

## ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	3
1 ИССЛЕДОВАНИЕ И АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	5
1.1 Описание поставленной задачи, ее обоснование	5
1.2 Обоснование актуальности исследуемой задачи	6
1.3 Современное состояние исследуемой задачи	7
1.4 Обзор методов решения подобных задач	9
1.5 Постановка задачи, системные требования, требования к входным данным и выходным формам	13
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

В современном мире большую роль играет контроль знаний учащихся школ, работников, студентов вузов и т.д. Как известно, под контролем понимается система научно обоснованной проверки результатов обучения. Более узкое определение гласит, что контроль - это выявление, измерение и оценка знаний, умений и навыков [13].

Существует множество форм контроля - экзамен, зачет, аттестация, контрольная. Но одним из самых популярных способов измерения знаний на сегодняшний день является тест [14]. Тест - это совокупность специальным образом подготовленных и подобранных заданий, позволяющая провести выявление требуемых характеристик процесса обучения [20]. Одно из главных преимуществ тестов состоит в том, что они позволяют опросить всех участников по всем вопросам нужного материала в одинаковых условиях, применяя при этом ко всем без исключения одну и ту же, заранее разработанную шкалу оценок. Это значительно повышает объективность, нерасплывчатость и обоснованность оценки по сравнению, скажем, с экзаменом.

Тесты и тестовые задания получили свое распространение в сферах, где нужно точно определить знания по всему курсу обучения, где много сдающих, где нужен жесткий отбор участников: экзамены в ГИБДД, аттестация работников предприятий, прием на работу, аттестация и контроль остаточных знаний студентов, дистанционное образование, экзаменационный тест и т.д.

В настоящее время, в эпоху информационного общества, глобальной компьютеризации, развития технологии Интернет и передачи данных все более актуальными становятся разнообразные компьютерные системы тестирования, способные дополнить или заменить традиционные методы контроля и методики преподавания [13]. Благодаря компьютерным системам тестирования стало намного удобнее проводить тестирования во всех сферах, где применялись и применяются обычные тесты. Например, дистанционное образование, которое стало распространенным способом получения знаний.

Компьютерное тестирование обладает рядом преимуществ перед традиционным тестированием. Оно отличается высокой оперативностью, производительностью процесса тестирования и объективностью результатов контроля знаний - преподаватель может провести опрос гораздо большего числа студентов за меньшее время по сравнению с очным опросом, и позволяет проанализировать качество подготовки тестируемых по большому кругу различных вопросов.

Компьютерные тесты позволяют использовать сложные методы контроля оценки знаний учащихся, снизить финансовые и временные затраты при проведении тестирования, применить в тестах мультимедийные задания, а также повысить открытость процесса тестирования. Но наряду с достоинствами, у компьютерных тестов есть и свои недостатки: повышается вероятность случайного выбора ответа, понижается внимание на оформление решения, теряется логика рассуждения, теряется информация о процессе выполнения отдельных заданий учащимися, отношение многих людей к компьютеру не как к средству получения и контроля знаний, а как к средству развлечения.

# 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 нужной 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 ответов 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 им. Т.Г. Шевченко.

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

Вначале преподавателю необходимо спроектировать и напечатать образец теста, потом сделать необходимые его копии. Затем нужно раздать копии теста, после чего студенты с разной скоростью их заполняют и сдают. 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 [11].

Помимо всего перечисленного, возможна потеря 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.

Стоит отметить, что для ПГУ уже была разработана автоматизированная система для проведения тестирования студентов, но она подразумевает проведения только классического вида тестирования с вариантами ответов, и при этом представляет собой классическое *Windows*-приложение.

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

### **1.3 Современное состояние исследуемой задачи**

Во многих странах мира компьютерное тестирование применяется уже довольно долгий промежуток времени и имеет широкую область применения в таких сферах как международные сравнительные исследования, мониторинг качества образования в масштабах страны, лицензирование и государственная аккредитация учебных заведений, аттестация учащихся и студентов, учителей и преподавателей, проверка профессиональной пригодности специалистов и т.д. [12].

В последнее время интерес к автоматизации различных видов учебной и административной деятельности все больше возрастает и в странах постсоветского пространства. В процессе обучения, прежде всего это коснулось информатизации контроля результатов обучения. Самым популярным видом такого контроля является тестирование, основанное на диалоге информационной системы и пользователя. Рост быстродействия технических средств, уменьшение цен на вычислительную технику, появление качественных и мощных систем программирования увеличило потребность в системах, позволяющих объективно, быстро и надежно оценивать знания учащихся, предлагая интересные формы взаимодействия с ними.

Проведение контроля достижения результатов обучения с использованием современных средств информационных технологий в процессе обучения по сравнению с другими методами контроля имеет ряд очевидных преимуществ, в числе которых: высокая степень стандартизации, объективность оценки результатов, удобная количественная форма выражения результатов, повышенная устойчивость к фальсификациям, высокая скорость обработки результатов,

единство требований ко всем учащимся, исключение субъективизма при оценке результатов [8, с.63].

При этом удобство в наличии количественных показателей выражается в возможности сравнения знаний и умений одних обучаемых с другими, или отслеживании динамики усвоения знаний одним учащимся в процессе обучения.

Итак, тестирование является эффективным видом проверки и контроля уровня знаний студентов на современном этапе развития высшего профессионального образования.

## 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> [21]. Данный сайт представляет собой бесплатный образовательный интернет сервис, который позволяет создавать и проходить тесты. Интерфейс добавления вопроса представлен на рисунке 1.1.

The screenshot shows a web application window titled "Тест номер 1". The interface includes a top navigation bar with links: "Добавить вопрос", "Изменить титул", "Проверить тест", and "Результат". A "Сохранить" button is located in the top right corner. On the left, a sidebar shows a tree view with "Тест" and "Вопрос 1". The main content area is a form for adding a question. It contains the following fields and controls:

- "Заголовок Вопроса:" followed by a text input field.
- A checked checkbox labeled "Дополнительно".
- "Дополнительный текст:" followed by a text input field.
- A section titled "Медиа Контент" containing a "Добавить Медиа" button.
- An unchecked checkbox labeled "Указать источник информации".
- "Тип вопроса:" with a dropdown menu currently showing "Однозначный Ответ".
- A section titled "Ответ" containing two radio buttons, each followed by a text input field, and a "Добавить Ответ" button.
- "Вес Вопроса:" with a dropdown menu currently showing "1".
- A "Готово" button at the bottom.

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.

**Опции Результата**

- ☐ Задать текст, отображаемый, если будет набранно определенное количество баллов
- ☒ Отображать процентное соотношение набранных баллов к максимальному количеству баллов
- ☒ Отображать количество набранных баллов
- ☒ Отображать правильные ответы после прохождения теста
- ☒ Отображать количество правильных ответов

Сообщение:

Здесь можно вписать текст, который будет отображаться после прохождения теста

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

При выводе результатов преподаватель может учитывать 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.

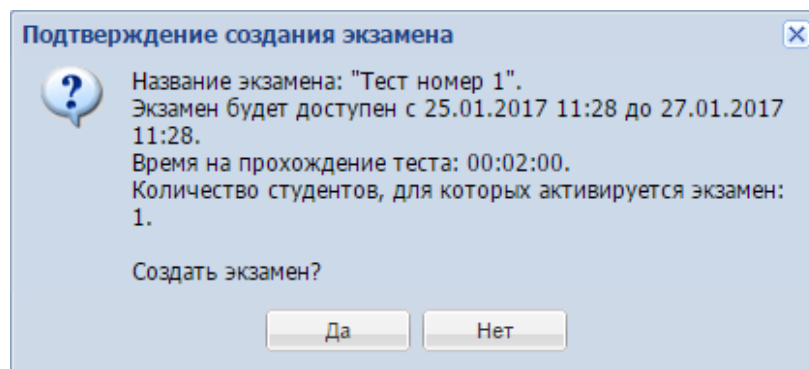


Рисунок 1.3 – Подтверждение публикации теста в системе *master-test.net*

Результат прохождения теста представлен на рисунке 1.4.

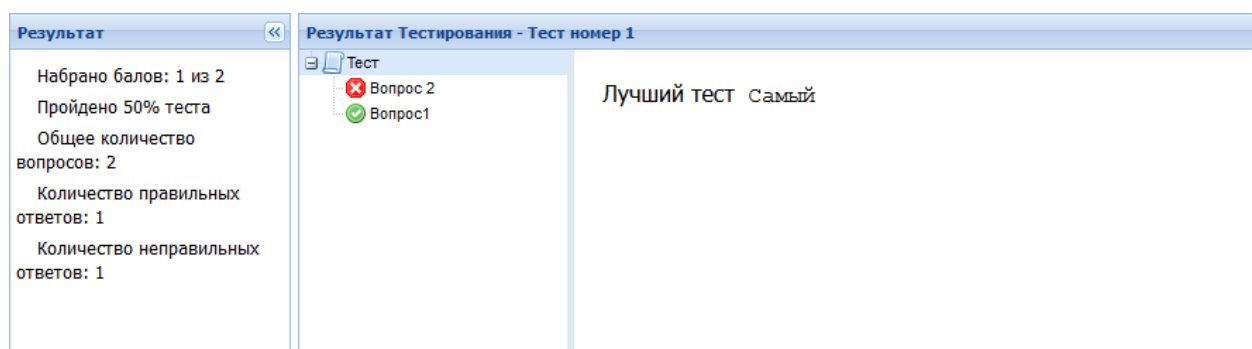


Рисунок 1.4 – Результаты прохождения теста в системе *master-test.net*

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 на наш взгляд имеет несколько неудачный интерфейс. 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.

