

**Министерство образования и науки Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  
**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**  
**МЕХАНИКИ И ОПТИКИ”**

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**РАЗРАБОТКА ИНФОРМАЦИОННОЙ СИСТЕМЫ СОЗДАНИЯ,**  
**СДАЧИ И ПРОВЕРКИ ЗАДАНИЙ ПО ДИСЦИПЛИНЕ**  
**"ПРОГРАММИРОВАНИЕ"**

Автор Цибин Андрей Игоревич \_\_\_\_\_  
(Фамилия, Имя, Отчество) (Подпись)

Направление подготовки (специальность) \_\_\_\_\_  
09.03.02 Информационные системы и технологии  
(код, наименование)

Квалификация бакалавр \_\_\_\_\_  
(бакалавр, магистр)\*

Руководитель Ефимчик Е.А. \_\_\_\_\_  
(Фамилия, И., О., ученое звание, степень) (Подпись)

**К защите допустить**

Зав. кафедрой Лисицына Л.С., д.т.н., проф. \_\_\_\_\_  
(Фамилия, И., О., ученое звание, степень) (Подпись)

“ \_\_\_\_\_ ” \_\_\_\_\_ 20 \_\_\_\_ г.

Санкт-Петербург, 2018 г.

Студент Цибин А.И. \_\_\_\_\_ Группа Р3420 Кафедра КОТ \_\_\_\_\_ Факультет ПИиКТ \_\_\_\_\_  
(Фамилия, И,О.)

Направленность (профиль), специализация:

Автоматизация и управление в образовательных системах

Консультант (ы):

а) \_\_\_\_\_  
(Фамилия, И., О., ученое звание, степень) (Подпись)

б) \_\_\_\_\_  
(Фамилия, И., О., ученое звание, степень) (Подпись)

ВКР принята “ \_\_\_\_ ” \_\_\_\_\_ 20 \_\_\_\_ г.

Оригинальность ВКР \_\_\_\_\_ %

ВКР выполнена с оценкой \_\_\_\_\_

Дата защиты “ \_\_\_\_ ” \_\_\_\_\_ 20 \_\_\_\_ г.

Секретарь ГЭК \_\_\_\_\_  
(ФИО) (подпись)

Листов хранения \_\_\_\_\_

Демонстрационных материалов/Чертежей хранения \_\_\_\_\_

**Министерство образования и науки Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  
**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**  
**МЕХАНИКИ И ОПТИКИ”**

**УТВЕРЖДАЮ**

Зав. кафедрой КОТ

Лисицына Л.С.  
(ФИО) (подпись)

« \_\_\_\_ » « \_\_\_\_ » 20 \_\_\_\_ г.

**ЗАДАНИЕ**  
**НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ**

Студенту Цибину Андрею Игоревичу Группа Р3420 Кафедра КОТ Факультет ПИиКТ

Руководитель Ефимчик Е.А., к.т.н., доцент, Университет ИТМО, доцент кафедры КОТ

(ФИО, ученое звание, степень, место работы, должность)

**1 Наименование темы:** Разработка информационной системы создания, сдачи и  
проверки заданий по дисциплине "Программирование".

**Направление подготовки (специальность)** 09.03.02 Информационные системы и технологии

**Направленность (профиль)** Автоматизация и управление в образовательных системах

**Квалификация** бакалавр

**2 Срок сдачи студентом законченной работы** « \_\_\_\_ » « \_\_\_\_ » 20 \_\_\_\_ г.

**3 Техническое задание и исходные данные к работе** Требуется разработать  
автоматизированную систему организации, сдачи и проверки испытаний по  
программированию с функциями автоматизированных проверок решений, качества кода и  
анализа работ на плагиат.

---

---

---

#### 4 Содержание выпускной квалификационной работы (перечень подлежащих разработке вопросов)

1. Анализ отрицательных факторов в механизмах сдачи и проверки заданий по программированию существующих систем.

2. Проектирование и разработка автоматизированной системы.

3. Проведение внедрения разработанной платформы в образовательный процесс и описание результатов.

---

---

---

---

---

---

---

---

---

---

---

---

#### 5 Перечень графического материала (с указанием обязательного материала)

---

---

---

---

---

---

---

---

---

---

---

---

#### 6 Исходные материалы и пособия

---

---

---

---

---

---

---

---

---

---

---

---

7 Дата выдачи задания « \_\_\_\_ » « \_\_\_\_\_ » 20 \_\_\_\_ г.

Руководитель ВКР \_\_\_\_\_  
(подпись)

Задание принял к исполнению \_\_\_\_\_ « \_\_\_\_ » « \_\_\_\_\_ » 20 \_\_\_\_ г.  
(подпись)

**Министерство образования и науки Российской Федерации**  
**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**  
**“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ**  
**УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ,**  
**МЕХАНИКИ И ОПТИКИ”**

## **АННОТАЦИЯ**

### **ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

Студент Цибин Андрей Игоревич

(ФИО)

Наименование темы ВКР: Разработка информационной системы создания, сдачи и  
проверки заданий по дисциплине "Программирование".

Наименование организации, где выполнена ВКР \_\_\_\_\_

### **ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ**

1 Цель исследования Создание автоматизированной системы организации, сдачи и проверки  
испытаний по программированию

2 Задачи, решаемые в ВКР Анализ отрицательных факторов в механизмах сдачи и проверки  
заданий по программированию существующих систем; проектирование и разработка  
автоматизированной системы; проведение внедрения разработанной платформы в  
образовательный процесс и описание результатов.

3 Число источников, использованных при составлении обзора 3

4 Полное число источников, использованных в работе 10

5 В том числе источников по годам

Отечественных			Иностранных		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
1	1		6	1	1

6 Использование информационных ресурсов Internet да, 1

(Да, нет, число ссылок в списке литературы)

7 Использование современных пакетов компьютерных программ и технологий (Указать, какие именно, и в каком разделе работы)

Пакеты компьютерных программ и технологий	Параграф работы
Dia, IBM Rational Rose, Erwin Process Modeller	2
Webpack, JavaScript, React	3.1
Gradle, Kotlin, Spring, Spek	3.2
Git, Github, Travis, Codacy, Moss	3.3
Java, Google Forms	4

8 Краткая характеристика полученных результатов В результате выполнения данной  
дипломной работы была спроектирована, разработана и протестирована распределенная  
интегрированная система создания, прохождения и сдачи заданий по программированию,  
сочетающая в себе большинство положительных сторон существующих на сегодняшний день

аналогичных систем, а также выдвигающая на новый уровень автоматизацию процессов создания обучающих курсов по программированию и проверки решений обучающихся с точки зрения прохождения тестов, качества кода и отсутствия плагиата.

9 Полученные гранты, при выполнении работы \_\_\_\_\_  
(Название гранта)

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы да  
(Да, нет)

а) 1 Цибин А.И., Ефимчик Е.А. Распределенная интегрированная информационная система организации сдачи и проверки заданий по программированию: применение и особенности технической реализации // Сборник тезисов докладов конгресса молодых ученых. Электронное издание [http://openbooks.ifmo.ru/ru/file/7344/7344.pdf]. - Режим доступа: ссылка на страницу с тезисом, своб.

(Библиографическое описание публикаций)

2 \_\_\_\_\_

3 \_\_\_\_\_

б) 1 Цибин А.И., Ефимчик Е.А. Распределенная интегрированная информационная система организации сдачи и проверки заданий по программированию: применение и особенности технической реализации : VII Конгресса молодых ученых, 2018.

(Библиографическое описание выступлений на конференциях)

2 \_\_\_\_\_

3 \_\_\_\_\_

Студент \_\_\_\_\_  
(ФИО) (подпись)

Руководитель \_\_\_\_\_  
(ФИО) (подпись)

“ \_\_\_\_\_ ” \_\_\_\_\_ 20\_\_ г.

## ОГЛАВЛЕНИЕ

Введение.....	5
1 Анализ требований и обзор аналогов.....	6
2 Проектирование.....	12
2.1 Пользовательские истории.....	12
2.2 Функциональное моделирование .....	14
2.3 Варианты использования .....	15
2.4 Сценарии работы .....	16
2.5 Модель базы данных .....	19
3 Разработка .....	22
3.1 Клиентская часть.....	22
3.2 Серверная часть .....	23
3.3 Выбор вендоров интегрированных сервисов.....	24
3.3.1 Сервис системы контроля версий.....	25
3.3.2 Сервис непрерывной интеграции .....	25
3.3.3 Сервис статического анализа кода .....	26
3.3.4 Сервис анализа плагиата.....	26
3.4 Пользовательский интерфейс .....	28
3.5 Реализация .....	32
4 Внедрение.....	35
Заключение .....	45
Обозначения и сокращения .....	46
Список иллюстративного материала.....	47
Список литературы .....	48

## **ВВЕДЕНИЕ**

В настоящее время дистанционное и электронное образование продолжает активно развиваться, создаются новые протоколы, системы и методы организации и проведения электронных курсов по множеству различных дисциплин. Между тем, программирование, являясь одной из самых многочисленных по количеству существующих на данный момент материалов дисциплин, по-прежнему имеет ряд ограничений и недостатков в механизмах выполнения заданий и механизмах проверки работ обучающихся.

Целью данной дипломной работы является создание системы организации, прохождения и проверки испытаний по программированию, которая одновременно будет являться удобным и функциональным средством достижения образовательных целей как для преподавателя, так и для обучающегося.

В данной дипломной работе приведены анализ отрицательных факторов в механизмах сдачи и проверки заданий по программированию существующих систем, пути решения описанных ограничений и борьбы с недостатками существующих систем, предложено автоматизированное решение, описано проектирование и технические подробности реализации созданной системы и механизмов проверки работ обучающихся.



## 1 АНАЛИЗ ТРЕБОВАНИЙ И ОБЗОР АНАЛОГОВ

Для определения требований и отличительных особенностей проектируемой системы было проведено исследование нескольких наиболее крупных из представленных на рынке решений для создания, проведение и проверки испытаний по различным компьютерным дисциплинам, связанным с программированием.

В исследовании участвовали следующие платформы:

- coursera.org,
- edx.org,
- hackerrank.com,
- stepik.org.

Помимо платформ были рассмотрены следующие виртуальные лаборатории, поддерживающие RLCP-протокол [1; 2]:

- Cyber-Net,
- MCE (Multi-style code editor) [1].

В ходе анализа были выявлены две основные группы систем: системы, поддерживающие автоматические проверки исключительно на уровне пар значений консольного ввода и консольного вывода, а также системы, в которых есть возможность кастомизации обработки решений, которая, однако, целиком является задачей, решаемой непосредственно самим автором курса.

В первую группу вошли следующие системы:

- Hackerrank,
- Stepic,
- Cyber-net,
- MCE.

Во вторую, соответственно:

- Coursera,
- Edx.

Однако, для обеих групп были выявлены несколько ограничений в работе большинства систем:

- низкая скорость выдачи результатов выполнения обучающимся;
- сокрытие набора тестов от обучающегося, отсутствие подробной информации об ошибках в решении и ограничение на количество попыток проверки решений;
- необходимость работать в браузере без поддержки знакомых обучающимся интегрированных сред разработки;
- ограничения на архитектуру возможных решений;
- отсутствие у обучающихся возможности использовать внешние библиотеки языка программирования;
- отсутствие автоматизированных проверок решений на плагиат;
- отсутствие автоматизированных проверок качества кода решений с точки зрения статического анализа кода.

Проведенный анализ дает понимание, что в существующих системах представлено большое количество ограничений на выполнение и проверку решений обучающихся, а также качество автоматизации обеспечивающих эти процессы механизмов находится на недостаточном уровне. Актуальность необходимости удаления наиболее критичных ограничений приведена далее в текущем разделе.

Для рассмотренных систем типичным является следующий алгоритм проверки решений обучающихся: студент оформляет решение, отправляет его через интерфейс системы на проверку и через некоторое время получает некоторый отчет по качеству выполненного им задания.

