**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 7](#_Toc187267412)

[1 Постановка задачи 8](#_Toc187267413)

[2 Существующие аналоги 9](#_Toc187267414)

[2.1 Обзор возможностей существующих аналогов 9](#_Toc187267415)

[2.2 Обзор способов автоматизированной проверки решений задач 15](#_Toc187267416)

[3 Обзор литературы 17](#_Toc187267417)

[3.1 Безопасность выполнения недоверенного кода 17](#_Toc187267418)

[ЗАКЛЮЧЕНИЕ 19](#_Toc187267419)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 20](#_Toc187267420)

ВВЕДЕНИЕ

В современном мире информационные технологии играют ведущую роль в развитии различных отраслей экономики и науки. Следовательно, спрос на IT-специалистов, включая разработчиков программного обеспечения, постоянно растет. В связи с этим, возникает необходимость в подготовке новых специалистов в области программирования. Образовательные информационные системы с алгоритмическими задачами по программированию становятся неотъемлемой частью этого процесса подготовки. Эти системы предоставляют студентам и начинающим программистам возможность развивать свои навыки, решая разнообразные задачи с различным уровнем сложности. Благодаря таким системам студенты могут улучшить свои навыки алгоритмизации, научиться эффективно решать задачи и практиковаться в программировании на различных языках. Более того, образовательные информационные системы обеспечивают доступ к обучающим материалам, а также предоставляют возможность соревноваться с другими пользователями, что мотивирует студентов к развитию и позволяет им оценить свой прогресс в сравнении с другими. Такие системы также часто предлагают системы автоматической проверки решений, что позволяет студентам получать обратную связь и исправлять ошибки, улучшая свои навыки программирования.

# **1 Анализ предметной области**

Для формирования требований к информационной системе обучения программированию был проведён анализ существующих решений и рассмотрен типовой сценарий использования платформы со стороны обучающегося.

Анализ пользовательского сценария взаимодействия с системой

Сценарий использования системы

Пользователь, обладающий базовыми навыками программирования, стремится улучшить свои компетенции в данной области. Его мотивация может быть связана с подготовкой к собеседованию, изучением нового языка программирования или закреплением ранее изученных концепций. В поисках подходящей платформы пользователь находит онлайн-ресурс (например, Codewars) и приступает к работе. На платформе пользователь проходит следующие этапы.

Регистрация и вход в систему. Пользователь авторизуется с использованием электронной почты и пароля, получая доступ к личному кабинету.

Выбор задачи для решения. На странице платформы представлен список задач, доступных для решения. Каждая задача сопровождается описанием, тегами, а также указанием поддерживаемых языков программирования. Пользователь выбирает интересующую задачу и переходит к её решению.

Решение задачи и проверка ответа. Пользователь пишет программный код в интегрированном редакторе и отправляет его на автоматическую проверку. Результат проверки формируется и возвращается в течение примерно 1–2 секунд.

Пример полученного отчёта:

* Время выполнения: 757 мс
* Тесты пройдены: 0
* Тесты не пройдены: 41
* Выходной код: 41

После внесения правок и повторной отправки решения результаты проверки обновляются:

* Время выполнения: 852 мс
* Тесты пройдены: 41
* Тесты не пройдены: 0

Таким образом, пользователь получает обратную связь по качеству решения и может переходить к следующей задаче.

Совместная работа над задачей. Платформа поддерживает возможность совместного решения задач несколькими пользователями в онлайн-режиме. Пользователь может пригласить другого участника, отправив ссылку на совместную сессию. Подключившийся участник получает доступ к редактору и может вносить изменения в код. Проверку решения осуществляет только пользователь, создавший сессию (владелец комнаты).

Сценарий взаимодействия с платформой со стороны преподавателя

Рассмотрим функциональные возможности платформы с точки зрения преподавателя — пользователя, обладающего достаточным уровнем знаний для формирования заданий по программированию.

Преподаватель, ранее сам прошедший путь обучения, имеет возможность создавать новые задачи для практики обучающихся. Для этого он переходит на страницу создания задачи, где выполняет следующие действия.