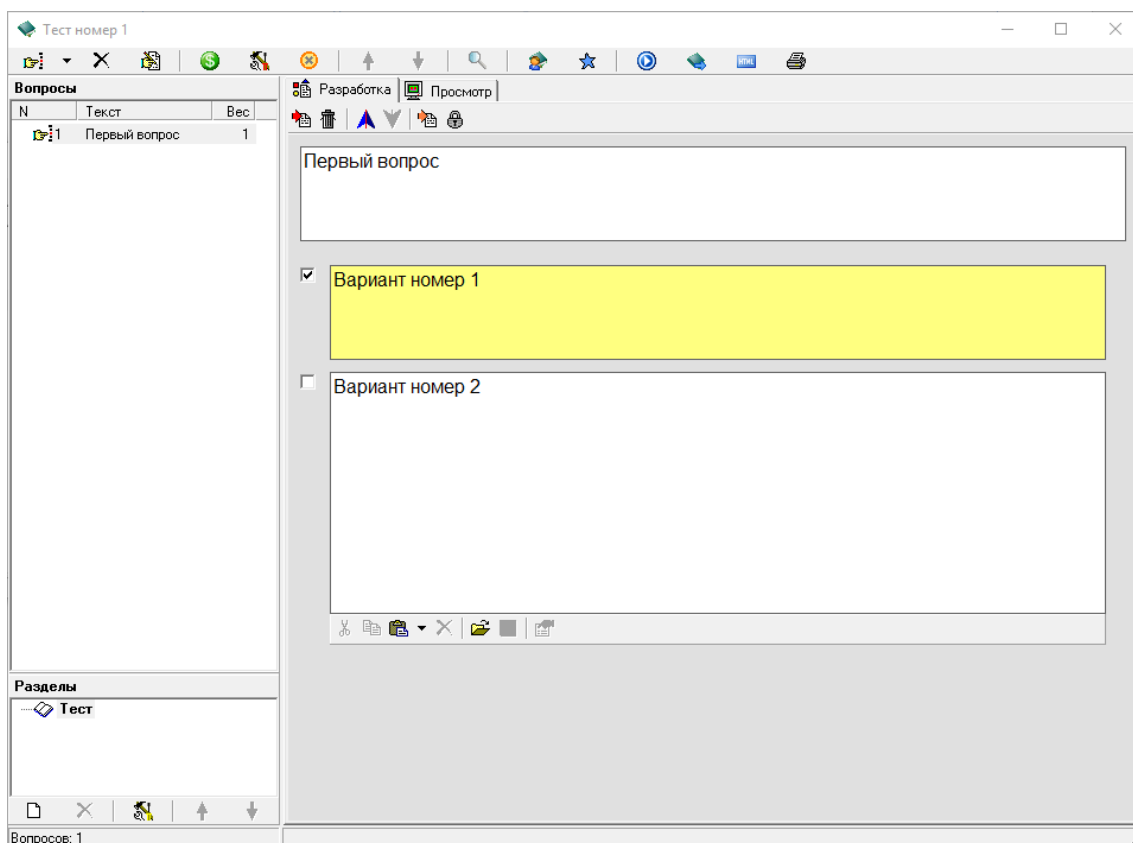


Рисунок 1.5 – Добавление вопроса в системе «Айрен»

В целом можно отметить, что 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 быть реализована в виде веб-приложения, имеющего 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 при этом 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 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.

Программный продукт необходимо разработать в виде веб-приложения, так как данное решение имеет над классическим настольным приложением ряд таких преимуществ, как кроссплатформенность, простота структурирования кода и гибкость в реализации интерфейса.

Входными данными программного продукта 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 практически 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 для операционных систем используемых пользователями 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 150 мегабайт свободной оперативной памяти.

## **1.6 Выводы к главе 1**

В первой главе был проведен подробный анализ предметной области; произведено обоснование актуальности исследуемой задачи на основании схожих по теме программных продуктов в различном представлении.

Также в данной главе было обозначено современное состояние исследуемой задачи, что позволило составить конкретные требования к программному продукту и его окружению, а также требования ко входным и выходным формам.

## 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* [10, с.18]. Было решено остановить свой выбор на языке *PHP*, так как это наиболее популярное решение для веб-приложений на сегодняшний день [20]. 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 [15].

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 программирования можно использовать разными способами:

1. Только во вспомогательных целях для придания относительной интерактивности 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 [19].

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*. Стоит отметить, что все технологии являются открытыми проектами и не требуют специального лицензирования даже при коммерческом использовании.



## 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 знакомится 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* [9, с.173].

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, содержащий ответы 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:

$$Y = \frac{P_{user}}{P_{max}} * T_{max} , \quad (2.1)$$

где  $P_{user}$  – общий вес вопросов, на которые правильно ответил студент;

$P_{max}$  – общий вес всех вопросов в тесте;

$T_{max}$  – максимальный балл за прохождение теста;

$Y$  – итоговый результат в баллах.

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

Приведем пример подсчета. Пусть в классическом тесте даны шесть вопросов, три из которых имеют вес равный 1, два вопроса даны с весом 2, и один, самый сложный, имеет вес 3. Максимальный балл же за прохождение теста – 12 баллов.

Таким образом, ответив правильно на все простые по весу вопросы и на 1 вопрос средней важности, студент получит  $Y = (3+2) / (3+2+3) * 12 = 6$  баллов.

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:

$$P_a = \frac{P_q}{K_q}, \quad (2.2)$$

где  $P_q$  – вес вопроса;

$K_q$  – 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 отличие 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 –  $P_{true}$ , суммы весов неверных ответов –  $P_{false}$ , сумма весов всех вопросов и сумма весов вопросов, на которые пользователь дал хотя бы один правильный ответ –  $P_{one}$ . Общая формула для расчета суммы весов представлена в формуле 2.3

$$P = \sum_{i=1}^n p_i , \quad (2.3)$$

где  $p_i$  – 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:

$$Y = T_{one} + T_{true} - T_{false} , \quad (2.4)$$

где  $T_{one}$  – оценка за вопросы, на которые дано хотя бы по одному правильному ответу;

$T_{true}$  – оценка за правильные ответы;

$T_{false}$  – штраф за неправильные ответы.

$T_{one}$  в свою очередь рассчитывается по формуле:

$$T_{one} = k * P_{one} \quad (2.5)$$

Для нахождения  $T_{true}$  используется формула 2.6:

$$T_{true} = k * P_{true} \quad (2.6)$$

$T_{false}$  находится по следующей формуле:

$$T_{false} = \frac{k}{2} * P_{false} \quad (2.7)$$

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

$$k = \frac{T_{max}}{2 * P_{max}}, \quad (2.8)$$

Где  $T_{max}$  – максимальная оценка за прохождение теста;

$P_{max}$  – 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.

Для наглядности алгоритма возьмем ситуацию, представленную на рисунке 2.1.



Рисунок 2.1 – Пример прохождения теста-сопоставления

Как видно по данному рисунку, студенту представлены три вопроса и шесть вариантов ответа. В овалах к вопросам обозначены веса, а в овалах к вариантам ответы их принадлежность к вопросам.

При этом максимальный балл за прохождение возьмем равным 15.

Таким образом у каждого вопроса по 2 варианта ответа. Для первого вопроса вес одного ответа (по формуле 2.2) будет равен 0.5, для второго вопроса – 1.5, а для третьего – 1. По формуле 2.3 вес всех вопросов  $P = 1 + 2 + 3 = 6$ .

Обратим внимание, что на первый вопрос студент дал два правильных ответа и один неправильный, на второй вопрос студент дал один правильный ответ из двух, а на третий вопрос ответил полностью правильно.

Произведем расчет количества баллов за единицу веса (формула 2.8). Получим  $k = 15 / (2 * 6) = 1.25$  баллов.

Таким образом, по формуле 2.6, количество баллов за правильные ответы  $T_{true} = 1.25 * (0.5 + 0.5 + 1.5 + 1 + 1) = 5.625$  баллов, штраф за неправильные ответы (формула 2.7)  $T_{false} = 0.5 * 1.25 / 2 = 0.3125$  баллов, а за вопросы, на которые дан хотя бы один правильный ответ (формула 2.5) студент получит  $T_{one} = (0.5 + 1 + 1.5) * 1.25 = 3.75$  баллов.

Итого, в данном случае, за прохождение теста студент получит  $Y = 5.625 + 3.75 - 0.3125 = 7.6875 \approx 10$  баллов из 15.

### 2.3 Структура, архитектура программного продукта

Архитектурно проектируемый программный продукт представляет собой классическое веб-приложение: существует сервер, на котором хранится сайт и база данных. Пользователи отправляют запросы на сервер на получение и изменение данных. Гораздо более интересна внутренняя архитектура серверной части приложения.

Для реализации серверной части в проектируемом программном продукте используется разделение данных с помощью концепции *MVC* (*Model-View-Controller*) [5, с.423].

Модель (*Model*) предоставляет данные и методы работы с ними: запросы в базу данных, проверка на корректность. Модель не зависит от представления, то есть не знает, как визуализировать данные, и контроллера, то есть не имеет точек взаимодействия с пользователем. Она просто предоставляет доступ к данным и управлению ими. Модель строится таким образом, чтобы отвечать на запросы,

изменяя своё состояние, при этом может быть встроено уведомление «наблюдателей». Модель, за счёт независимости от визуального представления, может иметь несколько различных представлений для одной «модели».

Представление (*View*) отвечает за получение необходимых данных из модели и отправляет их пользователю. Представление не обрабатывает введённые данные пользователя.

Контроллер (*Controller*) обеспечивает связь между пользователем и системой. Контролирует и направляет данные от пользователя к системе и наоборот. Использует модель и представление для реализации необходимого действия [20].

Такая концепция помогает грамотно дифференцировать код, что позволяет повысить количество повторно используемого кода, а также улучшить ориентирование в программном продукте.

Схема MVC представлена на рисунке 2.2.

