

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«Тольяттинский государственный университет»

Институт математики, физики и информационных технологий
Кафедра «Прикладная математика и информатика»

01.03.02 ПРИКЛАДНАЯ МАТЕМАТИКА И ИНФОРМАТИКА

СИСТЕМНОЕ ПРОГРАММИРОВАНИЕ И КОМПЬЮТЕРНЫЕ ТЕХНОЛОГИИ

БАКАЛАВРСКАЯ РАБОТА

на тему Разработка автоматизированной системы проверки заданий для
дистанционного обучения

Студент _____ Л.А. Петкина _____

Руководитель _____ Т.Г. Султанов _____

Консультант _____ Н.В. Яценко _____
по аннотации

Допустить к защите

Заведующий кафедрой _____ к.т.н., доцент А.В. Очеповский _____

« _____ » _____ 2017 г.

Тольятти 2017

АННОТАЦИЯ

Бакалаврскую работу выполнила студентка Тольяттинского государственного университета Петкина Любовь Анатольевна.

Название работы: «Разработка автоматизированной системы проверки заданий для дистанционного обучения».

Объект исследования – процесс проверки заданий с использованием дистанционных технологий. Целью данной работы является автоматизация проверки заданий при использовании дистанционных технологий.

Для достижения поставленной цели в работе решаются следующие задачи:

1. Изучение проблем процесса проверки заданий при дистанционном обучении.
2. Построение концептуальной модели предметной области для выявления процессов, требующих автоматизирования.
3. Построение архитектуры новой технологии.
4. Разработка автоматизированной системы проверки заданий.
5. Тестирование разработанной системы и описание процесса

Работа состоит из введения, трёх глав и заключения.

Во введении определены актуальность темы, цели и задачи, поставленные в работе, объект и предмет исследования.

Первая глава посвящена исследованию предметной области, постановке задачи, выявлению процессов автоматизации, нахождению проблем в существующей технологии проверки заданий.

Во второй главе выявлены требования к новой технологии, смоделирована архитектура новой системы проверки заданий для дистанционного обучения.

В третьей главе описаны разработка и тестирование системы автоматизированной проверки заданий в виде программного кода для дистанционного обучения, на основе предъявляемых требований.

В заключении подводятся итоги исследования, формируются окончательные выводы и результаты проведенной работы.

Работа состоит из пояснительной записки на 57 страницах, введения, трех глав, заключения, включая 25 рисунков, 2 таблицы, список использованной литературы содержит 30 источников.

Результаты данной работы могут быть использованы в модификации работы дистанционных технологий.

ABSTRACT

The title of the graduation work is «Development of an Automated Task Checking System for Distance Learning».

This graduation work is about the creation of an automated system for checking students' tasks.

The object of the graduation work is the process of checking of tasks using remote technologies.

The aim of the work is to give some information about to automated system job verification when using remote technologies.

This system can check tasks automatically, almost without the participation of the teacher. The student will only download the assignment, and the teacher will watch the result. The downloaded tasks are divided into 3 types: program code, documents and server applications. This task verification system is designed to automate the process, make it more convenient and faster. Overall, the results suggest that the developed system will simplify the life of teachers and students.

The graduation work consists of introduction, three parts, conclusions, the list of references and appendix.

The first part is devoted to the study of the subject area, setting the task, identifying the automation processes, finding problems in the existing task verification technology.

The second part identifies the requirements for the new technology, construction of the architecture of the new job verification system for distance learning.

The third part describes the development and testing of the automatic job verification system for program code.

The result of the work is an automated system for checking the tasks for distance learning, which allows to simplify the work of the teacher, increasing the quality of work.

The graduation work consists of 57 pages, introduction, three parts including 25 figures, 2 tables, the list of 30 references including 5 foreign sources.

Содержание

ВВЕДЕНИЕ.....	3
1 АНАЛИЗ ПРОЦЕССА ПРОВЕРКИ ЗАДАНИЙ ДЛЯ ДИСТАНЦИОННОГО ОБУЧЕНИЯ	5
1.1 Общая характеристика Тольяттинского государственного университета.....	5
1.2 Анализ существующей технологии проверки заданий.....	8
1.3 Выявление недостатков существующего процесса.....	13
1.4 Анализ существующих систем для проверки заданий.....	15
2 ПРОЕКТИРОВАНИЕ АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ПРОВЕРКИ ЗАДАНИЙ.....	17
2.1 Формализация требований к новой технологии	17
2.2 Функциональное моделирование автоматизированной системы ..	19
2.3 Архитектура автоматизированной системы проверки заданий	23
2.3.1 Модуль тестирования документов	25
2.3.2 Модуль тестирования серверных приложений	27
2.3.3 Модуль тестирования программного кода	30
3 РАЗРАБОТКА АВТОМАТИЗИРОВАННОЙ СИСТЕМЫ ПРОВЕРКИ ЗАДАНИЙ В ВИДЕ ПРОГРАММНОГО КОДА	33
3.1 Концептуальное и логическое моделирование данных	33
3.2 Выбор средств реализации автоматизированной системы.....	35
3.3 Физическая модель базы данных	36
3.4 Формальное описание процесса проверки заданий в виде программного кода	37
3.5 Описание основного принципа работы автоматизированной системы	39
ЗАКЛЮЧЕНИЕ	43
СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ	44
ПРИЛОЖЕНИЕ А. Листинг кода.....	46

Высшее образование – это важная составляющая в современной жизни человека. Качественное образование позволяет получить хорошую работу, открывая перед человеком огромные горизонты в будущем. Во всем мире уделяется большое внимание образовательному процессу. В частности в России с каждым годом открывается все больше высших учебных заведений, появляются разнообразные специальности, в которых нуждается наше общество. Для удобства получения знаний студентами существуют разные формы обучения: всеми известные очная и заочная формы. Но в последнее время все больше набирает популярность заочная форма обучения с применением дистанционных технологий.

В Тольяттинском государственном университете существует возможность получения высшего образования на основе дистанционного обучения, которое существенно дополняет и расширяет традиционные формы организации образовательного процесса. Неоспоримые плюсы дистанционного образования в том, что обучение проводится без отрыва от работы и повседневной жизни, учебный процесс осуществляется практически без выездов с места проживания обучающихся. В завершении некоторых модулей учебного процесса, студенты сдают работы и проходят тестирование.

Все три формы обучения объединяет необходимость тестирования знаний студентов, а именно проверка практических заданий. Сейчас этот процесс не автоматизирован, но это бы позволило значительно упростить работу преподавателей, и сэкономило бы значительную часть времени.

Цель выпускной квалификационной работы (ВКР) – автоматизация технологии проверки заданий, выполненных студентами по различным дисциплинам.

Объект исследования - процесс проверки заданий студентов, обучающихся с применением дистанционных технологий.

Предметом исследования ВКР является автоматизация проверки заданий студентов.

Актуальность работы обусловлена необходимостью автоматизации проверки заданий студентов, за счет сокращения большого количества времени, затраченного на данный процесс.

Для достижения поставленной цели, необходимо решить следующие **задачи**:

- проанализировать существующий процесс проверки заданий;
- выявить недостатки существующего процесса;
- сформировать требования к новой автоматизированной системе;
- разработать архитектуру новой автоматизированной системы проверки заданий;
- разработать и протестировать систему, на основе построенной архитектуры и выявленных требований.

Выпускная квалификационная работа состоит из введения, трёх глав, заключения, списка литературы и приложений.

Во введении определены актуальность темы, цели и задачи, поставленные в работе, объект и предмет исследования.

Первая глава посвящена исследованию предметной области, постановке задачи, выявлению процессов автоматизации, нахождению проблем в существующей технологии проверки заданий.

Во второй главе выявлены требования к новой технологии, смоделирована архитектура новой системы проверки заданий для дистанционного обучения.

В третьей главе описаны разработка и тестирование автоматизированной системы проверки заданий в виде программного кода, на основе предъявляемых требований.

В заключении подводятся итоги исследования, формируются окончательные выводы и результаты проведенной работы.

1.1 Общая характеристика Тольяттинского государственного университета

В соответствии с Федеральным законом об образовании РФ [16], высшим учебным заведением является образовательное учреждение, учрежденное и действующее на основании законодательства Российской Федерации об образовании, имеющее статус юридического лица и реализующее в соответствии с лицензией образовательные программы высшего профессионального образования.

Основными задачами высшего учебного заведения являются:

- обучение студентов по разным направлениям подготовки;
- развитие научной и творческой деятельности студентов, с возможностью последующего внедрения полученных результатов исследования в процесс обучения;
- подготовка, переподготовка и повышение квалификации специалистов;
- повышение культурного и образовательного уровня населения.

Тольяттинский государственный университет является высшим учебным заведением и был создан 29 мая 2001 г. на основе Тольяттинского политехнического института и Тольяттинского филиала Самарского государственного педагогического университета. ТГУ является довольно значимым университетом в Самарской области, так как здесь подготавливаются и обучаются высококвалифицированные специалисты в совершенно разных областях, которые смогут внести вклад в развитие своего региона. Университет включает в себя 11 институтов, в которых обучаются около 12 тысяч студентов и работают 100 профессоров и докторов наук и 450 доцентов и кандидатов наук.

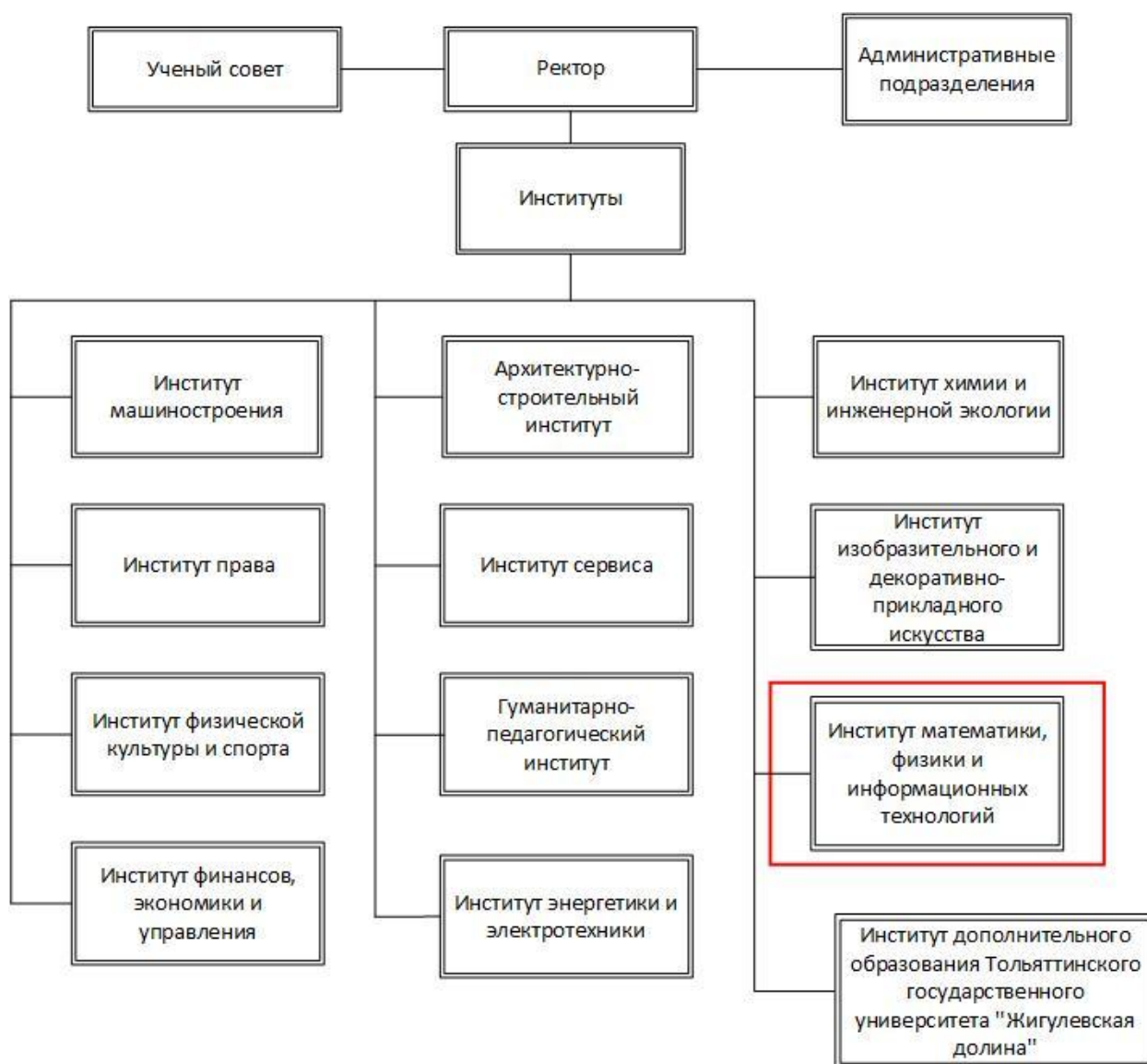


Рисунок 1.1 - Структура Тольяттинского государственного университета

В ТГУ существует возможность обучения по программам бакалавриата, специалитета, магистратуры и аспирантуры. На рисунке 1.2 представлены формы обучения в Тольяттинском Государственном Университете.

В университете проводятся различные виды контроля знаний студентов:

- Государственная итоговая аттестация выпускников университета – это обязательный вид аттестации выпускника; осуществляется после освоения основной образовательной программы высшего образования по направлению подготовки (включая выпускную квалификационную работу).
- Промежуточная аттестация (зачеты и экзамены).
- Текущая аттестация (контрольные, практические задания и т.п.).

