоненты системы и их взаимодействие. ARIS помогает выявить узкие места и возможности для оптимизации, а также обеспечить соответствие процессов стандартам качества.

BPMN (Business Process Model and Notation) используется для моделирования бизнес-процессов и создания диаграмм, которые наглядно отображают последовательность действий и взаимодействие различных компонентов системы. Это помогает улучшить понимание процессов, улучшить коммуникацию между участниками проекта и повысить качество разработки.

Бизнес-процесс: Разработка и тестирование проекта

Этот бизнес-процесс описывает основные этапы разработки и тестирования проекта с использованием интегрированной среды разработки для Java показан на рисунке 1.

|  |  |
| --- | --- |
|  | Рисунок 1 - бизнес-процесс “TO-BE” «Разработка и тестирование» |

Бизнес-процесс: управление задачами проекта

Этот бизнес-процесс описывает управление задачами проекта, включая создание, назначение и завершение задач показан на рисунке 2.

|  |  |
| --- | --- |
|  | Рисунок 2 – бизнес-процесс “TO-BE” «Управление задачами проекта» |

Бизнес-процесс: Развертывание приложения

Этот бизнес-процесс описывает этапы развертывания приложения на различные среды (тестовая, продуктная) показана на рисунке 3.

|  |  |
| --- | --- |
|  | Рисунок 3 – бизнес-процесс “TO-BE” «Развертывание приложения» |

## 2.3.6 Модель входов выходов

Модель входов и выходов — это метод моделирования бизнес-процессов, который используется для анализа и оптимизации процессов в организации. Она позволяет визуализировать, структурировать и анализировать входные и выходные данные каждой функции в рамках бизнес-процесса. В основе этой модели лежит идея того, что каждая функция в процессе требует определённых входных данных для своего выполнения и производит определённые выходные данные. Модель входов выходов проектируемой системы представлена на рисунке 4.

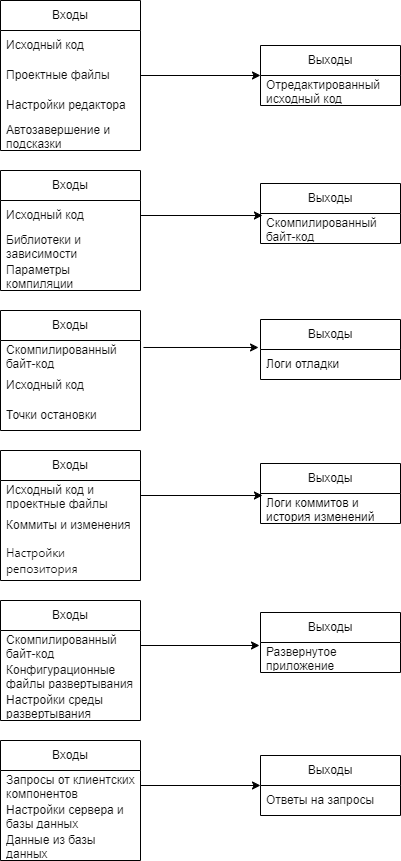


Рисунок 4 – Модель входов-выходов

## 2.4 Моделирование требований к интегрированной среде разработки

Для успешной разработки интегрированной среды необходимо определить и проанализировать требования, предъявляемые к системе. Эти требования можно разделить на функциональные и нефункциональные.

### 2.4.1 Функциональные требования

Функциональные требования (Functional Requirements) – определяют функциональность (поведение) программной системы, которая должна быть создана разработчиками для предоставления возможности выполнения пользователями своих обязанностей в рамках бизнес-требований и в контексте пользовательских требований. Вот некоторые из них:

* Поддержка написания, редактирования и форматирования исходного кода на языке Java;
* Инструменты для компиляции и выполнения кода;
* Средства для отладки и тестирования приложений;
* Интеграция с системами управления версиями, такими как Git;
* Поддержка рефакторинга и анализа кода;
* Возможность расширения функциональности через плагины;

### 2.4.2 Нефункциональные требования

Нефункциональные требования — требования, определяющие свойства, которые система должна демонстрировать, или ограничения, которые она должна соблюдать, не относящиеся к поведению системы. Например, производительность, удобство сопровождения, расширяемость, надежность, факторы эксплуатации. Вот некоторые из них:

* Производительность: система должна работать быстро и эффективно, не создавая задержек при работе с крупными проектами;
* Надежность: система должна быть стабильной и устойчивой к сбоям;
* Удобство использования: интерфейс должен быть интуитивно понятным и простым в освоении;
* Масштабируемость: система должна поддерживать возможность расширения функциональности и интеграции с новыми инструментами;

## 2.5. Архитектурные паттерны и подходы к проектированию

Для разработки сложных программных систем, таких как IDE, важно использовать проверенные архитектурные паттерны и подходы.

Паттерн MVC (Model-View-Controller) разделяет приложение на три компонента: модель, представление и контроллер. Это разделение облегчает управление сложностью системы и улучшает тестируемость. В контексте разработки IDE, MVC может помочь отделить логику обработки данных (модель) от пользовательского интерфейса (представление) и логики взаимодействия (контроллер). Этот паттерн особенно полезен для приложений с графическим интерфейсом, таких как редактор кода и инструменты отладки.

Модульная архитектура позволяет разделить систему на независимые модули, которые можно разрабатывать, тестировать и разворачивать отдельно. Это повышает гибкость разработки и позволяет легче управлять изменениями и обновлениями. В IDE, модульная архитектура позволяет создавать независимые компоненты для разных функций, таких как редактор кода, компилятор, отладчик и интеграция с системами управления версиями.

Микро-сервисная архитектура предполагает разбиение системы на небольшие, независимые сервисы, которые взаимодействуют через API. Этот подход улучшает масштабируемость и управляемость системы, позволяя независимо разрабатывать и развертывать отдельные компоненты. Для IDE это может означать создание отдельных микро-сервисов для выполнения различных задач, таких как компиляция кода, управление версиями и запуск тестов.

После тщательного анализа различных архитектурных подходов, было принято решение использовать модульную архитектуру для разработки интегрированной среды разработки для Java. Этот выбор обусловлен следующими причинами:

* Гибкость: модульная архитектура позволяет легко добавлять новые функции и компоненты без значительных изменений в существующем коде. Это особенно важно для IDE, которая должна поддерживать множество различных инструментов и расширений;
* Управляемость: разделение системы на независимые модули упрощает управление проектом и облегчает процесс тестирования. Каждый модуль можно разрабатывать и тестировать отдельно, что снижает риски и упрощает обнаружение и исправление ошибок;
* Производительность: в отличие от микро-сервисной архитектуры, модульная архитектура не требует дополнительных накладных расходов на межпроцессовое взаимодействие, что может быть критично для производительности IDE;
* Удобство разработки: модульная архитектура позволяет разработчикам сосредоточиться на отдельных частях системы, что упрощает процесс разработки и улучшает качество кода;

## 2.6. Технологии и инструменты для разработки среды

Для реализации интегрированной среды разработки будут использованы следующие технологии и инструменты:

Java Development Kit (JDK): основной набор инструментов для разработки на языке Java. В состав JDK входят:

* компилятор Java (javac);
* стандартные библиотеки классов Java;
* документация;
* исполнительная система Java (JRE);

JavaFX: платформа для создания пользовательских интерфейсов.

Spring Framework: фреймворк для разработки корпоративных приложений на Java, который может быть использован для создания серверной части IDE.

Hibernate: ORM (Object-Relational Mapping) фреймворк для работы с базами данных.

Maven: инструмент для автоматизации сборки проекта.

JUnit: фреймворк для модульного тестирования Java-приложений.

Git: система управления версиями, которая будет интегрирована в IDE для отслеживания изменений в коде.

# 3. Разработка технического задания

## 3.1.1 Определение функциональных требований

Функциональные требования к интегрированной среде разработки (IDE) для Java описывают, какой функционал (возможности) должны предоставляться пользователю и как система должна работать. Они нужны для того, чтобы определить, какие возможности должны быть реализованы в системе, чтобы она могла компетентно справляться со своими задачами.

Перечисленные ниже функциональные требования необходимы для того, чтобы IDE могла надежно и эффективно поддерживать процесс разработки программного обеспечения на языке Java. Без них работа разработчиков будет неорганизованной и менее продуктивной, что в конечном итоге может привести к снижению качества создаваемого ПО. Так, можно представить функциональные требования в виде следующего списка:

FE-1. Возможность написания и редактирования кода;

FE-2. Поддержка подсветки синтаксиса для языка Java;

FE-3. Реализация функции автозавершения кода на основе контекста;

FE-4. Реализации функции навигация по коду: переход к определению методов и классов;

FE-5. Поддержка рефакторинга кода: поддержка различных видов рефакторинга (переименование, изменение сигнатуры метода и т.д.);

FE-6. Поддержка компиляции Java-кода с выводом сообщений об ошибках и предупреждений;

FE-7. Поддержка интеграции с существующими инструментами сборки;

FE-8. Реализация функции установки и управления точками остановки(breakpoint), инспекция переменных и пошаговое выполнение кода;

FE-9. Реализации интеграция с Git для отслеживания изменений в коде;

FE-10. Возможность визуализации истории изменений: управление ветками и отображение истории изменений;

FE-11. Поддержка интеграция с JUnit для выполнения юнит-тестов;

FE-12. Реализация визуализации результатов тестов и покрытия кода;

FE-13. Реализации автоматизация процесса развертывания приложений;

FE-14. Поддержка развертывание на локальную, тестовую и продуктивную среды;

FE-15. Поддержка серверного фреймворка Spring Boot для выполнения серверных задач;

FE-16. Возможность управление базами данных;

FE-17. Обеспечение связи между приложением и базой данных PostgreSQL;

FE-18. Реализация управления запросами между клиентской и серверной частями;

## 3.1.2 Определение нефункциональных требований

Нефункциональные требования к интегрированной среде разработки (IDE) для Java описывают, как система должна выполнять свои функции, и включают аспекты, связанные с производительностью, надежностью, удобством использования и безопасностью.