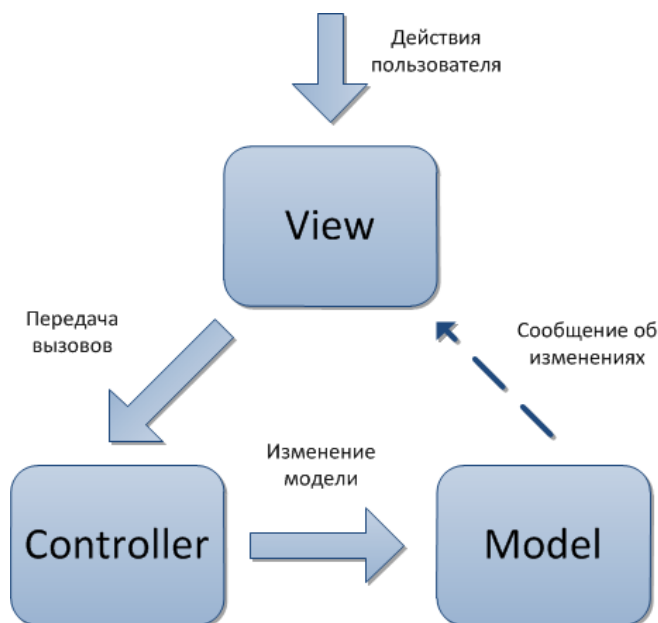


Рисунок 2.2 – Схема разделения данных MVC

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 2.3.





Рисунок 2.3 – Структурная схема программного продукта

Как видно по рисунку, 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 2.4.

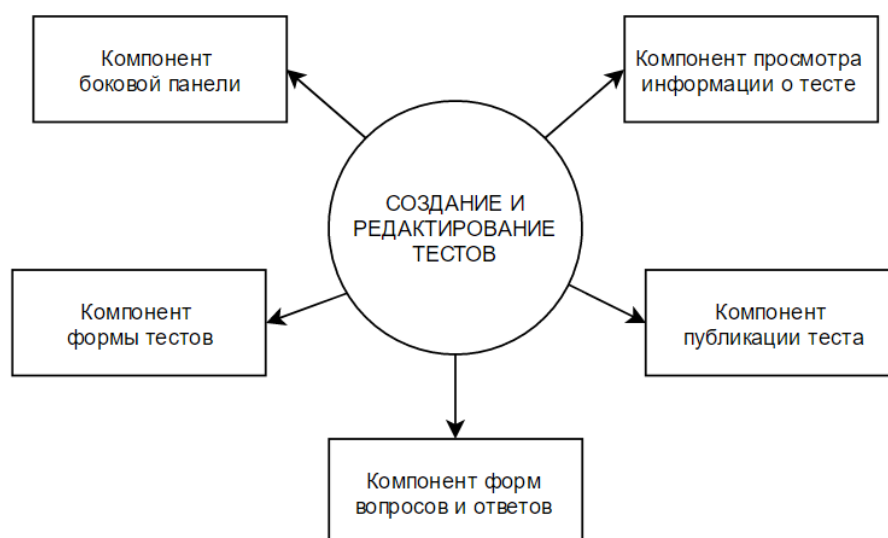


Рисунок 2.4 – Структура части создания и редактирования тестов

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.

В компоненте боковой панели содержится список всех тестов. Эта панель позволяет пользователю перейти к любому другому основному компоненту.

Компонент формы тестов предназначен для создания и изменения тестов.

Компонент форм вопросов и ответов предназначен для добавления и изменения вопросов и ответов теста.

Компонент публикации теста предназначен для публикации теста, промежуточными действиями которой является: формирование даты начала и окончания публикации и списка групп, для которых публикуется тест.

Компонент информации о тесте содержит информацию о всех полях и настройках теста, а также его вопросы и ответы.

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 2.5.

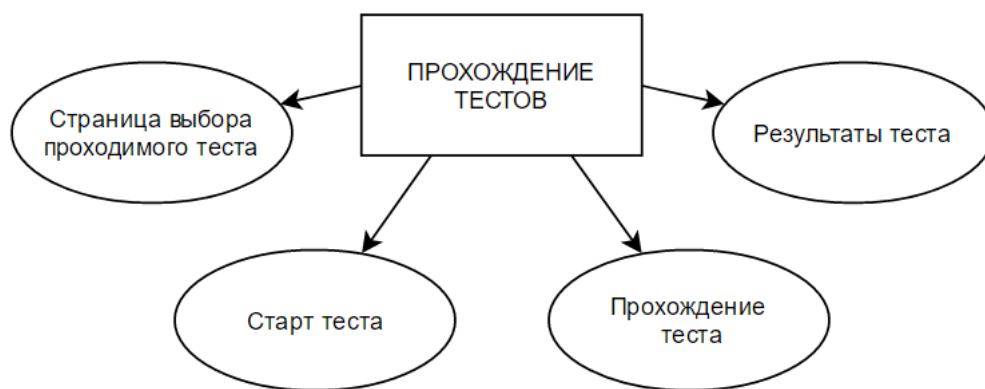


Рисунок 2.5 – Страницы структурной части прохождения тестов

Рисунок 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 просмотреть, насколько успешно он выполнил тест.

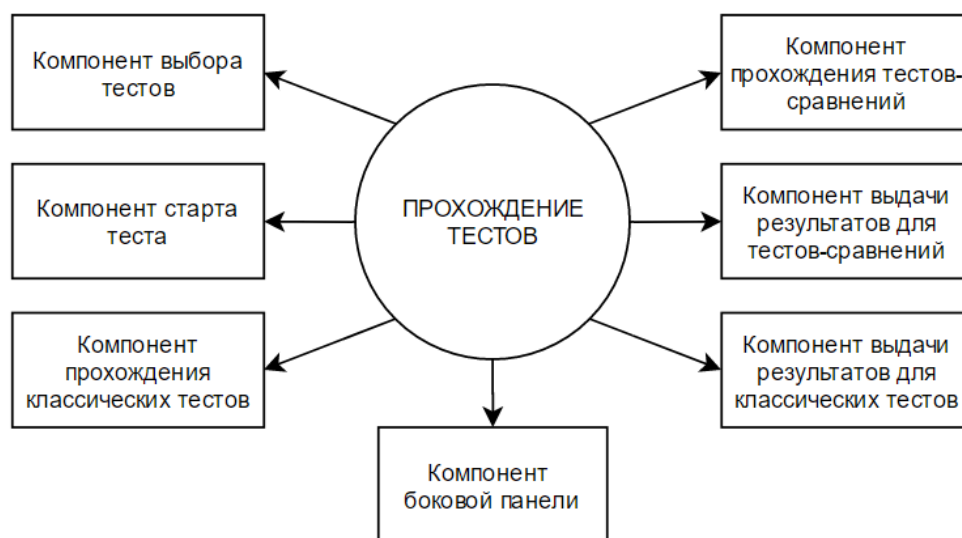


Рисунок 2.6 – Структура части прохождения тестов

Как видно по рисунку 2.6, структурная часть прохождения тестов содержит уже семь компонентов-модулей.

В компоненте выбора тестов студенту предоставляется информация о доступных для прохождения тестах.

В компоненте старта теста указывается информация о тесте и некоторые возможные предупреждения.

В компоненте прохождения классических тестов содержится структура и реализация алгоритмов для прохождения классического тестирования.

Компонент боковой панели содержит информацию о пройденных студентом тестах и их результатах.

В компоненте прохождения тестов-сравнений содержится структура и реализация алгоритмов для прохождения классического тестирования.

В компоненте выдачи результатов для классических тестов содержатся алгоритмы и структура выдачи результатов классического тестирования.

В компоненте выдачи результатов для тестов-сравнений содержатся алгоритмы и структура выдачи результатов тестов-сравнений.

## 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 [1, с.15]. База данных проектируется для информационного обслуживания редакторов 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 2.7 представлена *ER*-диаграмма проектируемого программного продукта.

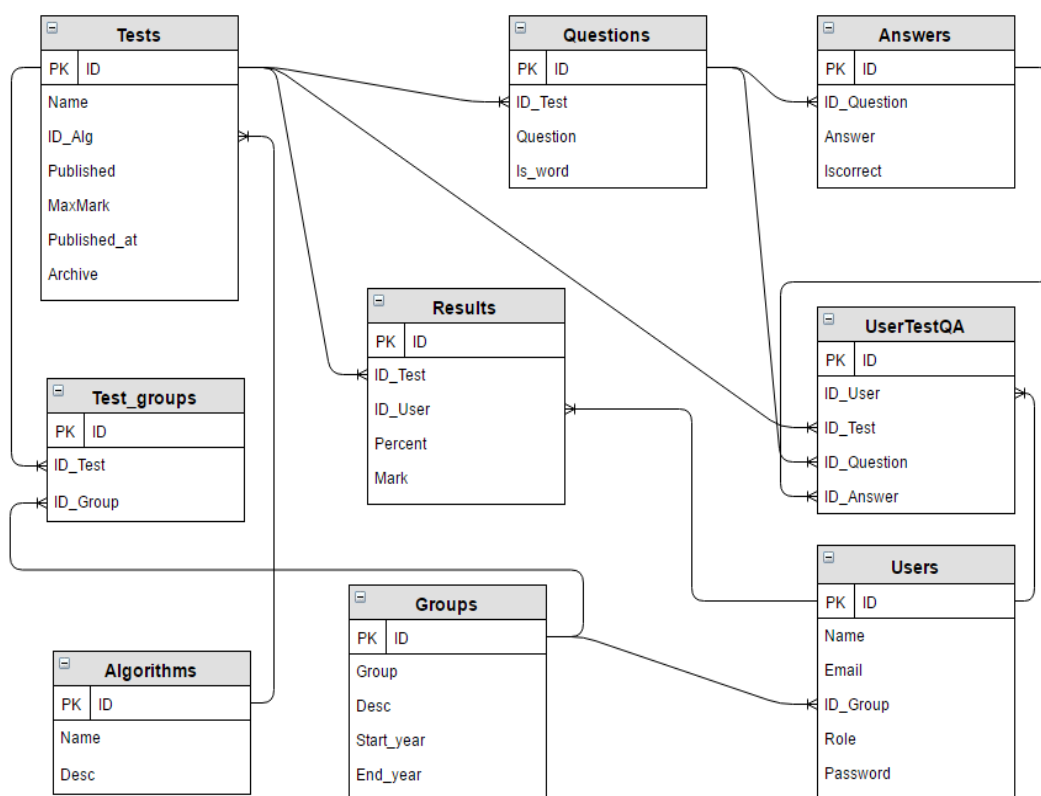


Рисунок 2.7 – ER-диаграмма проектируемой области

Как видно по схеме, таблицы БД будут отображать 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 теста, дата публикации, дата окончания публикации, факт публикации, время на прохождение теста.