Если говорить о скорости проверки, которая заключается в запуске всех тестов на серверной стороне платформы, то среднее время проверки решения занимает от нескольких секунд в лучшем случае и до нескольких десятков секунд в худшем. Скорость проверки решения тестами непосредственно влияет на частоту, с которой обучающийся запускает проверки своего решения. Частый запуск

проверок позволяет обучающемуся быстро наблюдать прогресс своего решения в соответствии с тем, насколько меньше тестов завершается с ошибкой. Зачастую, ошибки, совершаемые при написания кода, связаны не столько с самим алгоритмом, сколько с типичными ошибками для любого рода написания кода (опечатки, ошибки на единицу и пр.). Поэтому очень часто обучающийся тратит свое время на ожидание результатов проверки решения, которое изначально не было даже валидным. Если говорить о технической стороне вопроса, то задержка превышает реальное время прохождения тестов на локальной машине в несколько раз и связана с временем передачи решения и ответа по сети, а также с распределением нагрузки на стороне сервера. В связи с тем, что решения от обучающихся приходят очень часто, растет нагрузка на сервер, что в свою очередь увеличивает время проверки решений. И в этой ситуации теряет свое опять же обучающийся.

На многих платформах отчет, возвращаемый о результатах проверки работы крайне немногословен и, в худших случаях, содержит в себе только состояние: прошла работа проверку или нет, прошел ли тест проверку или нет. Для обучающегося такой ответ системы представляет крайне малую пользу, ему приходится тратить длительное время на изучение всей кодовой базы решения, хотя ошибка может быть детерминирована гораздо точнее, если бы для каждого теста имелись соответствующие модульные тесты, которые и запускались бы сервером в качестве проверки работ и описывали конкретный элемент поведения тестируемого кода. Если также учитывать, что на нескольких платформах существует также ограничение на количество возможных проверок решений, то обучающему становится крайне сложно в таких условиях заниматься полезным и удобным поиском ошибок в своем решении.

Первые раскрытые проблемы приводят рассуждение к проблеме сокрытия набора тестов. Используемая в большинстве анализируемых платформ инкапсуляция тест-кейсов не имеет ничего общего с реальной разработкой программного обеспечения, в которой зачастую тесты для определенных модулей написаны раньше, чем эти модули реализованы [3]. В некоторой степени

работоспособность некоего программного модуля можно оценивать по качеству и количеству модульных тестов к каждому из его элементов. При этом рефакторинг или исправление багов с большей вероятностью не испортит работу модуля, поскольку имеется возможность запустить модульные тесты и увидеть, осталось ли поведение модуля корректным. Эти аргументы приводят решение представленных выше проблем к следующему: тестовые кейсы, используемые сервером для проверки решений, должны быть открыты для обучающихся. В том числе, обучающиеся должны обладать возможностью запускать модульные тесты локально на своих персональных машинах. Такое решение оптимизирует время работы обучающихся над своими решениями и значительно уменьшит избыточную нагрузку на сервер.

Для большей части исследованных систем общим является необходимость оформления своего решения в специализированной форме окна браузера. В большинстве случаев данные поля ввода не имеют никакого дополнительного функционала и не позволяют на хоть каком-либо уровне проводить валидацию корректности введенного кода. Это накладывает ограничения на скорость написания кода, увеличивает вероятность простейших ошибок и, тем самым, увеличивает необходимое число раз, когда обучающийся будет отправлять решения на проверку. Для многих более привычным и удобным средством написания кода может являться определенная интегрированная среда разработки. Помимо этого, если задание включает в себя написание довольно большого количества кода (например, реализация некоторой структуры данных), то работа в форме на веб-странице становится крайне неудобным.

Ограничение на написание кода в браузере зачастую также не позволяет обучающемуся использовать полные возможности используемого языка: он ограничен в возможностях создания файлов и папок, подключении внешних зависимостей языка. Такие ограничения запрещают студенту использовать некоторые распространенные инструменты и практики, которые могут быть не менее важны в изучении, чем сами решаемые задания.

Помимо проверки прохождения решением тестов, крайне важно учитывать качество кода, использованного в решении. Из проанализированных систем ни в одной не было представленного автоматизированного средства оценки качества кода. В реальной разработке качество кода ценится немногим меньше, чем его работоспособность, поскольку код пишется один раз, а читается и развивается постоянно. Вынуждать обучающегося следить за качеством кода это практика, которая воспитывает дисциплину и внимание к деталям, то есть важные навыки для любого разработчика. Следовательно, возможность автоматического анализа качества кода и возможность включения его в оценку должно быть частью системы проверки заданий.

Сложно отрицать, что зачастую в образовательных курсах присутствует элемент списывания, и в связанных с программированием курсах подобное случается не реже, чем в любой другой дисциплине. На данный момент существуют средства установления цифровых отпечатков фрагментов кода, что позволяет говорить о возможности поиска плагиата среди работ по программированию. Возможность проверки работ студентов на плагиат является сильным средством оптимизации времени преподавателя. К сожалению, ни одна из рассмотренных систем не обладает автоматизированным средством поиска плагиата среди сданных работ. При этом, важно понимать, что любой автоматизированный анализ плагиата, это лишь указатель на некоторую вероятность присутствия плагиата, но ни в коем случае не доказательство его присутствия.

На основании проведенного обзора аналогов и анализа выявленных ограничений в процессах сдачи и проверки решений обучающихся одно из возможных решений заключается в создании системы, интегрированной с сервисами распределенных систем контроля версий и сервисами непрерывной интеграции, которые позволят автоматизировать процесс сдачи и проверки соответствия работ обучающихся спецификациям, указанным преподавателем.

При этом задания, тесты и решения должны находиться в открытом доступе на серверах одного из провайдеров систем контроля версий, проверка решений и

качества кода должна осуществляться сервисами непрерывной интеграции и статического анализа кода, а проверка на плагиат - специализированным сервисом поиск плагиата в коде.

## 2 ПРОЕКТИРОВАНИЕ

### 2.1 Пользовательские истории

Проектирование системы начинается с определения существующих в системе ролей, после чего для каждой роли описываются пользовательские истории, которые позволяют наглядно понять, в чем заключается цель использования системы различными пользователями и какую выгоду каждый из её участников хочет получить.

Для проектируемой системы удалось выделить 4 основных роли:

- Гость;
- Преподаватель;
- Обучающийся;
- Внешняя система.

Для каждой из ролей далее описаны наборы пользовательских историй.

*Пользовательские истории для роли гостя:*

- Как гость, я хочу зарегистрироваться в системе, чтобы иметь возможность создавать обучающие курсы;
- Как гость, я хочу авторизоваться в системе, чтобы иметь возможность пользоваться полным функционалом системы.

*Пользовательские истории для роли преподаватель:*

- Как преподаватель, я хочу создать курс, указав необходимые параметры (языки, фреймворки);
- Как преподаватель, я хочу иметь возможность быстро отключить подключенные проверки, а затем удалить курс вместе с репозиторием системы контроля версий, чтобы не тратить на это лишнее время;
- Как преподаватель, я хочу иметь возможность получить ссылку на репозиторий системы контроля версий созданного курса, чтобы иметь возможность самостоятельно добавить в него описания, задания и тесты;

- Как преподаватель, я хочу запустить ранее созданный курс, чтобы дать возможность обучающимся выполнять задания, и чтобы получать статистику об их успеваемости;
- Как преподаватель, я хочу просматривать отчет о результатах прохождения обучающимися заданий, чтобы видеть их прогресс, случаи плагиата и недобросовестного выполнения заданий;
- Как преподаватель, я хочу загрузить отчет о результатах прохождения обучающимися заданий в удобном мне формате, чтобы использовать его во внутренней системе своей организации;
- Как преподаватель, я хочу изменить настройки проверок решений студентов, чтобы установить другие множители оценивания заданий и проверяемых критериев;
- Как преподаватель, я хочу установить время завершения курса, чтобы для обучающихся существовал дедлайн сдачи решений.

*Пользовательские истории для роли внешней системы:*

- Как внешняя система, я хочу загрузить отчет о результатах прохождения обучающимися заданий конкретного курса в удобном мне формате, чтобы использовать полученную статистику с целью контроля успеваемости студентов;

*Пользовательские истории для роли обучающегося:*

- Как обучающийся, я хочу получить задания курса, чтобы иметь возможность их выполнить;
- Как обучающийся, я хочу просматривать правила прохождения заданий курса, чтобы иметь возможность получить максимальные оценки;
- Как обучающийся, я хочу иметь возможность отправить выполненное задание на проверку, чтобы получить оценку;
- Как обучающийся, я хочу иметь возможность увидеть результат проверки отправленного мною задания.



## 2.2 Функциональное моделирование

Описанные в пользовательских историях прецеденты использования позволяют обозначить причинно-следственные связи между ситуациями и событиями в проектируемой и выделить функциональные элементы системы в виде диаграмм IDEF3 PFDD в нескольких приближениях:

- Система в целом (Рисунок 1);
- Работа с курсами (Рисунок 2);
- Работа с решениями (Рисунок 3);
- Сбор статистики (Рисунок 4).