Перечисленные ниже нефункциональные требования необходимы для того, чтобы IDE могла эффективно поддерживать процесс разработки программного обеспечения на языке Java, обеспечивая при этом высокую производительность, надежность и удобство использования. Так, можно представить нефункциональные требования в виде следующего списка:

1. Требования к доступности

AVL-1. Система должна быть доступна круглосуточно, за исключением времени тех. работ.

AVL-2. Время между возникновением сбоя и восстановлением сервиса должно составлять не более пяти часов в год.

AVL-3. Система должна обеспечивать доступ к данным из любой точки мира через облачное решение.

AVL-4. Время простоя системы не должно превышать 0.1% от общего времени работы в году.

AVL-5. Все операции, связанные с редактированием и сохранением кода, должны быть доступны в любое время.

AVL-6. Система должна поддерживать горячее резервирование для минимизации времени простоя.

AVL-7. Восстановление системы после аварийного сбоя должно занимать не более 15 минут.

AVL-8. Система должна быть доступна для использования на всех поддерживаемых операционных системах (Windows, macOS, Linux).

AVL-9. Система должна обрабатывать все запросы пользователей без задержек в пиковые часы нагрузки.

2. Требования к целостности

INT-1. Система должна защищать от неправомерного добавления, удаления или изменения данных.

INT-2. Система должна хранить данные о проекте в течение 10 лет с момента завершения проекта.

INT-3. Система должна проверять корректность вводимых данных при компиляции и отладке кода.

INT-4. Все изменения в коде должны фиксироваться в системе контроля версий.

INT-5. Система должна поддерживать откат к предыдущим версиям проекта.

INT-6. Все транзакции в базе данных должны быть атомарными и изолированными.

INT-7. Система должна использовать контроль целостности данных для предотвращения ошибок при сохранении.

INT-8. Данные о пользователях должны быть защищены и доступны только авторизованным лицам.

INT-9. Система должна проводить регулярные проверки целостности данных.

INT-10. Все действия пользователей должны записываться в лог для последующего анализа.

3. Требования к совместимости

IOP-1. Система должна иметь возможность обмениваться данными с системами контроля версий, такими как Git.

IOP-2. Система должна поддерживать импорт и экспорт проектов в форматах XML, JSON, WSDL, JAR, WAR и PROJECT.

IOP-3. Система должна интегрироваться с популярными CI/CD инструментами.

IOP-4. Система должна поддерживать подключение к различным базам данных.

IOP-5. Система должна поддерживать работу с популярными Java фреймворками.

IOP-6. Система должна иметь возможность интеграции с облачными сервисами.

IOP-7. Система должна поддерживать работу с различными языками программирования через плагины.

IOP-8. Система должна быть совместима с различными операционными системами (Windows, macOS, Linux).

IOP-9. Система должна поддерживать интеграцию с инструментами мониторинга, такими как Prometheus и Grafana.

4. Требования к производительности

PER-1. Авторизация в системе не должна занимать более 2 секунд после нажатия кнопки «Войти».

PER-2. Время компиляции проекта не должно превышать 30 секунд для среднего проекта (250 файлов).

PER-3. Открытие проекта в IDE не должно занимать более 5 секунд.

PER-4. Время отклика системы на редактирование кода не должно превышать 0.1 секунды.

PER-5. Система должна поддерживать одновременную работу не менее 100 пользователей без снижения производительности на одном проекте.

PER-6. Время выполнения юнит-тестов не должно превышать 1 минуты для среднего проекта (250 тестов).

PER-7. Время развертывания приложения не должно превышать 7 минут.

PER-8. Время загрузки интерфейса IDE после запуска не должно превышать 5 секунд.

PER-9. Время выполнения операции поиска в проекте не должно превышать 1 секунды.

PER-10. Время синхронизации проекта с системой контроля версий не должно превышать 20 секунд.

5. Требования к надежности

REL-1. Не более 1 сбоя на 1000 операций должно быть допущено в течение года.

REL-2. Система должна создавать резервную копию данных каждый день.

REL-3. В случае сбоя пользователь должен иметь возможность восстановить не сохраненные данные.

REL-4. Время восстановления после сбоя не должно превышать 15 минут.

REL-5. Система должна обеспечивать непрерывность работы в случае отказа одного из серверов.

REL-6. Все критические ошибки должны быть исправлены в течение 24 часов после обнаружения.

REL-7. Система должна иметь возможность автоматического переключения на резервный сервер.

REL-8. Частота возникновения ошибок не должна превышать 0.1% от общего числа операций.

REL-9. Среднее время между отказами системы должно составлять не менее 30 дней.

REL-10. Система должна поддерживать журналирование всех действий для последующего анализа и восстановления.

6. Требования к безопасности

SEC-1. Все пароли должны храниться в зашифрованном виде.

SEC-2. Система должна требовать двухфакторную аутентификацию для доступа к критическим функциям.

SEC-3. Все данные должны передаваться по защищенному протоколу (HTTPS).

SEC-4. Система должна поддерживать контроль доступа на основе ролей (RBAC).

SEC-5. Система должна поддерживать журналирование всех действий пользователей.

SEC-6. Все критические данные должны быть зашифрованы на уровне базы данных.

7. Требования к удобству использования

USE-1. Пользователь, прошедший инструктаж, должен иметь возможность выполнить базовые операции за 10 минут.

USE-2. Интерфейс системы должен быть интуитивно понятным и поддерживать возможности настройки под пользователя.

USE-3. Все основные функции системы должны быть доступны в два клика.

USE-4. Система должна поддерживать темную и светлую темы оформления.

USE-5. Пользователь должен получать уведомления о завершении длительных операций.

USE-6. Все элементы интерфейса должны иметь всплывающие подсказки.

USE-7. Пользователь должен иметь возможность настраивать горячие клавиши для часто используемых функций.

USE-8. Система должна поддерживать автосохранение изменений каждые 30 секунд.

USE-9. Система должна предоставлять интерактивные учебные материалы и руководство пользователя.

USE-10. Пользователи должны иметь возможность оставить отзыв о системе и предложить улучшения.

8. Требования к возможности модификации

MOD-1. Изменения в системе должны быть реализованы без нарушения работы других функций.

MOD-2. Все изменения должны быть протестированы в тестовой среде перед внедрением в рабочую.

MOD-3. Время простоя системы при внесении изменений не должно превышать 15 минут.

MOD-4. Новые функции должны быть интегрированы через плагины или модули.

MOD-5. Система должна поддерживать откат изменений в случае возникновения ошибок.

MOD-6. Все модификации системы должны быть обратно совместимы.

MOD-7. Модификации не должны требовать изменения исходного кода системы.

MOD-8. Изменения в системе должны проходить через процесс проверки и утверждения.

MOD-9. Система должна поддерживать автоматическое обновление модулей и плагинов.

9. Требования к масштабируемости

SCA-1. Система должна справляться с увеличением объема данных на 50% без ухудшения производительности.

SCA-2. В случае увеличения нагрузки система должна автоматически масштабироваться.

SCA-3. Время ответа системы не должно превышать 2 секунд при максимальной нагрузке.

SCA-4. Система должна поддерживать горизонтальное и вертикальное масштабирование.

SCA-5. Система должна быть готова к интеграции с новыми сервисами и инструментами.

SCA-6. Система должна поддерживать масштабирование базы данных без простоев.

SCA-7. Система должна обеспечивать балансировку нагрузки между серверами.

SCA-8. Система должна поддерживать автоматическое распределение ресурсов в зависимости от нагрузки.

SCA-9. Время развертывания дополнительных ресурсов не должно превышать 10 минут.

10. Требования к проверяемости и тестируемости

VER-1. Все функции системы должны быть покрыты автоматическими тестами.

VER-2. Система должна поддерживать модульное тестирование.

VER-3. Тестирование производительности системы должно проводиться регулярно.

VER-4. Система должна поддерживать интеграционное тестирование.

VER-5. Все изменения в системе должны проходить через процесс тестирования перед внедрением.

VER-6. Система должна поддерживать нагрузочное тестирование.

VER-7. Все ошибки и сбои должны логироваться и анализироваться.

VER-8. Система должна поддерживать тестирование безопасности.

VER-9. Система должна предоставлять отчеты по результатам тестирования.

VER-10. Время выполнения автоматических тестов не должно превышать 30 минут.

# 3.2 Разработка архитектуры

## 3.2.1 Выбор архитектурного стиля

Для разработки интегрированной среды разработки (IDE) для Java был выбран модульный архитектурный стиль. Этот стиль позволяет разделить систему на независимые модули, что упрощает управление разработкой, тестированием и развертыванием отдельных компонентов системы. Модульная архитектура обеспечивает гибкость и масштабируемость, что позволяет легко добавлять новые функции и компоненты без значительных изменений в существующем коде.

На основе анализа требований и целей проекта, модульная архитектура была признана наиболее подходящей для реализации интегрированной среды разработки по следующим причинам:

Гибкость: модульная архитектура позволяет легко добавлять новые функции и модули, что критично для IDE, которая должна поддерживать множество различных инструментов и расширений;

Управляемость: разделение системы на независимые модули упрощает управление проектом, а также упрощает процесс тестирования и отладки. Каждый модуль можно разрабатывать и тестировать отдельно, что снижает риски и упрощает обнаружение и исправление ошибок;

Производительность: модульная архитектура не требует дополнительных накладных расходов на межпроцессовое взаимодействие, что может быть критично для производительности IDE;

Схема модульной архитектуры представлена на рисунке 5.