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 следующие 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**

Грамотное моделирование и проектирование структуры и архитектуры, а в частности функционального назначения программного продукта значительно облегчает его реализацию [2, с.27].

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 2.6.

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.

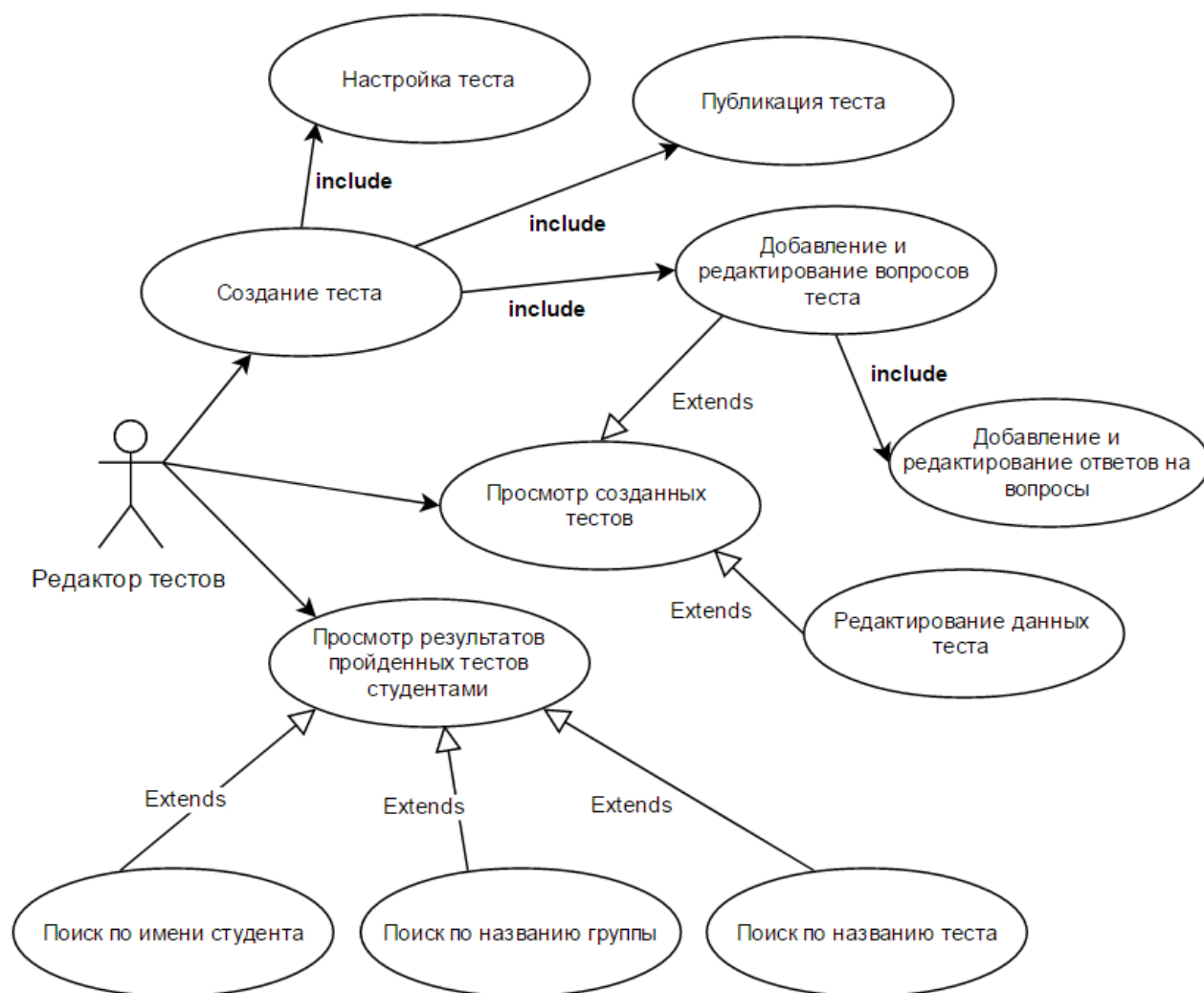


Рисунок 2.6 – Диаграмма вариантов использования приложения редактором тестов

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 2.7, 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.

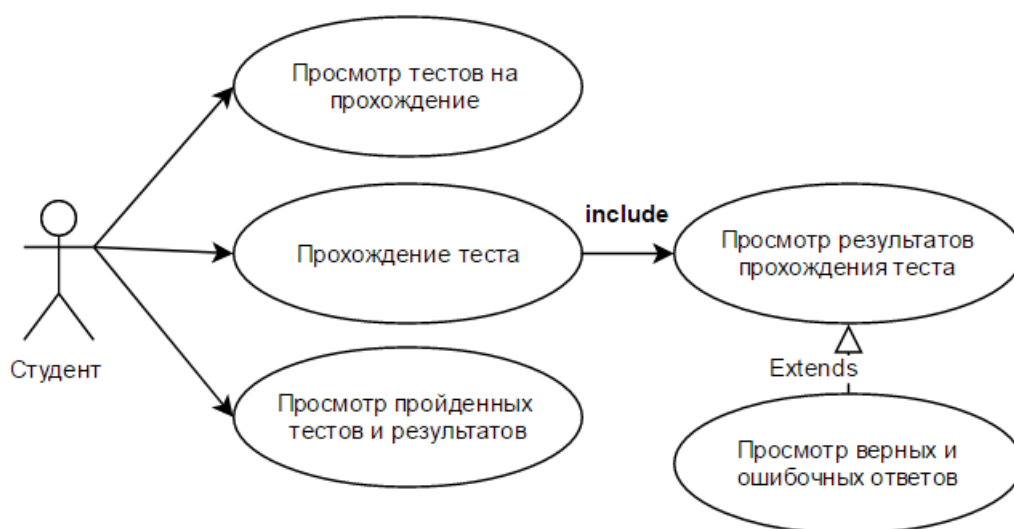


Рисунок 2.7 – Диаграмма вариантов использования приложения студентов

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 2.8 показана диаграмма прецедентов для администратора проектируемого программного продукта.

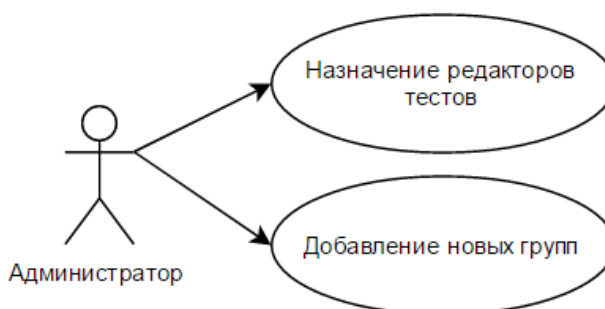


Рисунок 2.8 – Диаграмма вариантов использования приложения администратором

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 к главе 2**

Среди результатов работы над второй главой можно выделить следующие:

- произведен выбор инструментов и технологий для реализации программного продукта;
- приведены основные алгоритмы для решения поставленной задачи, в ходе описания которых были представлены наглядные примеры их работы;
- проработаны архитектура и структура программного продукта;
- разработана логическая структура;
- обозначены функциональные возможности различных групп пользователей разрабатываемого приложения.

По итогу, работа над данной главой позволила спроектировать программный продукт в соответствии со всеми требованиями, представленными в первой главе.

## 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 3.1.

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

Листинг 3.1 – Пример маршрута

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 базу данных с концепциями объектно-ориентированного программирования. В листинге 3.2 представлен пример определения отношения таблиц один ко многим.

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

Листинг 3.2 – Определение отношения один ко многим  
таблиц *Tests* и *Questions*

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 3.3 продемонстрирован пример работы с базой данных на основе функции *getAuthTests* из файла *Tests.php*.

```
public function getAuthTests($id) {
    $tests=$this
        ->with
            (['questions',
              'questions.answers',
              'algorithm'])
        ->where('id_user',$id)
        ->get();
    return $tests;
}
```

Листинг 3.3 – Функция выбора тестов, созданных текущим авторизованным пользователем

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 для обработки результатов 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]);
```

Листинг 3.4 – Функция store контроллера *TestController*

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 3.4,

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;
    }
  }
}
```

Листинг 3.5 – Реализация метода *switchMainView*

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* [3, с.78]. 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 3.6.

```
<i class="sidebar__icon sidebar__icon--blue fa fa-newspaper-o"
  @click.prevent="$parent.$parent.switchMainView
    ('test-publish', { testId: test.id })">
</i>
```

Листинг 3.6 – Пример реализации кнопки, позволяющей изменить основной компонент на странице

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

Вся работа редактора тестов с базой данных происходит без перезагрузки. На листинге ниже представлен пример обмена данными между клиентом и сервером – 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) {}); }
```

Листинг 3.7 – Реализация метода *setupTest*

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 [4, с.112]. После редактирования теста появляется соответствующее уведомление и обновляется информация в боковой панели. Если же тест только создается, то помимо уведомления и обновления боковой панели, 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;
```

Листинг 3.8 – Реализация метода *getFormData*

В данном методе собираются данные формы с помощью *jQuery* метода *serializeArray*, после чего происходит изменение представления данных полей типа *checkbox*. После этого происходит формирование выходного массива, пригодного для преобразования в *JSON* [6, с. 327].

Когда количество опубликованных тестов становится все больше, каждый из которых проходится несколькими группами, просмотр страницы результатов становится все сложнее, нахождение нужного студента или даже нужной группы может занимать большое количество времени. Для недопущения этого, на данной странице предусмотрен поиск с фильтром.

```
onSearch(str){ let self=this; self.b++;  
  switch(parseInt(self.searchIndex)){  
    case 0: self.searchTest(str); break;  
    case 1: self.searchGroup(str); break;  
    case 2: self.searchUser(str); break; } }
```

Листинг 3.9 – Реализация метода *onSearch*

На листинге выше представлен метод *onSearch*. Данный метод на вход получает строку поиска и анализирует, что именно нужно искать. Поле *searchIndex* содержит в себе целочисленное значение из множества {0,1,2}, что соответствует атрибуту *value* тэга *option* элемента *select*.