Из-за большого количества институтов, а соответственно студентов, преподавателей, отделов, сопровождающих обучение и занимающихся поддержкой и улучшением работы всего университета, возрастает необходимость автоматизации всего учебного процесса. В частности нуждается в автоматизации именно текущая аттестация, так как она менее всего автоматизирована в университете.

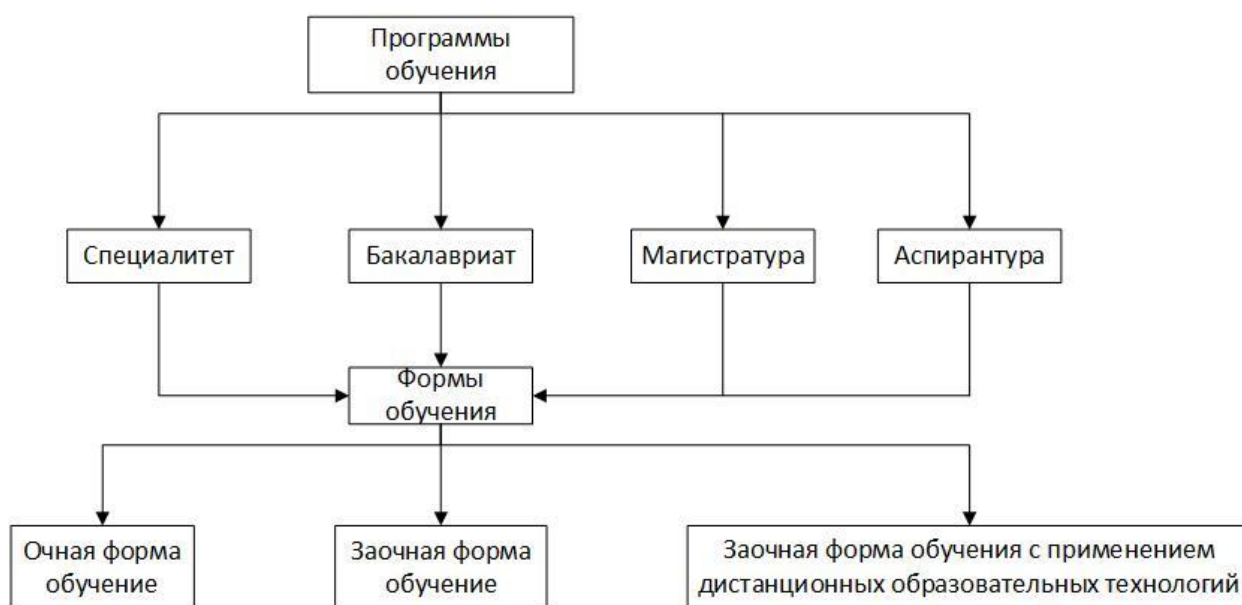


Рисунок 1.2 - Формы обучения в Тольяттинском Государственном Университете

Современный образовательный процесс включает в себя систему тестирования посредством web-сайта. Студент, используя логин и пароль, авторизуется на сайте, выбирает дисциплину, и проходит по ней тест, результаты теста обрабатываются в система, а затем студент видит свой результат. Результаты тестирования каждого студента находятся в базе данных, что позволяет преподавателю видеть статистику всех студентов.

В такой системе не полностью проверяются знания студента, так как его выбор на тесте ограничивается выбором ответа из нескольких вариантов. Таким образом, существует необходимость в разработке системы, которая будет обеспечивать автоматическую проверку практических заданий. Ниже рассмотрим существующий процесс проверки заданий у студентов.

1.2 Анализ существующей технологии проверки заданий

Процесс проверки заданий у студентов – это трудоемкий и отнимающий большое количество времени у преподавателей процесс, который, безусловно, является одним из важных в обучении. Ведь только так в течение учебного семестра можно понять уровень знаний каждого обучающегося. Один преподаватель может обучать большое количество групп, а в каждой группе в среднем около 25-30 студентов. При изучении одного семестрового предмета чаще всего дается на самостоятельное закрепление около 10 практических работ. Огромное количество времени требуется на проверку каждой работы каждого студента.

Рассмотрим данный процесс более подробно. Студент или преподаватель видят эту систему с разных сторон. Для этого необходимо построить модель «КАК ЕСТЬ», учитывая знания всех участников данного процесса. Для подробного анализа бизнес-процессов была использована диаграмма по методологии IDEF0 [17].

Так как в университете существуют очная и заочная форма обучения рассмотрим существующий процесс со всех сторон и с точки зрения преподавателя.

На очной форме обучения, как и на стандартной заочной, проверка заданий у студентов проходит во время пары по данной дисциплине. Студент открывает выполненную работу на персональном компьютере или, если требуется, распечатывает отчет и показывает преподавателю выполненные задания. Затем преподаватель проверяет работу на правильность и выставляет оценку (баллы), если работа не проходит проверку, то возвращается на доработку с указанием ошибок. Данный процесс проверки используется довольно давно и уже устарел. При такой проверке преподаватель не может проверить работу на уникальность или в полной мере протестировать все стороны данной работы.

На заочной форме с использованием технологий дистанционного обучения, процесс немного усовершенствован. Здесь появляется сайт, на

который студент загружает выполненное задание, а преподаватель проверяет работу на своем устройстве. Наличие сайта позволяет детальнее изучить и проверить работу, а также дает возможность преподавателю проверять работу в любое удобное для него время. Но и времени на проверку работы уходит большое количество.

Последовательность действий процесса проверки заданий у студентов, учащихся с использованием технологий дистанционного обучения:

- преподаватель заходит на сайт под своим логином/паролем;
- преподаватель выбирает студента, который загрузил выполненное задание;
- преподаватель скачивает работу и проверяет её на персональном компьютере;
- по результату проверки преподаватель оценивает работу или отправляет работу на доработку, оставляя рецензию.

На рисунке 1.3 представлена диаграмма процесса проверки заданий в методологии IDEF0 [18], как на очной форме обучения, так и на заочной с применением дистанционных технологий. То есть, преподаватель на входе получает выполненную студентом работу, затем проверяет задание. На выходе выставляет оценку или возвращает работу на доработку, то есть указывает на ошибки. Или на заочной форме, пишет рецензию по работе, в которой также указывает на неправильно сделанные задания. Управлением в данном процессе выступают рабочая программа по дисциплине и фонд оценочных средств, так как это нормативная база выполнения процесса.

Рабочая программа по дисциплине – это документ, который предназначен для реализации требований Федерального государственного образовательного стандарта по профессии или специальности. В ней содержится структура и содержание дисциплины, а также контроль и оценка результатов освоения учебной дисциплины.

Фонд оценочных средств – комплект методических материалов, которые нормируют оценивание результатов обучения.

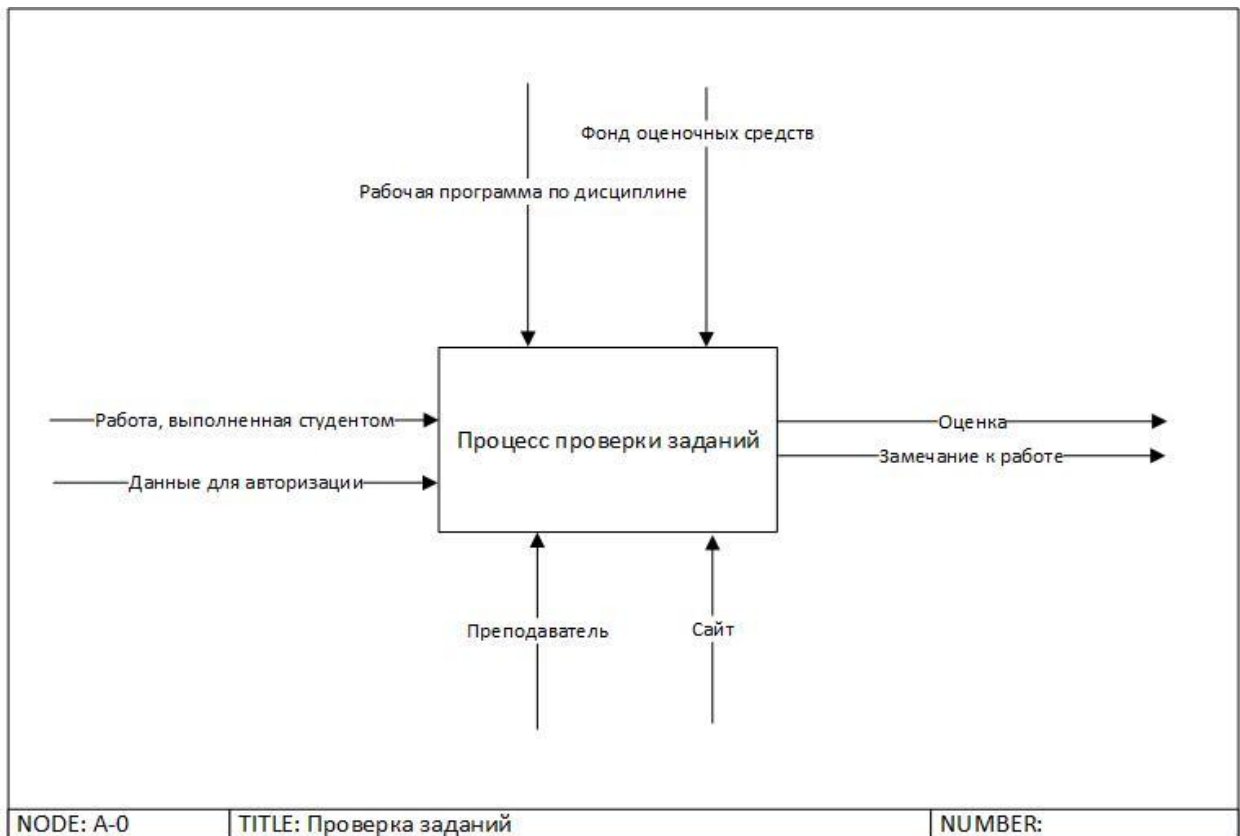


Рисунок 1.3 - Контекстная диаграмма проверки заданий для всех форм обучения «КАК ЕСТЬ» в методологии IDEF0 (0-й уровень)

Декомпозируем данную диаграмму процесса проверки заданий у студентов на заочной форме обучения с применением дистанционных технологий, потому что данный процесс включает в себя аспекты проверки заданий на очной и заочной форме. Хотя проверка заданий лично с преподавателем, широко используется практически во всех образовательных учреждениях до сих пор, но дистанционные технологии – это наше будущее, так как они помогают значительно упростить и усовершенствовать процесс обучения в целом.

Таким образом, на диаграмме (рисунок 1.4) представлена декомпозиция контекстной диаграммы (AS-IS) на основе проверки заданий у студентов на заочной форме обучения с применением дистанционных технологий, с помощью которой можно детальнее рассмотреть данный процесс. В ней находятся следующие процессы:

1. Вход на сайт.

У преподавателя, как и у студента, есть логин и пароль для входа в личный кабинет на сайт для дистанционного обучения. Преподаватель должен войти в свой личный кабинет для проверки работ студентов.

2. Выбор студента, загрузившего задание.

На сайте преподаватель выбирает дисциплину, период времени, группу студентов и уже конкретного студента, который загрузил задание.

3. Выгрузка выполненной работы.

Для того чтобы проверить выполненную работу преподаватель скачивает её на ПК. Работа может быть загружена в разных форматах (doc, txt и т.д.).

4. Проверка работы.

На вход данному процессу подается выполненная студентом работа, которая подразделяется на 2 типа: работа, выполненная на очном и заочном обучении, и работа, выполненная на заочном обучении с использованием дистанционных технологий.

Процесс проверки заданий – это процесс, охватывающий большое количество дисциплин, а соответственно заданий. Следовательно, задания можно классифицировать на следующие виды:

- Задания в виде программного кода – это задания, которые подразумевают написание программы на том или ином языке программирования, в соответствии с условиями задачи.
- Задания, выполненные в виде документов – это задания, подразумевающие выполнение в одном из видов программного обеспечения для работы с документами (например, Microsoft Office, OpenOffice и т.п.). Они могут быть выполнены в разных форматах, таких как .doc, .xls, .ppt и так далее. В них формируются отчеты, схемы, диаграммы, таблицы. Если задания, выполненные в виде документов, проверяются сейчас лишь в ручную, то для проверки программного кода во всех формах обучения используется компилятор, которые запускает программу и выдает результат. То есть преподаватель запускает,

программу, в программном обеспечении, выбор которого зависит от языка программирования. И по выведенному результату, оценивает работу.

- Задания в виде серверных приложений – это задания, которые реализованы чаще всего в виде программ, к коду которых нет доступа. Наглядным примером является создание сайта.

Все эти виды заданий имеют сейчас одинаковый подход к их проверке.

5. Выставление оценки.

На основе проверки работы, преподаватель выставляет оценку. Если работа не принята, то преподаватель возвращает работу на доработку и пишет рецензию к ней, в которой указывает ошибки.

