|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Приднестровский государственный университет им. Т.Г. Шевченко | | | | | | | | | |
| Инженерно-технический институт | | | | | | | | | |
| Инженерно-технический факультет | | | | | | | | | |
| Кафедра программного обеспечения вычислительной техники  и автоматизированных систем | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  | Допустить к защите | | | |
|  |  |  |  |  |  | зав. кафедрой ПОВТ и АС, | | | |
|  |  |  |  |  |  | к.т.н., доцент | | | |
|  |  |  |  |  |  |  |  | С.Г. Федорченко | |
|  |  |  |  |  |  | « \_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_ 2016 г. | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| **ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА** | | | | | | | | | |
|  | | | | | | | | | |
| **тема: «РАЗРАБОТКА ПРОГРАММНОГО**  **КОМПЛЕКСА ОПРОС СТУДЕНТОВ»** | | | | | | | | | |
|  | | | | | | | | | |
| БАКАЛАВРСКАЯ РАБОТА | | | | | | | | | |
|  | | | | | | | | | |
|  | | | | | | | | | |
| Направление 09.03.04 «Программная инженерия» | | | | | | | | | |
| Профиль: «Разработка программно-информационных систем» | | | | | | | | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | Студент | |  |  |  |  | Федоров | | |
|  | группы ИТ12ДР62ПИ1 | | |  |  |  | Артем Андреевич | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  | Научный руководитель, | | |  |  |  | Левицкий | | |
|  | к.т.н., доцент | | |  |  |  | Егор Андреевич | | |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| Тирасполь, 2016 | | | | | | | | | |

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
| ВВЕДЕНИЕ | | 4 |
| 1 ИССЛЕДОВАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ | | 5 |
|  | 1.1 Описание поставленной задачи, ее обоснование | 5 |
|  | 1.2 Обоснование актуальности исследуемой задачи | 5 |
|  | 1.3 Современное состояние исследуемой задачи | 6 |
|  | 1.4 Обзор методов решения подобных задач | 6 |
|  | 1.5 Постановка задачи, системные требования, требования к входным данным и выходным формам | 6 |
|  | 1.6 Выводы | 9 |
| 2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ И АРХИТЕКТУРЫ  ПРОГРАММНОГО ПРОДУКТА | | 9 |
|  | 2.1 Выбор методов и средств для реализации | 9 |
|  | 2.2 Описание применяемых алгоритмов | 10 |
|  | 2.3 Структура, архитектура программного продукта | 10 |
|  | 2.4 Описание логической структуры программного продукта | 10 |
|  | 2.5 Функциональная схема, функциональное назначение программного продукта | 11 |
|  | 2.6 Выводы | 11 |
| 3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА | | 12 |
|  | 3.1 Описание реализации | 12 |
|  | 3.2 Описание пользовательского интерфейса | 16 |
|  | 3.3 Полученные результаты и их анализ | 21 |
|  | 3.4 Методы и средства защиты программного продукта | 22 |
|  | 3.5 Тестирование и оценка надежности программного продукта | 22 |
|  | 3.6 Расчет себестоимости от внедрения результатов ВКРБ | 22 |
|  | 3.7 Охрана труда | 22 |
|  | 3.8 Выводы | 22 |
| ЗАКЛЮЧЕНИЕ | | 22 |
| ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ  И ТЕРМИНОВ | | 23 |
| СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ | | 24 |
| ПРИЛОЖЕНИЯ | | 27 |

**ВВЕДЕНИЕ**

Бла бла

**1 ИССЛЕДОВАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ**

**1.1 Описание поставленной задачи, ее обоснование**

Целью данной курсовой работы является создание программного комплекса «Опрос студентов».

Для проектируемой предметной области необходимо разработать программный продукт, обеспечивающий:

1. Формирование проекта:

1.1. Ввод названия проекта, сопутствующей информации.

1.2. Ввод информации об используемом тесте, алгоритме обработки результатов тестирования.

1.3. Подключение используемых тестов.

1.4. Формирование списка тестируемых с указанием сопутствующей информации.

2. Проведение опроса тестируемых в соответствии со сформированным проектом.

3. Обработка результатов опроса и формирование выходных форм.

4. Стыковка результатов опроса с массивом внешних данных.

Автоматизирование проведения тестирования значительно помогает преподавателям. Такой способ тестирования позволяет сэкономить уйму времени у преподавателя при составлении и проведении тестирования, также этот метод не требует использования бумаги и помогает исключить ошибки и неопределенности при проверке теста.

Самым большим преимуществом автоматических тестов является автоматизированный процесс проверки тестов. При составлении вопроса и вариантов ответа преподавателю достаточно один раз ввести и отметить правильные варианты ответов, и в процессе проверки программа автоматически будет искать совпадение с заданным вариантом, что поможет как ускорить процесс проверки проведенных тестов, так и улучшить качество проверки.

Но с другой стороны автоматические тесты могут повлечь за собой появление новых ошибок во время составления теста.

Например, при составлении теста возможна ситуация, когда у некоторых вопросов нет вариантов ответов вообще, нет правильных вариантов ответа, или же все варианты ответы правильные. Для предотвращения этой ситуации необходимо не допускать публикацию теста без исправления всех таких ошибок.

**1.2 Обоснование актуальности исследуемой задачи**

Исследуемая предметная область рассматривается, как альтернатива существующей системе проведения тестов в Приднестровском Государственном Университете.

На данный момент практически все тестирование студентов в ПГУ проводится преподавателями вручную при помощи бумаги и ручки. Данный метод тестирования требует некоторых лишних затрат.

Вначале необходимо напечатать образец теста, потом сделать необходимые его копии. Затем нужно раздать копии теста, после чего студенты с разной скоростью их заполнят и сдадут. После сдачи эти тесты нужно проверить, то есть затратить еще значительное количество ресурсов. Кто-то из студентов может неправильно понять принцип заполнения теста или после проставления ответа поменять свой выбор, это понесет за собой многочисленные правки, которые также затрудняют проверку теста. Также возможна утеря образца пройденного студентом теста.