Если выбран элемент списка со значением *value* равным нулю, то поиск будет вестись по названию теста. Если же значение *value* равно единице, то происходит поиск по названию группы. При значении *value* равном двум поиск ведется по имени пользователя.

Сам процесс поиска для каждого из фильтров аналогичен друг другу. На листинге 3.10 представлен код фильтрации результатов по названию группы.

```
searchGroup(str){  
  let self=this; let re = new RegExp(str, 'i');  
  for (let key in self.modResults){  
    for (let ke in self.modResults[key].groups){  
      let group=self.modResults[key].groups[ke];  
      if (group.group.group.search(re)<0){  
        self.modResults[key].groups[ke].active=false;  
      } else { self.modResults[key].groups[ke].active=true; } } } }
```

Листинг 3.10 – Реализация метода *searchGroup*

На вход данному методу поступает строка, по которой будет вестись фильтр. Данная строка используется в качестве регулярного выражения, и если какая-либо из групп содержит в названии поданную строку, то показываются результаты данной группы по всем тестам. Иначе данные скрываются.

Рассмотрим главные моменты компонента *pass\_id1.vue*, отвечающего за модуль прохождения классического вида тестов.

При прохождении классического вида тестов после загрузки страницы выполняется метод *modifyQuestions*. Код данного метода представлен в листинге, расположенном ниже.

```
modifyQuestions() {
  if (this.tests.shuffle_questions) {
    this.tests.questions=shuffle(this.tests.questions);
  }
  for (var key in this.tests.questions){
    this.tests.questions[key].index=parseInt(key);
    this.tests.questions[key].passed=false;
    for (var k in this.questionsAnswers){
      if (this.questionsAnswers[k].id_question
          == this.tests.questions[key].id){
        this.tests.questions[key].passed=true; } }
    let q = this.tests.questions[key];
    if (this.tests.shuffle_answers){
      q.answers=shuffle(q.answers);
    }
    var answs=this.tests.questions[key].answers;
    for (var k in answs){
      answs[k].checked=false;
    }
  }
}
```

Листинг 3.11 – Реализация метода *modifyQuestions*

Данный метод выполняет сразу несколько функций:

- перемешивание вопросов и ответов при наличии соответствующих настроек теста;
- проставление каждому вопросу свойства *passed*, отвечающего за то, отвечал ли на него уже студент;

– проставление каждому варианту ответа каждого вопроса свойства `checked`, которое отвечает за то, какие варианты ответа выбирал студент при ответе на вопрос.

При ответе на каждый из вопросов в классическом виде теста используется метод `setAnswers`. Этот метод срабатывает при нажатии кнопки ответа на вопрос.

В листинге 3.12 представлено объявление переменных метода `setAnswers`.

```
let self=this;
let idUser=self.$parent.$data.idUser;
let idTest=self.tests.id;
let idQuestion=self.currentQuestion.id;
self.tests.questions[self.currentQuestionNumber].passed=true;
let data={};
```

Листинг 3.12 – Объявление переменных в методе `setAnswers`

В переменную `idUser` записывается идентификатор студента, в `idTest` помещается себе идентификатор теста, в `idQuestion` – идентификатор вопроса. Создаваемый объект `data` предназначен для записи об ответах на вопрос. Также в данном листинге происходит пометка вопроса, на который отвечает студент, как пройденного.

Далее происходит проверка на вид вопроса. Если вопрос содержит варианты ответов, то выполняется код, представленный в листинге 3.13.

```
$('#pass-question-form .pass-answer-item input')
  .each(function(key,val) {
    if ($(this).prop('checked')) {
      self.tests.questions[self.currentQuestionNumber]
        .answers[key].checked=true;
      let answer=$(val).attr('answer-id');
      data={ id_user: idUser, id_test: idTest,
              id_question: idQuestion, id_answer: answer };
      self.$http.post('/api/usertestqa',data)
        .then(function(response) {
        },function(error){ console.log(error.data); });
    }
  });
self.questionsAnswers.push(data);
self.passedCount++;
self.incCurrentQuestionNumber();
```

Листинг 3.13 – Фрагмент кода из метода `setAnswers`



В данном коде представлен цикл, проходящий по всем вариантам ответа. В каждом проходе цикла происходит проверка на то, выбран ли вариант ответа, и, если он выбран, то происходит подготовка и отправка данных на сервер. Также после цикла увеличивается на единицу счетчик *passedCount*, отвечающего за количество пройденных вопросов, и происходит переход к следующему вопросу посредством метода *incCurrentQuestionNumber*.

Если вопрос, на который дается ответ, типа слово/число, тогда выполняется код, представленный на листинге 3.14.

```
data={ id_user: idUser, id_test: idTest,
      id_question: idQuestion,
      id_answer: parseInt
        ($('#pass-question-form [name="id_answer"]').val()),
      answer: ($('#pass-question-form [name="answer"]').val() );
self.$http.post('/api/usertestqa',data).then(function(response){
},function(error){ console.log(error.data); });
self.questionsAnswers.push(data);
self.passedCount++;
self.incCurrentQuestionNumber();
```

Листинг 3.14 – Фрагмент кода из метода *setAnswers*

В этом блоке кода формируется объект с данными и происходит его отправка на сервер. Также, как и в предыдущем варианте, происходит смена текущего вопроса и увеличивается счетчик вопросов, на которые ответы уже даны.

После прохождения теста происходит выполнение метода *onPassed*, код которого представлен в листинге 3.15.

```
var red = '/pass/test/'+this.tests.id+'/result';
let data=this.formData();
redirect(red, {'idResult': this.idResult,'data': data , _token:
  $('#[name="csrf-token"]').attr('content')},'POST'); }
```

Листинг 3.15 – Реализация метода *onPassed*

Данный метод собирает необходимые данные и совершает редирект методом *POST* на страницу с результатом теста.

Для обработки запроса на выдачу страницы с результатами прохождения теста выполняется метод *update* файла *ResultController.php*.

Данный метод производит обработку и выдачу результатов для обоих видов тестов.

В случае, если студент проходил классический тест, то вначале собираются данные об ответах на вопросы, которые заносились в базу данных во время прохождения теста. После получения данных происходит их преобразование, представленное в листинге 3.16.

```
$currentQ=null; $counter=-1;
foreach($qa as $k=>$item){
    $currentCounter=0;
    if ($item['answer']!=null){
        $questionsAnswers[]=['id_question'=> $item['id_question'],
            'answers' => [$currentCounter =>
                ['id_answer' => $item['id_answer'],
                    'answer'=>$item['answer']]
            ]];
        $counter++; continue; }
    if($currentQ!=$item['id_question']){
        $currentQ=$item['id_question'];
        $questionsAnswers[]=[
            'id_question'=> $item['id_question'],
            'answers' => [$currentCounter++ =>
                ['id_answer' => $item['id_answer']]
            ];
        $counter++; }
    else {
        $questionsAnswers[$counter]['answers'][] =
            ['id_answer'=>$item['id_answer']]; } }
```

Листинг 3.16 – Преобразование полученных данных о прохождении теста

Данные преобразовываются в массив *questionsAnswers* с ячейками видов, представленных на рисунках 3.1 и 3.2.

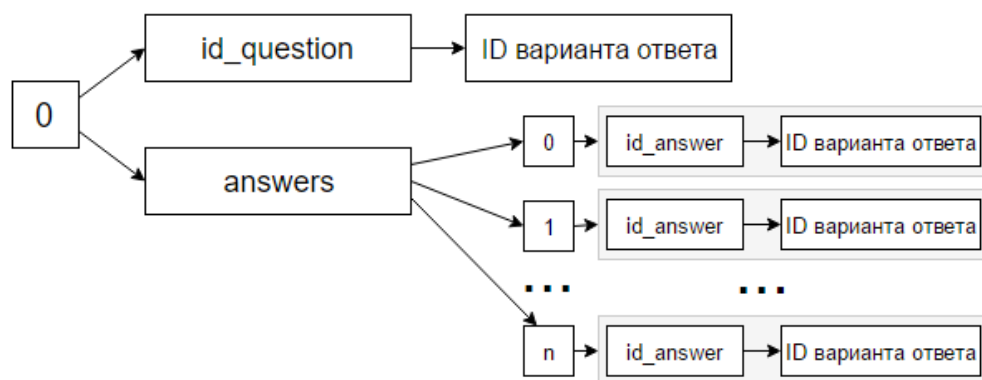


Рисунок 3.1 – Структура ячейки массива с преобразованными данными прохождения теста для вопроса с вариантами ответов

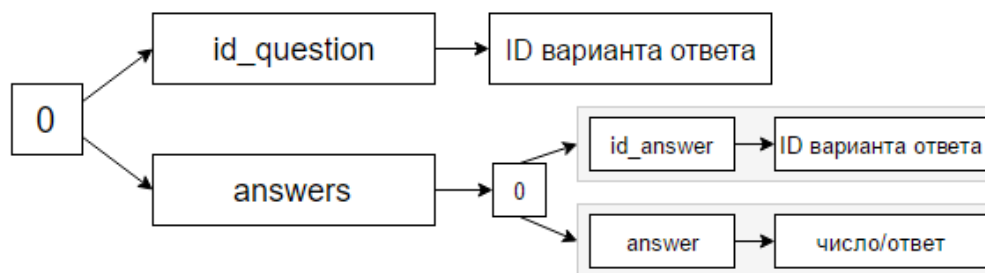


Рисунок 3.2 – Структура ячейки массива с преобразованными данными прохождения теста для вопроса вида число/слово

После приведения к вышепоказанному виду происходит дальнейшее преобразование массива *questionsAnswers*, предназначенное для определения того, правильные ли дал ответы студент.

```

if (strtolower($item['answers'][0]['answer']) ==
    strtolower($questions[$indexQuestion]['answers'][0]['answer'])) {
    $questionsAnswers[$key]['answers'][0]['incorrect']=true;
} else {
    $questionsAnswers[$key]['answers'][0]['incorrect']=false;
    $flag=false; }
  
```

Листинг 3.17 – Проверка ответа на вопрос типа слово/число

В листинге 3.17 представлена проверка правильности ответа студента на вопрос типа слово/число. Стоит отметить, что студенту можно не следить за начертанием прописных и строчных букв. В данном листинге переменная *\$flag* отвечает за правильность ответа.