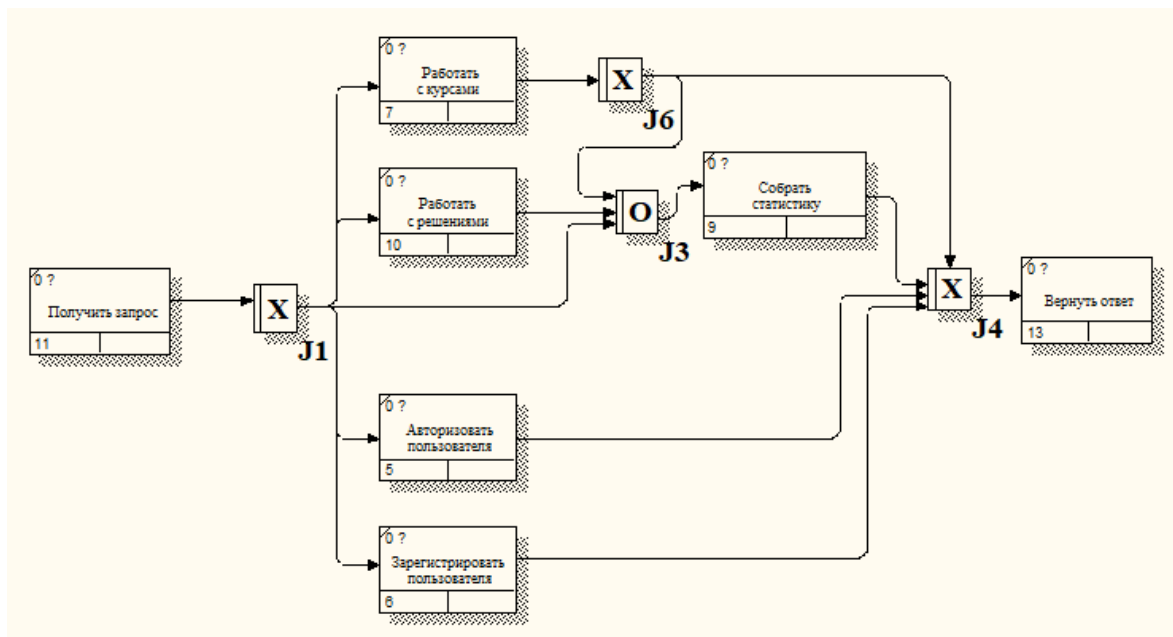


Рисунок 1 – Основная диаграмма IDEF3

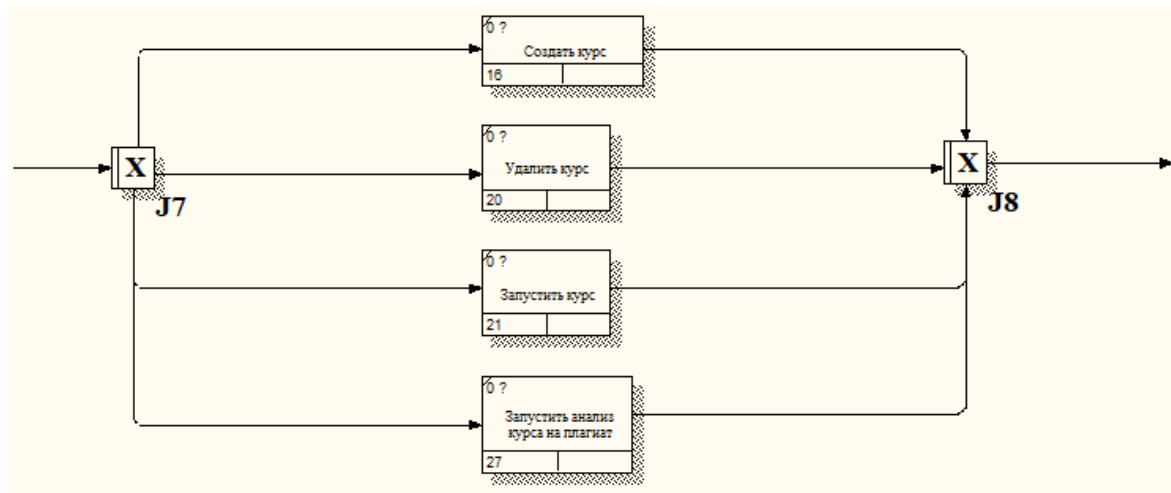


Рисунок 2 – Диаграмма IDEF3 работы с курсами

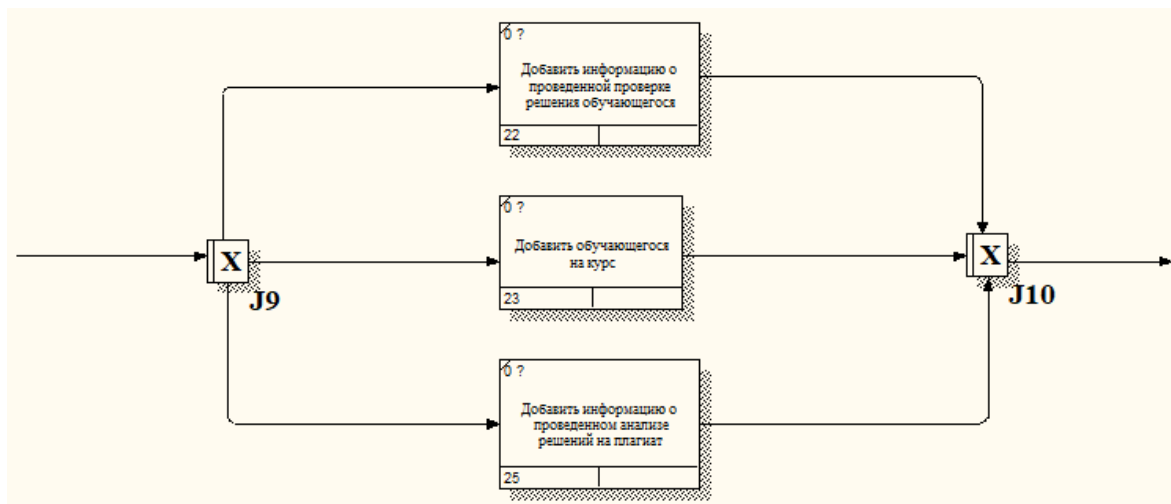


Рисунок 3 – Диаграмма IDEF3 работы с решениями

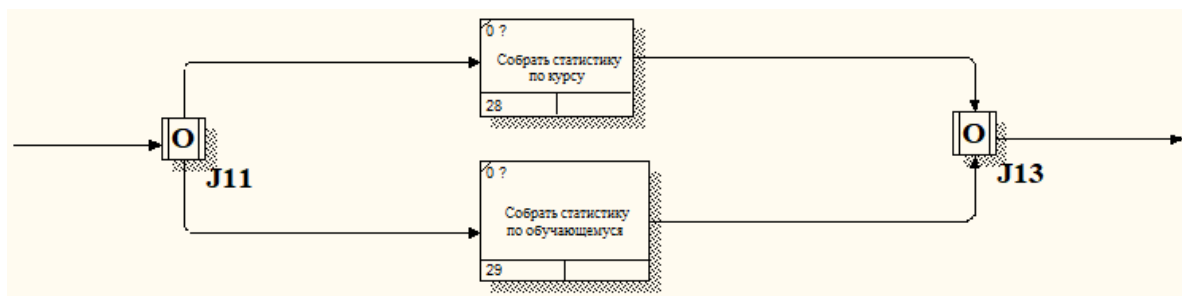


Рисунок 4 – Диаграмма IDEF3 сбора статистики

## 2.3 Варианты использования

На основании описанных пользовательских историй, составлена диаграмма вариантов использования или диаграмма прецедентов (Рисунок 5), позволяющая визуально определить спектр вариантов использования системы каждой из представленных в ней ролей.

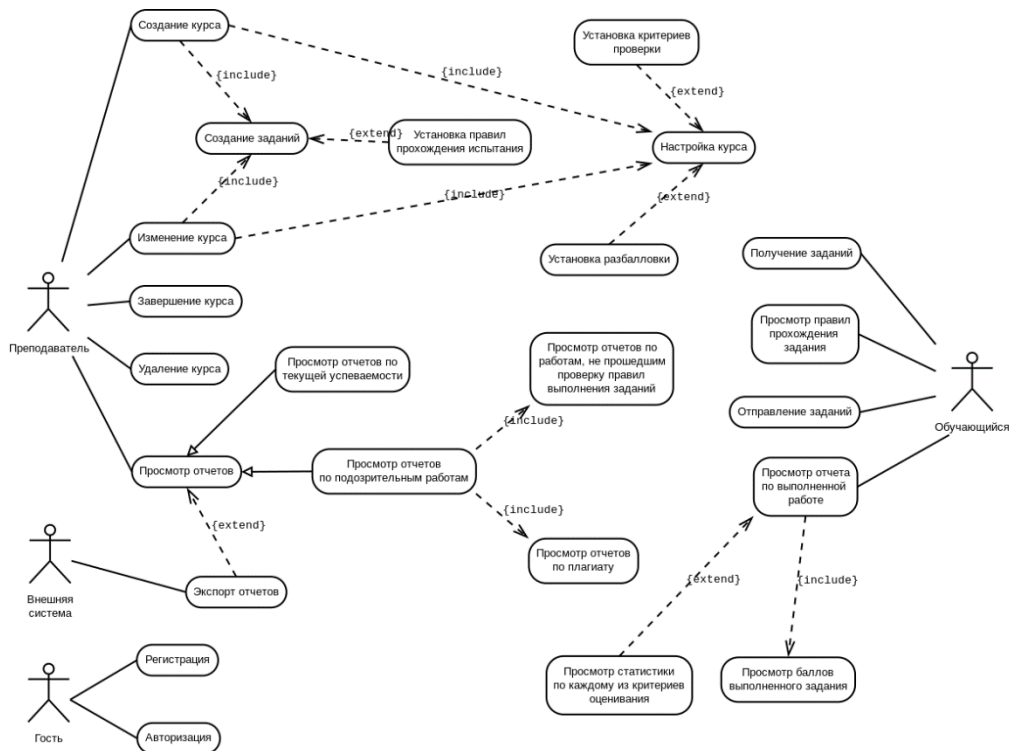


Рисунок 5 – Диаграмма вариантов использования

## 2.4 Сценарии работы

После того, как для системы описаны пользовательские истории, функциональные особенности и варианты использования, необходимо формально описать типичные сценарии использования системы. Для этого используются различные диаграммы, в данной дипломной работе для этой цели используются диаграммы последовательности.

Диаграммы основных сценариев использования системы приведены ниже:

- Сценарий создания и инициализации курса (Рисунок 6);

- Сценарий сдачи и проверки работы обучающегося системой (Рисунок 7);
- Сценарий анализа решений в курсе на плагиат (Рисунок 8).