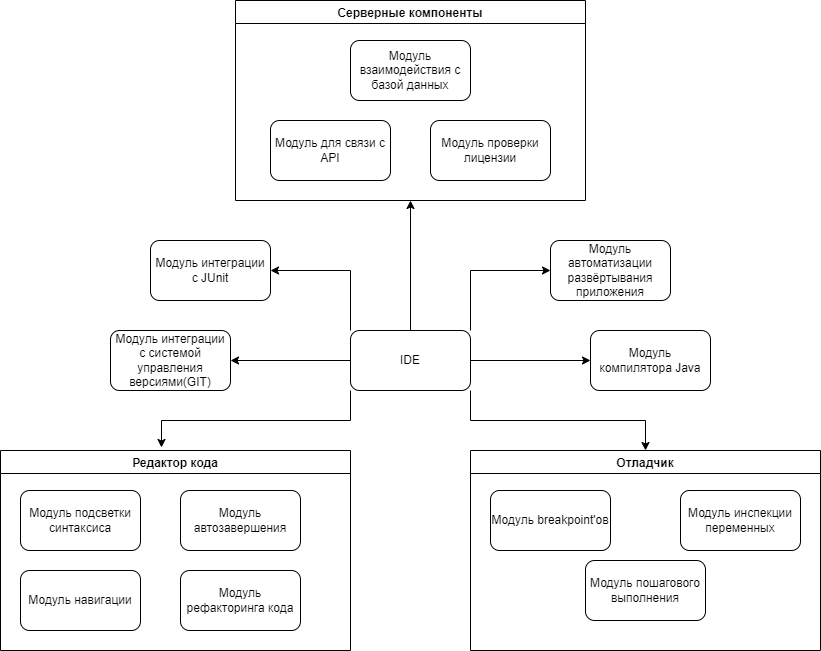


Рисунок 5 – Модульная архитектура приложения

## 3.2.2 Моделирование компонентов системы

Моделирование компонентов системы является важным этапом проектирования интегрированной среды разработки (IDE) для Java. Оно позволяет определить структуру и взаимодействие различных модулей системы, обеспечивая их согласованную работу. Ниже представлено подробное описание моделирования ключевых компонентов системы на основе выбранной модульной архитектуры.

Модуль подсветки синтаксиса поддерживает подсветку синтаксиса для языка Java, обеспечивая улучшенную читаемость кода;

Модуль автозавершения предоставляет функции автозавершения кода на основе контекста, помогая ускорить процесс написания кода и снизить количество ошибок;

Модуль навигации: обеспечивает переход к определению методов и классов, облегчая навигацию по коду;

Модуль рефакторинга кода: поддерживает различные виды рефакторинга, включая переименование переменных, методов и классов, а также изменение сигнатуры методов;

Модуль компилятора Java обеспечивает компиляцию Java-кода с выводом сообщений об ошибках и предупреждений, интегрируется с инструментами сборки, такими как Maven, Ant и Gradle;

Модуль управления точками остановки позволяет устанавливать и управлять точками останова в коде;

Модуль инспекции переменных позволяет просматривать значения переменных в реальном времени во время отладки;

Модуль пошагового выполнения: обеспечивает пошаговое выполнение кода с возможностью просмотра стека вызовов;

Модуль интеграции с Git интегрируется с системой управления версиями Git, поддерживает основные операции (commit, push, pull, merge) и визуализацию истории изменений.

Модуль интеграции с JUnit интегрируется с JUnit для выполнения юнит-тестов и отображения результатов тестирования.

Модуль автоматизации развёртывания приложений: автоматизирует процесс развертывания приложений, поддерживает развертывание на различные среды (локальная, тестовая, продуктивная).

Модуль взаимодействия с базой данных обеспечивает связь между приложением и базой данных PostgreSQL, выполняет операции CRUD (создание, чтение, обновление, удаление данных).

Модуль для связи с API управляет запросами между клиентской и серверной частями, маршрутизирует запросы и обеспечивает безопасность.

Модуль проверки лицензии управляет лицензиями пользователей, проверяет валидность лицензий при запуске и использовании IDE;

Компоненты системы взаимодействуют между собой для обеспечения согласованной работы интегрированной среды разработки. Основные взаимодействия между модулями описаны ниже:

* Редактор кода:
  + Передает написанный код компилятору для проверки и компиляции;
  + Взаимодействует с отладчиком для установки точек останова и выполнения кода;
  + Интегрируется с системой управления версиями для отслеживания изменений кода;
  + Использует инструменты для тестирования для запуска и отображения результатов юнит-тестов;
* Компилятор:
  + Получает исходный код от редактора кода и выполняет его компиляцию;
  + Сообщает об ошибках и предупреждениях обратно в редактор кода;
* Отладчик:
  + Получает информацию о точках останова от редактора кода.
  + Позволяет разработчику выполнять пошаговое выполнение кода и инспекцию переменных;
  + Сообщает о состоянии выполнения кода обратно в редактор;
* Система управления версиями:
  + Интегрируется с редактором кода для отслеживания изменений в коде;
  + Обеспечивает возможность выполнения операций с репозиторием (commit, push, pull, merge);
  + Визуализирует историю изменений и управление ветками.
* Инструменты для тестирования:
  + Получают тесты от редактора кода и выполняют их;
  + Отображают результаты тестирования и информацию о покрытии кода;
* Инструменты для развёртывания:
  + Получают собранное приложение от компилятора и выполняют его развертывание;
  + Поддерживают развертывание на различные целевые среды;
* Серверные компоненты:
  + Модуль взаимодействия с базой данных обеспечивает доступ к данным для всех других модулей;
  + API Gateway маршрутизирует запросы между клиентской и серверной частями;
  + Модуль проверки лицензии обеспечивает безопасность и управляет лицензиями пользователей;

# 3.3 Проектирование базы данных

Основная цель проектирования базы данных — создать структуру, которая будет поддерживать все необходимые функции системы, обеспечивая при этом целостность и производительность данных.

Для успешного проектирования базы данных были определены следующие основные требования:

Целостность данных: обеспечение целостности данных через использование ограничений и правил, таких как первичные и внешние ключи, уникальные индексы и ограничения целостности;

Производительность: оптимизация структуры базы данных для обеспечения быстрого доступа и обработки данных, включая использование индексов и оптимизацию запросов;

Масштабируемость: возможность расширения базы данных для поддержки увеличивающихся объемов данных и количества пользователей без значительных изменений в структуре;

Безопасность: обеспечение безопасности данных через контроль доступа, аутентификацию и авторизацию пользователей, а также защиту данных от несанкционированного доступа;

Поддержка транзакций: поддержка ACID (Atomicity, Consistency, Isolation, Durability) свойств для обеспечения надежности и целостности транзакций;

## 3.3.1 Разработка схемы базы данных

База данных для интегрированной среды разработки будет включать следующие основные таблицы:

Таблица 2 - Users

|  |  |
| --- | --- |
| Название | Описание |
| user\_id (PK) | Уникальный идентификатор пользователя. |
| username | Имя пользователя |
| password\_hash | Хеш пароля для аутентификации |
| email | Электронная почта пользователя |
| role | Роль пользователя |

Таблица 3 - Projects

|  |  |
| --- | --- |
| Название | Описание |
| project\_id (PK) | Уникальный идентификатор проекта |
| project\_name | Название проекта |
| created\_by (FK) | Идентификатор пользователя, создавшего проект |
| created\_at | Дата и время создания проекта |

Таблица 4 - Files

|  |  |
| --- | --- |
| Название | Описание |
| file\_id (PK) | Уникальный идентификатор файла |
| file\_name | Имя файла |
| file\_content | Содержимое файла |
| project\_id (FK) | Идентификатор проекта, к которому относится файл |
| created\_at | Дата и время создания файла |
| updated\_at | Дата и время последнего обновления файла |

Таблица 5 - Commits

|  |  |
| --- | --- |
| Название | Описание |
| commit\_id (PK) | Уникальный идентификатор коммита |
| project\_id (FK) | Идентификатор проекта, к которому относится коммит |
| user\_id (FK) | Идентификатор пользователя, создавшего коммит |
| commit\_message | Сообщение коммита |
| commit\_hash | Хеш коммита |
| created\_at | Дата и время создания коммита |

Таблица 6 - Issues

|  |  |
| --- | --- |
| Название | Описание |
| issue\_id (PK) | Уникальный идентификатор задачи |
| project\_id (FK) | Идентификатор проекта, к которому относится задача |
| title | Название задачи |
| description | Описание задачи |
| status | Статус задачи (например, открыта, в процессе, закрыта). |
| created\_by (FK) | Идентификатор пользователя, создавшего задачу |
| assigned\_to (FK) | Идентификатор пользователя, создавшего задачу |
| created\_at | Дата и время создания задачи |
| updated\_at | Дата и время последнего обновления задачи |

Таблица 7- TestResults

|  |  |
| --- | --- |
| Название | Описание |
| test\_result\_id (PK) | Уникальный идентификатор результата тестирования |
| project\_id (FK) | Идентификатор проекта, к которому относится результат тестирования |
| test\_name | Имя теста |
| status | Статус теста (например, успешно, неуспешно) |
| run\_at | Дата и время выполнения теста |
| log | Лог выполнения теста |

Таблица 8 - Deployments

|  |  |
| --- | --- |
| Название | Описание |
| deployment\_id (PK) | Уникальный идентификатор развертывания |
| project\_id (FK) | Идентификатор проекта, к которому относится развертывание |
| deployed\_by (FK) | Идентификатор пользователя, выполнившего развертывание |
| environment | Среда развертывания (например, тестовая, продуктивная) |
| status | Статус развертывания |
| deployed\_at | Дата и время развертывания |
| log | Лог развертывания |

На рисунке 6, изображена схема базы данных.