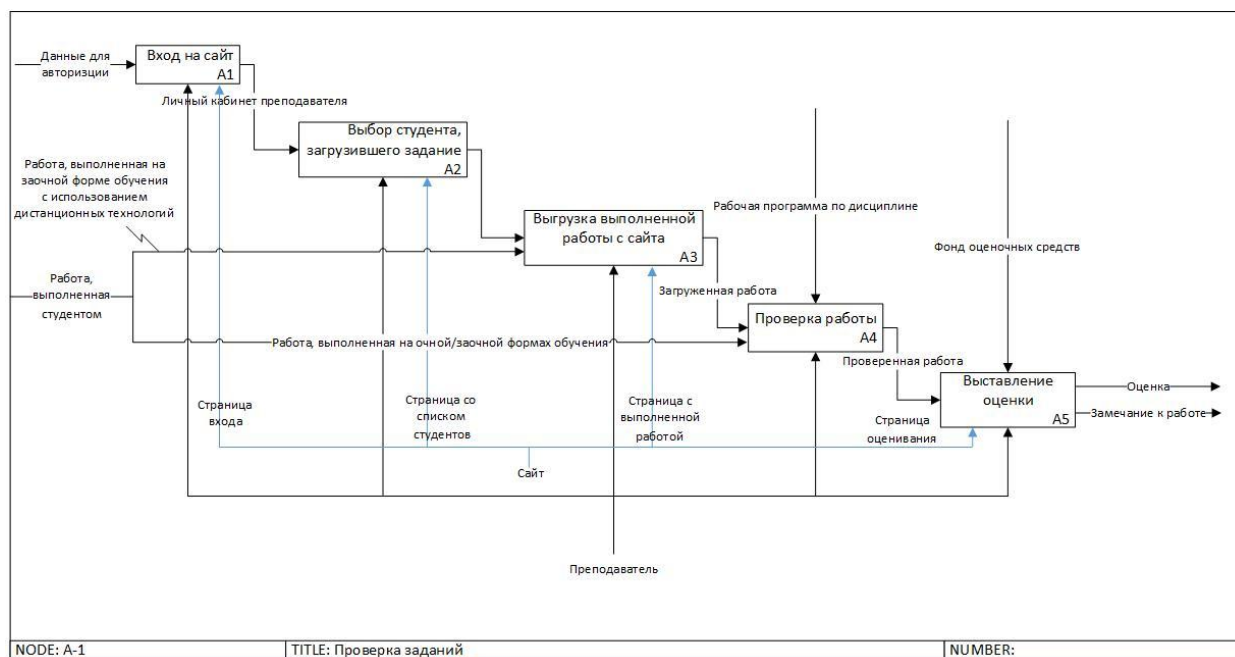


Рисунок 1.4. - Контекстная диаграмма проверки заданий для заочной формы обучения с применением дистанционных технологий «КАК ЕСТЬ» в методологии IDEF0 (1-й уровень)

На этой диаграмме изображены следующие элементы:

- Входные данные: данные для авторизации (логин/пароль), работа, выполненная студентом: работа, выполненная на очной/заочной форме обучения и на заочной форме обучения с применением дистанционных технологий;

- Выходные данные: оценка, замечание к выполненной работе;
- Управляющие воздействия: рабочая программа по дисциплине, фонд оценочных средств;
- Механизмы: преподаватель, сайт.

Данная модель является основой для анализа и дальнейшего совершенствования процессов.

1.3 Выявление недостатков существующего процесса

Анализ модели «КАК ЕСТЬ» позволил выделить недостатки на всех формах обучения во всех видах заданий, которые и будут рассмотрены далее.

Недостатки проверки работ, выполненных на очной и заочной форме обучения:

1. Ограниченное количество времени на проверку работы. Так как выполненная работа проверяется на паре по дисциплине, то проверке работы уделяется мало времени из-за большого количества студентов и временных рамок. Что порождает следующие недостатки.
2. Невозможность проверки работы на уникальность. Человеческий фактор не всегда может позволить преподавателю выявить работу, которую студент скачал с интернета или взял у своих одногруппников.
3. Неполный анализ работы. Из-за отсутствия достаточного количества времени на проверку, могут быть упущены ошибки или уникальные свойства работы.

Недостатки проверки работ, выполненных на заочной форме обучения с применением дистанционных технологий.

1. Отсутствие сроков на проверку работы преподавателем. Недостаток является существенным, если студент хочет как можно скорее узнать свой результат проверки.

2. Возможное отсутствие программного обеспечения у преподавателя для проверки работы.

Общие недостатки проверки работ для всех форм обучения по каждому типу заданий:

1. Недостатки проверки заданий, выполненных в виде программного кода.

Главным недостатком в проверке этого вида заданий является компиляция кода. Если программа сложная, а именно с множеством классов, объектов, то компиляция кода занимает большое количество времени даже за один запуск проекта, но программа чаще всего запускается несколько раз для тщательной проверки. Также если существует возможность написания работы на разных языках программирования, то преподавателю приходится иметь разные виды программного обеспечения.

2. Недостатки проверки заданий, выполненных в виде серверных приложений.

Проверка серверных приложений не предполагает доступа к коду. Поэтому преподавателю достаточно сложно проверить задание на соответствие требованиям. Например, если задание было создание сайта, и одно из требований к нему предполагало возможность регистрации и входа на сайт. Преподавателю придется проделывать регистрацию на всех сайтах студентов, что занимает огромное количество времени.

3. Недостатки проверки заданий, выполненных в виде документов.

Чаще всего практические работы выполняются в виде отчетов, таблиц и т.д. То есть однотипные задания, которые чаще всего подразумевают схожий результат, приходится проверять в ручную. Такая рутинная работа нуждается в автоматизации, хотя бы потому что это сэкономит достаточно времени.

1.4 Анализ существующих систем для проверки заданий

Для того чтобы корректнее и тщательней сформировать требования к новой автоматизированной системе проверки нужно рассмотреть существующие аналоги.

Moodle (Modular Object-Oriented Dynamic Learning Environment) – это система управления обучением. В системе создаются курсы для онлайн обучения. Курсы включают в себя материалы лекций, глоссарий, дополнительные ресурсы, также для коммуникации существует форум, обязательно все курсы содержат страницу с оценками и списком учащихся. Уровень знаний в таких курсах определяется с помощью тестов, что далеко не всегда позволяет верно, и со всех сторон оценить студента [8].

В общем, Moodle – это популярная среда обучения, которая находится в бесплатном доступе, но нам она не подходит, так как нет основной функциональной возможности – автоматической проверки заданий, а это перечеркивает весь список нужных возможностей для системы.

Ejudge, PCMS2, Contester – это автоматизированные системы, предназначенные для автоматической проверки программного кода. Системы давно применяются для проведения олимпиад по программированию. Данные решения также не удовлетворяют запрашиваемым требованиям, так как системы проверяют только один тип заданий: программный код, а документы и серверные приложения проверить не удастся. То есть система ориентирована не на учебный процесс, а лишь для проведения соревнований [14].

Вывод к первой главе

Была проанализирована структура Тольяттинского государственного университета и существующий в нем процесс проверки заданий для дистанционного обучения, в котором были обнаружены существенные недостатки, основным из которых является большое количество времени, затраченное преподавателем на проверку заданий студентов. В результате

анализа была выявлена потребность в создании автоматизированной системы проверки.

Проведение анализа существующих систем проверки заданий показал, что ни одно из рассмотренных программных средств полностью не удовлетворяет требованиям разрабатываемого сервиса.

Поэтому было принято решение о разработке нового сервиса для управления контекстными объявлениями.

2.1 Формализация требований к новой технологии

Целью создания новой технологии проверки заданий у студентов, учащихся с использованием технологий дистанционного обучения, является упрощение и автоматизирование существующего процесса, а также устранение недостатков данного процесса.

К целевой аудитории разрабатываемой технологии можно отнести:

- студентов;
- преподавателей.

Для формирования требований к новой технологии существуют различные классификации требований, одна из которых FURPS+ [1].

Классификация FURPS+ образована от первых букв слов:

- Functionality (функциональные требования);
- Usability (требования к удобству работы);
- Reliability (требования к надежности);
- Performance (требования к производительности);
- Supportability (требования к простоте поддержки). наилучшим образом поможет сформировать список требований, которые будут покрывать весь функционал.

В работе использовалась данная классификация, так как она наиболее универсальная и наилучшим образом поможет описать все требования к новой технологии (таблица 2.1).

Таблица 2.1 - Требования к системе

	Требование	Статус	Полезность	Риск	Стабильность
1	Автоматическая проверка работы студента	Одобрены	Критичное	Средний	Средняя
2	Поддержка тест-кейсов, в зависимости от выбранной дисциплины и номера задания.	Одобрены	Критичное	Средний	Низкая

	Требование	Статус	Полезность	Риск	Стабильность
3	Получение на выходе оценку или номер задания, в котором обнаружена ошибка.	Одобренные	Критичное	Средний	Средняя
4	Проверка работы на уникальность.	Одобренные	Критичное	Средний	Средняя
5	Информирование студента и преподавателя о результатах проверки.	Одобренные	Критичное	Средний	Средняя
6	Создание и редактирование преподавателем тест-кейсов	Одобренные	Критичное	Средний	Средняя
7	Разграничение доступа для студента и преподавателя	Одобренные	Критичное	Средний	Средняя
8	Формирование логов проверки	Одобренные	Критичное	Средний	Средняя
9	Создание отчета о выполненных и оцененных работ	Одобренные	Критичное	Средний	Средняя
10	Простота управления контентом	Одобренные	Критичное	Низкий	Низкая
11	Оповещение о результатах проверки должно приходиться на почту преподавателю и студенту	Одобренные	Критичное	Низкий	Низкая
12	Доступ пользователям должен быть обеспечен 24 часа в сутки.	Одобренные	Критичное	Низкий	Низкая
13	Время реакции системы на	Одобренные	Критичное	Средний	Средняя

	Требование	Статус	Полезность	Риск	Стабильность
	события должно быть не более пяти секунд				
14	Кроссбраузерность	Одобрённые	Критичное	Средний	Средняя
15	Система должна запускать подготовленные тест-кейсы сразу после загрузки работы на сайт.	Одобрённые	Критичное	Средний	Средняя
16	Время восстановления работоспособности не более 60 минут	Одобрённые	Критичное	Средний	Средняя

Разрабатываемая система проверки заданий должна автоматически проверять задания следующих видов: задания в виде программного кода, документов и серверных приложений. На вход системе подается загруженное студентом задание, на выходе – зачтено или ошибка, в которой указан тест-кейс, в котором она обнаружена.

Таким образом, на этапе анализа и выработки требований к новой системе было принято, что система должна реализовывать 9 функциональных требований и 7 нефункциональных требований.

2.2 Функциональное моделирование автоматизированной системы

Для отображения функционального аспекта системы построим диаграмму вариантов использования процесса «КАК ДОЛЖНО БЫТЬ».

Диаграммы вариантов использования (use case diagram) применяются при анализе для моделирования видов работ, выполняемых организацией, и для моделирования функциональных требований к проектируемой системе при ее проектировании и разработке.

Диаграмма вариантов использования описывает функциональные возможности рассматриваемой информационной системы «КАК ДОЛЖНО БЫТЬ», предоставляя дополнительную информацию об отношениях между

различными вариантами использования и внешними пользователями-актерами. Также диаграмма позволяет определить границы рассматриваемой системы.

Варианты использования проявляются только в терминах того, как они проявляются, когда рассматриваются внешним пользователем, при всем этом не описывают, какие функциональные возможности предоставлены внутри системы. Диаграмма процесса проверки задания изображена на рисунке 2.1.

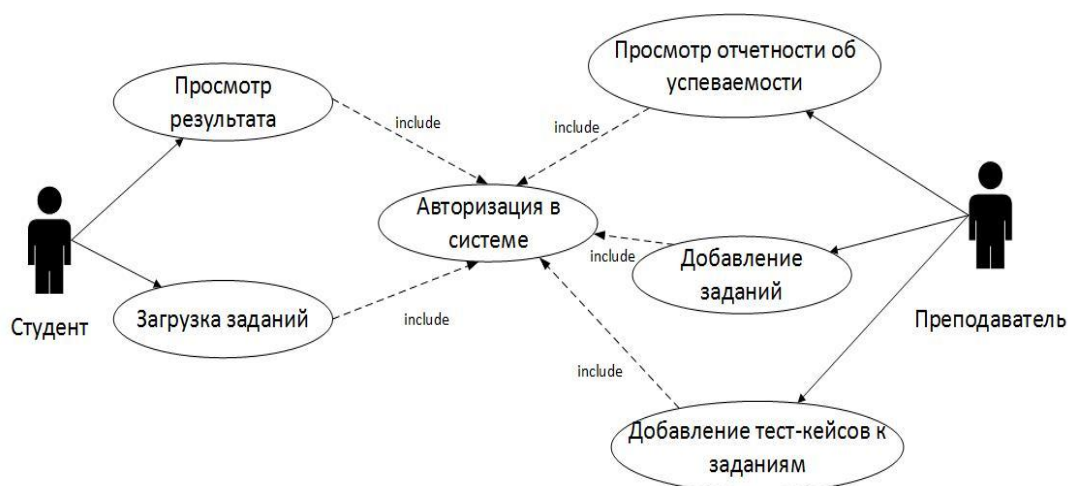


Рисунок 2.1 - Диаграмма вариантов использования

Актеры:

Студент – зарегистрированный пользователь, которому доступны функции загрузки заданий и авторизации на сайте, а также просмотр результатов проверки.

Преподаватель – зарегистрированный пользователь, которому доступны создания и редактирования тест-кейсов, а также просмотр отчетности об успеваемости студентов.