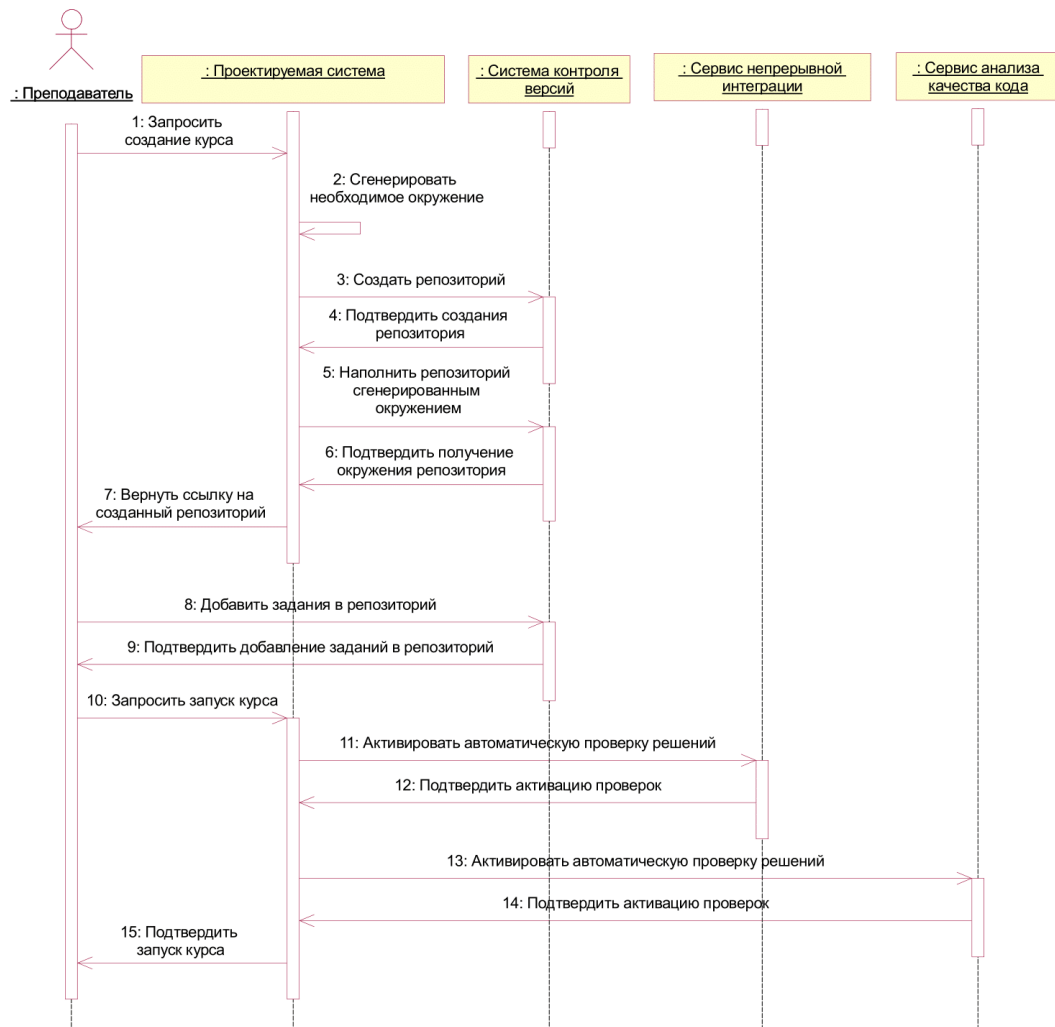


Рисунок 6 – Диаграмма последовательности сценария создания и инициализации курса

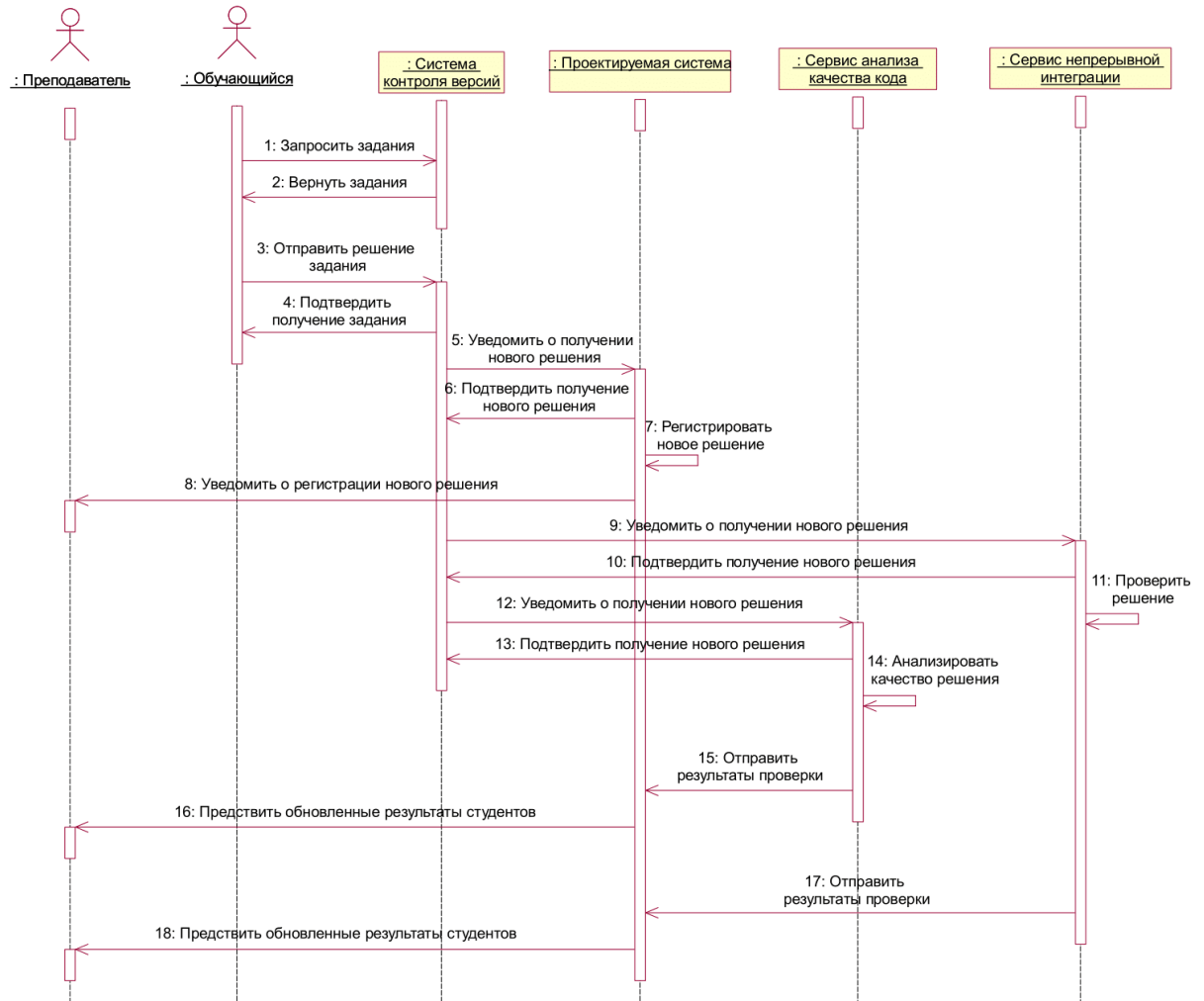


Рисунок 7 – Диаграмма последовательности сценария сдачи и проверки работы обучающегося системой

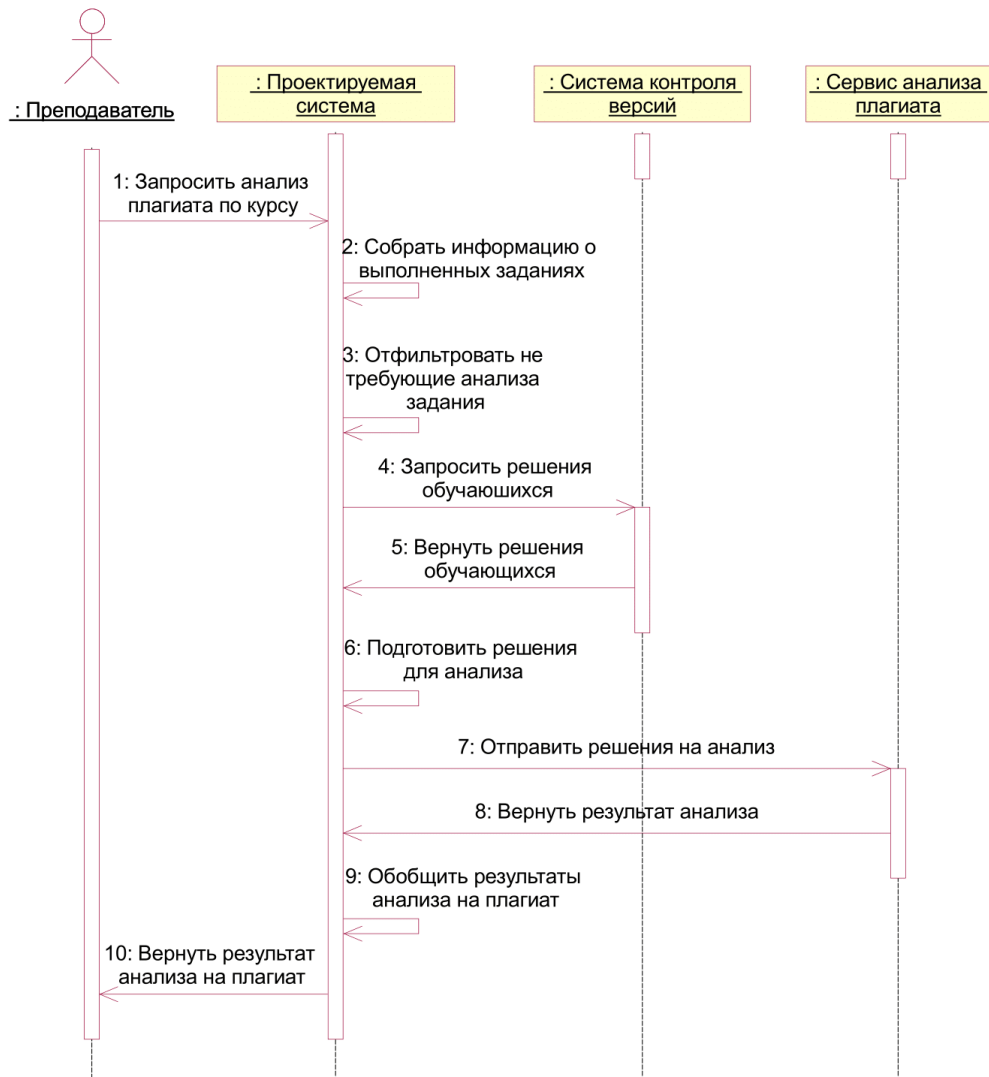


Рисунок 8 – Диаграмма последовательности сценария анализа решений в курсе на плагиат

## 2.5 Модель базы данных

На основании приведенных выше прецедентов, схем и диаграмм была спроектирована модель базы данных (Рисунок 9), отношения в которой удовлетворяют, по меньшей мере, первым трем нормальным формам.

Основными выделенными сущностями являются:

- User (Пользователь) – модель зарегистрированного в системе преподавателя.

- Credentials (Приватные данные) – модель, содержащая пароль от аккаунта и все токены авторизации интегрированных систем соответствующего пользователя.
- Course (Курс) – модель образовательного курса, созданного пользователем в системе.
- State (Состояние курса) – модель текущего состояния курса.
- Lifecycle (Жизненный цикл курса) – модель этапа жизненного курса цикла.
- Activated\_Service (Активированный сервис) – модель сервиса проверки, который может быть активирован для соответствующего курса.
- Student (Студент) – модель обучающегося на соответствующем курсе.
- Task (Задание) – модель задания в соответствующем курсе.
- Plagiarism\_Report (Отчет по плагиату) – модель отчета по проведенному анализу плагиата соответствующего задания в курсе.
- Plagiarism\_Match (Найденный случай потенциального плагиата) – модель отдельного замеченного случая, подозрительного на плагиат.
- Solution (Решение) – модель решения студента по определенному заданию.
- Commit (Коммит) – модель коммита, представляющего последние из сохраненных студентом изменений в решении.
- Build\_Report (Отчет по прохождению тестов) – модель отчета по проведению тестирования соответствующего решения.
- Code\_Style\_Report (Отчет по качеству кода) – модель отчета по анализу качества кода соответствующего решения.

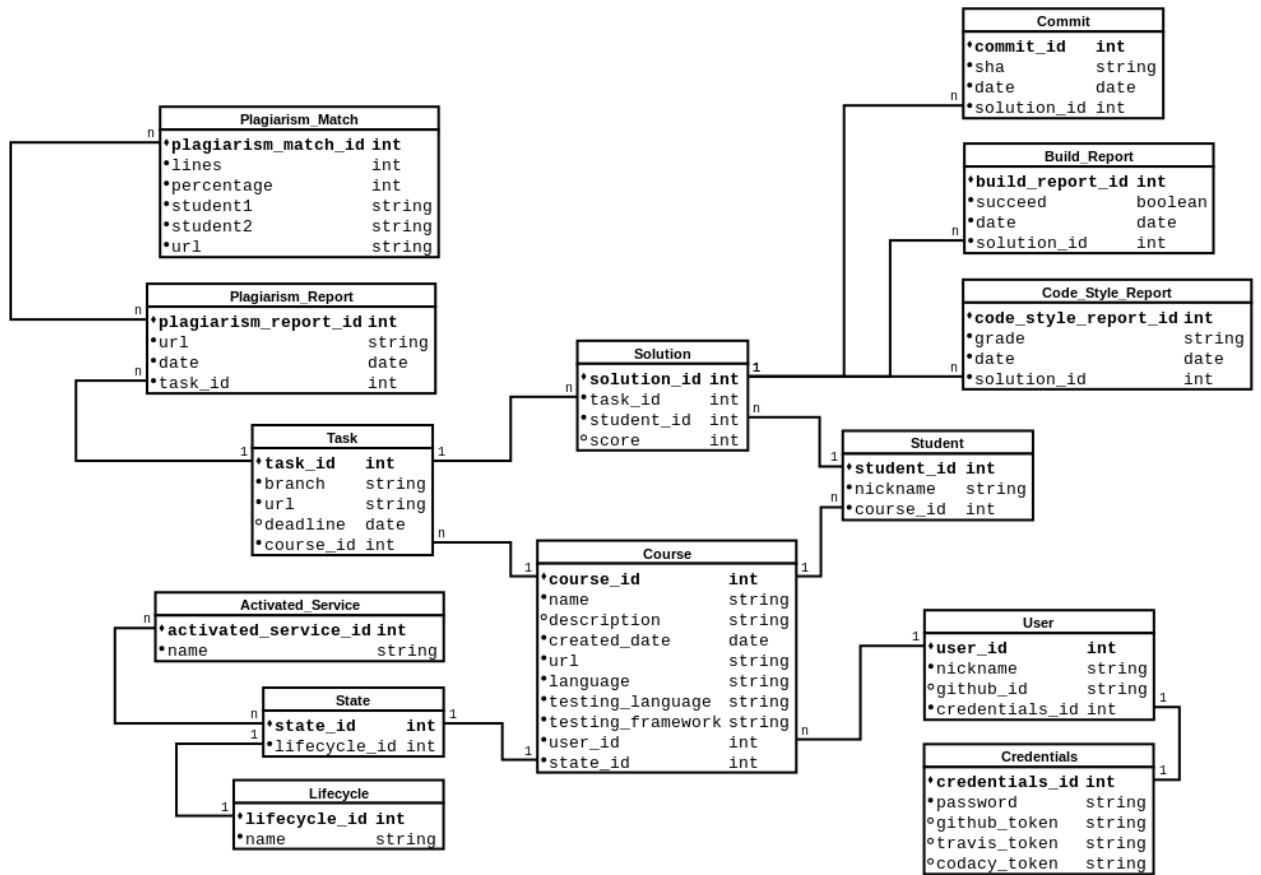


Рисунок 9 – Схема модели базы данных



### 3 РАЗРАБОТКА

Разрабатываемая система представляет из себя два крупных модуля: клиентский и серверный. На данный момент широко распространена схема разделения данных MVC, при которой клиентская часть системы является лишь представлением данных, а модель и контроллер находятся на стороне серверной части. У такого подхода существует множество преимуществ, но также существуют и значительные недостатки. Например, разработка пользовательского интерфейса неразрывна связана с разработкой серверной части и, зачастую, это вызывает множество проблем при внедрении изменений с той или иной стороны, а уследить за этим крайне сложно. Одним из подходов, заменяющим эту модель, является полное разделение серверной и клиентской частей. Единственным общим элементом, который остается между двумя частями проекта является HTTP API, предоставляемый сервером и используемый клиентом. Для описания такого процесса общения клиентской и серверной части системы используется понятие REST [4].