Чтобы избежать вышесказанных неудобств в совокупности с текущим развитием технологий и тем, что большинство аудиторий в ПГУ оборудованы персональными компьютерами, а у каждого студента имеется смартфон, планшет или ноутбук с доступом в интернет, метод проведения тестирования, описанный выше, является устаревшим и требует доработки в плане автоматизации.

Для ПГУ уже была разработана автоматизированная система для проведения тестирования студентов, но она подразумевает только один вид тестирования.

(КАКОЙ? КАКИЕ НЕУДОБСТВА)

Таким образом, рассматриваемая в данной работе предметная область в настоящий момент является актуальной.

**1.3 Современное состояние исследуемой задачи**

История тестирования насчитывает по крайней мере, несколько тысячелетий. Некоторые источники указывают на факты, которые свидетельствуют о еще более древнем применении тестов. Уже в середине третьего тысячелетия до н. э. в Древнем Вавилоне проводились испытания выпускников в школах писцов на знание арифметических действий, умение измерять поля, распределять рационы, делить имущество, владение искусством пения и игры на музыкальных инструментах.

Различные конкурсы и экзамены устраивались и в средневековом Вьетнамском государстве. В период с 1370 по 1372 год там была проведена переаттестация всех военных и гражданских чиновников, что позволило организовать проверку государственного аппарата по всей стране. Конкурсные испытания проводились по этапам и турам, а присвоение высших степеней на экзаменах сопровождалось большими почестями.

Первым этапом применения тестов в мировой практике можно считать период с восьмидесятых годов девятнадцатого века до двадцатых годов двадцатого века. Этот период характеризуется зарождением и становлением тестирования как такового.

Родоначальником тестологии по праву считается Дж. Кэттелл, так как именно он увидел в тестах средство измерения свойств человеческой психики. Ученый считал, что применение тестов к большому числу индивидов позволит открыть закономерности психических процессов. Исследователь полагал, что научная и практическая ценность тестов возрастет, если условия их проведения будут однообразными. Так было впервые сказано о необходимости стандартизации тестов, чтобы можно было сравнить результаты, полученные разными исследователями на разных испытуемых.

Дж. Кэттелл предложил в качестве образца 50 тестов, включавших измерения чувствительности, остроты зрения и слуха, времени реакции, времени, затрачиваемого на называние цветов и количества звуков, воспроизводимых после однократного прослушивания.

В 1904 году французский министр народного образования создал комиссию для изучения 341 проблемы отставания школьников в учебе и создания методики, с помощью которой можно было бы отделить детей, способных к учению, от детей, страдающих врожденными дефектами и не способных учиться в нормальной школе. Для этой комиссии психологи А. Бине и Т. Симон провели серию экспериментов по изучению внимания, памяти, мышления у детей разного возраста и разработали первую шкалу для вычисления коэффициента уровня интеллектуального развития.

В двадцатые – шестидесятые годы XX века тесты разрабатываются в рамках биологизаторской концепции развития личности. В это время начинают разделяться психологические и педагогические (дидактические) тесты. Дидактические тесты разрабатываются по всем школьным дисциплинам. Большое внимание уделяется обработке результатов тестирования. Намечается переход от единичных тестов к тестовым системам.

С начала шестидесятых по конец семидесятых годов тесты разрабатываются в рамках биосоциологизаторской концепции развития личности. Благодаря внедрению в образовательную практику программированного обучения начинает осуществляться машинный и безмашинный контроль знаний с помощью заданий в тестовой форме.

Начиная восьмидесятых годов и по настоящее время начинается широкое использование компьютеров и средств программирования для обучения и контроля. Во многих странах созданы службу, занимающиеся разработкой тестов, организацией массового тестирования и сбором информации для мониторинга качества образования.

Таким образом, история возникновения и развития тестирования показывает, что этот метод контроля стал важным методом оценки, с помощью которого можно достаточно объективно, надежно измерить, обработать, интерпретировать результаты учебного процесса, диагностировать выраженность у индивида определенных психологических качеств, свойств и состояний, а также измерить уровень их развития.

**1.4 Обзор методов решения подобных задач**

На сегодняшний день существует множество программных продуктов, связанных с автоматическим тестированием. Некоторые программные продукты разворачиваются на сайтах, но существуют и программы, которые необходимо устанавливать на компьютер.

Примером веб-решения для автоматизации проведения тестов является сайт [*http://master-test.net*](http://master-test.net)*.* Данный сайт представляет собой бесплатный образовательный интернет сервис, который позволяет создавать и проходить тесты. Интерфейс добавления вопроса представлен на рисунке 1.1.



Рисунок 1.1 – Добавление вопроса в системе *master-test.net*

При добавлении вопроса имеются такие необходимые функции, как добавление медиа-контента, выбор различных типов вопроса, его вес. После добавления всех вопросов имеется возможность проверить тест, чтобы выявить корректно ли преподаватель отметил правильные ответы.

На рисунке 1.2 представлено окно редактирования выводов результатов прохождения.



Рисунок 1.2 – Редактирование вывода результата в системе master-test.net

При выводе результатов преподаватель может учесть все необходимые опции.

После окончания настройки теста его нужно активировать. При активации можно выбрать одно из трех действий:

– провести тестирование студентов этим тестом;

– опубликовать тест как виджет;

– скачать тест как файл. После этого можно проходить тест без подключения к интернету.

Например, после выбора пункта тестирование студентов появляется окно с выбором времени доступности теста и списка людей, кому он будет доступен. После окончания всех настроек появляется окно подтверждения, представленное на рисунке 1.3.



Рисунок 1.3 – Подтверждение публикации теста в системе *master-test.net*

Результат прохождения теста представлен на рисунке 1.4.