В листинге 3.18 показана проверка правильности данных ответов студента на вопрос с вариантами ответов.

```

foreach ($item['answers'] as $k=>$idAnswer){
    $indexAnswer=ResultController::getArrayIndex
        ($answers,'id',$idAnswer['id_answer']);
    if (!$answers[$indexAnswer]['incorrect']) {
        $flag=false;
    } else {
        $questionsAnswers[$key]['answers'][$k]['incorrect']=true;} }
foreach($answers as $k=>$answer){
    if ($answer['incorrect']) {
        $x = array_search($answer['id'], array_column($item['answers'],
            'id_answer'));
        if ($x===false) $flag=false;
    }
}
}
  
```

Листинг 3.18 – Проверка ответов на вопрос с вариантами ответов

В данном коде объявлены два цикла. В первом из них проверяются на правильность все варианты ответов, данные пользователем, во втором – что не было упущено каких-либо правильных вариантов ответа.

После проставления правильности всем ответам студента выполняется листинг, представленный в Приложении А. В данном листинге объявлен цикл, перебирающий все вопросы теста, во время которого происходит:

- подсчет веса правильных ответов;
- подсчет веса всех ответов;
- деформирование массива *\$test*, который содержит всю информацию о тесте, заключающееся в представлении информации студенту после прохождения теста о правильных, неправильных и недостающих правильных ответах.

После формирования массива *\$test*, происходит подготовка результатов, а именно результат прохождения в процентах и полученном балле за тест. Также данные добавляются в таблицу с результатами.

```
$percent = round($result['weight']/$result['allWeight']*100);
$mark = round($result['weight']
              /$result['allWeight']
              *$test['maxmark']);
$result ['percent'] = $percent;
$result ['mark'] = $mark;
$res=[
    'id_test' => $data['id_test'],
    'id_user' => Auth::id(),
    'percent' => $percent,
    'mark' => $mark,
    'passed'=>true];
$resultModel->putResult($res,$idUser,$idT);
return
    view('passing.result',
        ['test'=>$test,
         'questionsAnswers'=>$questionsAnswers,
         'result'=>$result]);
```

Листинг 3.19 – Подготовка и выдача результатов прохождения классического вида тестов

После всех подсчетов и работы с БД происходит отправка страницы с данными о результатах и правильности отвеченных вопросов на клиентскую часть.

При прохождении теста-сопоставления интересной является функция *selectAnswer*, отвечающая за выбор ответа на выбранный вопрос.

```
let check=this.checkSelectedAnswer(answer.id);
if (check==1){
    this.questionsAnswers[this.searchQuestionIdIndex
        (this.selectedQuestion)].answers.push(answer);
    this.answersIsActive(answer.id,false);
} else
if (check==0) {
    if (confirm('этот ответ уже был выбран, переставить его?')){
        for (var key in this.questionsAnswers){
            for(var k in this.questionsAnswers[key].answers){
                if (this.questionsAnswers[key].answers[k].id==answer.id){
                    this.questionsAnswers[key].answers.splice(k,1);
                }
            }
        }
        this.questionsAnswers[this.searchQuestionIdIndex
            (this.selectedQuestion)].answers.push(answer);
    }
}
```

Листинг 3.20 – Реализация функции *selectAnswer* компонента *pass\_id2.vue*

В листинге, представленном выше, происходит проверка на то, был ли уже выбран данный ответ. Если он не был выбран, то весь объект ответа *answer* добавляется в массив ответов на вопросы *questionsAnswers*, в ячейку с выбранным текущим вопросом *selectedQuestion*. Если же данный вариант ответа уже был выбран для другого вопроса, то студенту будет предоставлено всплывающее окно с вопросом, хочет ли он его переставить. При подтверждении перестановки происходит удаление варианта ответа из вопроса, к которому он уже был приписан, и добавление к вопросу, который выбран сейчас.

Обработка результатов для теста-сопоставления по своей сути происходит в одном цикле. Листинг данного цикла представлен в Приложении Б.

В данном листинге происходит перебор всех ответов на вопросы студента,

В каждом проходе данного цикла вначале происходит перебор всех ответов, данных на текущий вопрос студентом. В процессе этого перебора наращивается счетчик количества и веса правильных и неправильных ответов. А также сами ответы помечаются правильными или неправильными и те ответы, которые

должны быть отмечены, но не были отвечены также учитываются для предоставления пользователю информации о его ошибках.

После перебора ответов выясняется, есть ли из них хотя бы один правильный, и если да, то увеличивается счетчик количества и веса вопросов с хотя бы одним правильным ответом.

По завершении этих операций происходит занесение данных в таблицу с ответами пользователя.

В листинге 3.21 представлен код, рассчитывающий итоговое количество баллов и процентное соотношение успеха студента к максимальному, а также эти результаты заносятся в таблицу с результатами.

```
$p=$test['maxmark']/2/$allWeight;  
$M1=$p*$result['atLeastWeight'];  
$Mtrue=$p*$result['trueWeight'];  
$Mfalse=$p/2*$result['falseWeight'];  
$result['markAtLeast']=$M1;  
$result['markTrue']=$Mtrue;  
$result['markFalse']=$Mfalse;  
$result['mark']=round($M1+$Mtrue-$Mfalse);  
$res=['id_test' => $data['id_test'],  
      'id_user' => Auth::id(),  
      'percent' =>  
          round($result['mark']/$test['maxmark']*100),  
      'mark' => $result['mark'],  
      'passed'=>true];  
$resultModel->putResult($res,$idU,$idT);  
return view('passing.result',  
    ['test'=>$test, 'questionsAnswers'=>$questionsAnswers,  
     'result'=>$result, 'res'=>$res]);
```

Листинг 3.21 – Подготовка и выдача результатов прохождения тестов-сопоставлений

После всех расчетов происходит выдача пользователю страницы с данными о результатах прохождения и конкретных ошибках.

### 3.2 Описание пользовательского интерфейса

При разработке программного продукта следует уделить немало времени на его графическую составляющую. Рассмотрим основные моменты пользовательского интерфейса.

При открытии приложения, его пользователь видит страницу приветствия, скриншот которой представлен на рисунке 3.3.

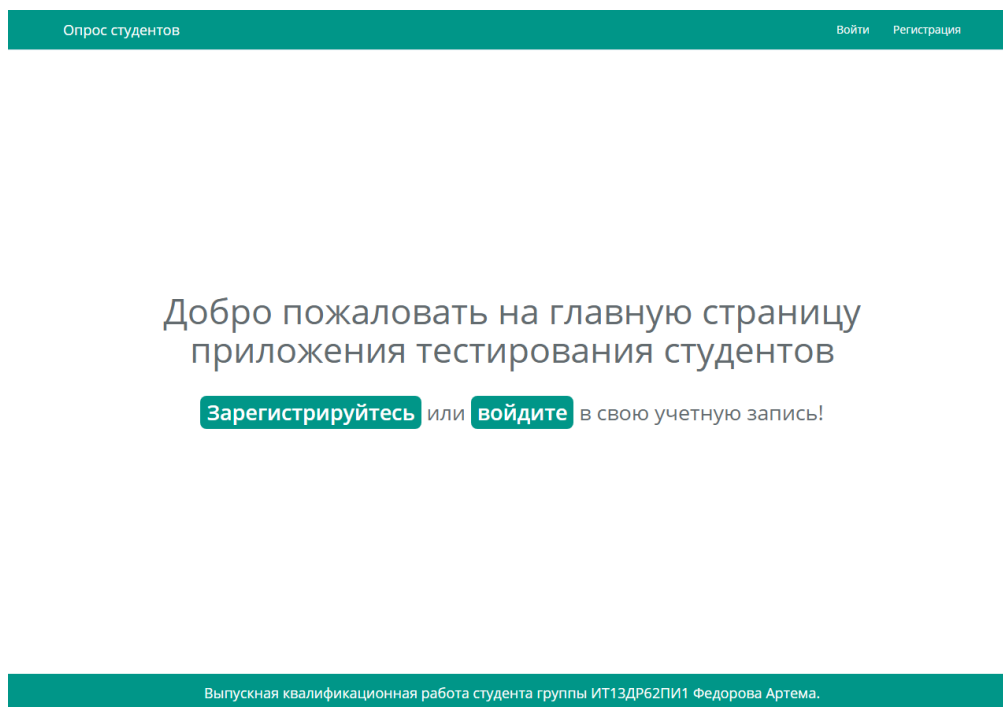


Рисунок 3.3 – Страница приветствия

С данной страницы незарегистрированному пользователю предлагается перейти на страницу входа или регистрации.

Форма регистрации представлена на рисунке 3.4

Рисунок 3.4 – Форма регистрации

Форма входа представлена на рисунке 3.5.

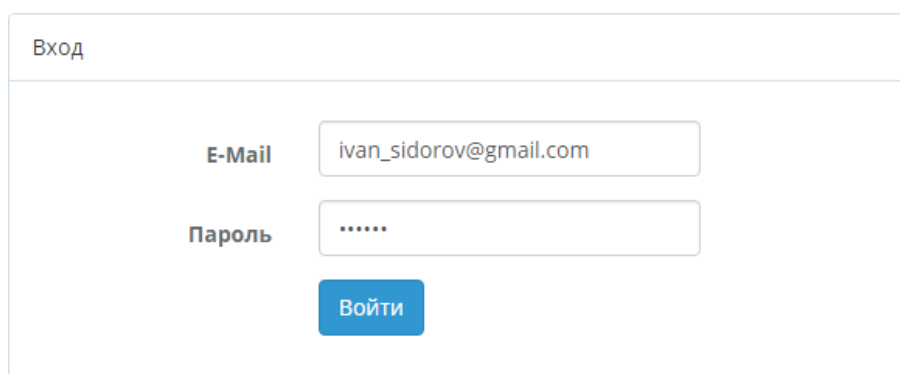


Рисунок 3.5 – Форма входа в приложение

На выше представленных формах все поля реализованы с помощью тега *input*, а выбор группы при регистрации с помощью тега *select*.