При разработке системы в данной дипломной работе было принято решения использовать именно такой подход. По этой причине описание инструментов и технологий для клиентской и серверной частей приведено независимо друг от друга.

#### 3.1 Клиентская часть

Для создания клиентской части разрабатываемой системы был выбран язык JavaScript, поддерживающий стандарт ECMAScript 6. Причиной выбора послужили: обилие документации, существующих библиотек и фреймворков, а также широкая поддержка функциональной и ООП парадигм.

В качестве основного фреймворка для написания клиентской части был выбран React [5], позволяющий создавать прагматичные и расширяемые

одностраничные веб-приложения, так называемые SPA, состоящие из переиспользуемых компонентов.

В качестве инструмента сборки клиентской части приложения используется Webpack, который позволяет собирать как пользовательские, так и библиотечные скрипты, и стили, в единый оптимизированный файл, который используется уже непосредственно в коде html-страницы. Это позволяет поддерживать кодовую базу в порядке, контролировать используемые зависимости, а также разбивать код, который будет отправлять на машину пользователя, на отдельные блоки. В свою очередь, это позволит ускорить работу клиентского веб-приложения.

### 3.2 Серверная часть

В качестве языка программирования серверной части системы был выбран язык Kotlin [6], разрабатываемый компанией JetBrains. Выбор данного достаточно нового языка (первая версия вышла в 2011 году) обусловлен многими критериями, основными преимуществами являются:

- Широкие возможности использования функциональной парадигмы программирования, иммутабельных структур данных, а также поддержка, так называемых, non-nullable типов;
- Поддержка DSL или декларативного стиля написания кода, в частности, существование высокоуровневых фреймворков для написания BDD спецификаций [7];
- Полная совместимость с платформой JVM, т.е. возможность запускать код, написанный на Kotlin на любой платформе, для которой существует соответствующая виртуальная машина;
- Поддержка существующей кодовой базы языка Java, возможность использовать большинство библиотек и фреймворков, написанных на Java;
- Полноценная поддержка современных средств сборки проектов.

В качестве инструмента сборки был выбран Gradle, поскольку обладает современным и функциональным императивным синтаксисом, позволяющим осуществлять любые задачи, связанные с организацией, сборкой, тестированием и развертыванием проекта.

Основным фреймворком, используемым в системе, является Spring, являющийся наиболее распространенным, свободным инструментом для создания гибких, надежных и безопасных веб-приложений, покрывающий большую часть возможных задач, возникающих при разработке подобных систем.

### **3.3 Выбор вендоров интегрированных сервисов**

Исходя из того, что разрабатываемая система интегрируется с несколькими внешними сервисами, одной из задач является выбор непосредственного вендора или поставщика той или иной услуги.

Согласно описанию разрабатываемой системы необходимо выбрать вендоров для четырех предоставляемых сервисов:

- сервис системы контроля версий,
- сервис непрерывной интеграции,
- сервис статического анализа кода,
- сервис анализа плагиата.

При выборе вендора каждого из сервисов учитываются несколько критериев:

- Сервис должен быть бесплатным для проектов с открытым исходным кодом.
- Сервис должен обладать API, позволяющим максимально автоматизировать процесс настройки всех необходимых условий функционирования системы.
- Сервис должен поддерживать как можно больше языков и технологий.

### 3.3.1 Сервис системы контроля версий

Первым делом стоит задача выбрать технологию для использования системы контроля версий. Из наиболее крупных на сегодняшний день распределенных систем контроля версий можно выделить:

- Mercurial;
- Git [8].

Исходя из фактической распространенности систем, а также количества крупных вендоров этих систем, выбор ложится на систему контроля версий git, как наиболее практичную.

Из представленных на рынке сервисов систем контроля версий можно выделить три наиболее распространенных:

- Bitbucket,
- GitLab,
- GitHub.

По объему необходимого функционала выбор сделан в пользу GitHub, который позволяет создавать неограниченное число публичных репозиторий для проектов с открытым исходным кодом, а также обладает качественным API. Помимо прочего, для GitHub существует несколько клиентов на языке Java, что упрощает разработку модуля работы с данным сервисом.

### 3.3.2 Сервис непрерывной интеграции

Выбор вендора системы непрерывной интеграции несколько ограничен, поскольку большая часть таких систем является коммерческими продуктами, которые непригодны для разрабатываемой платформы, поскольку накладывают значительные экономические и организационные ограничения. Наиболее распространенным сервисом, предоставляющим бесплатные услуги по

непрерывной интеграции для проектов с открытым исходным кодом, является Travis, который полностью удовлетворяет нуждам разрабатываемой системы.

Travis использует конфигурационный файл «.travis.yml», который должен находится в корне репозитория, для определения языка программирования и прочих настроек необходимого тестового окружения.

### **3.3.3 Сервис статического анализа кода**

Из представленных сервисов статического анализа кода на рынке наиболее распространенными являются:

- CodeFactor;
- Code Climate;
- Codacy.

Каждый из выбранных сервисов статического анализа кода бесплатен для проектов с открытым исходным кодом, поэтому выбор основывается на широте возможных для использования проверок, а также на полноте и качестве представленного платформами API. Наиболее полно удовлетворяющим всем требованиям разрабатываемой системы является Codacy.

Сервис Codacy, в отличие от Travis не использует собственный конфигурационный файл, а производит настройку автоматически, анализируя файлы в репозитории. При этом, более сложные конфигурации проверок качества кода можно изменять через сервис Codacy непосредственно.

### **3.3.4 Сервис анализа плагиата**

Открытых сервисов анализа плагиата [9] на текущий момент представлено небольшое количество, поэтому выбор осуществляется между двумя сервисами:

- JPlag;
- Moss.

Оба веб-сервиса анализа плагиата предоставляют консольный интерфейс взаимодействия с сервером. Однако, для сервиса Moss существуют программный клиент на языке Java, а количество поддерживаемых языков программирования превышает в несколько раз число языков JPlag: 23 языка против 6, соответственно. Именно эти критерии и обуславливают выбор Moss в качестве вендора сервиса анализа плагиата.

Moss (for a Measure Of Software Similarity) – автоматическая система определения схожести исходного кода. Основная идея, стоящая за механизмом анализа плагиата системой Moss, является алгоритм Winnowing [10], используемый для создания цифровых отпечатков и адаптированный для использования с исходным кодом на разных языках.

Модель, лежащую в основе работы системы Moss (Рисунок 10), можно разделить на основные части:

- алгоритм вычисления цифровых отпечатков (Fingerprinting engine), который полностью не зависит от типа данных, цифровые отпечатки которых требуется определить,
- специфичный для типа данных, например языка программирования, адаптер (Front-End).

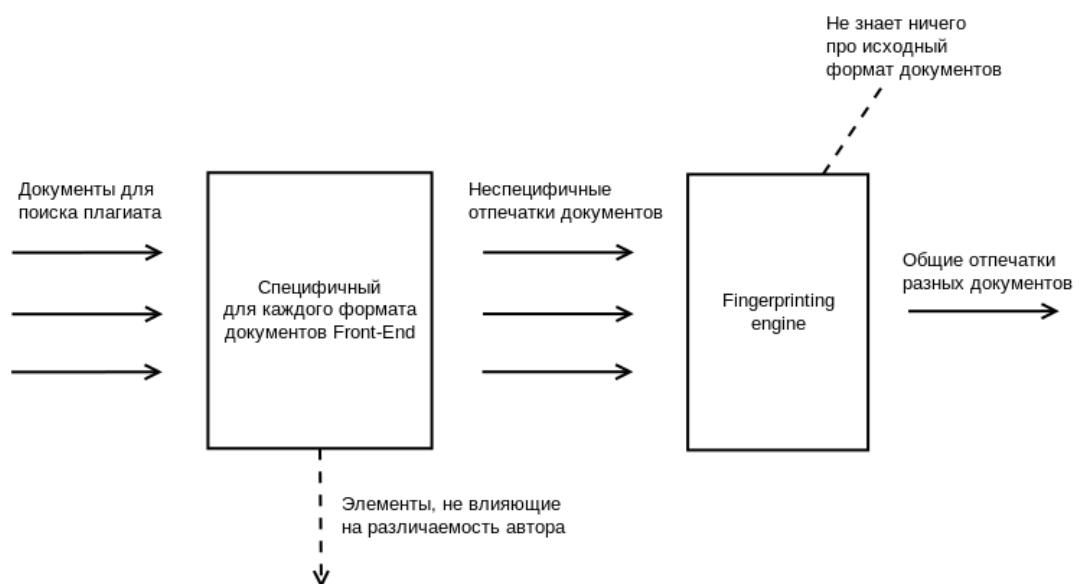


Рисунок 10 – Модель анализа плагиата системы Moss

### 3.4 Пользовательский интерфейс

Разработка пользовательского интерфейса начинается с создания макетов страниц или компонентов будущего приложения.

Основной задачей при создании пользовательского интерфейса разрабатываемой системы является получение одновременно функционального, прагматичного и минималистичного способа работы пользователя с системой.

С этой целью при разработке интерфейса используется понятие цветового кодирования, предусматривающего использование цветов в качестве элементов передачи дополнительной информации пользователю.

Согласно разработанному дизайну, если пользователь авторизован, то в любой момент времени на странице должна присутствовать панель управления с кнопкой выхода из системы и кнопками авторизации через сторонние сервисы или ярлыками, указывающими, что авторизация через сторонний сервис уже проведена.

Представление страницы всех курсов (Рисунок 11) содержит кнопку создания нового курса, а также набор карточек курсов, в каждой из которых отображается название и описание соответствующего курса, а также набор маркеров к нему.

Каждый маркер представляет из себя цветной ярлык, содержащий краткое обозначение некоторого состояния курса, название одной из используемых в курсе технологий или название совместимого внешнего сервиса.

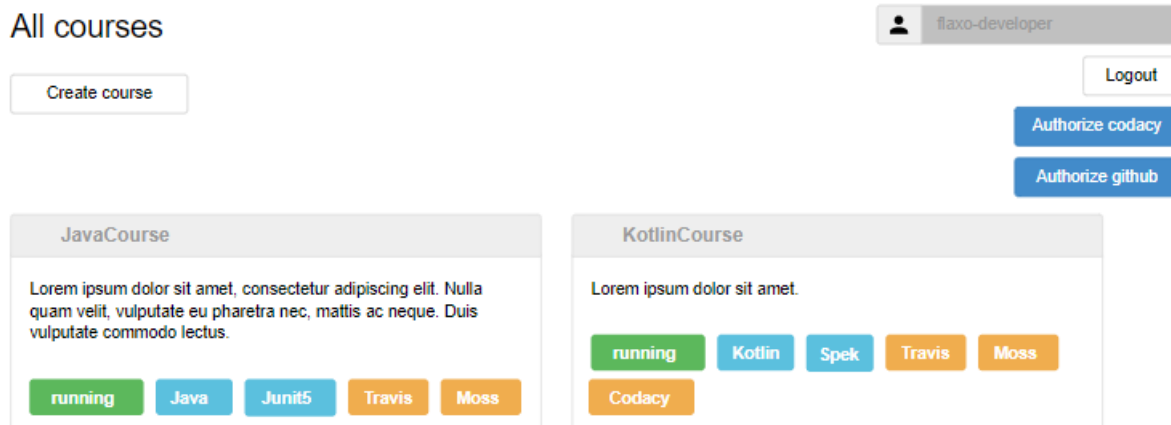


Рисунок 11 – Макет представления всех курсов пользователя

Представление выбранного курса (Рисунок 12) содержит набор соответствующих маркеров, кнопки запуска курса, анализа на плагиат, удаления курса, а также кнопка выпадающего списка форматов выгрузки отчета по курсу. Помимо прочего, представление содержит меню просмотра общей статистики курса, а также каждого отдельного задания курса.