Формирование задачи. В процессе создания задания преподаватель заполняет основные поля:

* Название задачи — краткий и ёмкий заголовок, отражающий суть задания.
* Описание задачи — подробное текстовое описание, поддерживающее формат Markdown для оформления (включая заголовки, списки, выделение кода и др.).
* Список тегов — ключевые слова, позволяющие классифицировать задачу по темам и уровням сложности.

Настройка параметров задачи по языкам программирования. Преподаватель указывает поддерживаемые языки программирования для задачи. Для каждого выбранного языка настраиваются следующие файлы:

* Файл с кодом полного решения — эталонное решение, необходимое для проверки корректности тестов.
* Файл с кодом шаблона для решения — минимальная заготовка кода, предоставляемая обучающемуся для начала работы.
* Файл с бойлерплейт-кодом (необязательный) — дополнительный файл, содержащий вспомогательные структуры, которые могут быть использованы при решении задачи.
* Файл с кодом тестов — скрипт, автоматически выполняющий проверку корректности решения обучающегося.

Управление задачами. После создания задачи преподаватель имеет следующие возможности:

* Сохранить задачу как черновик.
* Опубликовать задачу для общего доступа обучающимся.
* Обновить существующую задачу и повторно опубликовать её в виде новой версии.
* Снять задачу с публикации, сделав её недоступной для решения.
* Удалить задачу полностью из системы.

Каждое редактирование задачи сопровождается созданием новой ревизии, для которой фиксируются следующие данные:

* Номер ревизии.
* Дата создания.
* Автор изменений.

При повторной публикации задачи используется последняя ревизия, что обеспечивает контроль версионности и воспроизводимость материалов.

Автоматизированная проверка решений

Ключевым компонентом системы является модуль автоматизированной проверки решений, обеспечивающий обратную связь пользователям в режиме реального времени. Процесс проверки решения задачи включает следующие этапы.

Отправка задачи на проверку. При отправке решения на проверку система получает от клиента следующие данные:

* Идентификатор пользователя (userId);
* Идентификатор задачи (problemId);
* Идентификатор выбранного языка программирования (languageId);
* Исходный код решения (code).

Подготовка к выполнению проверки. Система выполняет следующие действия:

* Находит задачу в базе данных по problemId.
* Извлекает файлы с кодом задачи, включая тесты и шаблоны, соответствующие выбранному языку программирования (languageId).
* Получает конфигурацию запуска контейнера (ПОДа) для выполнения решения на указанном языке.

Запуск проверки в изолированной среде. Для обеспечения безопасности и изоляции выполнения кода создаётся контейнер (ПОД), настроенный для запуска выбранного языка программирования. В контейнер передаются:

* Код задачи (тесты и вспомогательные файлы);
* Код решения, написанный пользователем.

Выполнение проверки и анализ результатов. Внутри ПОДа запускается тестовый фреймворк, который выполняет проверку решения пользователя по заранее подготовленным тестам. По завершении выполнения система:

* Получает результаты выполнения тестов;
* Анализирует их на предмет успешного прохождения всех тестов.

Формирование и возврат результата. Система возвращает пользователю информацию о результате проверки, включая количество успешно пройденных и не пройденных тестов, а также возможные сообщения об ошибках.

Режим Code Playground

Дополнительной возможностью платформы является предоставление пользователям среды для запуска кода вне контекста конкретной задачи (Code Playground). Данная функциональность аналогична инструменту, доступному на платформе LeetCode, и предоставляет следующие возможности.

Выбор языка и написание кода. Пользователь выбирает язык программирования из поддерживаемых системой и вводит исходный код в редакторе.

Запуск кода. Система находит конфигурацию запуска ПОДа для выбранного языка; создаёт изолированный контейнер, запускает его и передаёт в него введённый пользователем код.

Получение результатов выполнения. После выполнения кода в ПОДе система возвращает пользователю консольный вывод программы, что позволяет отлаживать и тестировать код в интерактивном режиме.

**Описание предметной области и сущностей системы**

Система интерактивного обучения программированию представляет собой совокупность взаимосвязанных сущностей, каждая из которых выполняет строго определённую роль в процессе обучения, создания и проверки задач. Ниже приведены ключевые сущности, их описания и пояснения к полям.