Варианты *использования*:

Авторизация в системе – вход в систему под личным логином и паролем.

Загрузка заданий – студент загружает выполненное задание в систему.

Проверка заданий – информационная система автоматически проверяет задания.

Добавление заданий и тест-кейсов – преподаватель составляет тест-кейсы для дисциплины, для того чтобы система могла проверить задания, то

есть чтобы у системы для конкретного задания при определенных входных данных был эталонный результат.

Просмотр отчетности об успеваемости – список студентов с результатами проверки заданий. Результат: «Зачтено»/«Не зачтено». При выводе на экран «Не зачтено» нужно выводить входные значения, в которых была допущена ошибка, для того чтобы студент мог исправить выполненное им задание.

Диаграмма вариантов использования предоставляет дополнительную информацию об отношениях между различными вариантами использования и внешними пользователями-актерами. Данная диаграмма вариантов использования отображает отношения между преподавателем и студентом и позволяет обрисовать систему на концептуальном уровне. Данные варианты использования системы распространяются на проверку заданий и программного кода, и документов, а также серверных приложений.

На основе диаграммы вариантов использования построим диаграмму последовательности для отображения временных особенностей передачи и приема сообщений объектами (рис. 2.2). Основными элементами диаграммы последовательности являются объекты (прямоугольники с названиями объектов), вертикальные «линии жизни», которые отображают течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, которые показывают обмен сигналами или сообщениями между объектами.

На данной диаграмме изображены следующие объекты: студент, преподаватель, информационная система, вертикальные «линии жизни», которые отображают течение времени, прямоугольники, отражающие деятельность объекта и стрелки, которые показывают обмен сигналами или сообщениями между объектами.

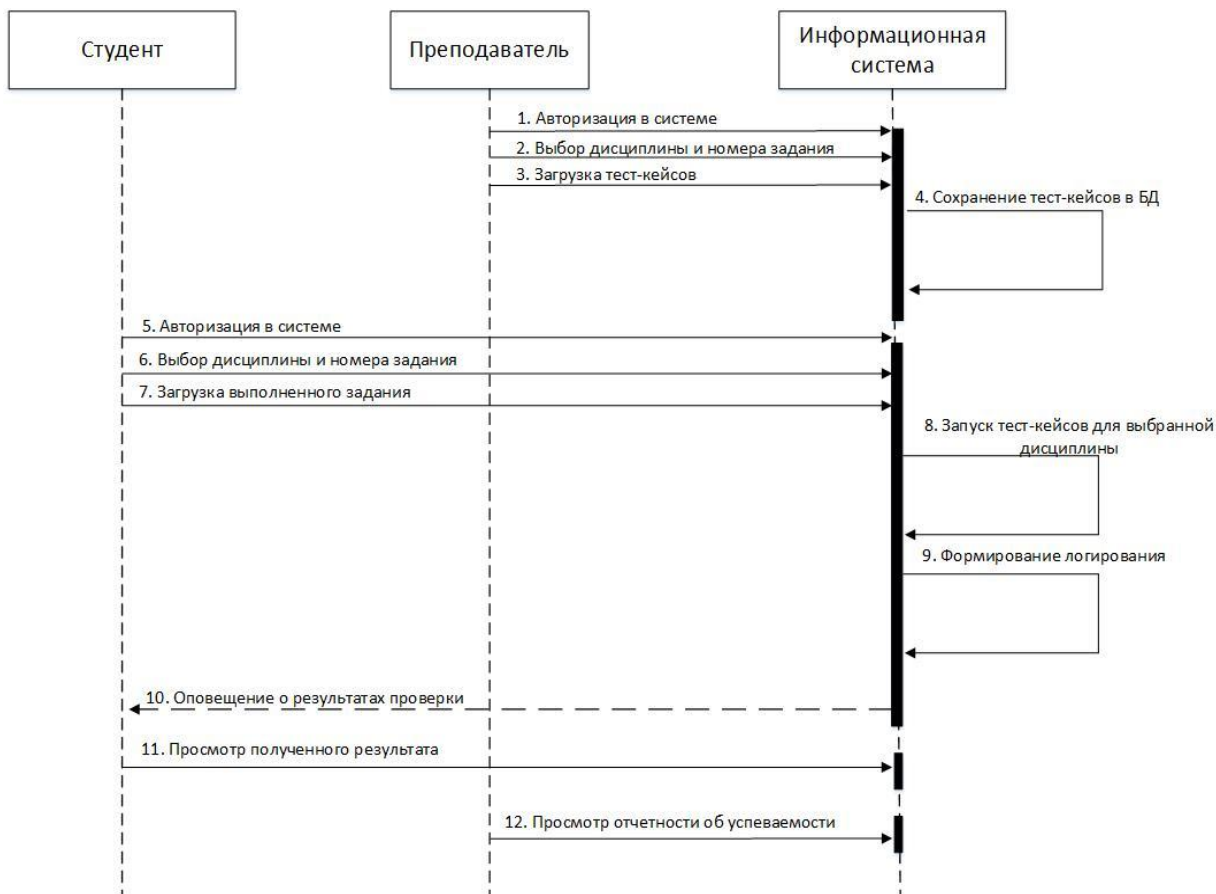


Рисунок 2.2 – Диаграмма последовательности автоматизированного процесса проверки заданий

На этих двух диаграммах хорошо показаны взаимодействия преподавателя и студента с новой системой, продемонстрированы возможности, которые могут быть использованы при работе с данной системой.



Рисунок 2.3 - Схема системы проверки заданий

На рисунке 2.3 видно, что наша система разделяется на 3 подсистемы в зависимости от вида проверяемого задания. Далее рассмотрим варианты автоматизации для всех типов заданий: заданий в виде программного кода, документов и серверных приложений.

2.3 Архитектура автоматизированной системы проверки заданий

На рисунке 2.4 приведена диаграмма компонентов системы проверки заданий для дистанционного обучения.

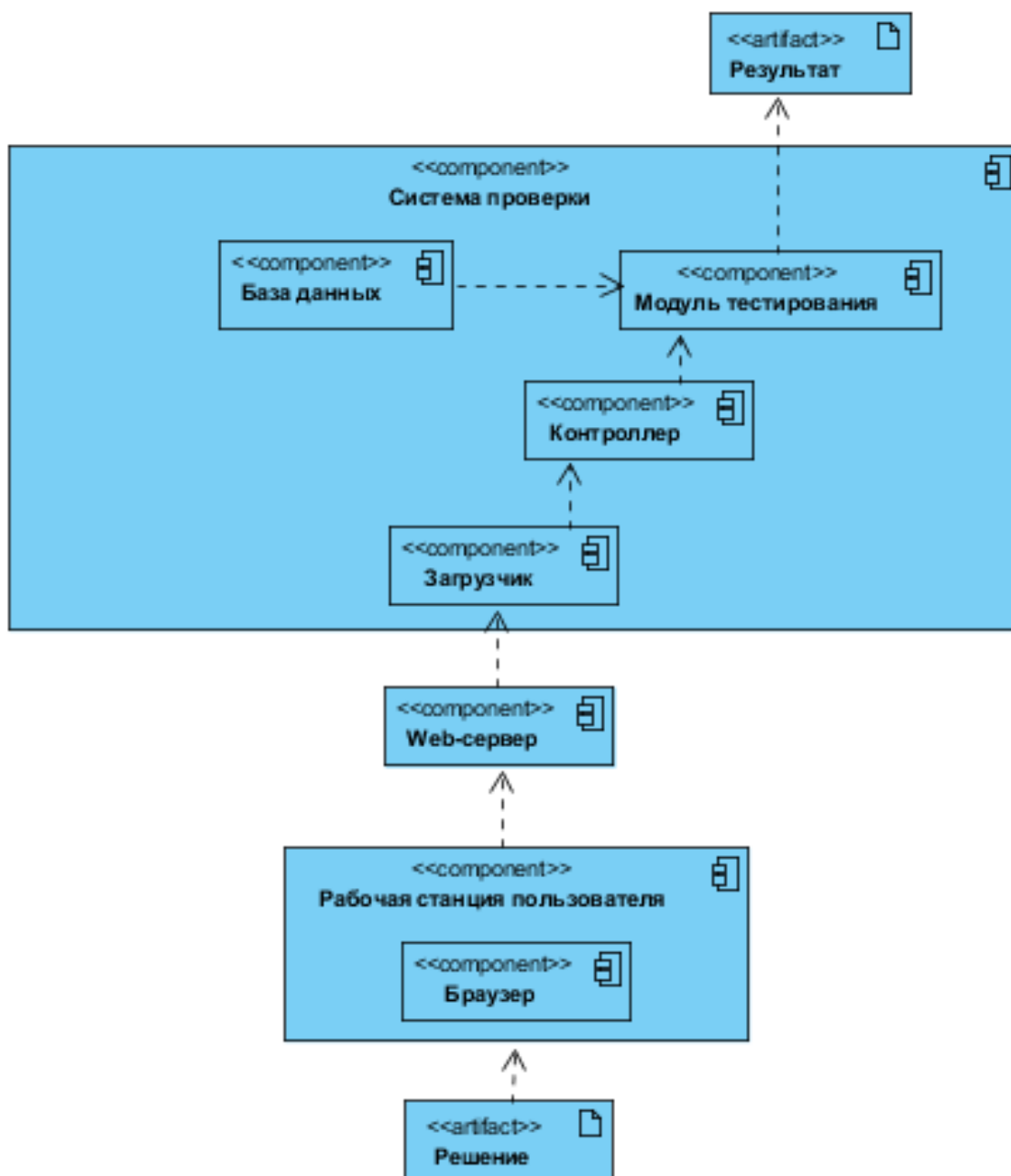


Рисунок 2.4 - Диаграмма компонентов

Пользователь (студент) загружает задание в систему через браузер, выбирая дисциплину и номер задания. Затем web-сервер передает загруженное решение загрузчику.

Данная система проверки состоит из трех основных модулей: загрузчика, контроллера и модуля тестирования.

Контроллер – это один из модулей автоматизированной системы проверки заданий, который определяет к какой дисциплине, заданию принадлежит данное загруженное решение. В зависимости от номера задания определяется и его тип: решение в виде программного кода, документа или серверного приложения. Рассмотрим алгоритм работы контроллера на рисунке 2.5.

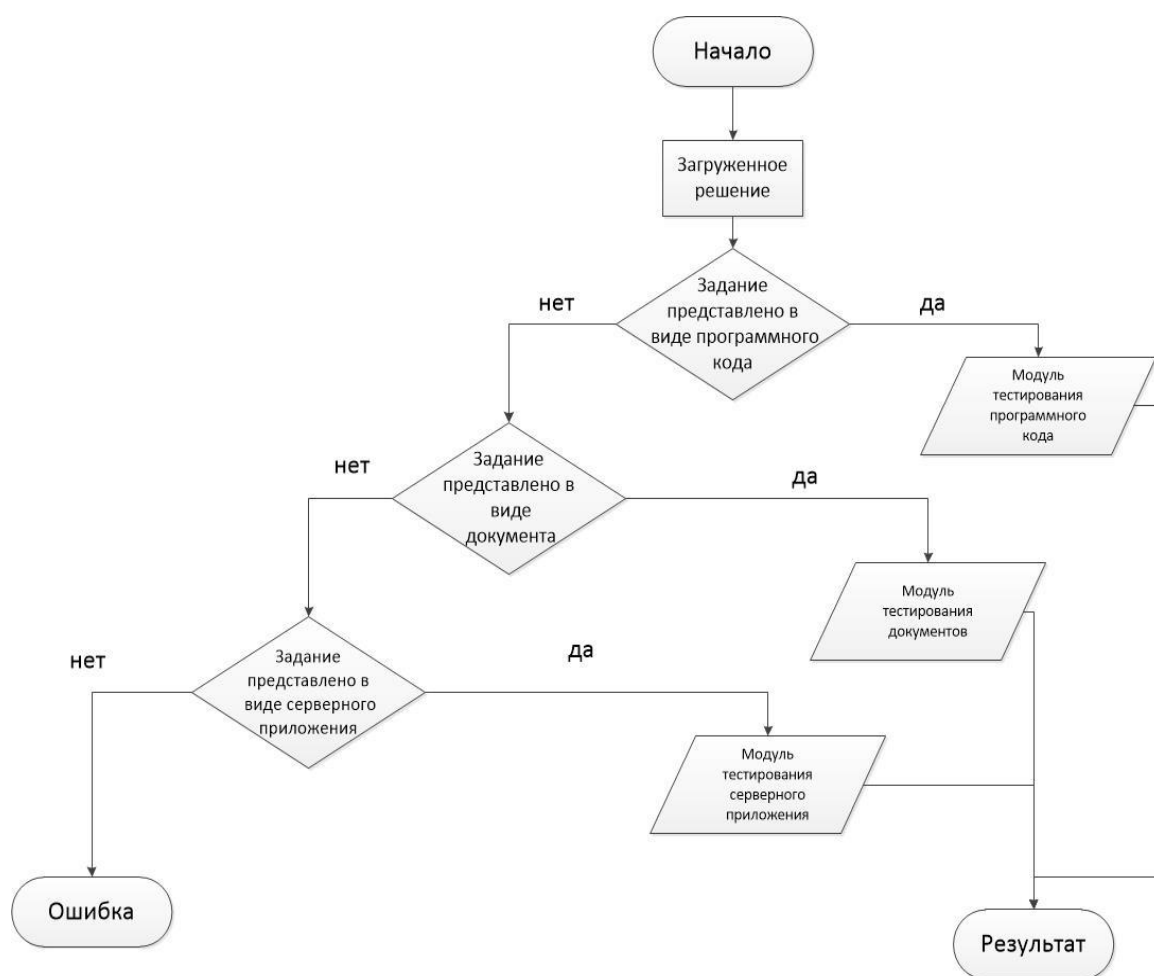


Рисунок 2.5 - Алгоритм работы контроллера

Из данной схемы видно, что контроллер по типу задания определяет в какой модуль тестирования отправить решение. Если решение имеет вид программного кода, то задание переходит в модуль тестирования программного кода, если имеет вид документа, то в модуль тестирования документа, если имеет вид серверного приложения, то в модуль тестирования

серверного приложения. Иначе выводится ошибка, о том, что данное решение не подходит ни под один тип допустимого вида заданий.