Рассмотрим интерфейс редактора тестов. На рисунке 3.6 показана боковая панель главной страницы.

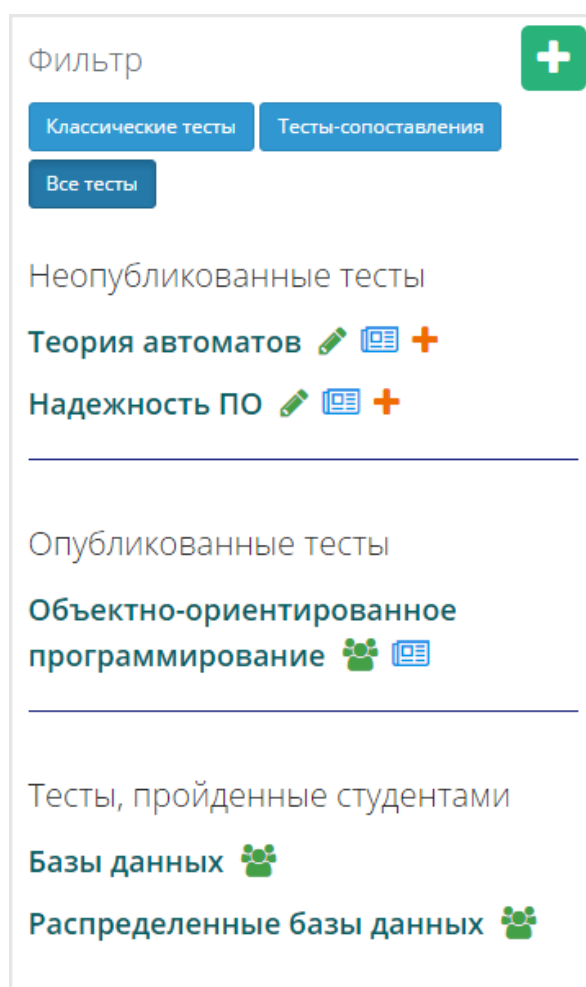


Рисунок 3.6 – Боковая панель редактора тестов



Верхняя часть данной панели содержит фильтр по видам тестов, а также кнопку перехода на компонент создания теста. Основной частью панели является список тестов, разделенный на три части:

- неопубликованные тесты, то есть тесты, которые находятся в разработке;
- опубликованные тесты, то есть тесты, которые уже опубликованы, но еще не пройдены студентами;
- тесты, пройденные студентами.

На рисунке 3.7 показан скриншот, содержащий компонент добавления/изменения теста.

Назад Вперед

### Изменение теста

Название теста: Надежность ПО

Описание теста: Тест для студентов 4 курса кафедры ПОВТ и АС

Категория: Классический ▼

Максимальная оценка (от 1 до 100): 25

Время на прохождение в формате ЧЧ:ММ:СС: 00:20:00

Перемешать порядок вопросов ☒

Перемешать порядок ответов ☒

Показывать, если правильных вариантов ответа больше одного ☐

Позволить произвольное перемещение по вопросам ☐

Показать правильные ответы после прохождения теста ☒

Изменить тест

Добавить вопрос Опубликовать тест Удалить тест

Рисунок 3.7 – Компонент добавления теста

На данном скриншоте представлена форма для работы с описанием и настройками теста, а также кнопки, позволяющие удалить тест, опубликовать его или добавить новый вопрос. Стоит также отметить, что в верхней части компонента располагаются кнопка «Назад» и «Вперед». Действие данных кнопок сравнимо с кнопками перемещения по страницам в браузере, но в отличие от одного, эти кнопки отвечают за перемещение редактора теста по компонентам.

На рисунке 3.8 представлен компонент добавления/изменения вопросов к тесту.


Тест "Надежность ПО" →

Изменение вопроса

В I U Normal Sans Serif

Т<sub>x</sub>

Чем характеризуется надежность ПО?



Вес вопроса (от 1 до 10)

Вид вопроса Варианты ответа ▼

Изменить вопрос

Варианты ответа

<input checked="" type="checkbox"/>	<input type="text" value="устойчивостью к дефектам"/>	<span>Обновить</span>	<span>✕</span>
<input type="checkbox"/>	<input type="text" value="временной эффективностью;"/>	<span>Обновить</span>	<span>✕</span>
<input checked="" type="checkbox"/>	<input type="text" value="восстанавливаемостью"/>	<span>Обновить</span>	<span>✕</span>
<input checked="" type="checkbox"/>	<input type="text"/>	<span>Сохранить</span>	

Добавить следующий вопрос Удалить вопрос

Рисунок 3.8 – Компонент изменения вопроса и ответов теста

Для добавления вопроса используется расширенный редактор, имеющий такие возможности, как изменение параметров шрифта, добавление изображений и т.д. Варианты ответа слева содержат элемент checkbox, позволяющий отметить верные из них, а справа расположены кнопки закрепления изменений и удаления для уже добавленных ответов.

Также стоит отметить в верхней части компонента возможность перемещения между уже добавленными вопросами теста.

При переходе на компонент публикации теста можно увидеть ошибки тесты и предупредительные оповещения (рисунок 3.9).

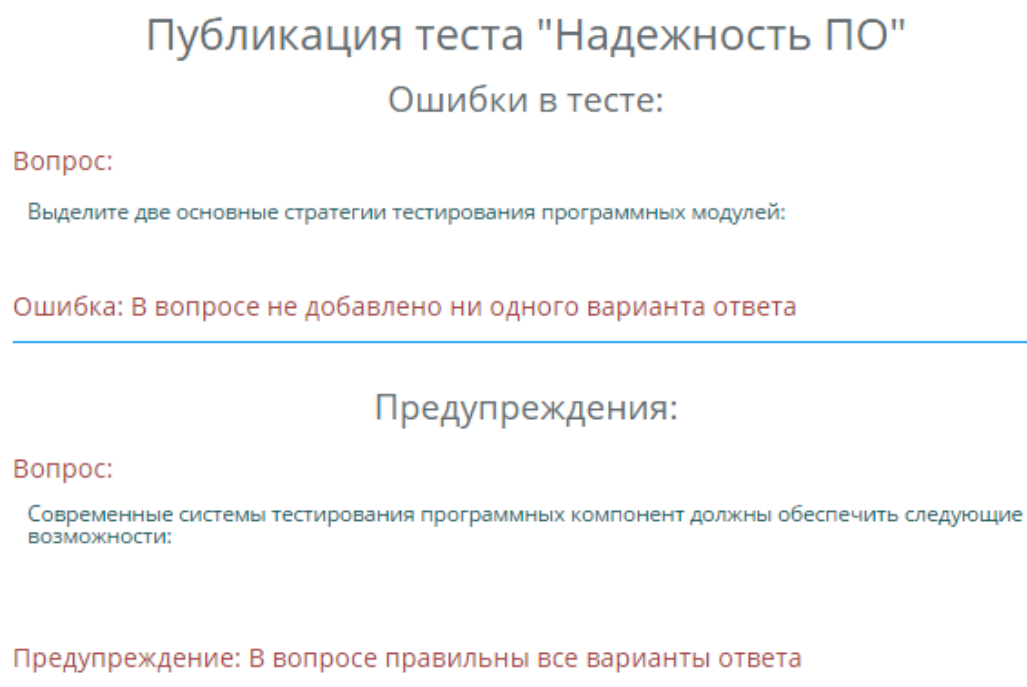


Рисунок 3.9 – Ошибки и предупреждения в компоненте публикации теста

Общими ошибками для обоих разрабатываемых видов тестов являются:

- отсутствие вопросов в тесте;
- отсутствие ответов на какой-либо вопрос.

Индивидуальной ошибкой в классическом виде тестов считается такая ситуация, когда в каком-либо из вопросов не выбрано ни одного правильного варианта ответа.

В компоненте публикации пользователю предлагается проверить информацию о тесте.

Также в данном компоненте можно определить дату начала, окончания публикации и группы, для которых публикуется тест, скриншот представлен на рисунке 3.10.

Публикация: Выбрать дату

Дата публикации (формат ГГГГ-ММ-ДД ЧЧ:ММ:СС): 2017-05-20 15:20:47

Окончание публикации: Выбрать дату

Дата публикации (формат ГГГГ-ММ-ДД ЧЧ:ММ:СС): 2017-05-27 15:20:48

Назначаемые группы: 14ПИ [ОК]

13ПИ 13ИВ 13ИС

Опубликовать

Рисунок 3.10 – Выбор дат публикации и групп в компоненте публикации теста

Тест может быть опубликован как сразу, так и через некоторое время. Стоит отметить, что ведется проверка на устанавливаемую пользователем дату, и если дата публикации будет превышать дату окончания, или же даты будут меньше текущей, то редактор не сможет опубликовать тест, системы выдаст ошибку. Также не получится опубликовать тест, если в нем есть ошибки (предупреждения не влияют на публикацию и несут чисто информативный характер), не выбрав ни одной группы, или если формат даты не верен.

Представление списка пройденных и доступных к прохождению тестов представлен на рисунке 3.11.

**Список пройденных тестов**

Базы данных  
25 баллов из 25 возможных  
100% правильных ответов

**Доступные к прохождению тесты:**

<p>Распределенные базы данных</p> <p>Тест для студентов 4 курса кафедры ПОВТ и АС</p> <p>Пройти тест</p>	<p>Тест по С#</p> <p>Тест для студентов 4 курса кафедры ПОВТ и АС</p> <p>Пройти тест</p>
--	--

Рисунок 3.11 – Списки пройденных и доступных к прохождению тестов для студента

На рисунке 3.12 показано, как представляется информация о тесте перед его прохождением.



Рисунок 3.12 – Информация о тесте перед его прохождением

Интерфейс при прохождении классического теста представлен на рисунке 3.13.