Рисунок 6 – ER-диаграмма в нотации Crow’s Foot

Нотация Crow's Foot (англ. Crow's Foot Notation) — один из методов представления сущностей и их взаимосвязей в диаграммах "сущность-связь" (ERD, Entity-Relationship Diagram). Эта нотация была разработана в 1970-х годах и получила своё название из-за характерного символа в виде "воронки" (Crow's Foot), который используется для обозначения связей "один ко многим" и "многие ко многим".

Нотация Crow's Foot широко используется в моделировании баз данных благодаря своей наглядности и интуитивности.

Crow's Foot была разработана Гордоном Эверестом (Gordon Everest) в 1970-х годах в рамках его работы по моделированию данных. Первоначально она была представлена в его книге "Data Base Management: Theory and Application" в 1976 году. Целью создания этой нотации было упрощение визуализации и анализа сложных структур данных, а также улучшение коммуникации между разработчиками и аналитиками.

Использование нотации Crow's Foot обусловлено несколькими ключевыми преимуществами:

* Наглядность. Нотация Crow's Foot предоставляет чёткое и легко читаемое представление взаимосвязей между сущностями, что упрощает понимание структуры базы данных;
* Интуитивность. Символы и обозначения в нотации легко воспринимаются и запоминаются, что делает её удобной для работы как для технических специалистов, так и для людей без глубоких технических знаний;
* Широкое распространение. Благодаря своей популярности и стандартности, нотация Crow's Foot используется в большинстве инструментов для моделирования данных, что облегчает совместную работу и обмен информацией между различными командами и проектами;

Эта структура поддерживает все необходимые функции системы, включая управление пользователями, проектами, файлами, коммитами, задачами, результатами тестирования и развертываниями.

## 3.3.2 Оптимизация запросов и структуры данных

Оптимизация запросов и структуры данных является ключевым аспектом разработки эффективных и высокопроизводительных информационных систем. Целью оптимизации является обеспечение быстрого доступа к данным, снижение времени выполнения запросов и минимизация потребления ресурсов системы. В данном разделе будет рассмотрена оптимизация структуры базы данных и запросов на примере интегрированной среды разработки (IDE) для Java.

Оптимизация структуры данных была проведена с помощью таких операций как:

* Нормализация данных. Таблицы находятся как минимум в третьей нормальной форме для устранения избыточности и предотвращения аномалий при обновлении данных;
* Индексация. Создание индексов для полей, которые часто используются в условиях WHERE, JOIN и ORDER BY;
* Кластеризация данных. Распределение данных по кластеризованным индексам для улучшения производительности запросов;
* Разделение таблиц. Разделение больших таблиц на более мелкие секции для ускорения доступа к данным;

Для наилучшей оптимизации запросов были использованы такие средства:

* Использование объединений. Оптимизация запросов с помощью JOIN для уменьшения времени выполнения операции;
* Кэширование результатов запросов. Использование кэша для хранения результатов часто выполняемых запросов;
* Переписывание подзапросов в виде JOIN для повышения производительности;
* Оптимизация UPDATE и DELETE операций. Использование WHERE условий для ограничения объема изменяемых данных;

# 3.4 Разработка интерфейса

Интерфейс пользователя играет ключевую роль в интегрированной среде разработки для Java, так как он является основным средством взаимодействия разработчика с инструментами и функциями системы.

Разработка интерфейса включает создание интуитивно понятного и удобного интерфейса, который способствует повышению продуктивности пользователей и минимизации ошибок.

Компоненты интерфейса состоят из:

* Редактор кода. Основной компонент IDE, предоставляющий функции подсветки синтаксиса, автозавершения, рефакторинга и поиска;
* Панель проекта. Отображает структуру проекта, включая файлы и папки, и обеспечивает навигацию по проекту;
* Панель отладки. Предоставляет инструменты для установки точек останова, просмотра значений переменных и выполнения кода пошагово;
* Консоль вывода. Отображает результаты выполнения программы, сообщения об ошибках и системные логи;
* Инструменты управления версиями. Интеграция с системами контроля версий, такими как Git, для управления коммитами, ветками и слияниями.
* Панель задач и уведомлений. Отображает текущие задачи, уведомления и напоминания для пользователя.

Для успешного пользовательского опыта были оптимизированы и реализованы:

* Использование асинхронных операций для минимизации времени отклика интерфейса на действия пользователя.
* Функции автоматического сохранения изменений и восстановления незавершенной работы при сбоях.
* Встроенные подсказки и обучающие материалы для пользователей, позволяющие быстро освоить функции интерфейса.
* Возможность выбора и настройки тем оформления для улучшения визуального восприятия и снижения утомляемости глаз.

На рисунках 7, 8 и 9 отображен прототип интерфейса для интегрированной среды разработки для Java.