Общая статистика по курсу содержит ссылку на репозиторий курса и включает таблицу с баллами обучающихся по каждому из заданий с полями ввода окончательной оценки за курс. К каждому полю ввода прилагается рекомендуемое значение оценки, рассчитанное по следующей формуле:

$$V = \frac{\sum_{i=1}^t v_i}{t}, \quad (1)$$

где  $v_i$  – количество баллов обучающегося за задание под номером  $i$ ;

$t$  – количество заданий в курсе.



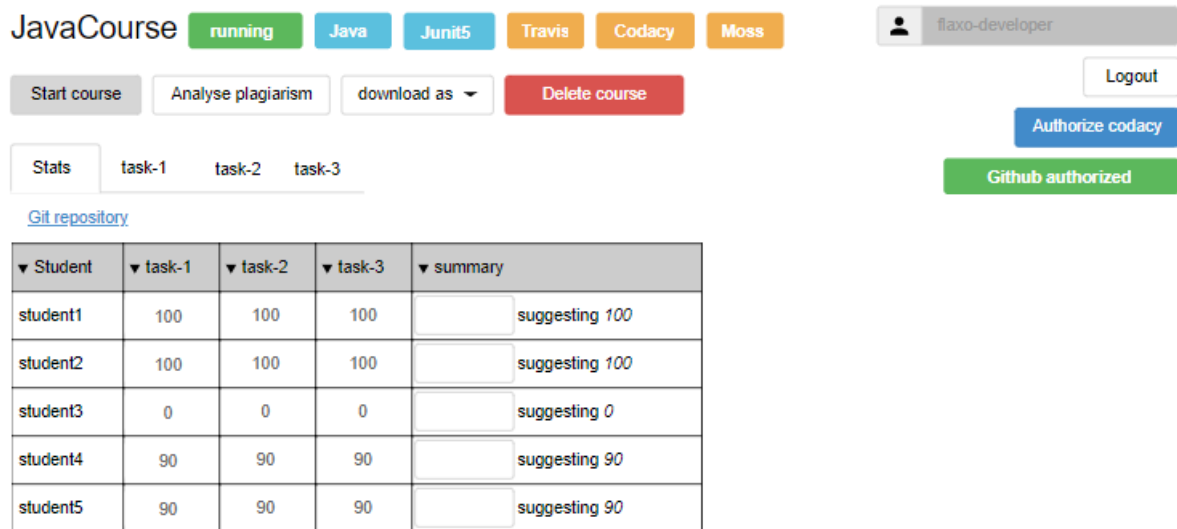


Рисунок 12 – Макет представления обобщенной статистики по курсу

Статистика по заданию (Рисунок 13) содержит ссылки на результат анализа плагиата, ветку задания в репозитории и включает в себя таблицу, содержащую результаты каждого из обучающихся по следующим критериям:

- прохождение тестов,
- качество кода,
- найденные случаи подозрительные на плагиат,
- сдача в рамках установленного срока.

Помимо этого, таблица для каждого обучающегося содержит поле ввода оценки за задание. К каждому полю ввода прилагается рекомендуемое значение оценки, рассчитанное по следующей формуле для каждого обучающегося:

$$v = 60 \cdot b + s \cdot 5 + d \cdot 10, \quad (2)$$

где  $b \in [0,1]$  – параметр прохождения решением тестов: 0 – задание не прошло часть тестов, 1 – задание прошло все тесты;

$s \in [1,6]$  – значение оценки качества кода: 1 – плохое качество, 6 – хорошее качество;

$d \in [0,1]$  – параметр сдачи задания в срок: 0 – задание сдано позже дедлайна, 1 – задание сдано до дедлайна.

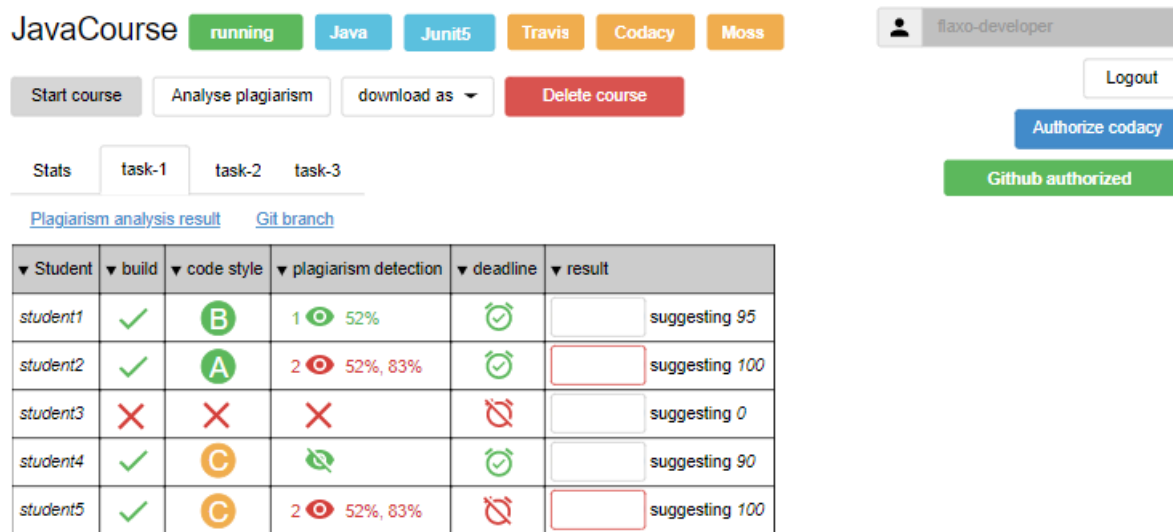
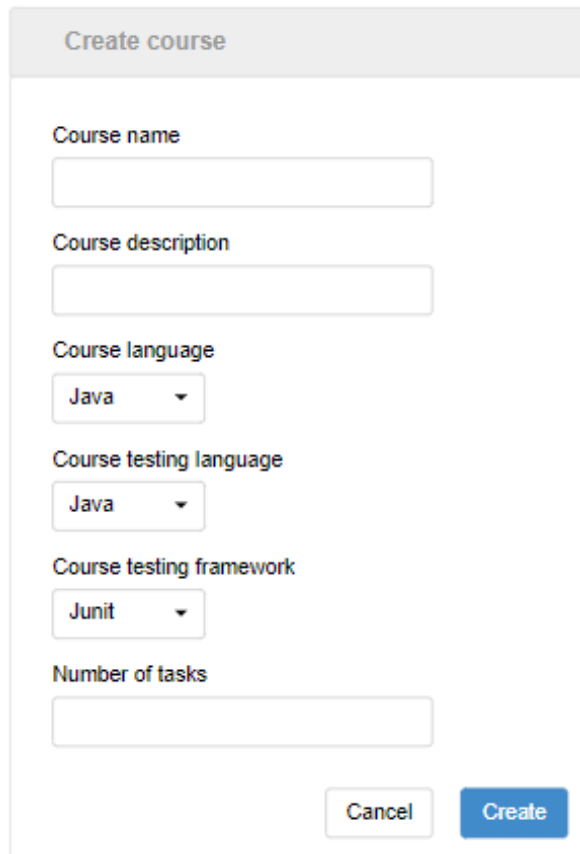


Рисунок 13 – Макет представления статистики по заданию курса

Модальное окно создания курса (Рисунок 14) содержит поля ввода названия курса, описания курса, числа заданий в курсе, а также выпадающие списки выбора языка, языка тестов и тестового фреймворка курса.



The image shows a modal window titled "Create course". It contains several input fields and dropdown menus. The fields are: "Course name" (text input), "Course description" (text input), "Course language" (dropdown menu with "Java" selected), "Course testing language" (dropdown menu with "Java" selected), "Course testing framework" (dropdown menu with "JUnit" selected), and "Number of tasks" (text input). At the bottom right, there are two buttons: "Cancel" and "Create".

Рисунок 14 – Макет представления модального окна создания курса

### 3.5 Реализация

По окончании проектирования, выбора технологий, инструментов и вендоров интегрированных сервисов следует этап непосредственной реализации системы.

Тестирование осуществляется непосредственно до, в момент и после написания любого кода и бизнес-логики. На момент окончания разработки всего функционала, описанного на этапе проектирования, программная реализация представляет из себя надежную и покрытую тестами систему, способную к развертыванию и использованию в любой момент на подходящем веб-сервере.

Исходный код проекта доступен по адресу <https://github.com/tcibinan/flaxo>, подробная документация к разработанной платформе присутствует в файле README.md, документация к исходному коду доступна в формате KDoc и может быть сгенерирована средствами языка Kotlin.

Скриншоты разработанной системы представлены на следующих рисунках:  
Рисунок 15, Рисунок 16, Рисунок 17.

The screenshot shows a modal window titled "Create course" with a close button (X) in the top right corner. The form contains several input fields and dropdown menus, each with a label and a small hint text below it:

- Course name**: A text input field. Hint: "Course name should be a valid git repository name".
- Course description**: A text input field. Hint: "Course description won't be visible for students".
- Language**: A dropdown menu with "java" selected. Hint: "Language solutions will be written on".
- Testing language**: A dropdown menu with "java" selected. Hint: "Language tests will be written on".
- Testing framework**: A dropdown menu with "junit" selected. Hint: "Test framework to use in course".
- Number of tasks**: A text input field. Hint: "Number of git branches for tasks".

At the bottom right of the modal, there are two buttons: "Create" (blue) and "Cancel" (grey).

Рисунок 15 – Модальное окно создания курса

The screenshot shows a web interface for managing courses. At the top, there is a header bar with "Flaxo" on the left and "Signed as admin Options" on the right. Below the header, there is a "Create course" button. The main content area displays a list of courses:

- JavaCourse**: Includes tags for `running`, `travis`, `codacy`, `java`, `java`, and `junit`. It shows "2 tasks, 5 students" and a description consisting of five "Ambiguous course description." entries. It was created on "Thu Mar 08 2018" and has a "Repository" link.
- KotlinCourse**: Includes tags for `init`, `java`, `kotlin`, and `spek`. It shows "4 tasks, 0 students" and was created on "Thu Mar 01 2018". It also has a "Repository" link.

Рисунок 16 – Страница просмотра всех курсов

Flaxo Signed as admin Options ▾

**JavaCourse**
running
travis
codacy
java
java
junit

[Start course](#)
[Analyse plagiarism](#)
[Delete course](#)
[Download as ▾](#)

[Course summary](#)
[task1](#)
[task2](#)

**task1**
[Git branch](#)
[Plagiarism report](#)
[Save results](#)
[Rules](#)

#	Student	Build	Code style	Plagiarism detection	Deadline	Result
1	student1	✓	B	1 85%	🕒 95	<a href="#">suggesting 95</a>
2	student2	✓	A	1 22%	🕒 100	<a href="#">suggesting 100</a>
3	student3	✗	B	🕒	🕒 35	<a href="#">suggesting 35</a>
4	student4			🕒	0	<a href="#">suggesting 0</a>
5	student5	✓	C	2 85%, 22%	🕒 80	<a href="#">suggesting 80</a>

Рисунок 17 – Страница просмотра статистики задания

## 4 ВНЕДРЕНИЕ

Завершающим этапом написания данной дипломной работы является внедрение спроектированной и разработанной системы в реальный образовательный процесс.