Полученные данные контроллер передает модулям тестирования, которые подгружают тест-кейсы, опираясь на номер задания и производит его проверку. Модуль тестирования берет данные из базы данных, которые являются абсолютно верными (ожидаемые значения должны быть предоставлены преподавателем) и сравнивает их с данными, которые студент предоставил в задании. Если актуальный результат (полученный в решении студента) равен ожидаемому, то студент получает в результате «Зачтено», в ином случае «Незачтено».

Модуль тестирования разделен на 3 подмодуля: модуль тестирования программного кода, модуль тестирования документов и модуль тестирования серверных приложений.

2.3.1 Модуль тестирования документов

Обычно задания проверки знаний работы с документами сводятся к следующим видам:

- составлений документа с заданными параметрами (шрифт, отступы и т.д.)
- наличие в документе заданных элементов (рисунки, таблицы и т.д.)

На данный момент для того чтобы проверить знания студентов при изучении офисных приложений (чаще всего это Microsoft Office, поэтому его и будем рассматривать далее) преподаватель должен делать это в ручную, а именно открывать каждый документ и проверять шрифт, наличие таблицы и так далее. Для автоматизации данного процесса и разработан модуль тестирования документов, архитектура которого представлена на рисунке 2.6.

Как известно любой документ можно представить в виде XML-документа. Конвертер – это преобразователь документа в XML-файл, работа

которого сводится к стандартным функциям офисных приложений. XML – это расширяемый язык разметки, который используется для текстового выражения информации в стандартном виде [10].

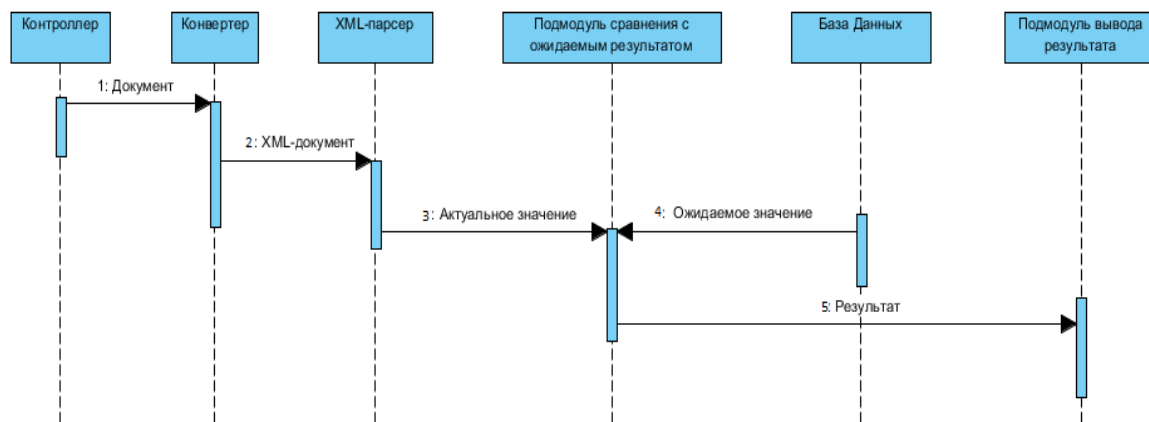


Рисунок 2.6 – Диаграмма последовательности проверки документов

XML-документ состоит из тегов, которые имеют свою семантику и определенный смысл (пример на рисунке 2.7). То есть XML документ содержит все об обычном документе: шрифт, стиль, цвет, рисунки и так далее.

```

<w:document mc:Ignorable="w14 wp14" xmlns:wpc="http://schemas.microsoft.com/office/word/2010/wordprocessingC
<w:body>
  <w:pPr>
    <w:rPr>
      <w:b/>
    </w:rPr>
  </w:pPr>
  <w:r>
    <w:rPr>
      <w:b/>
    </w:rPr>
    <w:t>Модуль тестирования документов</w:t>
  </w:r>
</w:p>
<w:p w:rsidR="00126539" w:rsidRPr="00053895" w:rsidRDefault="00126539">
  <w:pPr>
    <w:rPr>
      <w:rFonts w:ascii="Times New Roman" w:hAnsi="Times New Roman" w:cs="Times New Roman"/>
      <w:sz w:val="36"/>
      <w:szCs w:val="36"/>
    </w:rPr>
  </w:pPr>
  <w:r w:rsidRPr="00053895">
    <w:rPr>
      <w:rFonts w:ascii="Times New Roman" w:hAnsi="Times New Roman" w:cs="Times New Roman"/>
      <w:sz w:val="36"/>
      <w:szCs w:val="36"/>
    </w:rPr>
    <w:t>Модуль тестирования серверных приложений.</w:t>
  </w:r>
</w:p>
<w:p w:rsidR="00126539" w:rsidRPr="00053895" w:rsidRDefault="00126539">
  <w:pPr>
    <w:rPr>
      <w:sz w:val="20"/>
      <w:szCs w:val="20"/>
    </w:rPr>
  </w:pPr>
  <w:r>
    <w:rPr>
      <w:sz w:val="20"/>
      <w:szCs w:val="20"/>
    </w:rPr>
    <w:t>Модуль тестирования серверных приложений.</w:t>
  </w:r>
</w:p>
</w:body>
</w:document>
  
```

Рисунок 2.7 - Пример части XML-документа, который получен из документа формата doc

XML-парсер работает уже с XML-документом. XML-парсер - это часть кода, которая читает XML-документ и анализирует его структуру. Забираем данные из XML с помощью xpath (язык программирования Java поддерживает работу с xml и xpath, при импорте специальных библиотек). Xpath — это язык запросов к элементам xml или xhtml документа. То есть для того, чтобы получить из XML-документа необходимые данные, нужно всего лишь правильно составить xpath, который будет описывать путь к этим данным. Например, в предыдущем XML-документе с помощью xpath = “/pkg:package/pkg:part/pkg:xmlData/w:document/w:body/w:p/w:r/w:t”, получим в результате «Модуль тестирования документов» и «Модуль тестирования документов», так как мы в xpath прописали путь к этим элементам.

С помощью вышеописанного пути мы можем выполнять следующие проверки:

1. Проверка наличия фрагментов текста.
2. Проверка соответствия параметров текста (стиль, размер шрифта, отступы и т.д.).
3. Проверка правильности составления таблиц.
4. Наличие рисунков.

После того как мы получили из документа нужное актуальное значение, сравниваем его с ожидаемым результатом, который хранится в базе данных. Таким образом, если значения будут равны, то значит документ составлен верно, иначе выводится ошибка.

2.3.2 Модуль тестирования серверных приложений

Под серверными приложениями мы будем понимать продукты, к коду которых либо нет доступа, либо проверка кода – не является оптимальным решением. Это сайты и различные приложения. Допустим, заданием является написание сайта. Если нам нужно проверить наличие таблицы или формы регистрации, то очень трудозатратно будет искать это в коде всех html страниц.

Поэтому метод автоматизированной проверки обычных программ, в данном случае будет малоэффективен.

Рассмотрим подходы к тестированию приложений через графический пользовательский интерфейс.

1. Утилиты записи и воспроизведения. Такой инструмент записывает и сохраняет ручные действия тестировщика для дальнейшего воспроизведения. Данный подход нам не подходит, так как оболочка сайта у каждого студента будет разная, и нельзя будет предугадать расположение кнопок, ссылок и т.д.

2. Написание сценария – это программа для каждого конкретного тестового случая, которая написана на языках специально разработанных для автоматизации тестирования. В данном подходе также остается проблема в непредсказуемости оболочки приложения.

3. Тестирование по ключевым словам. В данном случае тест-кейс – это не программный код, а описание действий с параметрами.

Третий подход является самым оптимальным для проверки заданий в виде серверных приложений. Инструментов для проверки приложений существует огромное множество. В этой работе был выбран Selenium WebDriver [19].

Рассмотрим достоинства данного инструмента для автоматизации серверных приложений:

- эмуляция пользовательского взаимодействия с браузером;
- много поддерживаемых платформ, браузеров, языков программирования;
- многопоточность;
- высокая скорость выполнения;
- гибкость использования, с возможностью расширения.

Для работы с Webdriver необходимо иметь три основных программных компонента:

1. Браузер, работа которого будет автоматизирована.

2. Для управления браузером необходим driver браузера. Driver на самом деле выполняет функции веб-сервера, он запускает браузер и отправляет ему команды для прохождения теста, а затем его закрывает. Каждый браузер имеет свой driver. Происходит это из-за того что у каждого браузера своя реализация команд.
3. Тест, который имеет определенный набор команд на выбранном языке программирования для драйвера браузера.

Рассмотрим последовательность действий с использованием Selenium Web-driver на рисунке 2.8.

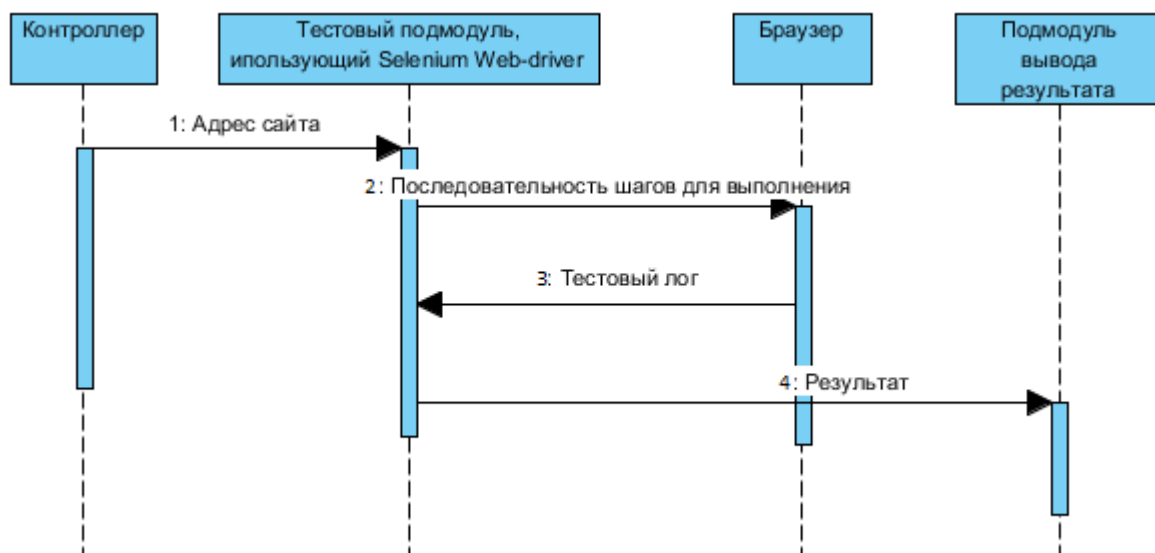


Рисунок 2.8 - Диаграмма последовательности модуля тестирования

На вход модулю подается адрес сайта, каждый тест начинается с указания адреса приложения и используемого браузера.

Тестовый подмодуль будет включать в себя последовательность шагов тест-кейса, написанных на языке Java, с использованием Selenium Web-driver. Selenium Web-driver – это набор инструментов для автоматизации тестирования веб-приложений.


```
driver.get("https://mail.ru/");
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
driver.findElement(By.id("mailbox_login")).sendKeys("petkina.9595@mail.ru");
Thread.sleep(1000);
driver.findElement(By.id("mailbox_password")).sendKeys("password");
Thread.sleep(1000);
driver.findElement(By.id("mailbox_auth_button")).click();
Thread.sleep(2000);
```

Рисунок 2.9 - Пример тест-кейса для серверного приложения

Рассмотрим пример на рисунке 2.9. Этот фрагмент кода предназначен для того, чтобы проверить работает ли форма входа на сайт или нет. То есть драйвер введет логин и пароль, а затем нажмет кнопку войти. Тест-кейс для проверки серверных приложений представляет собой программу, которая будет выполнять последовательные действия.

Пройден тест успешно или нет, мы можем увидеть из лога результатов теста.

Так как каждый сайт может быть написан по разному, и в нем могут использоваться разные теги и тому подобное, то при составлении задания лучше ввести жесткие требования по его созданию, для облегчения его проверки. Например в нашем случае, в задании нужно указать, что окно ввода логина должно называться «mailbox_login», а пароля - mailbox_password. Для того чтобы данная проверка на рисунке 2.8 смогла проверить все сайты, сделанные студентами.

2.3.3 Модуль тестирования программного кода

Когда контроллер передает программный код модулю тестирования программного кода, то поступивший код выполняется с помощью компилятора с входными значениями, которые лежат в базе данных для этого задания (рисунок 2.10) [2].