Рисунок 1.4 – Результаты прохождения теста в системе master-test.net

После окончания прохождения теста, тестируемому представляется вся информация по проходимому тесту: количество правильных и неправильных ответов, процентная оценка тесту. Так же можно посмотреть ответ по каждому из вопросов. Данная информация доступна и преподавателю в специально отведенной для этого графе.

Подводя итоги, можно отметить, что сервис *master-test.net* обладает хорошей функциональностью, но имеет устаревший дизайн. Для того, чтобы научиться пользоваться данным сервисом, нужно достаточно малое количество времени.

В пример настольной программы можно привести программу «Айрен». Данная программа предоставляет мощный комплекс для создания и прохождения тестов как по сети интернет, так и по локальной сети.

При сетевом тестировании преподаватель видит на своем компьютере подробные сведения об успехах каждого из учащихся. По окончании работы эти данные сохраняются в архиве, где их в дальнейшем можно просматривать и анализировать с помощью встроенных в программу средств.

Кроме того, предусмотрено создание тестов в виде автономных исполняемых файлов (пример), которые можно раздать учащимся для прохождения тестирования без использования сети и без сохранения результатов. Такой режим ориентирован прежде всего на тесты, предназначенные для самопроверки. Учащемуся, чтобы приступить к тестированию, достаточно запустить полученный файл на любом компьютере с *Windows*, установка каких-либо программ для этого не требуется.

Интерфейс добавления вопроса представлен на рисунке 1.5.



Рисунок 1.5 – добавление вопроса в системе «Айрен»

В целом данная программа перекрывает свой веб-аналог в функциональности.

Например, данная программа позволяет перемешивать варианты ответов, настраивать разрешение исправления выбранного варианта ответа для теста и имеет еще множество полезных мелочей.

При том, что функциональность программы на высоте, слабыми сторонами являются так же является интерфейс программы и необходимость ее установки на машины.

Для обучения пользованием этого продукта придется потратить уже более значительное количество времени, чем в предыдущем случае.

**1.5 Постановка задачи, системные требования, требования к входным**

**данным и выходным формам**

Необходимо разработать программный продукт «Опрос студентов». Программа должна собой представлять веб-приложение, имеющее следующие функции:

1. Возможность регистрации и авторизации пользователей.

2. Разделение прав пользователей на три категории: администратор, редактор тестов и обычный пользователь:

– обычный пользователь при регистрации указывает свою принадлежность к группе и после авторизации может выбирать доступные к прохождению тесты, а также просматривать результаты пройденных тестов;

– редактор тестов создает и публикует для разных групп тесты, а также просматривает результаты прохождения тестов всеми студентами;

– администратор имеет возможность назначать обычных пользователей редакторами тестов, а также добавлять и изменять группы.

3. Возможность создания теста, указав информацию, такую как:

– название и описание теста;

– максимальный балл, который можно получить, пройдя тест;

– время на прохождение;

– вид теста:

– дополнительные настройки:

а) перемешивание вопросов и ответов;

б) показывание правильных ответов после прохождения теста;

в) произвольное перемещение по вопросам (только для классического вида тестов);

г) подсказка, если вариантов ответа больше одного (только для классического вида тестов).

4. При публикации теста редактор указывает список групп для которых публикуется тест, а также дату публикации (можно опубликовать тест сразу или через время) и дату закрытия публикации (по умолчанию тест доступен до того времени, пока студент его не пройдет).

5. Для теста доступно добавление, изменение, удаление вопросов теста и ответов на них. При формировании вопроса необходимо следить, чтобы хотя бы один вариант ответа был верным, так же имеется возможность выбора веса вопроса.

6. Студентам доступны только опубликованные тесты для их группы. После прохождения теста студентов происходит обработка ответов на вопросы теста и формирование результатов по окончании прохождения теста.

Необходимо реализовать два вида тестирования студентов.

Первый вид – классический. Студенту предоставляется вопрос и варианты ответов, после ответа на один вопрос студент переходит к другому. Для этого вида тестов доступны вопросы двух разны типов: варианты ответа и число/вопрос

Второй же вид тестирования представляет собой такую систему, когда предоставляется список всех вопросов и список всех ответов и требуется сопоставить каждому вопросу необходимые варианты ответов. Сложность заключается в том, что для каждого вопроса по сути приходится выбирать из всех вариантов ответа.

Входными данными является информация, которую заносит преподаватель, создающий тесты. Эти данные могут быть как текстовыми, так и медиа-контентом. Название теста не должно превышать 255 символов, для описания теста – количество символов неограниченно. Что касается вопросов и ответов – нужно придерживаться правила: длинный вопрос, короткие ответы, поэтому длина вопроса не ограничена, а вот длина ответа на вопрос ограничена 255 символами. Так же в качестве вопроса можно использовать картинки и красиво оформленный код.

Выходными формами рассматриваемой предметной области являются данные о результатах прохождения студентом теста. В частности, это данные о количестве вопросов, о количестве верных и неверных ответов, успех прохождения теста в процентах.

Так как разрабатываемый программный продукт является веб-приложением, то ограничений операционных систем для пользователей нет. Для успешного использования программы требуется интернет-браузер со включенным *JavaScript.*

Минимальные требования к версиям популярных браузеров:

– Internet Explorer 11;

– EDGE – версия 12 и выше;

– Браузер Firefox не ниже 30 версии;

– Браузер Chrome – 20;

– Safari версии 6.1 и более;

– Opera 17;

– Другие браузеры – желательны одни из последних версий.

Для комфортной работы с программой необходимо минимум 150мегабайтсвободной оперативной памяти.

**1.6 Выводы**

Налил водички в первой главе

**2 ПРОЕКТИРОВАНИЕ СТРУКТУРЫ И АРХИТЕКТУРЫ**

**ПРОГРАММНОГО ПРОДУКТА**

**2.1 Выбор методов и средств для реализации**

Выбор методов и средств для реализации проектируемого программного продукта проходит в несколько этапов.