**Сущность Пользователь (*User*)**

**Пользователь** — это зарегистрированный участник системы, который может выполнять действия в зависимости от своих ролей и прав доступа.

**Атрибуты:**

* email — строка; адрес электронной почты пользователя, используется для входа в систему и связи.
* username — строка; отображаемое имя пользователя.
* password — строка; хеш пароля для аутентификации.
* image — строка; путь или URL к изображению профиля пользователя.
* roles — массив строк; роли пользователя в системе. Возможные значения: "user" (обычный пользователь) или "admin" (администратор).
* permissions — массив строк; конкретные права пользователя (например, право на создание задач, модерацию и т.д.).

**Основной функционал:**

* Регистрация и авторизация пользователя.
* Удаление пользователя из системы (доступно администратору).
* Смена пароля и изображения профиля.

**Сущность Задача (*Problem*)**

**Задача** — основная единица контента системы, предназначенная для обучения и проверки навыков программирования.

**Атрибуты:**

* author — ссылка на сущность *User*; автор задачи.
* name — строка; название задачи.
* description — строка; описание задачи в формате Markdown.
* difficulty — строка; уровень сложности задачи (например, "легкая", "средняя", "сложная").
* tags — массив сущностей *Tag*; тематические теги задачи.
* revision — число; номер текущей ревизии задачи.
* isPublished — логический тип; признак публикации задачи (true — задача опубликована, false — находится в черновике).
* languages — массив объектов, описывающих реализацию задачи для разных языков программирования:
  + completeSolution — строка; файл с полным, правильным решением задачи.
  + initialSolution — строка; файл с шаблоном для начала решения задачи.
  + preloaded — строка; дополнительный (необязательный) файл с предзагруженным кодом.
  + tests — строка; файл с тестами для проверки решения.

**Основной функционал:**

* Создание, редактирование, удаление и публикация задач (при наличии соответствующих прав доступа).
* Просмотр списка задач.
* Получение детальной информации о задаче.

**Сущность Тег (*Tag*)**

**Тег** — метаданные, помогающие классифицировать задачи по тематикам.

**Атрибуты:**

* name — строка; название тега (например, "массивы", "рекурсия", "строки").

**Основной функционал:**

* Создание, редактирование и удаление тегов (для администратора).
* Получение списка тегов (для всех пользователей).

**Сущность Язык программирования (*Language*)**

**Язык программирования** описывает поддерживаемый язык, на котором пользователь может писать решения задач.

**Атрибуты:**

* name — строка; название языка программирования (например, "Python", "JavaScript").
* image — строка; Docker-образ, содержащий рантайм языка и настроенный тестовый фреймворк для запуска кода.
* podManifest — строка; Kubernetes-манифест для создания изолированного контейнера (ПОДа) для запуска решений.
* testRunner — строка; Java-класс, который выполняет запуск тестов для проверки решений.
* taskRunner — строка; Java-класс, выполняющий произвольный код в режиме *Code Playground*.

**Основной функционал:**

* Получение списка поддерживаемых языков.
* Интеграция новых языков в систему через механизм плагинов.

**Сущность Решение (*Solution*)**

**Решение** — код, отправляемый пользователем для проверки задачи.

**Атрибуты:**

* user — ссылка на сущность *User*; пользователь, отправивший решение.
* problem — ссылка на сущность *Problem*; задача, для которой предоставлено решение.
* language — ссылка на сущность *Language*; язык программирования решения.
* code — строка; исходный код решения.

**Основной функционал:**

* Отправка решения на автоматизированную проверку.
* Просмотр списка отправленных решений и их результатов.

**Сущность Результат проверки решения (*SolutionResult*)**

**Результат проверки решения** фиксирует исход выполнения проверки кода, отправленного пользователем.

**Атрибуты:**

* solution — ссылка на сущность *Solution*; связанное решение.
* exitCode — число; код завершения процесса выполнения решения.
* stderr — строка; стандартный поток ошибок.
* stdout — строка; стандартный поток вывода.
* timedOut — логический тип; признак того, что выполнение превысило лимит времени.
* time — число; время выполнения решения (в миллисекундах).
* completed — логический тип; признак завершения проверки.
* tests — массив объектов, каждый из которых содержит:
  + name — строка; название теста.
  + passed — логический тип; результат прохождения теста (успешно/неуспешно).