Для проведения образовательного эксперимента была выбрана дисциплина «Информатика», преподаваемая студентам университета ИТМО первого курса, обучающимся по программе «Нейротехнологии и программирование».

Студентам было предложено пройти курс (на данный момент курс доступен по ссылке <https://github.com/tcibinan/data-structures-course>), созданный с помощью разработанной системы, содержащий два задания по программированию на Java, которые заключались в реализации следующих структур данных, соответственно:

- Range – ряд целых чисел, обладающий следующим набор методов:
  - leftBound, rightBound - методы, возвращающие границы ряда,
  - isBefore, isAfter - методы проверки порядка следования двух рядов,
  - isConcurrent - метод проверки пересечения двух рядов,
  - contains - метод проверки вхождения элемента в ряд,
  - asList - метод получения ряда в виде списка чисел,
  - asIterator - метод получения ряда в виде итератора чисел.
- IntSet – множество целых неотрицательных чисел, обладающее следующим набором методов:
  - add - добавление элемента во множество,
  - remove - удаление элемента из множества,
  - contains - проверка присутствия элемента во множестве,
  - size - количество чисел во множестве,
  - isSubsetOf - проверка на вхождение одного множества в другое,
  - union - объединение множеств,

- intersection - пересечение множеств,
- minus - вычитание множеств,
- difference - исключающее ИЛИ на множествах.

Для курса было подготовлено общее описание (Рисунок 18), детальное описание алгоритма получения и сдачи заданий, включающее скриншоты и примеры необходимых для ввода команд git. Для каждого из заданий были созданы документы с описанием того, что из себя представляет соответствующая структура данных. Помимо этого, были разработаны:

- классы-заглушки (Рисунок 19), повторяющие каждую из описанных структур данных и содержащие подробное описание контракта в виде JavaDoc,
- наборы тестов на языке Kotlin с использованием фреймворка тестирования Spek (Рисунок 20), покрывающие большую часть аспектов работы описанных структур данных.

## Структуры данных на Java

В данном учебном курсе вы сможете потренироваться в написании простейших структур данных на языке Java.

Для прохождения данного курса необходимо:

- установить систему контроля версий `git` и обладать базовыми навыками работы с `git` или внимательно читать и использовать предложенные команды;
- установить `java 8`;
- использовать удобную IDE, например, IntelliJ Idea.

## Правила курса

Для получения максимально возможной оценки, вам необходимо написать решение, удовлетворяющее тестам и имеющее высокий уровень качества кода.

Также все решения будут проверяться на плагиат. Поэтому списывать **крайне не рекомендуется**, но подглядывать **можно**.

Переотправлять решения (пушить новые коммиты в открытый pull request) можно неограниченно число раз до наступления дедлайна.

Рисунок 18 – Главная страница описания курса

```

/**
 * Добавляет число ко множеству.
 *
 * @param value Число, которое необходимо добавить во множество.
 */
public void add(final int value) {
    // todo: Необходимо добавить реализацию метода
    throw new UnsupportedOperationException("Method is not implemented yet");
}

/**
 * Удаляет число из множества.
 *
 * @param value Число, которое необходимо удалить из множества.
 */
public void remove(final int value) {
    // todo: Необходимо добавить реализацию метода
    throw new UnsupportedOperationException("Method is not implemented yet");
}

/**
 * Проверяет, содержится ли значение во множестве.
 *
 * @param value Число, наличие которого во множестве необходимо проверить.
 * @return true если множество содержит значение, иначе - false.
 */
public boolean contains(final int value) {
    // todo: Необходимо добавить реализацию метода
    throw new UnsupportedOperationException("Method is not implemented yet");
}

```

Рисунок 19 – Фрагмент класса-заглушки IntSet

```

on("minus") {
    val mainSetValues = intArrayOf(1, 2, 3)
    val otherSetValues = intArrayOf(2, 3, 4)
    val mainSet = IntSet.of(*mainSetValues)
    val otherSet = IntSet.of(*otherSetValues)

    val minus = mainSet.minus(otherSet)

    it("should return a set that contains elements from the first set " +
        "which are not presented in the second one") {
        mainSetValues.minus(otherSetValues.toSet())
            .forEach {
                assertTrue("set should contain $it") { minus.contains(it) }
            }
    }
}

```

Рисунок 20 – Фрагмент теста структуры данных IntSet

На момент начала решения заданий студентами был подготовлен видео ролик (Рисунок 21), целиком показывающий и описывающий процесс работы студента с курсом. Данный ролик первоначально не являлся необходимостью, однако, как оказалось, большинство студентов никогда до этого момента не работали с системами контроля версий, и поэтому для них не было самой простой задачей использовать git для получения и сдачи заданий.



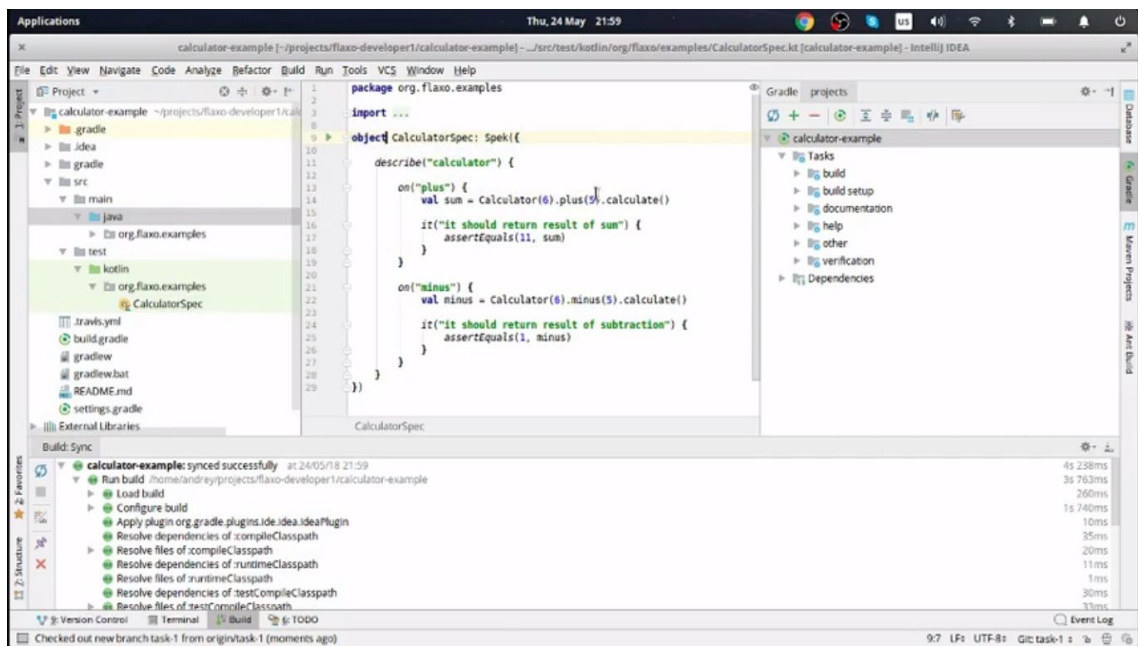


Рисунок 21 – Фрагмент обучающего видеоролика

В ходе всего эксперимента система показала стабильность своей работы, а некоторые выявленные недостатки и баги удалось оперативно исправить, не затронув процессов сдачи студентами решений и их оценки. Система выполнила возложенную на нее функцию и смогла обработать более 60 сданных студентами решений. Помимо прочего, системой был произведен поиск случаев плагиата в работах и подготовлен подробный отчет для каждого из решений с числом всех замеченных подозрительных случаев и процентами похожих строчек кода для каждого из них (Рисунок 22). Если количество строчек кода в файлах разных авторов совпадало (с точки зрения плагиата) более, чем на 80 процентов, решения обоих помечались красным цветом, как подозрительные.

## task-1

[Git branch](#)
[Plagiarism report](#)
[Save results](#)
[Rules](#)


#	Student	Build	Code style	Plagiarism detection
1	KuzmichevaKsenia	✓	15	50%, 40%, 40%, 38%, 37%, 34%, 27%, 27%, 27%, 27%, 22%, 21%, 14%, 14%, 13%
2	EduardDepo	✓	15	94%, 94%, 84%, 39%, 38%, 24%, 24%, 22%, 20%, 15%, 14%, 14%, 14%, 13%, 12%

Рисунок 22 – Подозрительное на плагиат решение студента под номером два

На протяжении всего курса система агрегировала результаты выполнения работ обучающимися и составляла отчет по текущей успеваемости.

По окончании курса из системы был выгружен отчет об успеваемости в формате csv, который был использован преподавателем для проставления баллов студентам во внутренней системе Университета ИТМО – de.ifmo.ru.

По окончании проведения эксперимента, со всех обучающихся была собрана обратная связь. Обучающимся было предложено пройти небольшой опрос, результаты анализа которого приведен ниже в данном разделе. Всего в опросе приняли участие 25 человек.

Как видно на рисунках ниже (Рисунок 23, Рисунок 24), в своем большинстве (84%) студентам понравилось использование системы, а также 68% из них хотели бы получать задания по программированию с использованием разработанной системы и дальше в процессе обучения в университете.

### Общее впечатление

25 ответов

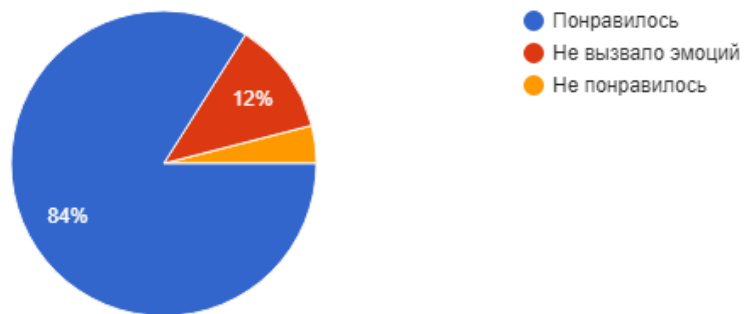


Рисунок 23 – Диаграмма общего впечатления от прохождения курса

### Хотели бы вы получать и проходить задания по программированию с использованием такой системы и дальше в обучении?

25 ответов

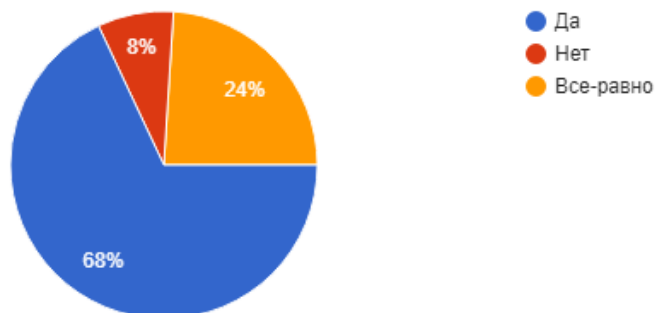


Рисунок 24 – Диаграмма востребованности системы в дальнейшем обучении с точки зрения студентов

Отдельной частью исследования опыта прохождения курса обучающимися было выявление сильных и слабых сторон каждого из видов описания заданий:

- через непосредственное описание заданий в виде README.md,
- через описание отдельных методов и классов в виде JavaDoc,
- через написание тестов таким образом, чтобы они покрывали отдельные части поведения разрабатываемого модуля.