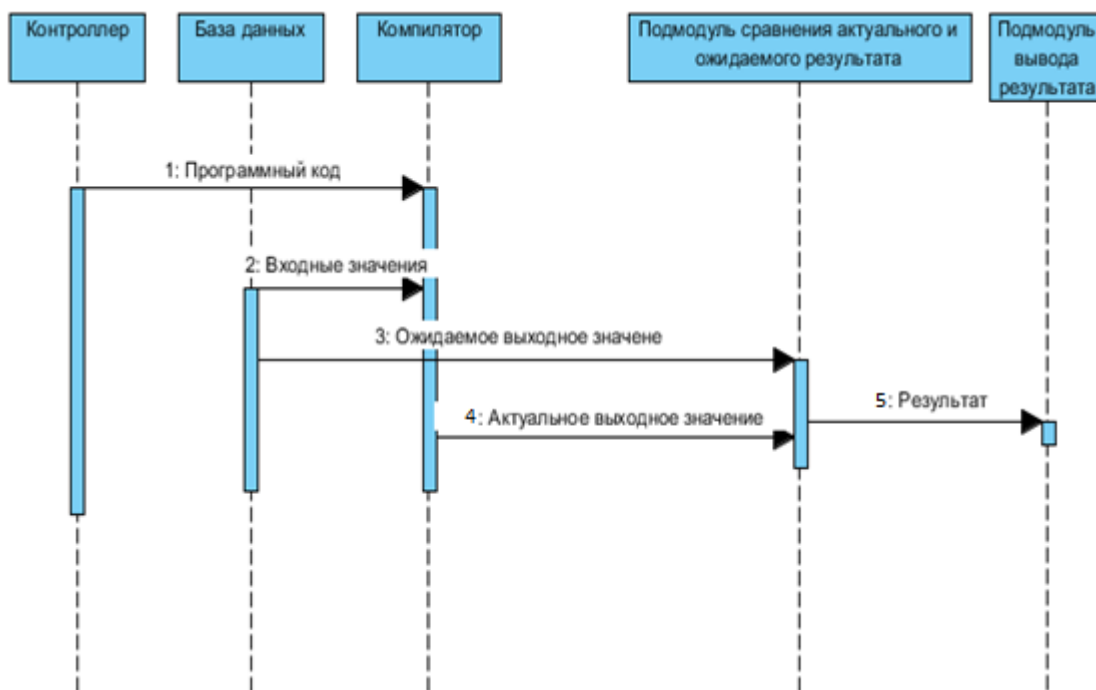


Рисунок 2.10 - Диаграмма последовательности модуля тестирования программного кода

База задач составляется преподавателем. Задача представляет собой совокупность условий и тест-кейсов (проверочных данных). Проверочные данные в простейшем случае представляют собой набор тестов. Каждое решение, отправляемое пользователем, сохраняется в базе решений. То есть для каждого задания есть база с входными и выходными ожидаемыми результатами. Задание проверяется для определенных входных значений, сравнивая с ожидаемым результатом. Преподаватель имеет возможность изучения базы решений. Содержимое стандартного потока вывода, получаемое в результате проверки решения на тесте, сравнивается с правильным ответом [23].

Подмодуль сравнения актуального и ожидаемого результата сравнивает:

1. Актуальный результат – это результат, который получен при компиляции кода, с входными значениями записанными в базе данных.
2. Ожидаемый результат – это результат, записанный в базе данных с входным значением для получения актуального результата.

Система тестирования должна поддерживать компиляторы всех языков, на которых могут быть написаны решения (возможные языки программирования для того или иного задания определяются преподавателем). В данный момент поддерживается язык Java. Система тестирования запускает нужный компилятор, в зависимости от выбранного студентом задания.

Реализация модуля тестирования программного кода рассмотрена в следующей главе.

Вывод ко второй главе

Во второй главе проведено моделирование автоматизированной системы проверки заданий для дистанционного обучения. Основываясь на выявленных недостатках существующего процесса, были сформированы требования для новой системы. Построена диаграмма компонентов, которая является основным этапом в реализации системы и явным образом показывает архитектуру системы. Опираясь на эти компоненты, будет реализоваться проектируемая система.

Система была разделена на три подсистемы – это проверка заданий, в виде программного кода, в виде документов и в виде серверных приложений.

3.1 Концептуальное и логическое моделирование данных

Концептуальная модель – это систематизированное содержательное описание моделируемой системы на неформальном языке. Концептуальная модель разрабатывается для того, чтобы на основе неформализованного описания осуществлялась разработка более строгого и подробного формализованного описания [13]. На рисунке 3.1 изображена концептуальная модель, в которой присутствуют сущности с атрибутами, а также связи с отношениями между сущностями.

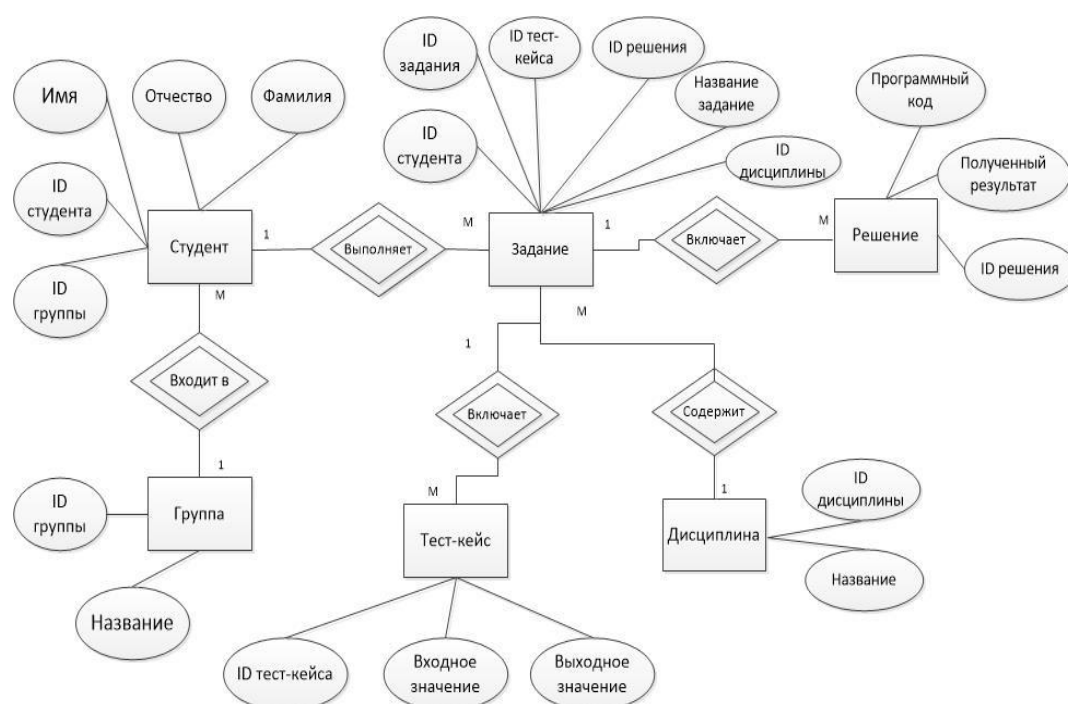


Рисунок 3.1 - Концептуальная модель данных

Сущность «Студент» имеет атрибуты «ID студента», «ID группы», «Имя», «Отчество», «Фамилия» связан с сущностью «Группа» при помощи идентифицирующей связи «Входит в», имеющей мощность М:1. И с сущностью «Задание» при помощи связи «Выполняет», имеющей мощность 1:М.

Сущность «Задание» имеет атрибуты «ID задания», «ID решения», «ID студента», «ID дисциплины», «ID тест-кейса», «Название задание» связан с сущностью «Решение» при помощи идентифицирующей связи «Включает», имеющей мощность 1:М. Также связан с сущностью «Тест-кейс» при

помощи идентифицирующей связи «Включает», имеющей мощность 1:M. И с сущностью «Дисциплина», имеющей мощность M:1.

Сущность «Решение» имеет атрибуты «ID решения», «Программный код», «Полученный результат». Сущность «Тест-кейс» имеет атрибуты «ID тест-кейса», «Входное значение», «Выходное значение» связан с сущностью «Задание» и имеет мощность M:1.

С помощью данной модели данных были выделены основные сущности и связи между ними. Для построения логической модели необходимо выполнить нормализацию данных, которые выявлены в концептуальной модели. Нормализация – это процесс преобразования базы данных к виду, отвечающему нормальным формам.

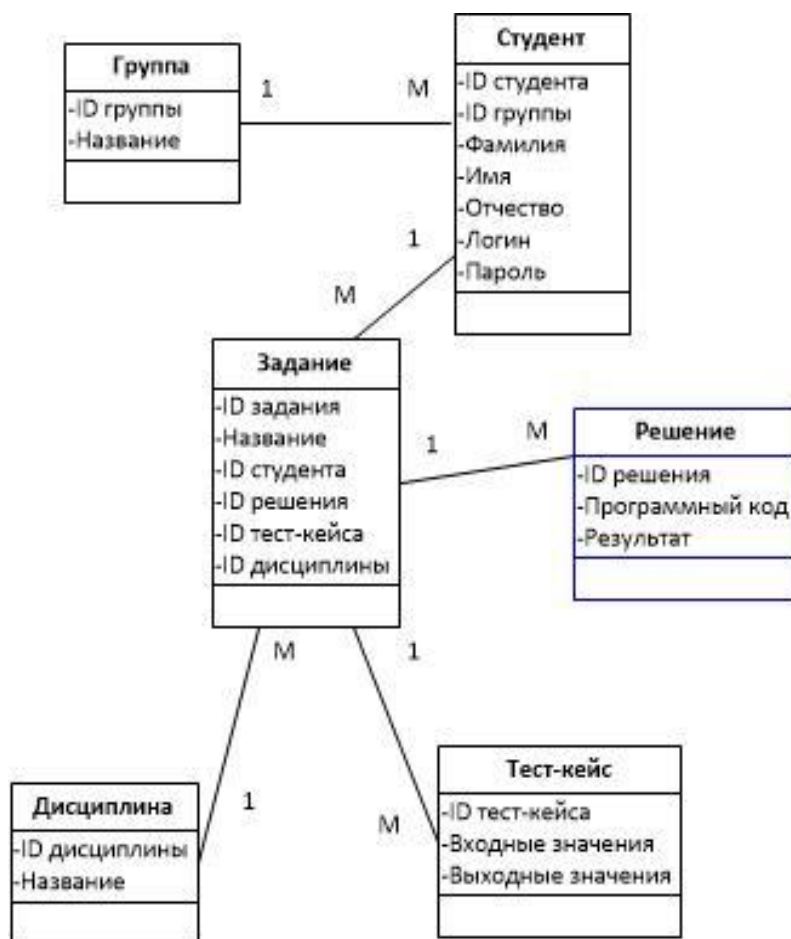


Рисунок 3.2 – Логическая модель данных

На рисунке 3.2 представлена логическая модель данных системы проверки заданий, выполненных в виде программного кода. В данной модели

выделены 6 сущностей: «Студент», «Решение», «Дисциплина», «Тест-кейс», «Группа», «Задание».

3.2 Выбор средств реализации автоматизированной системы

Выбор системы управления базами данных (СУБД) – это один из главных вопросов, которые возникают в процессе разработки системы.

Выбор СУБД основывался на следующих критериях:

1. Опыт работы с СУБД.
2. Кроссплатформенность.
3. Бесплатность.
4. Распространенность СУБД.
5. Поддержка реляционных баз данных.
6. Использование многомерных массивов.
7. Хранение большого количества данных.

Таблица 3.1 - Сравнительный анализ СУБД

Характеристики	PostgreSQL	MySQL	FireBird
Опыт работы с СУБД	+	+	+
Кроссплатформенность	+	+	+
Бесплатность	+	+	+
Распространенность СУБД.	+	+	-
Поддержка реляционных баз данных	+	+	+
Использование многомерных массивов	+	-	-
Хранение большого количества данных.	+	-	-
Итого:	7 из 7	5 из 7	4 из 7

Из сравнительной таблицы можно сделать вывод, что PostgreSQL, лучше удовлетворяет нашим требованиям, поэтому выбираем ее в качестве СУБД.

3.3 Физическая модель базы данных

Физическое проектирование базы данных – это описание способа физической реализации логического проекта базы данных.