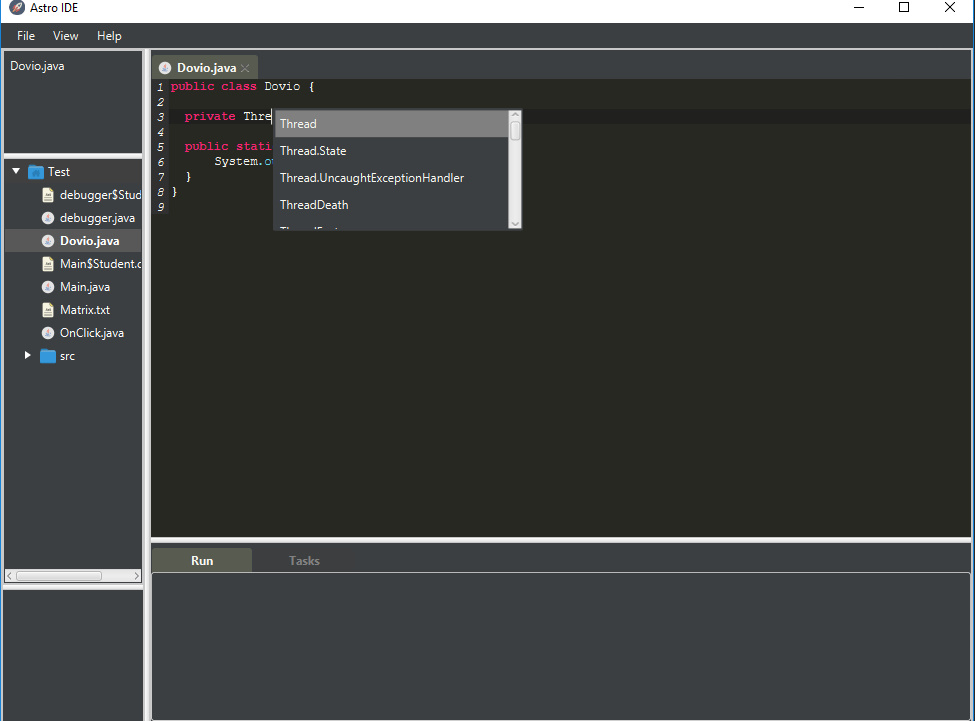


Рисунок 7 – Прототип авто-заполнения

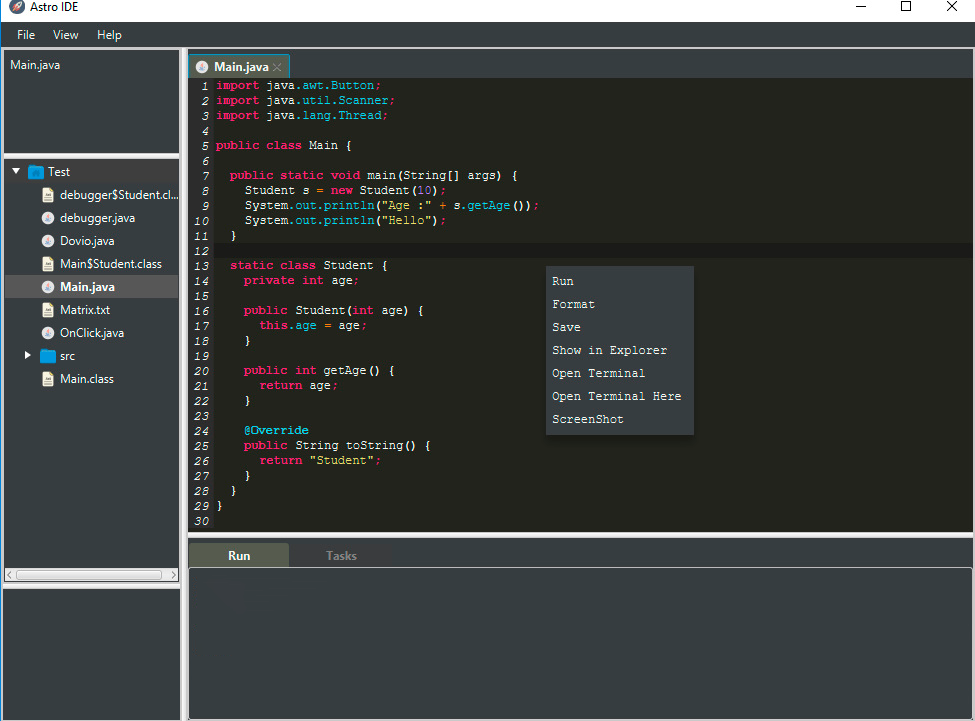


Рисунок 8 – Прототип контекстного меню

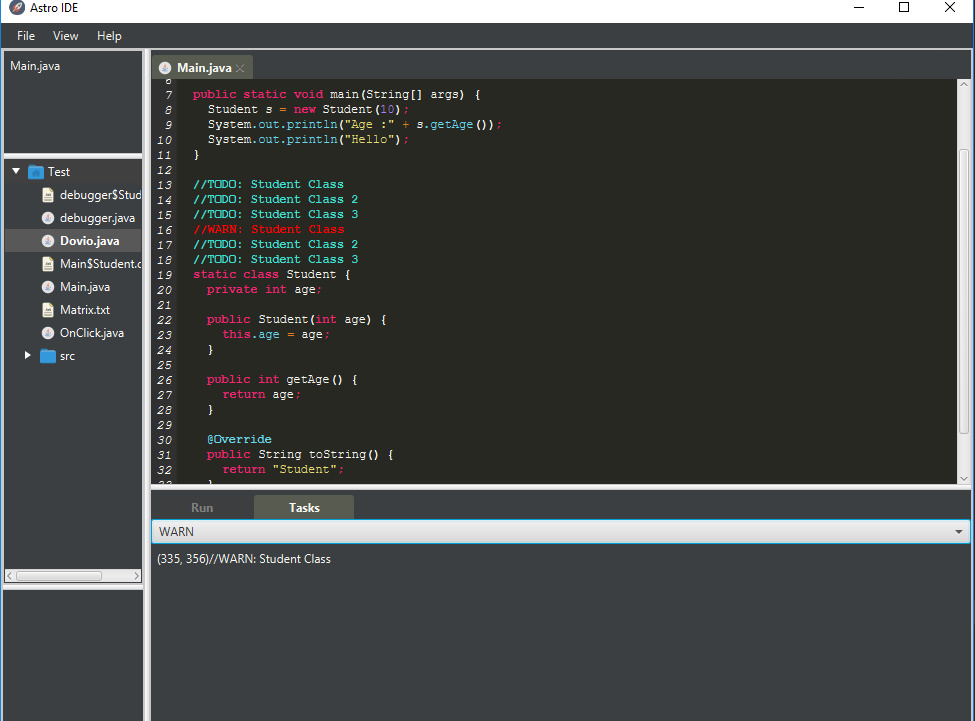


Рисунок 9 – Прототип уведомления об ошибке

# 3.5 Разработка логики работы

Разработка логики работы интегрированной среды разработки (IDE) для Java является важным этапом, включающим проектирование и реализацию внутренних механизмов, которые обеспечивают функционирование всех компонентов системы. В ходе выполнения этой работы я подробно исследовал и разработал логику обработки пользовательских действий, управления данными, выполнения компиляции и тестирования, а также интеграции с внешними системами.

## 3.5.1 Обработка пользовательских действий

Для создания удобной и эффективной среды разработки необходимо обеспечить быструю и надежную обработку пользовательских действий. В процессе исследования были проанализированы основные сценарии взаимодействия пользователя с IDE, такие как редактирование кода, выполнение компиляции, запуск тестов и управление проектами.

При редактировании кода пользователем, система мгновенно обновляет интерфейс редактора, отображая подсветку синтаксиса и авто-завершение. Для этого был разработан компонент, который отслеживает изменения в коде и применяет соответствующие стили и подсказки в реальном времени.

При сохранении изменений система записывает данные в базу данных и обновляет состояние проекта в системе контроля версий. Для этого используется асинхронная обработка запросов, что позволяет минимизировать задержки и обеспечить плавное взаимодействие.

В случае выполнения компиляции, пользователь инициирует процесс через интерфейс IDE. Система передает команду компилятору, который обрабатывает исходный код, проверяя его на наличие ошибок. Если ошибки обнаружены, пользователю предоставляется подробная информация для их исправления. Этот процесс также реализован с использованием асинхронной обработки, что позволяет пользователю продолжать работу, не дожидаясь завершения компиляции.

## 3.5.2 Управление данными

Эффективное управление данными является ключевым аспектом логики работы IDE. Были исследованы и внедрены механизмы взаимодействия с базой данных для обеспечения надежного хранения, обновления и извлечения информации. Основное внимание уделялось целостности и согласованности данных.

Все операции с данными выполняются в рамках транзакций, что гарантирует их атомарность и предотвращает частичное применение изменений в случае сбоя. Например, при создании нового проекта система создает запись в таблице проектов и инициализирует соответствующие директории и файлы в файловой системе. Все изменения завернуты в транзакцию, что обеспечивает целостность данных и возможность отката в случае ошибки.

Для обеспечения быстрого доступа к данным были внедрены индексации ключевых полей, что позволило значительно сократить время выполнения запросов. Также была реализована система кэширования для часто запрашиваемых данных, что дополнительно повысило производительность системы.

## 3.5.3 Выполнение компиляции и тестирования

Компиляция и тестирование являются основными процессами в IDE для Java. Было проведено исследование и разработан механизм, обеспечивающий последовательное выполнение этих процессов, управление зависимостями и обработку ошибок.

Процесс компиляции включает преобразование исходного кода в байт-код Java с последующей проверкой на наличие ошибок. При обнаружении ошибок система предоставляет пользователю подробную информацию для их исправления. Для этого был разработан компонент, который анализирует выходные данные компилятора и отображает их в удобном для пользователя виде.

После успешной компиляции может быть выполнен запуск тестов. Их реализует модуль, который автоматически запускает тесты, собирает результаты и представляет их пользователю в виде отчетов. В случае неудачного выполнения тестов система предоставляет информацию о причинах сбоя, что позволяет пользователю быстро их исправить. Один из методов реализующий компиляцию в проекте изображён на рисунке 10.

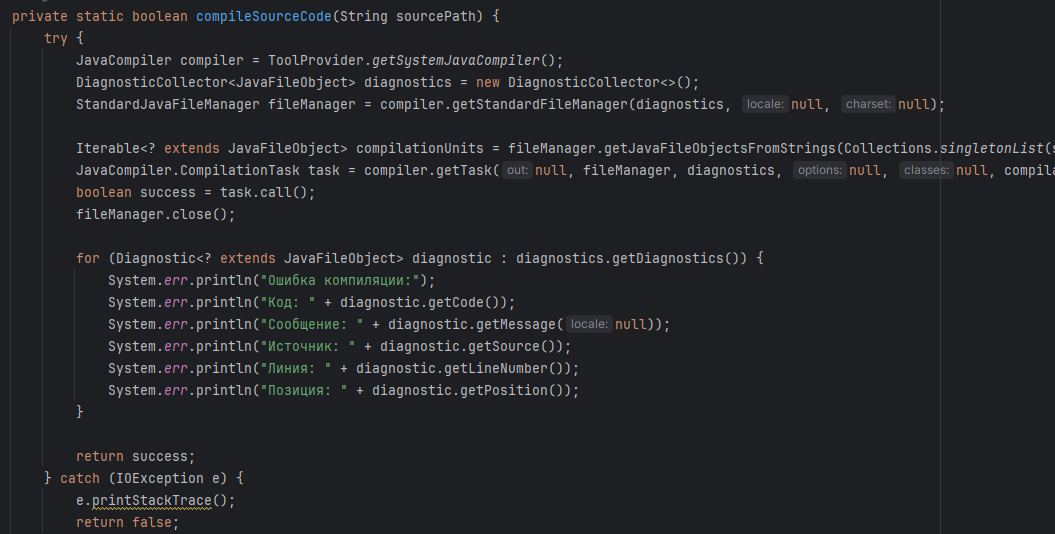


Рисунок 10 – Метод компиляции кода

## 3.5.4 Интеграция с внешними системами

Для полноценного функционирования IDE необходимо обеспечить интеграцию с внешними системами, такими как системы контроля версий, CI/CD инструменты, базы данных и облачные сервисы. Были разработаны механизмы взаимодействия с этими системами через API, обеспечивая безопасный и надежный обмен данными.

Интеграция с системой контроля версий (Git) позволяет пользователям выполнять коммиты, управлять ветками и разрешать конфликты непосредственно из интерфейса IDE. Данный компонент, который обеспечивает синхронизацию данных между локальным хранилищем и удаленным репозиторием, а также предоставляет пользователю уведомления о статусе операций.

Для интеграции с CI/CD инструментами было реализовано API взаимодействие, которое позволяет автоматически запускать процессы сборки и развертывания приложения. Это обеспечивает непрерывную интеграцию и развертывание, что способствует улучшению качества программного обеспечения и снижению времени выхода на рынок. Некоторые методы для реализации отображены на рисунках 11, 12, 13 и 14.