Очевидно, что для данной предметной области должна использоваться клиент-серверная архитектура, поэтому первый этап - это выбор серверного языка программирования и базы данных, с помощью которых будет реализована система. Выбор стоит в основном между тремя языками программирования: *C#, PHP, Server-side JavaScript.* Было решено остановить свой выбор на языке *PHP*, так как это наиболее популярное решение для веб*-*приложений на сегодняшний день. Основным преимуществом данного выбора стало то, что настройка сервера для данного языка программирования не должна доставить больших неудобств. Наиболее выгодным решением относительно выбора базы данных является *MySQL*. Данное решение отлично подходит для низко- и средненагруженных приложений и великолепно сочетается с *PHP*.

Второй этап – это выбор того, как именно будет использоваться выбранный в первом этапе язык программирования. Будет ли изобретаться что-то с нуля для реализации программного продукта, или же будет взят какой-либо программный каркас. Для *PHP* создано большое количество фреймворков, абсолютной большинство из них базируется на архитектурной модели *MVC*. Примерами таких фреймворков являются *Zend Framework, Symphony, Kohana, Laravel, Yii*. Выбор был остановлен на фреймворке *Laravel* в пользу того, что это самый быстроразвивающийся фреймворк в СНГ и самый развитый в западных странах. Так же *Laravel* отлично подходит для программистов, которые до этого не имели опыта работы с *MVC* фреймворками.

В современных веб-приложениях, на клиентской стороне сегодня повсеместно используется язык *JavaScript.* Поэтому третьим этапом нужно выбрать его место в проектируемом программном продукте. Данный язык программированиям можно использовать разными способами:

1. Только во вспомогательных целях для придания относительной живости веб-страницам. То есть сервер по запросу чаще всего в качестве ответа предоставляет сформированную *html-*страницу.

2. В данном варианте *JavaScript* по сути становится основным языком программирования, а сервер используется как *API.* То есть клиентам поступают данные, которые обрабатываются и выводятся с помощью вычислительных способностей клиента. Такие приложения называются *single page application* (*SPA*), или по-русски одностраничные приложения, потому что в таких приложениях не происходит перезагрузки страницы.

3. Смесь первого и второго пунктов. Приложение имеет несколько значительных обособленных модулей, взаимодействие внутри которых осуществляется с помощью *JavaScript*, то есть с помощью второго пункта,перемещение же между самими модулями побуждает полноценный запрос на сервер с последующей выдачей новой *html*-страницы.

В проектируемой области решено использовать третий вариант использования *JavaScript*, таким образом, приложение будет выглядеть достаточно живо, при этом не будет ограничений в функциональности.

Для оптимизации процесса написания кода на *JavaScript* было принято решение использовать *JS*-фреймворк *VueJS*. Этот фреймворк активно продвигается *PHP*-фреймворком *Laravel*, и довольно успешно. Данный фреймворк находится в числе наиболее активно развивающихся, имеет отличную скорость работы, и хорошо подходит для относительно небольших проектов.

Итак, для проектируемого программного продукта в качестве серверного языка программирования было решено использовать *PHP* версии 5.6 в связке с базой данных *MySQL.* На клиентской стороне будет использоваться JavaScript фреймворк *VueJS* с небольшой поддержкой *JQuery*.

**2.2 Описание применяемых алгоритмов**

Алгоритм прохождения теста состоит из трех этапов:

1. Ознакомление с информацией о тесте (старт теста).

2. Прохождение теста (процесс ответа на вопросы студентом).

3. Подсчет и выдача результатов прохождения.

Первый этап формальный, пользователь ознакомляется с правилами проведения теста и подтверждает, что хочет пройти именно этот тест.

Второй и третий этап для разных видов тестов значительно отличаются.

На этапе прохождения (втором этапе) классического вида теста с сервера загружаются все данные о тесте с массивом вопросов, каждый элемент которого содержит свой массив с ответами, после чего они поступают в компонент, написанный на *JavaScript*.

После получения данных модулем происходит их обработка, заключающаяся в установке порядка следования вопросов и ответов случайным образом (при наличии соответствующих настроек теста).

В классическом виде тестов пользователю предоставляется в момент времени один вопрос. Для перехода к следующему вопросу необходимо дать минимум один вариант ответа или напечатать один символ для соответствующих видов вопроса либо же, при наличии соответствующих настроек, пользователь может пропустить вопрос и вернуться к нему позже.

После ответа на вопрос, этот ответ добавляется в массив с ответами пользователя.

Завершение классического теста происходит при установлении соответствующей переменной в положительное значение. Это происходит если:

– студент ответил на все вопросы;

– вышло время на прохождение теста;

– студент добровольно завершил тест, нажав соответствующий элемент управления.

По завершению классического вида теста происходит *POST* запрос на сервер со передачей все ответов на вопросы пользователя. По принятию этих данных происходит обход всех вопросов, на которые дал ответы студент, на каждом проходе которого делается в свою очередь еще один обход проверки правильности ответов, данных студентом. На этом этапе не учитываются вопросы, на которые не было дано ответа, и ответы, которые в тесте отмеченные как правильные, но не отмечены студентом.

Следующим этапом обработки результатов является формирование массива для предоставления пользователю вопросов, на которые он дал правильные или неправильные ответы (условимся, что вид вопроса слово/число подразумевает один правильный вариант ответа), а также подсчет баллов пользователя. На этом этапе модифицируется массив вопросов теста.

Во время обхода вопроса теста собирается такая информация как вес вопроса, количество правильных ответов в вопросе, наличие неверных ответов. также проставление условной пометки каждому затребованному варианту ответа, а таковыми являются все правильные варианты ответа на вопрос в совокупности с ответами пользователя.

Пометки бывают трех типов:

– вариант ответа правильный и пользователь отметил его;

– вариант ответа неверен и пользователь отметил его;

– вариант ответа верен, но пользователь не отметил его.

Данные пометки нужны для представления студенту его ошибок после прохождения теста.