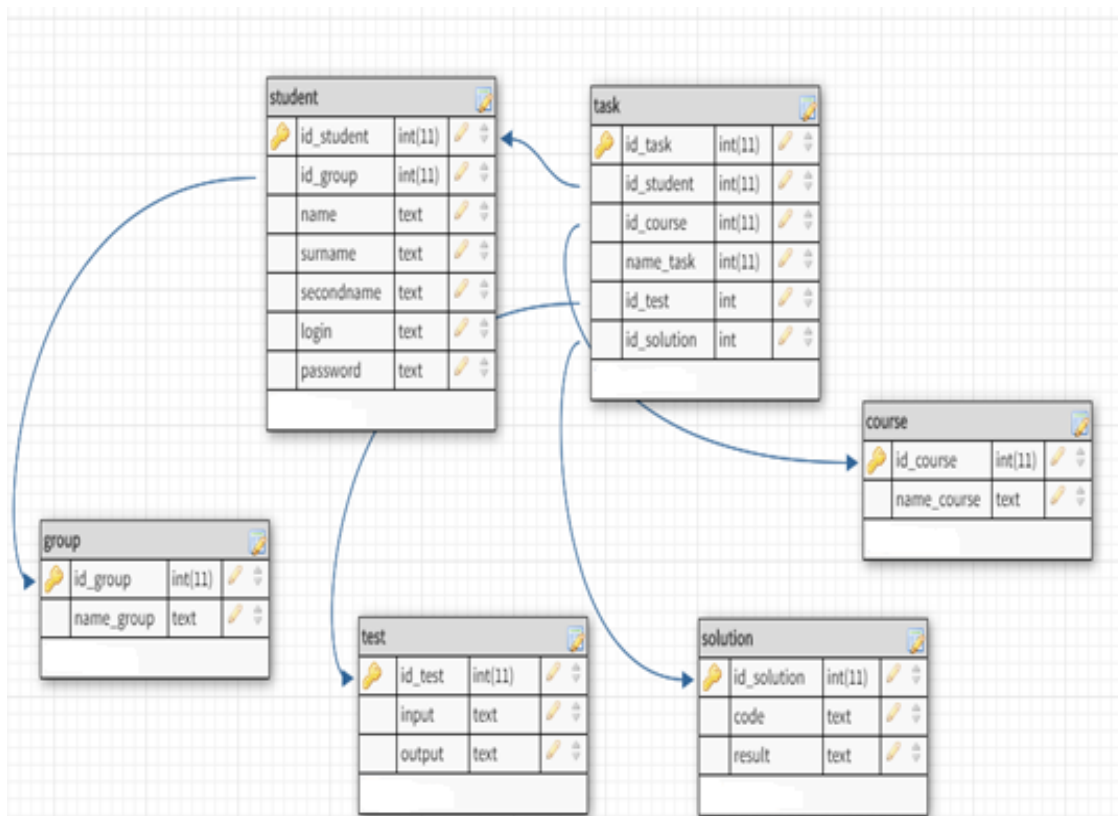


Рисунок 3.3 - Физическая модель базы данных

Все необходимое для того, чтобы создать базу в конкретной СУБД находится в физической модели базы данных, которая представлена на рисунке 3.3, а именно название столбцов, таблиц, формат данных, первичные и внешние ключи. Данная физическая модель данных описывает реализацию объектов логической модели на уровне объектов базы данных PostgreSQL.

Для реализации была выбрана трехзвенная «клиент-серверная» архитектура. В разработке системы необходимо использование сервера базы данных, сервера приложений и клиентского ПО. Преподаватель и студент осуществляют работу с системой через браузер. Все решения, которые отправляют студенты на проверку, сохраняются в базе данных. Также в нашей системе присутствует дополнительный узел – это проверка заданий. Студент

отправляет решение через браузер на сервер, для получения результата. На сервере формируется запрос к базе данных для получения списка тест-кейсов для проверяемой задачи. Далее получаем ответ из базы данных, после чего запускает процесс тестирования (проверки) задания. В итоге генерируется результат, который записывается в базу данных. Далее студент может увидеть результат проверки задания.

3.4 Формальное описание процесса проверки заданий в виде программного кода

На рисунке 3.3 представлена модель базы данных, которую мы и будем рассматривать далее. Для того чтобы получить из базы данных список студентов с их результатами по определенной дисциплине можно воспользоваться следующим запросом (рисунок 3.4):

```
select course.name_course, task.name_task, student.surname, student.name,
student.secondname, group.name_group, solution.result
from task
inner join course on course.id_course = task.id_course
inner join student on student.id_student = task.id_student
inner join student on solution.id_solution = task.id_solution
where name_course = 'Программирование'
```

Рисунок 3.4 – SQL-запрос для получения данных списка студентов с их результатами по определенной дисциплине

Результат запроса представлен на рисунке 3.5.

name_corse	name_task	surname	name	secondname	name_group	result
Программирование	Нахождение объема куба	Иванов	Иван	Иванович	ПМИ6-1301	Зачтено
Программирование	Нахождение квадратного корня	Петров	Петр	Петрович	ПМИ6-1301	Не зачтено
Программирование	Нахождение квадратного корня	Лобов	Денис	Николаевич	МО6-1301	Зачтено

Рисунок 3.5 - Пример результата запроса

Полученный результат в общем случае может быть описан кортежем:

$$UD = \langle Course, NameTask, Res, Surname, Name, Name2, Group \rangle, \quad (3.1)$$

где *Course* – название дисциплины;

NameTask –название задания;

Res – результат;

Surname – фамилия студента;

Name - имя студента;

Name2 – отчество студента;

Group – группа студента.

С помощью операции реляционной алгебры тета-соединения (\bowtie -join) объединяем таблицы с общими атрибутами получаем следующее:

$$Z = T \bowtie_{T.id_{course}=C.id_{course}} C \bowtie_{T.id_{student}=ST.id_{student}} ST \bowtie_{T.id_{solution}=S.id_{solution}} S \quad (3.2)$$

То есть Z – это объединение отношений T (таблица task), C (таблица course), ST (student) и S (solution), из которых выбираются только те кортежи, которые удовлетворяют следующим условиям $course.id_course = task.id_course$, $student.id_student = task.id_student$, $solution.id_solution = task.id_solution$.

Используя операцию проекции π , построим вертикальное подмножество отношений, т.е. подмножества кортежей, получаемых выбором одних и исключением других атрибутов [12]. Отношение Y возвращает таблицу, в которой остаются следующие атрибуты: Course, NameTask, Res, Surname, Name, Name2, Group.

$$Y = \pi_{Course, NameTask, Res, Surname, Name, Name2, Group}(Z). \quad (3.3)$$


Далее воспользуемся операцией выборки. Выборка σ – это построение подмножества кортежей, обладающих определенными заданными свойствами. Построим отношение X :

$$X = \sigma_{id_{course}=\text{Программирование}}(Y). \quad (3.4)$$

Таким образом, отношение X определяет результирующее отношение, содержащее только те кортежи (строки) отношения Y , которые удовлетворяют заданному условию (предикату), а именно $id_course=\text{"Программирование"}$.

3.5 Описание основного принципа работы автоматизированной системы

Когда студент заходит в автоматизированную систему для загрузки и проверки задания, то перед ним открывается главное меню, которое представлено на рисунке 3.6.



Главное меню

Выберите задачу:

a + b

JAVA код (должен быть класс Main с функцией main):

Вставьте ваш код сюда

Рисунок 3.6 - Главная страница

На главной странице пользователю предоставляется выбор задачи, которую он будет загружать на проверку. При нажатии на стрелку будет виден полный список задач.

После того как студент выбрал задачу, он загружает свой программный код в поле, которое называется «JAVA код». Также обязательным условием при загрузке задания является обязательное наличие класса Main с функцией main.

Допустим, перед студентом стоит задача, написать программный код на языке Java, который будет находить корни квадратного уравнения (рисунок 3.7). Он вставляет код и нажимает кнопку «Отправить на проверку».

Тут нужно отметить, что система проверки будет ждать на выходе загруженного решения тот результат, который хранится в базе данных. Поэтому в задании должно быть четко описано, что должно быть в результате работы программы.

```

import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        double a, b, c;
        double D;
        //System.out.println("Программа решает квадратное уравнение вида:");
        //System.out.println("ax^2 + bx + c = 0");
        //System.out.println("Введите a, b и c:");
        Scanner in = new Scanner(System.in);
        a = in.nextDouble();
        b = in.nextDouble();
        c = in.nextDouble();
        D = b * b - 4 * a * c;
        if (D > 0) {
            double x1, x2;
            x1 = (-b - Math.sqrt(D)) / (2 * a);
            x2 = (-b + Math.sqrt(D)) / (2 * a);
            System.out.println(x1);
            System.out.println(x2);
        }
        else if (D == 0) {
            double x;
            x = -b / (2 * a);
            System.out.println(x);
        }
        else {
            System.out.println("Уравнение не имеет действительных корней!");
        }
    }
}

```

Рисунок 3.7 – Пример программного кода задачи

После запуска мы удаляем предыдущий скомпилированный класс Main, компилируем новый и запускаем его (рисунок 3.8).

```

// Удаление предыдущего скомпилированного класса
File oldFile = new File("Main.class");
oldFile.delete();

// Компиляция кода
Process process1 = new ProcessBuilder("javac" , "Main.java").start();
process1.waitFor();

// Запуск кода
ProcessBuilder processBuilder = new ProcessBuilder("java", "-cp", ".", "Main");

// Редирект ввод/вывод в файлы
File input = new File("input");
input.createNewFile();
File output = new File("output");
output.createNewFile();

PrintWriter writer = new PrintWriter(input);
writer.print(inputS);
writer.close();

```

Рисунок 3.8 - Компиляция программного кода

На рисунке 3.9 показаны тест-кейсы, которые хранятся в базе данных. Тест-кейсы для нашей задачи нахождения корней имеют «task_id integer»=3.

	id [PK] serial	task_id integer	input text	output text
1	1	1	2 4	6
2	2	1	0 0	0
3	3	3	2 4 0	-2.0 0.0
4	4	3	1 6 9	-3.0
5	5	3	1 -8 12	2.0 6.0
6	6	3	1 -6 9	3.0
7	7	3	3 7 9	Уравнение не имеет действительных корней!

Рисунок 3.9 - Таблица с тест-кейсами

После прохождения тест-кейса система может выдать два результата: либо все текст-кейсы прошли успешно, то есть задание выполнено правильно, либо возникает ошибка при определенных входных значениях (рисунок 3.10 и 3.11).

Результаты

Все тесты пройдены. Задание зачтено.

Рисунок 3.10 – Результат, когда тест-кейсы прошли успешно

Если же при каких либо тест-кейсах актуальный результат не равен ожидаемому, то выводится сообщение, о том, что тест не пройден, а также выводятся входные значения, в которых возникла ошибка (рисунок 3.10).

Тест не пройден на следующих входных данных:

2 4

Рисунок 3.11 - страница получения отрицательного результата

Добавление и удаление тест-кейсов осуществляется через добавление входных и выходных параметров в базу данных.

Вывод к третьей главе

В третьей главе описано проектирование и разработка автоматизированной системы проверки заданий в виде программного кода, для чего и были выбраны средства реализации. Были рассмотрены концептуальная и логическая модель системы, на основе которых построена физическая модель данных. Реализованная система получила весь необходимый функционал.

ЗАКЛЮЧЕНИЕ

Выполненная работа посвящена исследованию процесса проверки заданий и проблемам контроля заданий в системах дистанционного обучения, проверки практических заданий. Цель работы определила ее основное направление – автоматизация проверки заданий при использовании дистанционных технологий. В ходе выполнения работы были решены следующие теоретические задачи:

1. Проведен анализ процесса проверки заданий при дистанционном обучении, в ходе которого были выявлены недостатки существующего процесса и определены пути решения выявленных проблем.

2. Изучена информация по реализации дистанционного обучения в образовательных учреждениях, и определен спектр взаимодействия обучающегося и преподавателя.

3. Рассмотрены этапы обучения, существующий процесс отправки заданий студентом и проверки заданий преподавателем.

В процессе выполнения проектных задач бакалаврской работы:

1. Построена модель новой технологии, обладающая новыми требованиями и особенностями.

2. Разработана автоматизированная система проверки заданий в виде программного кода.

В результате работы была создана автоматизированная система проверки заданий в виде программного кода для дистанционного обучения, позволяющая автоматически проверять такого типа задания.

Учебники и учебные пособия

1. Вигерс, К. Разработка требований к программному обеспечению // К. Вигерс, Д. Битти. - СПб:ВНУ, 2014.-736с.
2. Верещагин А.Г. Автоматизация тестирования программ как средство повышения эффективности учебного процесса. / А.Г. Верещагин //Известия МГИУ – 2012 - № 3 (27), с. 28–41.
3. Виштак, О.В. Направления программной реализации электронных образовательных ресурсов. Т. 2. / О.В. Виштак - М: Научные труды SWorld, 2013. - 540 с.
4. Гвоздева, В.А. Лаврентьева И. Ю. Основы построения автоматизированных информационных систем. М.: ФОРУМ, 2014. - 320с.
5. Гриневич, Е.А. Методика дистанционного изучения информатики студентами экономических специальностей/ Е. А. Гриневич //Информатизация образования. - 2011. - №1. - С. 36-44.
6. Исаев, Г. Н. Проектирование информационных систем - М.: Омега-Л, 2012. - 432 с.
7. Исаченко, А.Н. Модели данных и системы управления базами данных / А.Н. Исаченко, СП. Бондаренко. Минск: БГУ, 2007. 220 с.
8. Кравченко, Г. В. Работа в системе Moodle: руководство пользователя: учебное пособие. — Барнаул, 2012. - 116 с.
9. МакГрат, М. Программирование на Java для начинающих / Майк Мак - Грат; [пер. с англ. М.А. Райтмана]. – Москва: Издательство 8. «Э», 2016. – 192с.
10. Одиночкина, С.В. Основы технологий XML - СПб: НИУ ИТМО, 2013. – 56 с.
11. Саак, А.Э. Информационные технологии управления : учеб.по спец. «Гос. и муницип. управление» / А. Э. Саак, Е. В. Пахомов, В. Н. Тюшняков. – 2-е изд. ; гриф УМО. – Санкт-Петербург : Питер, 2012. – 318с.