Как видно из диаграммы (Рисунок 25) наиболее полезным оказалось описание структур данных через JavaDoc, а написание подробных тестов поведения структур данных оказались наименее полезными. Изначально предполагалось, что JavaDoc и тесты должны быть одинаково полезны обучающимся, поэтому такой результат не является ожидаемым. Однако, причина данного отклонения может крыться в том, что, как видно на рисунке выше (Рисунок 26) только 36% обучающихся регулярно запускали тесты, а 56% запускали тесты только в самом конце разработки, чтобы удостовериться в работоспособности своей структуры. Хотя и в описании курса были рекомендации по процессу выполнения заданий, как видно, не все студенты следовали им. В результате чего, статистика по полезности того или иного вида непосредственного описания заданий не может являться показательной.

#### Какие из видов описания заданий были полезны?

25 ответов

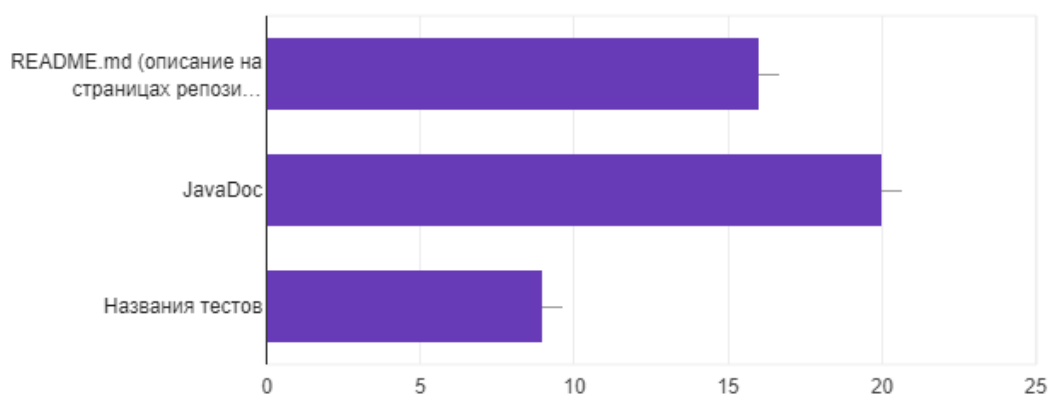


Рисунок 25 – Диаграмма предпочтений вида описаний заданий

### Часто ли вы запускали тесты?

25 ответов

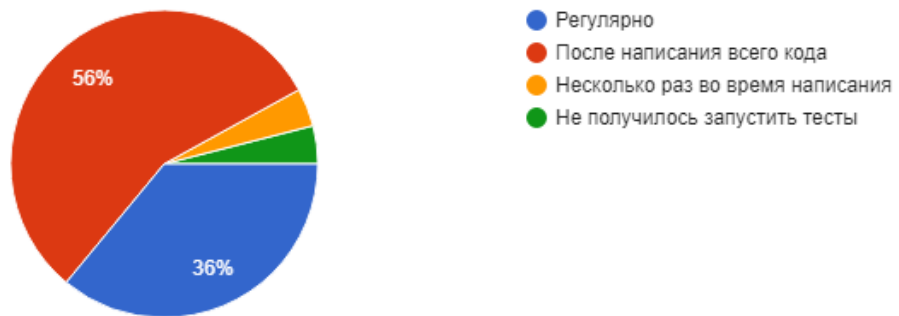


Рисунок 26 – Диаграмма частоты запуска тестов обучающимися

Одной из сложностей, с которыми столкнулись обучающиеся были установка и использование различных инструментов и технологий: git, gradle, IntelliJ Idea. Как оказалось, только 20% обучающихся (Рисунок 27) имели опыт работы с системой контроля версий git.

### Были ли вы знакомы с системой git до прохождения этого курса?

25 ответов

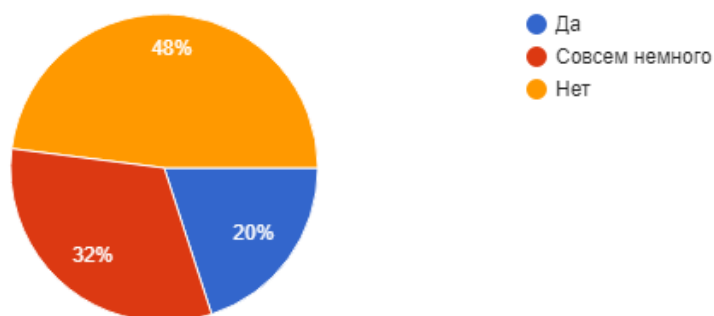


Рисунок 27 – Диаграмма опыта работы студентов с git

Как уже было сказано, для студентов был подготовлен вспомогательный видеоролик, который показывал, как происходит процесс получения и сдачи заданий с использованием разработанной системы. Видеоролик оказался крайне

полезен обучающимся (Рисунок 28), несмотря на отсутствие нескольких неосвещенных тем, которые были отданы на самостоятельную проработку студентов (установка Java 8, git, IntelliJ Idea с плагином для тестирования).

#### Было ли полезно вспомогательное видео?

25 ответов

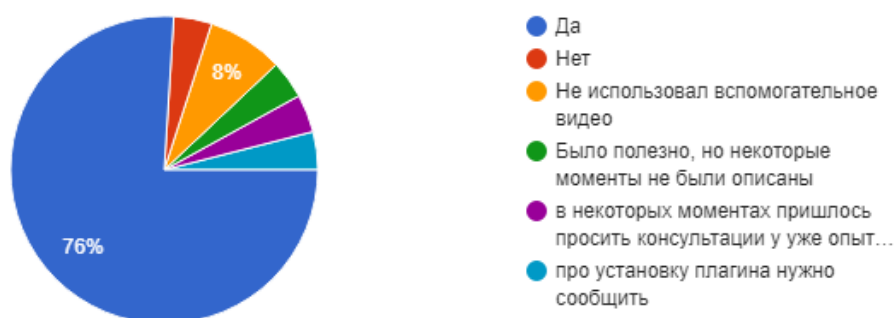


Рисунок 28 – Диаграмма полезности вспомогательного видео

Обратная связь была получена и от преподавателя. Внедренная система произвела положительное впечатление, позволив сократить большое количество времени за счет автоматизации процессов сдачи и проверки работ.

Наиболее интересными и полезными функциями системы, с точки зрения преподавателя, оказались:

- автоматический подсчет предлагаемого балла за решение,
- экспорт статистики для использования во внутренних системах университета,
- возможность проверки работ на плагиат.

Однако, по мнению преподавателя, существует также ряд функций, присутствие которых является востребованным, но на данный момент не реализованным:

- возможность добавлять несколько вариантов для заданий,
- возможность кастомизации функции вычисления предлагаемого балла за решение.

Оба требования являются совершенно обоснованными и сложность их добавления в разработанную систему не предполагает большого количества изменений.

Собранная и проанализированная обратная связь позволяет сделать некоторый вывод о текущем состоянии системы и о ее дальнейшем развитии. Разработанная система на данный момент готова для использования в реальном процессе обучения и позволяет сократить большое количество времени как преподавателю, так и обучающемуся за счет автоматизированного процесса создания и проведения образовательных курсов, а также автоматического анализа решений обучающихся.

## **ЗАКЛЮЧЕНИЕ**

В результате выполнения данной дипломной работы была спроектирована, разработана и протестирована распределенная интегрированная система создания, прохождения и сдачи заданий по программированию, сочетающая в себе большинство положительных сторон существующих на сегодняшний день аналогичных систем, а также выдвигающая на новый уровень автоматизацию процессов создания обучающих курсов по программированию и проверки решений обучающихся с точки зрения прохождения тестов, качества кода и отсутствия плагиата.

Собранная обратная связь позволяет говорить о возможном предстоящем внедрении разработанной системы в образовательный процесс студентов Университета ИТМО. Также система предположительно является востребованной в современной среде промышленного программирования и может быть использована в качестве инструмента обеспечения образовательного процесса. На данный момент, разрабатывается план применения разработанной системы для обучения программированию на стажировках одной коммерческой компании.



## **ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

В настоящей работе применяются следующие сокращения.

IDEF3 – Integrated Definition for Process Description Capture Method

PFDD – Process Flow Description Diagrams

RLCP – Remote Laboratory Control Protocol

MVC – Model-View-Controller

API – Application programming interface

REST – Representational state transfer

ООП – Объектно-ориентированное программирование

SPA – Single page application

DSL – Domain-specific language

BDD – Behavior driven-development

JVM – Java virtual machine

## СПИСОК ИЛЛЮСТРАТИВНОГО МАТЕРИАЛА

Рисунок 1 – Основная диаграмма IDEF3 .....	14
Рисунок 2 – Диаграмма IDEF3 работы с курсами .....	15
Рисунок 3 – Диаграмма IDEF3 работы с решениями .....	15
Рисунок 4 – Диаграмма IDEF3 сбора статистики .....	15
Рисунок 5 – Диаграмма вариантов использования .....	16
Рисунок 6 – Диаграмма последовательности сценария создания и инициализации курса.....	17
Рисунок 7 – Диаграмма последовательности сценария сдачи и проверки работы обучающегося системой .....	18
Рисунок 8 – Диаграмма последовательности сценария анализа решений в курсе на плагиат .....	19
Рисунок 9 – Схема модели базы данных .....	21
Рисунок 10 – Модель анализа плагиата системы Moss .....	27
Рисунок 11 – Макет представления всех курсов пользователя.....	29
Рисунок 12 – Макет представления обобщенной статистики по курсу .....	30
Рисунок 13 – Макет представления статистики по заданию курса .....	31
Рисунок 14 – Макет представления модального окна создания курса.....	32
Рисунок 15 – Модальное окно создания курса .....	33
Рисунок 16 – Страница просмотра всех курсов .....	33
Рисунок 17 – Страница просмотра статистики задания.....	34
Рисунок 18 – Главная страница описания курса.....	36
Рисунок 19 – Фрагмент класса-заглушки IntSet.....	37
Рисунок 20 – Фрагмент теста структуры данных IntSet.....	37
Рисунок 21 – Фрагмент обучающего видеоролика.....	38
Рисунок 22 – Подозрительное на плагиат решение студента под номером два.....	39
Рисунок 23 – Диаграмма общего впечатления от прохождения курса.....	40
Рисунок 24 – Диаграмма востребованности системы в дальнейшем обучении с точки зрения студентов .....	40
Рисунок 25 – Диаграмма предпочтений вида описаний заданий.....	41
Рисунок 26 – Диаграмма частоты запуска тестов обучающимися .....	42
Рисунок 27 – Диаграмма опыта работы студентов с git.....	42
Рисунок 28 – Диаграмма полезности вспомогательного видео .....	43

## СПИСОК ЛИТЕРАТУРЫ

1. Andrey V. Lyamin, Oleg E. Vashenkov. Virtual laboratory: Multi-Style code editor // 25th International Conference on Data Engineering. - 2009. - 9 с.
2. Vladimir L. Uskov, Robert J. Howlett, Lakhmi C. Jain. Smart Education and e-Learning 2017 // 4th International KES Conference on Smart Education and Smart e-Learning. - 2017. - 498 с.
3. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship // Prentice Hall. - 2008. - 464 с.
4. Leonard Richardson, Mike Amundsen, Sam Ruby. RESTful Web APIs: Services for a Changing World // O'Reilly Media. - 2013. - 406 с.
5. Alex Banks, Eve Porcello. Learning React: Functional Web Development with React and Redux // O'Reilly Media. - 2017. - 350 с.
6. Kotlin Language Documentation. URL: <http://kotlinlang.org/docs/kotlin-docs.pdf> (дата обращения: 03.06.2018). - 242 с.
7. John Ferguson Smart. BDD in Action: Behavior-driven development for the whole software lifecycle // Manning Publications; 1 edition. - 2014. - 384 с.
8. Scott Chacon, Ben Straub. Pro Git // Apress. - 2014. - 456 с.
9. Software Source Code Plagiarism Detection Using Latent Semantic Analysis // International Journal of Science and Research. - 2016. - 5 с
10. Saul Schleimer, Daniel S. Wilkerson и Alex Aiken. Winnowing: Local Algorithms for Document Fingerprinting. - 2003. - 10 с.