Если все варианты ответов, данные пользователем на ответ верны, то вес вопроса добавляется как в общий счетчик веса, так и в счетчик веса правильных ответов.

После окончания обхода формируется процент правильных ответов и округленный балл за прохождение теста, рассчитывающийся по формуле:

, (2.1)

где P*user* – общий вес вопросов, на которые правильно ответил студент, *Pmax* – общий вес всех вопросов в тесте, *Tmax*– максимальный балл за прохождение теста, а *Y* – итоговый результат в баллах.

После всех необходимых подсчетов данные заносятся в базу и студенту возвращается страница с данными о результатах прохождения.

В случае, если в качестве проходимого теста был выбран тест-сопоставление, алгоритм разнится. Во-первых, для данного вида теста доступна функция указания количества предоставляемых студенту вопросов и ответов, которые выбираются случайным образом. Это задумано для уменьшения вероятности списывания ответов одним студентом у другого.

Алгоритм выбора случайных вопросов и ответов заключается в удалении случайных элементов из общего массива с вопросами или ответами. Количество удаляемых элементов получается с помощью разности количества элементов в тесте вообще и количества элементов, показываемых студенту.

После выбора случайных вопросов и ответов, если установлены соответствующие настройки, происходит их перемешивание, после чего данные об этих вопросах и ответах заносятся в базу данных и эти же массивы передаются на клиентскую сторону.

После получения данных модулем *JavaScript* данные располагаются в виде двух списков: вопросов и ответов. Для того, чтобы дать ответ на вопрос требуется для начала выделить вопрос нажатием на него, а затем вписать в него номер из списка ответов, либо нажатием на этот ответ в списке. Номера отмеченных вариантов ответов после выбора становятся рядом с вопросом, на который они были даны, при этом в основном списке показывается, что этот вариант ответа уже дан. Для извлечения варианта ответа из данных на вопрос ответов требуется нажать на номер этого варианта в области рядом с вопросом, либо нажать на него в общем списке ответов.

Так как для прохождения случается случайный набор вопросов и ответов, то возможен вариант, что на некоторые вопросы в выборке не будет правильных ответов. Для этого требуется предусмотреть вариант ответа «Нет правильных ответов». Этот вариант ответа можно поставить для всех вопросов, а не только для одного, как в случае с обычными вариантами ответов.

Прохождение теста данного вида завершается двумя способами: по завершению действия таймера и с помощью кнопки завершения теста. После завершения прохождения теста данного вида также, как и для классических тестов, отправляется *POST* запрос на сервер с данными о вопросах и ответах, данных на них студентом.

Помимо данных пользователем ответов на вопросы делается выборка случайным образом выбранных вопросов и ответов по записям в базе данных.

Затем подсчитывается сумма всех весов вопросов в тесте, а также вес правильного ответа в вопросе. Этот вес рассчитывается по формуле:

,(2.2)

где *Pq* – вес вопроса, *Kq* – количество правильных вариантов ответов на вопрос.

После данной процедуры происходит подсчет количества верных вариантов ответов в выборке. Это понадобится при подсчете процента правильных ответов, данных студентом при прохождении.

В данном виде теста, в отличии от предыдущего, на сервер поступает информация обо всех вопросах, выданных студенту, а не только о тех, на которые были даны ответы, как в случае с классическим видом тестов.

После определения количества верных ответов для всего теста происходит сбор информации, необходимой для подсчета результатов.

Стоит отметить, что в данном виде тестов применяется особый алгоритм подсчета количества баллов. Он складывается из трех метрик:

– количество верных ответов, данных студентом;

– количество неверных ответов;

– количество вопросов, на которые дан хотя бы один правильный ответ.

В процессе обхода вопросов и ответов теста находятся значения суммы весов правильных ответов – *Ptrue*, суммы весов неверных ответов – *Pfalse*, сумма весов всех вопросов и сумма весов вопросов, на которые пользователь дал хотя бы один правильный ответ – *Pone*. Общая формула для расчета суммы весов представлена в формуле 2.3

, (2.3)

где *pi* – вес одного вопроса

Так же в процессе данного обхода ставятся пометки на вариантах ответов, рассмотренные в первом виде тестов.

После сбора всех значений подводятся итоги. Количество баллов за тест рассчитывается по следующей формуле:

, (2.4)

где *Tone* – оценка за вопросы, на которые дано хотя бы по одному правильному ответу, *Ttrue* – оценка за правильные ответы, *Tfalse* – оценка за неправильные ответы.

*Tone* в свою очередь рассчитывается по формуле:

(2.5)

Для нахождения *Ttrue* используется формула 2.6:

(2.6)

*Tfalse* находится по следующей формуле:

(2.7)

Для формул 2.5 – 2.7 переменная *k* – количество баллов за единицу веса вопроса. Данная переменная рассчитывается по формуле:

, (2.8)

Где *Tmax* – максимальная оценка за прохождение теста, *Pmax* – общий вес вопросов, выданных случайным образом студенту.

После нахождения всех результатов данные о результате заносятся в базу данных и отправляются для представления на клиентскую часть.

**2.3 Структура, архитектура программного продукта**

Программный продукт своей структурой представляет две главных независимых друг от друга части: создание и редактирование тестов и прохождение тестов. Визуализация этой структуры представлена на рисунке 2.1.



Рисунок 2.1 – Структурная схема программного продукта

Как видно по рисунку 2.1 дополнительными частями структуры программного продукта являются страница приветствия, аутентификация, просмотр результатов тестов студентами и редакторами тестов и редактирование групп и пользователей администратором.

Каждая из двух главных частей также подразделяется на компоненты.

Создание и редактирование тестов структурно представляет собой практически полноценное одностраничное приложение (*Single Page Application*), то есть все запросы на сервер происходят асинхронно без перезагрузки страницы. Данная структурная часть состоит из множества *JavaScript* компонентов, представленных на рисунке 2.2.