12. Ульман, Д.Д. Введение в системы баз данных : [пер. с англ.] / Джеффри Д. Ульман, Дженнифер Уидом. – М., 2000. – VIII, 374 с.
13. Хорстманн, К. Java SE 8. Базовый курс // К. Хорстманн. – Москва:Вильямс, 2016. – 464 с.
14. Филинов, А.Н. Система автоматического тестирования Ejudge. / А.Н. Филинов //Информатика и образование – 2012 - № 9 - С. 63–64.
Электронные ресурсы
15. Грекул, В. Проектирование информационных систем [Электронный ресурс]: <http://www.intuit.ru/studies/courses/2195/55/info>.
16. Закон Российской Федерации «Об образовании» (от 29 декабря 2012 года №273-ФЗ). [Электронный ресурс]: сайт Министерства образования и науки Российской Федерации: <http://минобрнауки.рф/документы/2974>.
17. Сайт Проект IDEF.RU [Электронный ресурс]: статья / Проект компании «IDEF.RU» – М., 2013. Режим доступа: <http://www.ideal.ru/ideal.php>, свободный (дата обращения 1.03.2017).
18. Сайт Проект IDEF.RU [Электронный ресурс]: <http://www.ideal.ru/ideal.php> (дата обращения 1.06.2017).
19. Нагаева, И.А. Организация электронного тестирования: преимущества и недостатки / И.А. Нагаева // Интернет-журнал «Наукоеведение». – 2013. – №5 (18). [Электронный ресурс]: <http://naukovedenie.ru/PDF/111pvn513.pdf> (дата обращения 02.06.2017).
20. Пишем тесты на Selenium IDE [Электронный ресурс]: <http://automated-testing.info/t/selenium-ide/2455>, (дата обращения: 22.05.2017).
21. Продукты Visio и Project / MicrosoftCorporation [Электронный ресурс]: <https://www.microsoft.com/ru-ru/office/vip/visio.aspx/>, (дата обращения: 14.03.2017).
22. Проектирование информационных систем // II Основы методологии проектирования информационных систем // п. 2.3. Содержание и организация проектирования / [Электронный ресурс]: <http://webmath.exponenta.ru/db/02.html>, (дата обращения: 22.04.2017).

23. Скоробогатов, С.Ю. Автоматизированная система для проведения практических занятий по программированию / С.Ю. Скоробогатов // Инженерный журнал: наука и инновации: Электронное научно-техническое издание. – 2014. – №11 (35). [Электронный ресурс]: <http://engjournal.ru/catalog/pedagogika/hidden/1330.html> (дата обращения 02.06.2017).

24. Система для проведения турниров и индивидуального решения задач по олимпиадному программированию Contester. [Электронный ресурс]: <http://www.contester.ru> (дата обращения 1.06.2017).

25. Ejudge Contest Management System. [Электронный ресурс]: <https://ejudge.ru> (дата обращения 12.03.2017).

Литература на иностранном языке

26. Deitel, H. Java How to Program / H. Deitel, P. Deitel. – 9th edition, Prentice Hall, 2015.

27. Galusha Jill M. Barriers to Learning in Distance Education. / Galusha Jill M., 2016. - 425 p.

28. Kaplan, A. Higher education and the digital revolution: About MOOCs, SPOCs, social media, and the Cokie Monster/ Andreas Kaplan, Michael Haenlein, 2016. - 325 p.

29. Krochmalski, J. IntelliJ IDEA Essentials // J. Krochmalski. – Birmingham: Packt Publishing, 2014.-263p.

30. Maksimenkova, O. Educational tests in “Programming” academic subject development /O. Maksimenkova, V. Podbelskiy // Proceedings of the Spring/Summer Young Researchers’ Colloquium on Software Engineering. – 2010.

```

package com.tasks.tasks;

/*Проверка задачи*/

import java.io.IOException;
import java.util.LinkedHashMap;
import java.util.Мар;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.tasks.readDB.GetTests;
import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;
import com.tasks.writeTODB.*;
import com.google.gson.Gson;

@WebServlet("/private/CheckTask")
public class CheckTask extends HttpServlet {
    private static final long serialVersionUID = 1L;
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
        throws ServletException, IOException {
request.setCharacterEncoding("UTF-8"); //Кодировка данных
        /*Обертка получаемых данных в JSON */
        Map<String, String[]> options = new LinkedHashMap<>();
options = request.getParameterMap();

        String jsonString = new Gson().toJson(options);
        JSONParser parser = new JSONParser();

```

```

JSONObject json = new JSONObject();
    try {
        json = (JSONObject) parser.parse(jsonString);
    } catch (ParseException e) {
        e.printStackTrace();
    }
    response.sendError(400, "Parse error");
}
response.setContentType("text/html");
response.setCharacterEncoding("UTF-8"); //Кодировка данных
if (json.get("task_id") == null || json.get("code") == null) {
    response.sendError(400, "No task specified");
} else {
    try {
        Integer task_id = Integer.parseInt((String) ((JSONArray)
json.get("task_id")).get(0));

        JSONArray tests = new GetTests().get(task_id); // скачать тесты
        String code = (String) ((JSONArray) json.get("code")).get(0);
        for (int i = 0; i < tests.toArray().length; i++) {
            String input = (String) ((JSONObject) tests.get(i)).get("input");
            String output = (String) ((JSONObject) tests.get(i)).get("output");
            boolean res = CodeChecker.checkCode(code, input, output); //
проверить работу программы не тесте
            if (!res) {
                String doc = "<!DOCTYPE html PUBLIC "-//W3C//DTD
HTML 4.01 Transitional//RU\" \"http://www.w3.org/TR/html4/loose.dtd\">\n"
+
                "<html lang=\"ru\">\n" +
                "<head>\n" +
                "\t<meta charset=\"UTF-8\">\n" +

```

```

        "\t<title>Меню</title>\n" +
        "\t<meta name=\"viewport\" content=\"width=device-
width, initial-scale=1\">\n" +
        "\t<link rel=\"stylesheet\" href=\"bootstrap-3.3.6-
dist/css/bootstrap.min.css\">\n" +
        "\t<link rel=\"stylesheet\" href=\"css/main.css\">\n" +
        "</head>\n" +
        "<body>\n" +
        "<div class=\"container-fluid\">\n" +
        "\t<div class=\"row header\">\n" +
        "\n" +
        "\t\t<div class=\"col-sm-10\">\n" +
        "\t\t\t<h1>Результаты</h1>\n" +
        "\t\t</div>\n" +
        "\t\t</div>\n" +
        "\t\t<div class=\"col-sm-3\"></div>\n" +
        "\t\t<div class=\"col-sm-6\">\n" +
        "\t\t\t<div class=\"form-group\">\n" +
        "\t\t\t\t<label for=\"task_id\">Тест не пройден на
следующих входных данных:</label>\n" +
        "\t\t\t\t\t<textarea class=\"form-control\" rows=\"50\"
cols=\"80\" name=\"code\">" + input + "</textarea>\n" +
        "\t\t\t\t\t</div>" +
        "\t\t\t\t</div>\n" +
        "\t\t\t</div>\n" +
        "\t\t</div>\n" +
        "</body>\n" +
        "</html>";
response.getWriter().println(doc);
return;

```

```

    }
} catch (ParseException e) {
    e.printStackTrace();
    response.sendError(400, "Parse error");
}
}

String doc = "<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01
Transitional//RU\" \"http://www.w3.org/TR/html4/loose.dtd\">\n" +
    "<html lang=\"ru\">\n" +
    "<head>\n" +
    "\t<meta charset=\"UTF-8\">\n" +
    "\t<title>Меню</title>\n" +
    "\t<meta name=\"viewport\" content=\"width=device-width, initial-
scale=1\">\n" +
    "\t<link rel=\"stylesheet\" href=\"bootstrap-3.3.6-
dist/css/bootstrap.min.css\">\n" +
    "\t<link rel=\"stylesheet\" href=\"css/main.css\">\n" +
    "</head>\n" +
    "<body>\n" +
    "<div class=\"container-fluid\">\n" +
    "\t<div class=\"row header\">\n" +
    "\n" +
    "\t\t<div class=\"col-sm-10\">\n" +
    "\t\t\t<h1>Результаты</h1>\n" +
    "\t\t</div>\n" +
    "\t\t</div>\n" +
    "\t\t<div class=\"col-sm-3\"></div>\n" +
    "\t\t<div class=\"col-sm-6\">\n" +

```

```

        "    <h1 style=\"text-align:center;color:#396E8D\">Все тесты
пройжены. Задание зачтено.</h1>" +
        "    </div>\n" +
        "    <div class=\"col-sm-3\"></div>\n" +
        "</div>\n" +
        "</body>\n" +
        "</html>";
        response.getWriter().println(doc);
    }
}

```

Содержание ConnectionDB.java

```

package com.tasks.connection;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
public class connectionDB {
    public static Connection createConnection() {
        Connection connection = null;
        System.out.println("----- PostgreSQL JDBC Connection Testing -----
----");

        /*
        * Проверка наличия драйвера PostgreSQL
        */
        try {
            Class.forName("org.postgresql.Driver");
        } catch (ClassNotFoundException var3) {
            System.out.println("Where is your PostgreSQL JDBC Driver? Include
in your library path!");

```

```

        var3.printStackTrace();
    }
    System.out.println("PostgreSQL JDBC Driver Registered!");
    /*
     * Проверка подключения к базе данных
     */
    try {
        connection =
DriverManager.getConnection("jdbc:postgresql://localhost:5432/tasks",
"postgres", "postgres");
    } catch (SQLException var2) {
        System.out.println("Connection Failed! Check output console");
        var2.printStackTrace();
    }

    if(connection != null) {
        System.out.println("You made it, take control your database now!");
    } else {
        System.out.println("Failed to make connection!");
    }

    return connection;
}
public static void closeConnection(Connection con)
{
    /*
     * Закрывать соединение с базой
     */
    try {
        con.close();
    } catch (SQLException e) {

```

```

        /*
        * Catch the exception if occurred
        */
        e.printStackTrace();
    }
}

```

Содержание файла GetTests.java

```

package com.tasks.readDB;
import com.tasks.connection.connectionDB;
import org.json.simple.JSONArray;
import org.json.simple.parser.ParseException;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
public class GetTests {
    public static JSONArray get(Integer task_id) throws ParseException {
        Connection db = connectionDB.createConnection(); //Подключение к
базе
        String sql = "SELECT * FROM tests WHERE task_id =" +
task_id.toString(); //SQL для выполнения на базе
        JSONArray res = new JSONArray();
        System.out.println(sql);
        try {
            Statement sqlStat = db.createStatement();
            res = ResultSetConverter.convert(sqlStat.executeQuery(sql));
//Конвертировать результат в JSON
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}

```



```

        connectionDB.closeConnection(db); //Закрывать соединение с базой
    return res;
    }
}

```

Содержание файла GetTasks.java

```

package com.tasks.readDB;
/*
 * Подключение к базе, для получения информации по задачам
 */
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import com.tasks.tasks.CodeChecker;
import org.json.simple.JSONArray;
import org.json.simple.parser.ParseException;
import com.tasks.connection.connectionDB;
public class GetTasks {

    public static JSONArray get() throws ParseException{
        Connection db = connectionDB.createConnection();
//Подключение к базе

        String sql = "SELECT name, id FROM tasks"; //SQL для
выполнения на базе
        JSONArray res = new JSONArray();

        System.out.println(sql);
        try {
            Statement sqlStat = db.createStatement();

```

```

        res =
ResultSetConverter.convert(sqlStat.executeQuery(sql)); //Конвертировать
результат в JSON
        } catch (SQLException e) {
            e.printStackTrace();
        }
        connectionDB.closeConnection(db); //Закрывать соединение с
базой
        return res;
    }
}

```

Содержание файла CodeChecker.java

```

package com.tasks.tasks;
import javax.tools.*;
import java.io.*;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Comparator;
import java.util.Scanner;
/**
 * Created by iharshulhan on 01/06/2017.
 */
public class CodeChecker {
    static public boolean checkCode(String code, String inputS, String outputS)
    {
        try{
            // Запись кода в файл
            File file = new File("Main.java");
            file.createNewFile();
            PrintWriter writer = new PrintWriter(file);

```

```

        writer.print(code);
        writer.close();
    } catch (FileNotFoundException e) {
        e.printStackTrace();
        return false;
    } catch (IOException e) {
        e.printStackTrace();
        return false;
    }
}
try {
    // Удаление предыдущего скомпилированного класса
    File oldFile = new File("Main.class");
    oldFile.delete();
    // Компиляция кода
    Process process1 = new ProcessBuilder("javac" , "Main.java").start();
    process1.waitFor();
    // Запуск кода
    ProcessBuilder processBuilder = new ProcessBuilder("java", "-cp", ".",
"Main");
    // Редирект ввод/вывод в файлы
    File input = new File("input");
    input.createNewFile();
    File output = new File("output");
    output.createNewFile();

    PrintWriter writer = new PrintWriter(input);
    writer.print(inputS);
    writer.close();

    processBuilder.redirectInput(input);

```

```
processBuilder.redirectOutput(output);
Process process2 = processBuilder.start();
process2.waitFor();
// Проверка вывода
Scanner scanner = new Scanner(output);
StringBuilder stringBuilder = new StringBuilder();
while (scanner.hasNext()) stringBuilder.append(scanner.next());
if (stringBuilder.toString().compareTo(outputS) != 0) return false;
} catch (IOException e) {
    e.printStackTrace();
    return false;
} catch (InterruptedException e) {
    e.printStackTrace();
    return false;
}
return true;
}
}
```