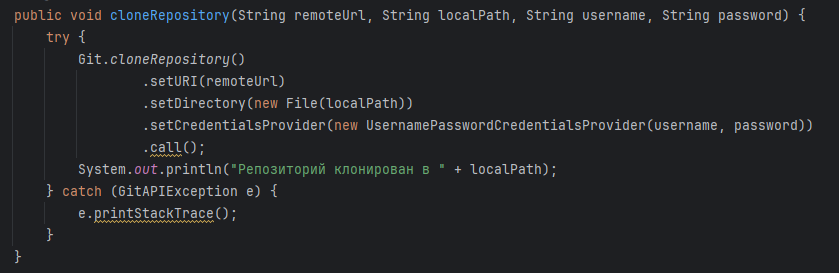


Рисунок 11 – Реализация клонирования репозитория

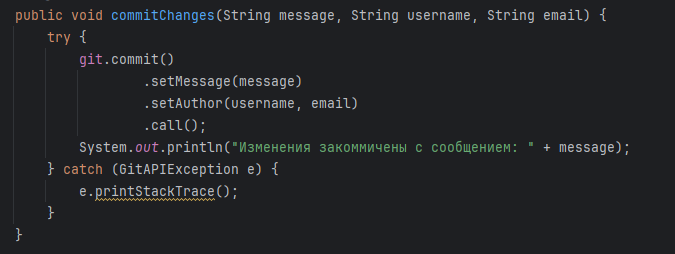


Рисунок 12 – Реализация коммитов

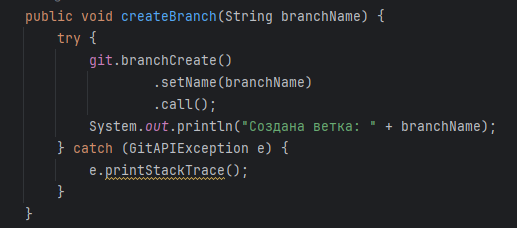


Рисунок 13 – Реализация функции создания веток

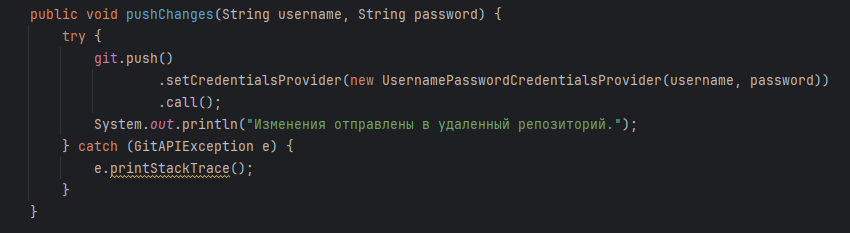


Рисунок 14 – Реализация функции push

## 3.5.5 Интеграция с другими инструментами и сервисами

Интеграция с внешними инструментами и сервисами является важной составляющей для создания функциональной и эффективной интегрированной среды разработки для Java. В условиях современных требований к импортозамещению и использования отечественных продуктов, особое внимание уделяется интеграции с российскими решениями, такими как ИИ от СберТех и Яндекса для помощи в программирование, а также иными отечественными аналогами. В этом разделе были представлены результаты исследования и реализации интеграции IDE с различными инструментами и сервисами.

## 3.5.6 Интеграция с системами контроля версий

Система контроля версий (VCS) является неотъемлемой частью процесса разработки программного обеспечения. В ходе работы была внедрена интеграция с различными системами контроля версий, включая как международные, так и отечественные решения.

Для интеграции с Git был разработан компонент, который позволяет пользователям выполнять основные операции, такие как коммиты, управление ветками и разрешение конфликтов, непосредственно из интерфейса IDE.

## 3.5.7 Интеграция с CI/CD инструментами

Непрерывная интеграция и развертывание (CI/CD) играют ключевую роль в автоматизации процесса сборки, тестирования и развертывания приложений. Была исследована и реализована интеграция с различными CI/CD инструментами, включая отечественные решения.

Для интеграции с Jenkins и GitLab CI был разработан модуль, который позволяет автоматически запускать процессы сборки и развертывания приложения. Особое внимание было уделено интеграции с CI/CD системами, такими как TeamCity от JetBrains.

Интеграция с CI/CD инструментами включает следующие ключевые функции:

* Сборка кода. Автоматическая сборка проекта при каждом коммите;
* Тестирование. Автоматическое выполнение тестов для проверки корректности кода;
* Развертывание. Автоматическое развертывание приложения в тестовую или продуктивную среду;
* Мониторинг. Отслеживание состояния сборок и уведомление о результатах;

На рисунке 15 отображён метод реализации интеграции с GitLab CI.



Рисунок 16 – Интеграция с GitLab CI

## 3.5.8 Интеграция с облачными сервисами

Облачные сервисы обеспечивают масштабируемость и гибкость, необходимые для современных разработок.

Для интеграции с облачными платформами, такими как AWS, Azure и отечественной платформой Yandex Cloud, был разработан модуль, который позволяет управлять развертыванием приложений и ресурсами непосредственно из интерфейса IDE.

Интеграция с облачными сервисами включает следующие ключевые функции:

* Развертывание приложений. Автоматическое развертывание приложений на облачной инфраструктуре;
* Мониторинг. Отслеживание состояния ресурсов и производительности приложений;

На рисунке 17 отображена работа с ключами облачных сервисов.

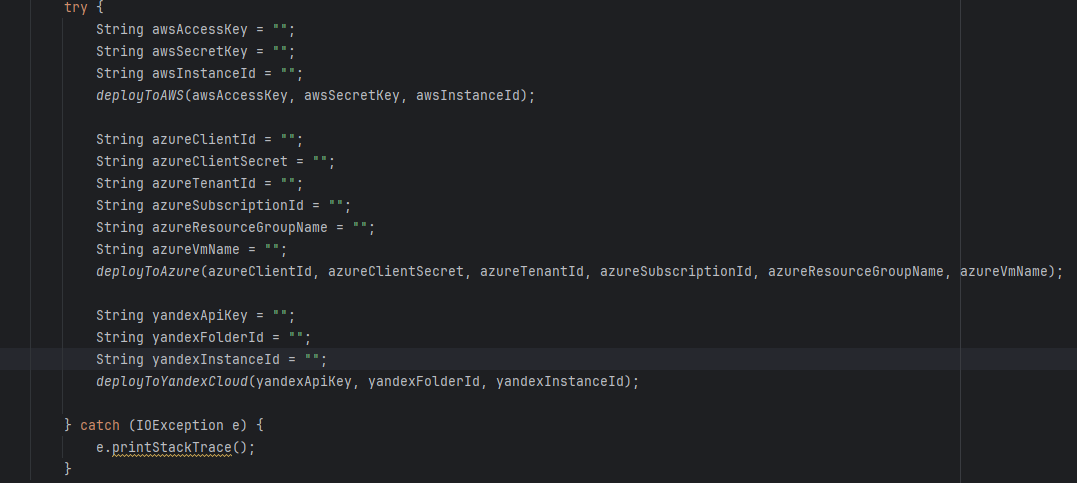


Рисунок 17 – Интеграция с облачными сервисами.

## 3.5.9 Интеграция с инструментами анализа и помощи в разработке

Современные разработки активно используют инструменты на базе искусственного интеллекта для анализа кода и помощи в программировании. Я исследовал возможности интеграции с отечественными решениями, такими как СберИИ.

Для интеграции с GigaChat был разработан API-модуль, который позволяет пользователям получать рекомендации и автоматические исправления кода. Эти инструменты помогают повысить качество кода и ускорить процесс разработки.

## 3.5.10 Интеграция с системами управления проектами

Эффективное управление проектами требует интеграции IDE с системами управления проектами. В ходе работы была реализована интеграция с отечественными и международными решениями.

Для интеграции с Jira и отечественной системой управления проектами "Мегаплан" был разработан модуль, который позволяет пользователям управлять задачами, отслеживать прогресс и получать уведомления непосредственно из интерфейса IDE.

Интеграция с системами управления проектами включает следующие ключевые функции:

* Управление задачами. Создание, назначение и отслеживание задач.
* Отчеты и аналитика. Генерация отчетов по проектам и задачам.
* Уведомления. Получение уведомлений о статусе задач и важных событиях.
* Календарное планирование. Планирование сроков выполнения задач и контроль дедлайнов.

# 3.6 Тестирование и отладка

Тестирование и отладка являются важнейшими этапами разработки программного обеспечения, позволяющими обеспечить качество и надежность создаваемого продукта. Эти процессы направлены на выявление и исправление ошибок, проверку соответствия программного обеспечения требованиям и оценку его производительности. В ходе выполнения данной выпускной квалификационной работы я подробно исследовал и применил различные методы и подходы к тестированию интегрированной среды разработки (IDE) для Java. Целью данных процессов является не только выявление дефектов, но и обеспечение соответствия системы требованиям пользователей, улучшение её производительности и надежности.

Тестирование включает в себя широкий спектр методов, начиная от начальных проверок стабильности системы (smoke-тестирование) до сложных сценариев, охватывающих различные функциональные и нефункциональные аспекты системы. Отладка же позволяет не только выявить, но и исправить ошибки в программном коде, улучшить его качество и производительность. В данном разделе будет представлено методы и подходы к тестированию, используемые в разработке IDE для Java, и опишу процесс отладки.

## 3.6.1 Методы и подходы к тестированию

Тестирование программного обеспечения — это процесс проверки системы на наличие ошибок и соответствие её поведения требованиям. Основные цели тестирования включают обеспечение качества продукта, предотвращение дефектов, улучшение производительности и надежности. Тестирование включает различные методы и подходы, которые позволяют проверять функциональность, производительность, безопасность и удобство использования системы.

## 3.6.2 Тестирование с использованием классов эквивалентности

Для проведения тестирования системы на основе описанных классов эквивалентности рассмотрим различные аспекты ввода данных пользователем в IDE, таких как имена файлов, параметры компиляции и значения переменных. Рассмотрим таблицу классов эквивалентности:

Таблица 9 – Классы эквивалентности

|  |  |  |
| --- | --- | --- |
| Наименование параметра | Допустимый класс | Недопустимый класс |
| Имена файлов | Корректные | Некорректные |
| Параметры компиляции | Валидные | Невалидные |
| Значение переменных | Допустимые | Недопустимые |

Тестовый вариант: некорректные имена файлов:

Имена файлов содержат недопустимые символы.

Ожидаемый результат: вывод сообщения об ошибке и отказ от сохранения файла.

Тестовый вариант: валидные параметры компиляции:

Все параметры компиляции заданы корректно.

Ожидаемый результат: успешная компиляция кода.

Тестовый вариант: недопустимые значения переменных:

Переменные содержат недопустимые значения (например, деление на ноль).

Ожидаемый результат: сообщение об ошибке при выполнении программы.