Рисунок 2.2 – Структура части создания и редактирования тестов

На рисунке видно, что структурная часть создания и редактирования тестов содержит пять компонентов: боковую панель, содержащую информацию о всех созданных тестах и рычаги управления компонентами основной части, и четыре сменяющих друг друга компонента на основной части экрана.

Компонент того-то делает то-то и тд.

Второй основной структурной частью является часть прохождения тестов. Она представляет собой также множество компонентов, но в этом случае предусмотрен и переход между страницами.

Для этой части существует четыре основных страницы, которые представлены на рисунке 2.3.



Рисунок 2.3 – Страницы структурной части прохождения тестов

Рисунок 2.3 показывает, что структурная часть прохождения тестов содержит страницы для выбора теста на прохождения, а также три страницы ориентированных на работу с конкретным выбранным тестом, а именно:

– старт теста, где пользователь получает информацию о выбранном тесте, его основных настройках;

– прохождение теста, где пользователь занимается непосредственно прохождением теста, именно на этой странице пользователь будет проводить больше всего времени;

– результаты прохождения теста, где пользователь может



Рисунок 2.4 – Структура части прохождения тестов

Как видно по рисунку 2.4, структурная часть прохождения тестов содержит уже семь компонентов-модулей.

На компоненте бла-бла-бла

Что в архитектуре?

Архитектура Laravel MVC?

**2.4 Описание логической структуры программного продукта**

Логическая структура (структура базы данных) является одной из важнейших составляющих программного продукта, от нее в существенной мере зависит возможность масштабирования и быстродействие при поиске необходимой информации.

База данных проектируется для информационного обслуживания редакторов (создателей) тестов и студентов, проходящих эти тесты. БД должна содержать информацию о пользователях системы, созданных ими тестах, которые содержат вопросы и ответы, результаты прохождения тестов пользователями, а также некоторые дополнительные данные.

На рисунке 2.х представлена *ER*-диаграмма для проектируемого программного продукта.

Возможно стоит сделать как у вас в диссертации только сущности?

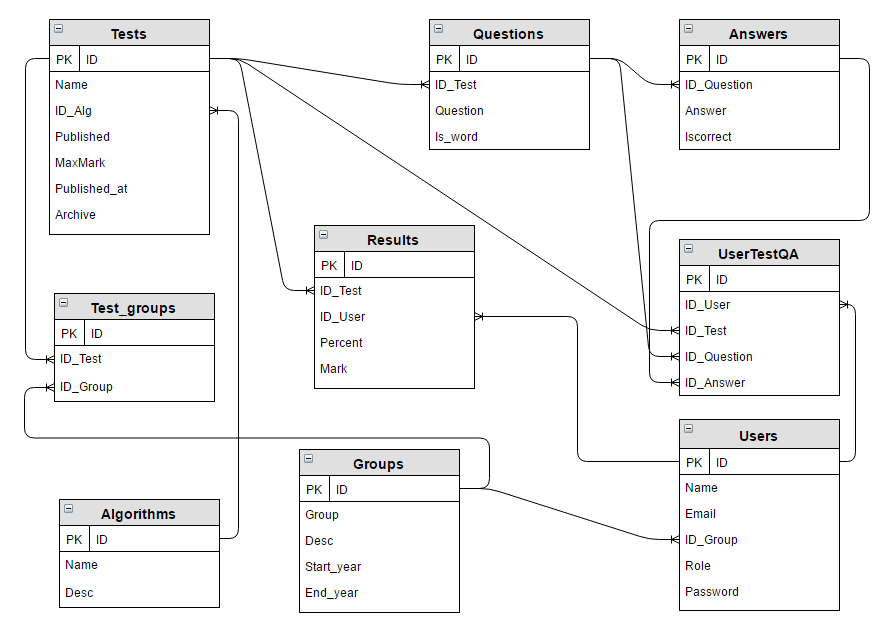


Рисунок 2.x – ER-диаграмма проектируемой области

Как видно по схеме, база данных должна иметь девять сущностей.

Сущность *«Users»* предназначена для хранения пользователей системы. Атрибутами таблицы являются: идентификатор, имя, *e-mail*, пароль, роль пользователя и идентификатор группы, к которой принадлежит зарегистрированный пользователь-студент.

Сущность *«Tests»* хранит в себе данные о тестах, создаваемых редакторами тестов. Атрибутами таблицы являются название теста, описание теста, идентификатор редактора создаваемого теста, максимальный балл за прохождение, идентификатор алгоритма, используемого при прохождении теста, дата публикации, дата окончания публикации, факт публикации, время на прохождение теста. (не все атрибуты написал)

Сущность *«Algorithms»* хранит в себе информацию об алгоритмах, используемых для теста. Атрибутами данной сущности являются идентификатор, название и описание.

Сущность *«Questions»* хранит в себе информацию о вопросах теста. Атрибутами данной сущности являются текст вопроса, его идентификатор, вес вопроса и тип вопроса.

Сущность *«Answers»* хранит в себе информацию об ответах на вопросы. Атрибутами данной сущности являются идентификатор, текст ответа на вопрос и факт правильности ответа.

Сущность *«Groups»* хранит в себе информацию о группах студентов. Атрибутами данной сущности являются идентификатор, наименование группы, описание группы, дата образования и дата выпуска группы.

Сущность *«Test\_groups»* хранит в себе информацию о группах, для которых назначен тест. Атрибутами данной сущности являются идентификатор записи, идентификатор теста и идентификатор группы.

Сущность *«UserTestQA»* хранит в себе информацию об ответах студентов на вопросы. Атрибутами данной сущности являются идентификаторы студента, теста, вопроса и ответа, а также идентификатор записи.

Сущность *«Results»* хранит в себе информацию о результатах прохождения тестов студентами. Атрибутами данной сущности являются идентификатор записи, студента и теста, а также процент правильных ответов и оценка за тест.