**Основной функционал:**

* Создание и хранение результатов проверки (выполняется системой автоматически).

**Сущность Комната для совместного редактирования (*CollabRoom*)**

**Комната** — виртуальное пространство для совместной работы пользователей над решением задачи.

**Атрибуты:**

* problem — ссылка на сущность *Problem*; задача, которая редактируется в комнате.
* owner — ссылка на сущность *User*; владелец комнаты.
* link — строка; уникальная ссылка для подключения к комнате.
* members — массив сущностей *User*; участники комнаты.
* code — строка; текущий код решения в комнате.

**Основной функционал:**

* Создание комнаты для совместной работы.
* Подключение участников к комнате по уникальной ссылке.
* Совместное редактирование кода.

# **2 Проектирование системы**

На основе проведённого анализа предметной области разработано проектное решение системы.

*Проектирование базы данных*

Основные сущности и их взаимосвязи выявлены, описаны и формализованы в виде схемы базы данных. Для наглядности и удобства дальнейшей реализации создана ER-диаграмма, отражающая структуру данных системы. Она представлена на рисунке 2.1.

ER-диаграмма включает все основные сущности:

* Пользователи (User)
* Задачи (Problem)
* Теги (Tag)
* Языки программирования (Language)
* Решения (Solution)
* Результаты проверки решений (SolutionResult)

Схема базы данных, представленная в виде ER-диаграммы, позволяет визуализировать и структурировать данные проекта, а также служит основой для построения реляционной модели данных в выбранной системе управления базами данных.

Изображение выглядит как текст, диаграмма, План, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.1 – ER диаграмма базы данных платформы

## **2.1 Обзор возможностей существующих аналогов**

На сегодняшний день существует множество онлайн-платформ для решения алгоритмических задач и проверки знаний в области программирования. Одной из самых известных и популярных платформ на международном уровне является «LeetCode» [1], а в России — «Информатикс» [2].

## **2.2 Обзор способов автоматизированной проверки решений задач**

Автоматизированное тестирование играет ключевую роль в проверке решений задач на платформах программирования. В данном разделе рассматриваются подходы к формату тестовых данных и процессу добавления задач на платформах «LeetCode», «CodeWars», «HackerRank» и «Информатикс».

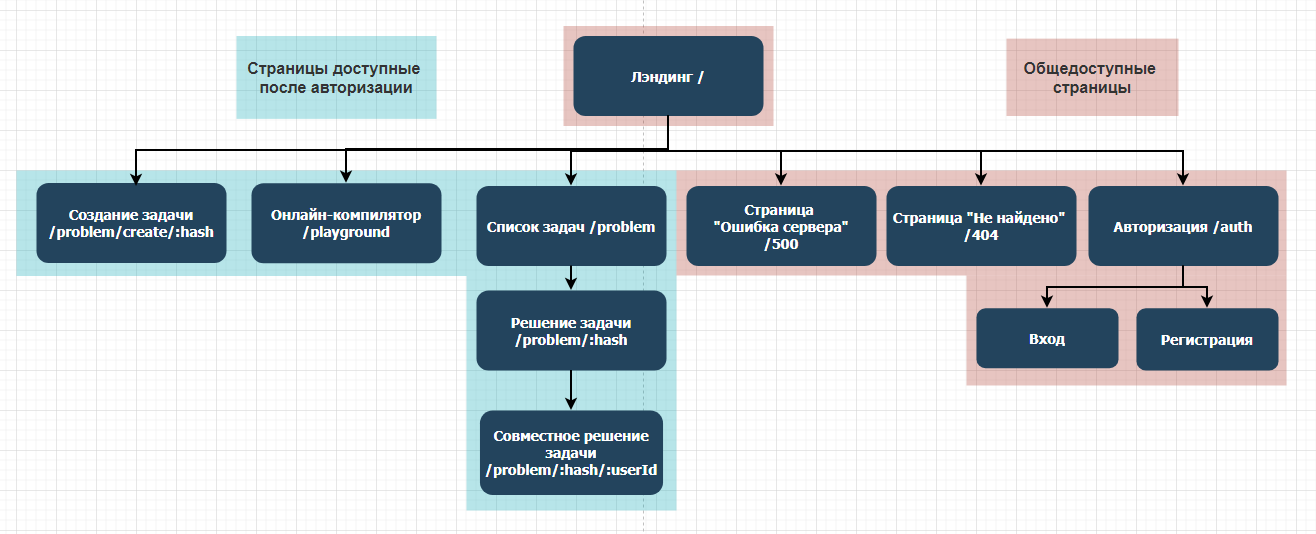
Изображение выглядит как текст, диаграмма, План, линия