## 3.6.3 Тестирование с использованием базовых путей

Проанализируем процесс компиляции и тестирования кода, который является одним из ключевых процессов в IDE. Для анализа используем метод белого ящика с помощью базовых путей:

1. Открыть проект в IDE.
2. Если проект открыт успешно, перейти на шаг 3, иначе перейти к шагу
3. Начать компиляцию кода.
4. Если компиляция успешна, перейти к шагу 5, иначе перейти к шагу 6.
5. Начать выполнение юнит-тестов.
6. Если тесты прошли успешно, перейти к шагу 7, иначе вернуть результаты тестов с ошибками.
7. Показать сообщение об успешном завершении процесса и перейти к шагу 9.
8. Показать ошибку открытия проекта.
9. Конец выполнения алгоритма.

Граф алгоритма отображён на рисунке 7.

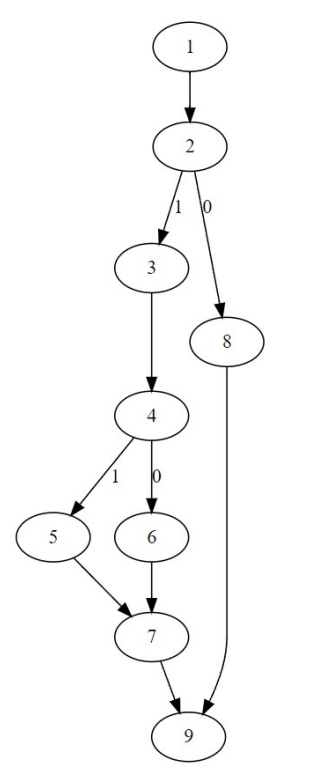


Рисунок 18 - Граф

Проведем расчеты цикломатической сложности:

V(G) = (Количество регионов) = 3.

V(G) = E(Количество дуг) – N(Количество узлов) + 2 = 10 - 9 + 2 = 3

V(G) = p(Количество предикатных узлов) + 1 = 2 + 1 = 3

Варианты путей по шагам:

* 1-2-8-9.
* 1-2-3-4-5-7-9.
* 1-2-3-4-6-7-9.

Создадим тестовые данные, которые покроют только обнаруженные пути:

* Тест по пути 1-2-8-9:
  + - На шаге 1 открыт проект в IDE, перейти к шагу 2.
    - На шаге 2 проект не был открыт, перейти к шагу 8.
    - На шаге 8 возвращена ошибка открытия проекта.
    - На шаге 9 конец выполнения алгоритма.
  + Тест по пути 1-2-3-4-5-7-9:
    - На шаге 1 открыт проект в IDE, перейти к шагу 2.
    - На шаге 2 проект был открыт, перейти к шагу 3.
    - На шаге 3 начата компиляция кода, перейти к шагу 4.
    - На шаге 4 компиляция успешна, перейти к шагу 5.
    - На шаге 5 начаты юнит-тесты, перейти к шагу 7.
    - На шаге 7 показано сообщение об успешном завершении процесса, перейти к шагу 9.
    - На шаге 9 конец выполнения алгоритма.
  + Тест по пути 1-2-3-4-6-7-9:
    - На шаге 1 открыт проект в IDE, перейти к шагу 2.
    - На шаге 2 проект был открыт, перейти к шагу 3.
    - На шаге 3 начата компиляция кода, перейти к шагу 4.
    - На шаге 4 компиляция не успешна, перейти к шагу 6.
    - На шаге 6 возвращены результаты тестов с ошибками, перейти к шагу 7.
    - На шаге 7 показано сообщение об успешном завершении процесса, перейти к шагу 9.
    - На шаге 9 конец выполнения алгоритма.

## 3.6.4 Тестирование критического пути

Тестирование критического пути фокусируется на проверке наиболее важных и часто используемых сценариев использования системы. Основные цели данного метода — гарантировать, что основные функции системы работают корректно и надежно. В таблице 10 указаны тесты для выполнения тестирования критического пути.

Таблица 10 – Тест-кейс

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Описание сценария | Шаги выполнения | Ожидаемый результат |
| 1 | Создание нового проекта | 1. Открыть IDE 2. Создать новый проект 3. Ввести имя проекта 4. Выбрать расположение 5. Нажать "Создать" | Проект успешно создан и открыт в IDE |
| 2 | Редактирование кода | 1. Открыть файл в редакторе кода 2. Внести изменения 3. Сохранить изменения 4. Проверить синтаксис | Изменения успешно сохранены, синтаксических ошибок нет |

Продолжение таблицы 10

|  |  |  |  |
| --- | --- | --- | --- |
| 3 | Компиляция проекта | 1. Открыть проект в IDE 2. Выбрать "Компилировать" | Проект успешно скомпилирован, ошибок компиляции нет |
| 4 | Запуск тестов | 1. Открыть проект в IDE 2. Выбрать "Запустить тесты" | Все тесты прошли успешно, результаты отображены в удобном виде |
| 5 | Отладка кода | 1. Открыть файл в редакторе кода 2. Установить точки остановки 3. Запустить отладку | Отладка проходит корректно, пользователь может пошагово выполнять код и просматривать значения переменных |

## 3.6.5 Интеграционное тестирование

Интеграционное тестирование направлено на проверку взаимодействия между интегрированными модулями системы. Цель этого метода — выявление дефектов, возникающих при интеграции отдельных компонентов, и обеспечение корректного обмена данными между ними. В таблице 11 указаны тесты для проведения интеграционного тестирования.

Таблица 11 – Тест-кейс

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Описание сценария | Шаги выполнения | Ожидаемый результат |
| 1 | Интеграция редактора кода и компилятора | 1. Открыть файл в редакторе кода 2. Внести изменения 3. Сохранить и компилировать | Изменения переданы компилятору, компиляция проходит без ошибок |
| 2 | Интеграция компилятора и отладчика | 1. Скомпилировать проект 2. Запустить отладку | Проект успешно компилируется и отлаживается |
| 3 | Интеграция с системой управления версиями | 1. Внести изменения в код 2. Сохранить и закоммитить изменения 3. Синхронизировать с удаленным репозиторием | Изменения успешно закоммичены и синхронизированы |
| 4 | Интеграция с инструментами тестирования | 1. Открыть проект 2. Запустить тесты 3. Проверить передачу результатов тестирования и генерацию отчетов | Тесты успешно проходят, результаты отображены в отчетах |
| 5 | Интеграция с внешними сервисами | 1. Открыть проект 2. Настроить взаимодействие с облачным сервисом 3. Запустить развертывание | Корректное взаимодействие с облачными сервисами, успешное развертывание проекта |

## 3.6.6 Системное тестирование

Системное тестирование — это комплексная проверка всей системы на соответствие требованиям. Цель системного тестирования заключается в проверке всей функциональности системы в интегрированном состоянии и оценке её готовности к выпуску.

Таблица 12 – Тест-кейс для системного тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Описание сценария | Шаги выполнения | Ожидаемый результат |
| 1 | Полный цикл разработки | 1. Создать новый проект 2. Написать и сохранить код 3. Скомпилировать проект 4. Запустить тесты | Все этапы проходят успешно без ошибок |
| 2 | Обработка ошибок и исключений | 1. Внести ошибку в код 2. Сохранить и скомпилировать проект | Компилятор выявляет ошибку, система отображает сообщение об ошибке |
| 3 | Производительность системы | 1. Запустить несколько проектов одновременно 2. Выполнить компиляцию и тестирование | Система справляется с нагрузкой, время отклика приемлемо |
| 4 | Совместимость и переносимость | 1. Проверить работу системы на различных платформах (Windows, macOS, Linux) 2. Проверить совместимость с различными версиями Java | Система корректно работает на всех платформах и совместима с различными версиями Java |
| 5 | Безопасность и доступность | 1. Проверить контроль доступа и аутентификацию пользователей 2. Проверить доступность системы при различных сценариях | Система обеспечивает контроль доступа и аутентификацию, остается доступной при различных сценариях |

## Приемочное тестирование

Приемочное тестирование проводится для оценки системы с точки зрения конечного пользователя и проверки соответствия бизнес-требованиям.

Цель данного тестирования — убедиться, что система удовлетворяет потребностям пользователей и готова к развертыванию.

Необходимые тесты указаны в таблице 13.

Таблица 13 – Тест-кейс для приемочного тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| № теста | Описание сценария | Шаги выполнения | Ожидаемый результат |
| 1 | Удобство создания проектов. | 1. Открыть IDE 2. Создать новый проект | Процесс создания проекта интуитивно понятен, проект создается без проблем |
| 2 | Удобство редактирования кода. | 1. Открыть файл в редакторе кода 2. Внести изменения 3. Сохранить изменения | Редактирование и сохранение кода проходят без проблем, интерфейс удобен для пользователя |
| 3 | Удобство отладки кода | 1. Открыть файл в редакторе кода 2. Установить точки остановки 3. Запустить отладку | Отладка проходит корректно, пользователь может легко следить за выполнением кода и изменять точки остановки |

# 4. Документация пользователя

## 4.1 Введение

Данная документация предназначена для пользователей интегрированной среды разработки (IDE) для Java, разработанной в рамках выпускной квалификационной работы на тему «Проектирование интегрированной среды разработки для Java». В документации описаны основные функции и возможности среды, а также приведены инструкции по установке, настройке и использованию IDE.

## 4.2 Установка

Для установки интегрированной среды разработки (IDE) для Java необходимо выполнить следующие шаги:

* Скачайте установочный файл с официального сайта.
* Запустите установочный файл.
* Следуйте инструкциям мастера установки, выбирая необходимые параметры.
* После завершения установки запустите IDE, используя соответствующий ярлык на рабочем столе или в меню «Пуск».