Для проектируемой базы данных между таблицами существуют различные связи:

– сущности *Groups* и *Users* связаны отношением один ко многим;

– сущности *Users* и *Results* связаны отношением один ко многим;

– сущности *Users* и *UserTestQA* связаны отношением один ко многим;

– сущности *Groups* и *Test\_groups* связаны отношением один ко многим;

– сущности *Algorithms* и *Tests* связаны отношением один ко многим;

– сущности *Tests* и *Test\_groups* связаны отношением один ко многим;

– сущности *Tests* и *Questions* связаны отношением один ко многим;

– сущности *Questions* и *Answers* связаны отношением один ко многим;

– сущности *Questions* и *UserTestQA* связаны отношением один ко многим;

– сущности *Answers* и *UserTestQA* связаны отношением один ко многим;

– сущности *Tests* и *Results* связаны отношением один ко многим;

– сущности *Tests* и *UserTestQA* связаны отношением один ко многим.

**2.5 Функциональная схема, функциональное назначение**

**программного продукта**

Основным назначением проектируемого программного продукта является проведение тестирования студентов. Для проведения тестирования необходимо наличие тестов, которые составляет редактор тестов. Диаграмма вариантов использования приложения редактором тестов представлена на рисунке 2.х

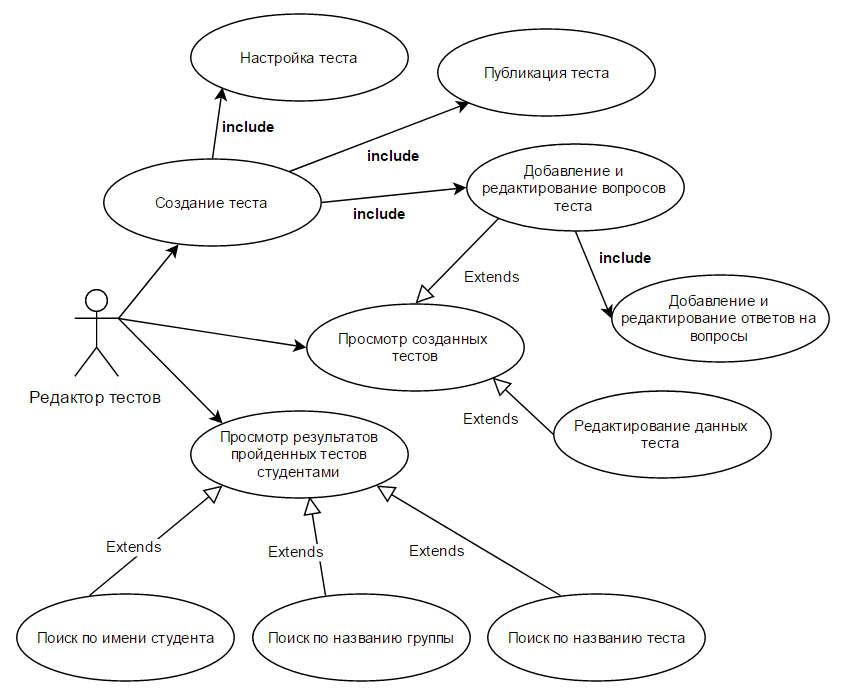


Рисунок 2.х – диаграмма вариантов использования приложения редактором тестов

По данной диаграмме видно, что основной функцией редактора тестов в приложении является работа с тестами. В процессе работы с программным продуктом редактор тестов сначала создает тест, после чего добавляет и редактирует вопросы и ответы на вопросы теста, а также имеет возможность проводить настройку теста и его публикацию. Публикация теста происходит исключительно для установленных редактором групп студентов и в установленную дату и время.

После публикации, если тест опубликован для группы, в которой зарегистрирован студент и наступила дата публикации, тесты становятся доступными для прохождения студенту. Как видно по рисунку 2.х, из этих тестов он выбирает тот, который нужно или хочет пройти в данный момент. Подтверждая выбор проходимого теста, студент вовлекается непосредственно в процесс прохождения. После окончания данного процесса студент просматривает результаты прохождения, где вместе с результатами он может увидеть, на какие вопросы какие ответы являются правильными.

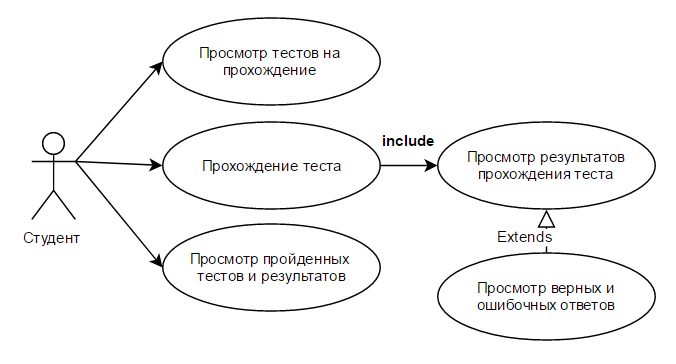


Рисунок 2.х – диаграмма вариантов использования приложения студентов

Пройдя несколько тестов, студент с помощью проектируемого приложения может освежить в памяти результаты обо всех этих тестах. Редакторы тестов также могут зафиксировать результаты прохождения этих тестов студентами.

На рисунке 2.х показана диаграмма прецедентов для администратора проектируемого программного продукта.

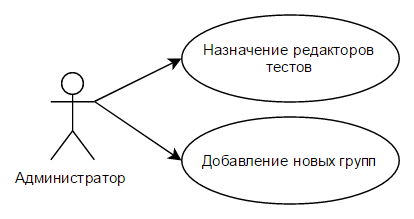


Рисунок 2.х – диаграмма вариантов использования приложения администратором

По умолчанию все зарегистрированные пользователи являются студентами. Для становления редактором тестов необходимо, чтобы пользователя им назначил администратор. Также функцией администратора является ежегодное добавление новых групп для студентов.

**2.6 Выводы**

В результате спроектировали программный продукт.