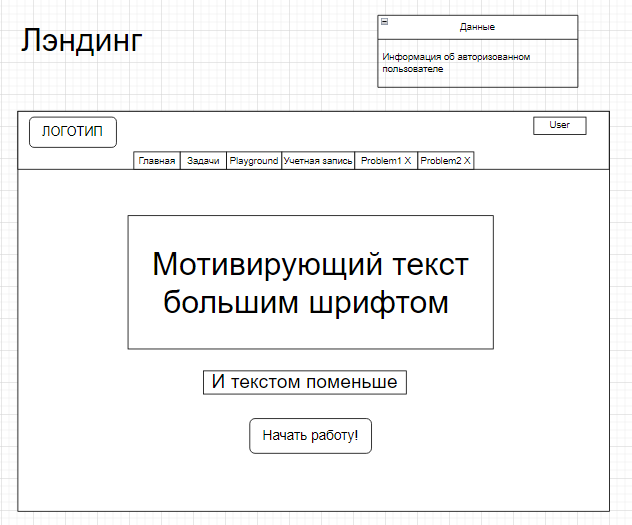
Контент, сгенерированный ИИ, может содержать ошибки.

Рисунок 2.1 – ER диаграмма базы данных платформы

# **3 Обзор литературы**

Глава 3 содержит информацию о тех источниках, которые понадобятся для написания диссертационной работы.





Изображение выглядит как текст, снимок экрана, диаграмма, линия

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, Параллельный, число

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, диаграмма, Параллельный

Контент, сгенерированный ИИ, может содержать ошибки.

Изображение выглядит как текст, снимок экрана, диаграмма, число

Контент, сгенерированный ИИ, может содержать ошибки.Изображение выглядит как текст, диаграмма, снимок экрана, План

Контент, сгенерированный ИИ, может содержать ошибки.

## **3.1 Безопасность выполнения недоверенного кода**

Тема безопасности на платформах, где запускается пользовательский код, является особенно актуальной. Запуск пользовательского кода создает потенциальные риски, включая возможность выполнения вредоносного кода и несанкционированного доступа к данным или ресурсам системы. Решением данной проблемы является применение контейнеров приложений. Docker [1] — это платформа для контейнеризации, которая позволяет создавать, развертывать и управлять приложениями в изолированных средах, называемых контейнерами. Контейнер — это легковесный, автономный пакет, содержащий все необходимое для запуска приложения, включая код, библиотеки, зависимости и конфигурации.

ЗАКЛЮЧЕНИЕ

В рамках данной работы сформулирована тема диссертационной работы, определены объект, предмет и цель исследования, поставлены задачи по ее достижению.

Проведен подбор и анализ существующих аналогов информационных систем с автоматизированной проверкой решений задач. Проведен анализ литературных источников по применению контейнеров для безопасного запуска недоверенного кода.

Недостатки существующих платформ достаточно значительны и подчеркивают необходимость создания собственной онлайн-платформы «SupremeCode».

Цель разрабатываемой платформы – предоставить пользователям возможность учиться программированию и информационным технологиям с возможностью практического решения задач, которые будут автоматически проверяться.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Leetcode платформа с задачами [Электронный ресурс],   
   URL: https://leetcode.com/ (дата обращения: 01.05.2025).