## 4.3 Настройка

После установки рекомендуется выполнить начальную настройку среды разработки:

* Откройте настройки IDE через меню «Файл» -> «Настройки».
* В разделе «Язык и фреймворки» выберите версию Java SDK и настройте параметры компиляции.
* В разделе «Редактор» настройте параметры подсветки синтаксиса, автозавершения и рефакторинга кода.
* В разделе «Интеграция с системой управления версиями» укажите параметры подключения к репозиториям Git.
* В разделе «Инструменты» настройте параметры автоматической сборки и развертывания.

## 4.4 Основные функции

1. Редактор кода предоставляет следующие возможности:

* Подсветка синтаксиса для языка Java.
* Автозавершение кода.
* Встроенные функции рефакторинга.
* Поддержка навигации по коду.
* Компиляция и выполнение

2. Для компиляции и выполнения кода используйте следующие шаги:

* Откройте проект или файл с исходным кодом.
* Нажмите кнопку «Компилировать» на панели инструментов или выберите соответствующую опцию в меню «Сборка».
* Для запуска программы нажмите кнопку «Запустить» на панели инструментов или выберите соответствующую опцию в меню «Выполнение».

3. Управление проектами включает в себя следующие шаги:

* Создание нового проекта.
* Импорт существующего проекта.
* Управление зависимостями проекта.
* Настройка параметров сборки.
* Интеграция с системами управления версиями

4. Для работы с системами управления версиями (Git) доступны следующие возможности:

* Клонирование репозиториев.
* Создание и управление ветками.
* Выполнение коммитов и слияний.
* Просмотр истории изменений и разрешение конфликтов.

5. Для тестирования и отладки кода используйте следующие возможности:

* Установка точек остановки.
* Пошаговое выполнение кода.
* Просмотр и изменение значений переменных.
* Запуск и управление юнит-тестами.
* Автоматизация сборки и развертывания

6. Для автоматизации процессов сборки и развертывания используйте следующие функции:

* Настройка автоматической сборки проекта при изменении кода.
* Интеграция с системами CI/CD (например, Jenkins).
* Настройка параметров развертывания на серверы и в облачные среды.

# Заключение

В ходе выполнения данной выпускной квалификационной работы была рассмотрена тема "Проектирование интегрированной среды разработки для Java". В процессе анализа существующих решений и требований к современной интегрированной среде разработки были выявлены ключевые недостатки, которые могут препятствовать эффективной работе разработчиков. Одним из важнейших моментов было выявление тех аспектов, которые могут быть улучшены за счет создания новой, более функциональной и удобной интегрированной среды разработки.

Было проведено тщательное изучение и анализ существующих IDE, таких как IntelliJ IDEA, Eclipse и NetBeans. В результате проведенных исследований были определены основные преимущества и недостатки этих инструментов. Основным принципом при разработке новой интегрированной среды разработки было объединение лучших качеств существующих решений и добавление инновационных возможностей для улучшения процесса разработки программного обеспечения.

Одной из основных целей данной работы было создание интегрированной среды разработки, которая поддерживает все этапы жизненного цикла разработки на языке Java, включая написание, компиляцию, отладку, тестирование и деплой. Для этого было спроектировано и реализовано множество модулей, таких как редактор исходного кода с функциями подсветки синтаксиса и автозавершения, компилятор Java, отладчик, система управления версиями, инструменты для тестирования и автоматизации деплоя.

Кроме того, была разработана серверная часть, включающая компоненты для управления базами данных, API Gateway и модуль аутентификации, обеспечивающие надежное и эффективное взаимодействие между клиентской и серверной частями системы. Эти серверные компоненты были реализованы с использованием Spring Boot и PostgreSQL, что обеспечило высокую производительность и масштабируемость системы.

Результаты, полученные в ходе разработки и тестирования интегрированной среды разработки, показали, что новая IDE значительно упрощает и ускоряет процесс разработки программного обеспечения на Java. Она предоставляет разработчикам мощные инструменты для написания и отладки кода, управления версиями и автоматизации тестирования и деплоя, что позволяет значительно повысить их продуктивность и качество создаваемых приложений.

Таким образом, данная работа может стать полезным руководством для разработчиков и инженеров, работающих над созданием и улучшением интегрированных сред разработки. Она демонстрирует подходы и методологии, которые могут быть использованы для разработки современных и эффективных инструментов для программирования, способных удовлетворить требования быстро меняющегося технологического ландшафта.

# Список использованных источников

1. ГОСТ 7.32-2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления (введен в действие Приказом Росстандарта от 24.10.2017 N 1494-ст)

2. Технологии разработки информационных систем. Лекции.// Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина». Автор: Громов А.Ю. – 20201.

3. Бизнес-анализ. Лекции.// Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина». Автор: Громов А.Ю. – 2023.

4. Моделирование бизнес-процессов. Лекции.// Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Рязанский государственный радиотехнический университет имени В.Ф. Уткина». Автор: Громов А.Ю. – 2021.

5. Гамма, Э., Хелм, Р., Джонсон, Р., Влиссидес, Дж. (1995). Приемы объектно-ориентированного проектирования. Паттерны проектирования.

6. Фаулер, М. (2006). Архитектура корпоративных программных приложений. Москва: Вильямс.

7. Блох, Дж. (2018). Java. Эффективное программирование. Второе издание.

8. Фримен, Э., Робсон, Э. (2005). Паттерны проектирования.

9. Репин В. Бизнес-процессы. Моделирование, внедрение, управление/В. Репин. – М.: Манн, Иванов и Фербер, 2013

10. Гордеев, С. И. Организация баз данных в 2 ч. Часть 1: учебник для вузов / С. И. Гордеев, В. Н. Волошина. — 2-е изд., испр. и доп. — Москва: Издательство Юрайт, 2021. — 310 с.

11. Данилин, А. В. Архитектура предприятия: учебное пособие / А. В. Данилин, А. И. Слюсаренко. — 3-е изд. — Москва: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. — 439 c.

12. Каменнова, М. С. Моделирование бизнес-процессов. В 2 ч. Часть 2: учебник и практикум для вузов / М. С. Каменнова, В. В. Крохин, И. В. Машков. — Москва : Издательство Юрайт, 2021. — 228 с. — (Высшее образование).

13. Джонсон, К., Хоар, Т. (2013). Микросервисы: от монолитов к облакам.

14. Саймон Браун (2018). Современная архитектура программных систем.

15. Томпсон, Р. (2019). Spring Boot: руководство по созданию приложений.

16. Позняк, А. (2017). PostgreSQL для профессионалов.

17. Вейз, Л., Кили, П. (2005). JUnit in Action.

18. Kim Moir (2018). The Architecture of Open Source Applications Eclipse.

# Техническая документация

# Исходный код

*Скрипт создания БД*

CREATE TABLE Users (

user\_id SERIAL PRIMARY KEY,

username VARCHAR(50) NOT NULL UNIQUE,

password\_hash VARCHAR(255) NOT NULL,

email VARCHAR(100) NOT NULL UNIQUE,

role VARCHAR(50) NOT NULL

);

CREATE TABLE Projects (

project\_id SERIAL PRIMARY KEY,

project\_name VARCHAR(100) NOT NULL,

created\_by INT NOT NULL REFERENCES Users(user\_id),

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE Files (

file\_id SERIAL PRIMARY KEY,

file\_name VARCHAR(255) NOT NULL,

file\_content TEXT NOT NULL,

project\_id INT NOT NULL REFERENCES Projects(project\_id) ON DELETE CASCADE,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

CREATE TABLE Commits (

commit\_id SERIAL PRIMARY KEY,

project\_id INT NOT NULL REFERENCES Projects(project\_id) ON DELETE CASCADE,

user\_id INT NOT NULL REFERENCES Users(user\_id),

commit\_message TEXT NOT NULL,

commit\_hash VARCHAR(255) NOT NULL UNIQUE,

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP

);

CREATE TABLE Issues (

issue\_id SERIAL PRIMARY KEY,

project\_id INT NOT NULL REFERENCES Projects(project\_id) ON DELETE CASCADE,

title VARCHAR(255) NOT NULL,

description TEXT NOT NULL,

status VARCHAR(50) NOT NULL,

created\_by INT NOT NULL REFERENCES Users(user\_id),

assigned\_to INT REFERENCES Users(user\_id),

created\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

updated\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP ON UPDATE CURRENT\_TIMESTAMP

);

CREATE TABLE TestResults (

test\_result\_id SERIAL PRIMARY KEY,

project\_id INT NOT NULL REFERENCES Projects(project\_id) ON DELETE CASCADE,

test\_name VARCHAR(255) NOT NULL,

status VARCHAR(50) NOT NULL,

run\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

log TEXT NOT NULL

);

CREATE TABLE Deployments (

deployment\_id SERIAL PRIMARY KEY,

project\_id INT NOT NULL REFERENCES Projects(project\_id) ON DELETE CASCADE,

deployed\_by INT NOT NULL REFERENCES Users(user\_id),

environment VARCHAR(50) NOT NULL,

status VARCHAR(50) NOT NULL,

deployed\_at TIMESTAMP NOT NULL DEFAULT CURRENT\_TIMESTAMP,

log TEXT NOT NULL

);

CREATE INDEX idx\_user\_username ON Users(username);

CREATE INDEX idx\_project\_created\_by ON Projects(created\_by);

CREATE INDEX idx\_file\_project\_id ON Files(project\_id);

CREATE INDEX idx\_commit\_project\_id ON Commits(project\_id);

CREATE INDEX idx\_issue\_project\_id ON Issues(project\_id);

CREATE CLUSTERED INDEX idx\_project\_id ON Files(project\_id);