**3 РЕАЛИЗАЦИЯ И ТЕСТИРОВАНИЕ ПРОГРАММНОГО ПРОДУКТА**

**3.1 Описание реализации**

Для реализации серверной части приложения используется *Laravel Framework* версии 5.3.

Для определения доступа к страницам проекта используется файл маршрутизации *routes/web.php*. В данном файле находится множество всех доступных маршрутов. Пример одного из них представлен в листинге 3.х.

Route::*post*(**'/pass/test/{id}/result'**,[**'uses'** => **'ResultController@update'**, **'as'** => **'pass.update'**]);

Данный маршрут предназначен для *POST* запроса на выдачу результатов прохождения теста. В этом листинге показывается, что для доступа к *URL* {домен сайта}*/pass/test/*{идентификатор теста}*/result* будет использоваться функция *update* контроллера *ResultController*, а ассоциируемое с этим маршрутом имя – *pass.update.*

Для работы с базой данных используется *Eloquent ORM* – технология, связывающая БД с концепциями объектно-ориентированного программирования. В листинге 3.х представлен пример отношения таблиц один ко многим.

**public function** questions()  
{  
 **return** $this->hasMany(**'App\Models\Question'**,**'id\_test'**);  
}

Данная функция находится в модели *Test,* которая хранит все запросы к таблице *Tests.* Представленная функция показывает, что в одном тесте может быть много вопросов (модель *Question*) связанных с таблицей *Tests* полем *id\_test*.

На листинге 3.х продемонстрирован пример выборки из базы данных.

**public function** getAuthTests($id){  
 $tests=$this->with([**'questions'**,**'questions.answers'**,**'algorithm'**])

->where(**'id\_user'**,$id)->get();  
 **return** $tests;  
}

На данном листинге происходит получение всех тестов, созданных пользователем с идентификатором *$id*. Метод *with* позволяет вместе с данными тестов загрузить из базы данных еще и данные, сопряженные отношениями с таблицей *Tests*. В данном запросе это вопросы теста, ответы на вопросы и информация об использованном алгоритме теста.

Для логической обработки данных используются контроллеры, которые после этой самой обработки могут выдавать представление с данными, либо отправлять какие-либо данные на клиента без представления. Эти операции можно делать также и в файле маршрутизации.

$data=Input::*except*([**'id'**]);  
$test=**new** Test();  
$testModel->insert($data);  
$data=$testModel->getLastTest();  
**return** response()->json([**'data'**=>$data]) ;

Так как реализуемое приложение подразумевает большое количество *AJAX* запросов, большинство обработчиков контроллера используется в качестве *api*, то есть результатом их работы являются отправляемые данные. В листинге, представленном выше, показан код обработчика *store* контроллера *TestController*, который выполняет добавление нового теста в базу данных и возвращает данные о нем обратно на сторону клиента.

Для переключения компонентов на клиентской стороне приложения используется метод switchMainView.

switchMainView: function(view,ar = {}){

if (this.current.mainView!=view||!isEqual(this.current.array,ar)){

if (this.current.mainView!=view){

this.previous.push(clone(this.current));

}

this.current.mainView=view;

if (Object.keys(ar).length>0){

this.current.array=ar;

}

}

},

В данном методе происходит сравнение названия и массивов нового и текущего компонентов. Если они не совпадают, то компонент меняется и данные о предыдущем компоненте заносятся в стек *this.previous*. Если же не совпадают только массивы, а название компонента одинаково, то меняется только этот самый текущий массив.

Изменение основного компонента происходит после наступления какого-либо события (нажатие кнопки, отправка формы, изменение данных). Пример использования метода switchMainView представлено в листинге 3.х.

<i class="sidebar\_\_icon sidebar\_\_icon--blue fa fa-newspaper-o"

@click.prevent="$parent.$parent.switchMainView('test-publish',

{ testId: test.id })">

</i>

Например, в сайдбаре, на странице редактирования тестов, при нажатии на иконку «опубликовать тест» происходит изменение текущего главного компонента на компонент *test-publish*.

Вся работа редактором тестов с базой данных происходит без перезагрузки. На листинге ниже представлен метод *setupTest*, предназначенный для создания и редактирования теста.

let data= getFormData($('#test-form'));

if (this.edited){

this.$http.put('/api/test/'+this.test.id, data).then(function(response) {

this.test=response.data.data;

this.$parent.showPopupDelay(1500);

this.$parent.popup.header="Тест успешно обновлен";

this.$parent.$refs.testSidebar.getTests();

}, function(response) {

});

}

else {

this.$http.post('/api/test', data).then(function(response) {

this.test=response.data.data;

this.$parent.showPopupDelay(2000);

this.$parent.popup.header="Тест успешно создан";

this.$parent.$refs.testSidebar.getTests();

this.switchTestInfo();

}, function(response) {

});

}

Данный метод срабатывает по нажатию на кнопку создать/обновить тест. В начале собираются данные формы, необходимые для отправки на сервер. Затем в зависимости от того, создан уже тест или нет происходит *AJAX* запрос на создание или обновление данных. После редактирования теста появляется соответствующее уведомление и обновляется информация в сайдбаре. Если же тест только создается, то помимо уведомления и обновления сайдбара, произойдет переход на страницу с информацией о тесте.

Для сбора информации формы используется метод *getFormData*.

var unindexed\_array = $form.serializeArray();

$.each($($form).find('input[type=checkbox]')

.filter(function(idx){

return $(this).prop('checked') === false

}),

function(idx, el){

unindexed\_array.push({name:$(el).attr('name'),value:0});

}

);

var indexed\_array = {};

$.map(unindexed\_array, function(n, i){

n['value']=n['value']=='on'?1:n['value'];

indexed\_array[n['name']] = n['value'];

});

return indexed\_array;

В данном методе собираются данные формы с помощью метода *jQuery* *serializeArray*, после чего происходит изменение представления данных полей типа *checkbox*. После этого происходит формирование выходного массива, пригодного для преобразования в *JSON*.