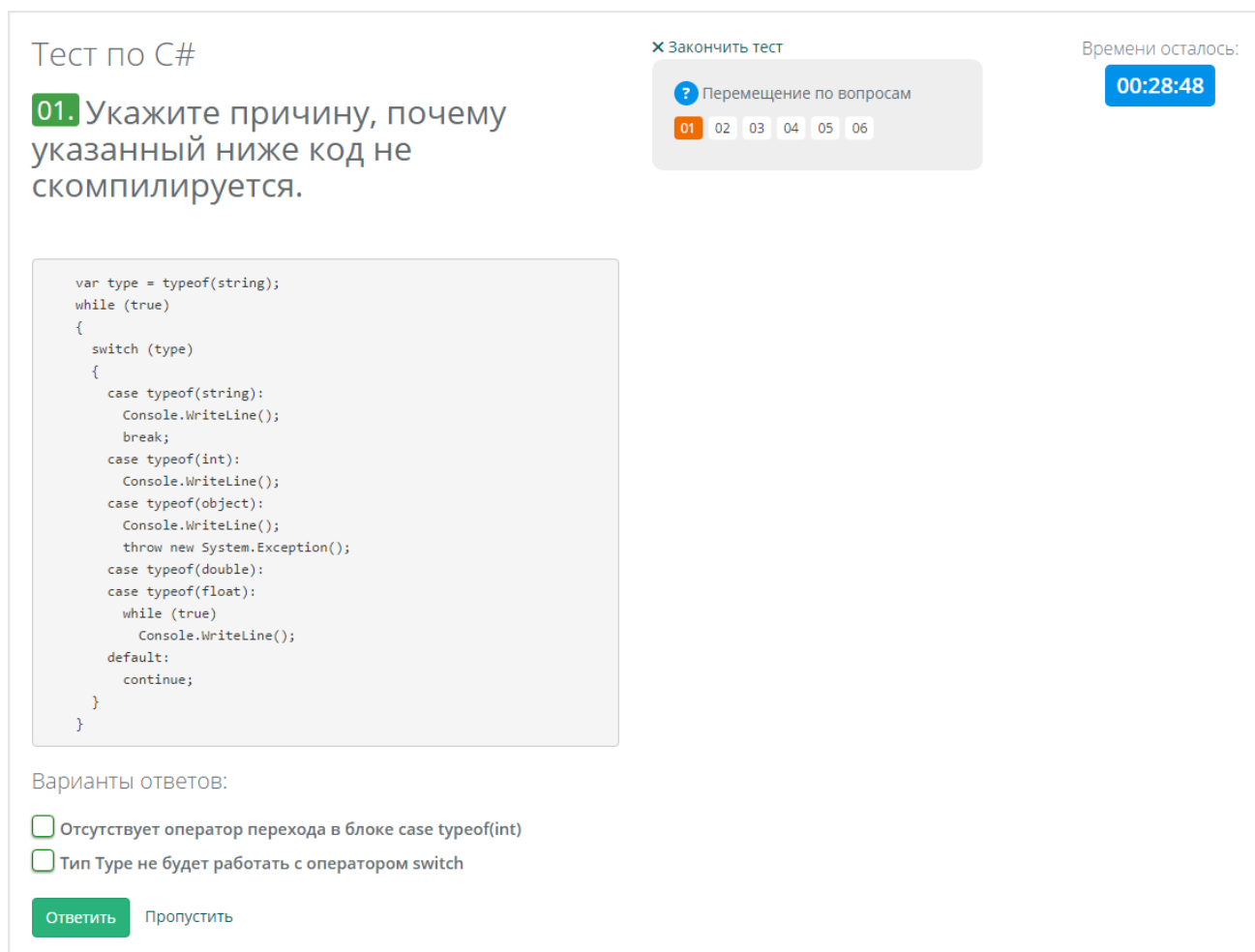


Рисунок 3.13 – Интерфейс при прохождении классического вида тестов

По данному рисунку видно, что в левой части экрана располагается блок с вопросами и вариантами ответов, а справа – блок, позволяющий перемещаться между вопросами, таймер и кнопка, по нажатию на которую происходит преждевременное завершение теста.

На рисунке 3.14 показан интерфейс прохождения тестов-сопоставлений.

Тест "ООП" Времени осталось: 00:44:55

Вопросы

модификаторы класса	10 3 4 7
Конструктор	5 6
Стратегии доступа к свойствам	9 12 14 13 <input type="button" value="OK"/>
По умолчанию модификатор доступа класса	15
Private	16
Свойства - это	1 8
public	2

Ответы

1. интеллектуальные поля 2. доступен вне класса 3. sealed 4. abstract 5. создает объект класса 6. метод 7. protected  
8. Архитектурная единица программы 9. Read, Write 10. new 11. Write-only 12. set 13. Read-only 14. get 15. public 16. Нет правильных ответов

Рисунок 3.14 – Интерфейс прохождения тестов-сопоставлений

Работа с данным интерфейсом подразумевает две операции:

- выбор вопроса, на который нужно ответить, с помощью нажатия на него, после чего данное поле окрашивается в желтоватый цвет;
- проставление желаемого варианта ответа двумя способами: введя его номер в поле, и нажав кнопку «ОК», либо нажатие на сам вариант ответа в нижней части интерфейса.

Страница с результатами прохождения теста имеет интерфейс, показанный на рисунке 3.15.

## Результаты прохождения теста

Вы прошли тест на **88%**

Ваша оценка: **22** (из 25)

Ошибки в тесте:

Вопросы:

1 2 3 4 5 6 7

6 Свойства - это

интеллектуальные поля

Архитектурная единица программы

set

get

[Вернуться к списку тестов на прохождение](#)

Рисунок 3.15 – Интерфейс выдачи результатов

На данном рисунке видно, что на странице результатов присутствует главным образом информация о успешности прохождения теста в процентном соотношении и в баллах, а также кнопка перехода на страницу, содержащую списки всех пройденных и доступных тестов.

В блоке с ошибками тестов красным цветом отмечаются вопросы, на которые студент ответил полностью неправильно, желтым цветом – вопросы, на которые пользователь дал как правильные, так и неправильные ответы (либо не дав неправильных ответов, не поставил нужный), а зеленым цветом помечаются вопросы, на которые студент ответил безошибочно.

### 3.4 Методы и средства защиты программного продукта

Для обеспечения защиты сайта были предусмотрены несколько методов и средств.

Защита от SQL-инъекций производится с помощью привязки параметров к запросам средствами PDO (это освобождает от необходимости экранирования строк перед их подачей в запрос).

Чтобы запретить доступ к данным посторонним лицам, используется аутентификация пользователей с разделением по ролям. При этом используется защита маршрутов как методом перенаправления на главную страницу, так и методом выдачи ошибки при попытке доступа к странице, которая недоступна данной группе пользователей.

Для защиты приложения от межсайтовой подделки запроса используются случайные токены, помещаемые как в сеанс пользователя, так и в формы.

### 3.5 Тестирование и оценка надежности программного продукта

В ходе тестирования программного продукта как при разработке, так и после нее в программном продукте было выявлено множество неисправностей. Рассмотрим некоторые из них.

1. Проходя классический вид теста, не по истечении времени или досрочном завершении теста, если не был дан ответ ни на один вопрос, то в результате выбрасывается исключение, представленное на рисунке 3.16.

1/1 ErrorException in ResultController.php line 56:  
Undefined offset: 0

```
1. in ResultController.php line 56
2. at HandleExceptions->handleError('8', 'Undefined offset: 0', 'C:\xampp\htdocs\diplom.local\app\Http\Controllers\ResultController.php', '56',
  array('request' => object(Request), 'resultModel' => object(Result), 'testQA' => object(UserTestQA), 'testModel' => object(Test), 'data' =>
  array('data' => array(), 'id_test' => '1', 'id_user' => '1', 'id_alg' => '1'), 'idAlg' => '1', 'test' => object(Test), 'testQuestions' => object(Collection),
  'answersQuestions' => array(), 'resTest' => array(), 'resultArray' => array())) in ResultController.php line 56
3. at ResultController->store(object(Request), object(Result), object(UserTestQA), object(Test))
4. at call_user_func_array(array(object(ResultController), 'store'), array(object(Request), object(Result), object(UserTestQA), object(Test))) in
  Controller.php line 55
5. at Controller->callAction('store', array(object(Request), object(Result), object(UserTestQA), object(Test))) in ControllerDispatcher.php line 44
6. at ControllerDispatcher->dispatch(object(Route), object(ResultController), 'store') in Route.php line 190
7. at Route->runController() in Route.php line 144
8. at Route->run(object(Request)) in Router.php line 653
9. at Router->Illuminate\Routing\{closure}(object(Request)) in Pipeline.php line 53
10. at Pipeline->Illuminate\Routing\{closure}(object(Request)) in SubstituteBindings.php line 41
11. at SubstituteBindings->handle(object(Request), object(Closure)) in Pipeline.php line 137
12. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in Pipeline.php line 33
13. at Pipeline->Illuminate\Routing\{closure}(object(Request)) in Authenticate.php line 43
14. at Authenticate->handle(object(Request), object(Closure)) in Pipeline.php line 137
15. at Pipeline->Illuminate\Pipeline\{closure}(object(Request)) in Pipeline.php line 33
```

Рисунок 3.16 – Ошибка при выдаче результатов классического вида теста, во время прохождения которого не было дано ответов

Ошибка была найдена в файле *ResultController.php* методе *update* (листинг 3.22).

```
$currentQuestion = $answersQuestions[0]['id_question'];
```

Листинг 3.22 – Строка с ошибкой при выдаче результатов классического теста



Переменная *\$answersQuestions* содержит массив с данными об ответах пользователя. Соответственно, если пользователь не ответил ни на один вопрос, то количество ячеек массива будет равняться нулю и попытка взять первый элемент такого массива приводит к ошибке [18]. Для исправления этой ошибки нужно добавить проверку длины массива, представленную на листинге 3.23.

```
if (count($answersQuestions)>0) {  
    $currentQuestion = $answersQuestions[0]['id_question']; }  
}
```

Листинг 3.23 – Исправление ошибки при выдаче результатов классического теста

В данном листинге проверяется количество элементов массива *\$answersQuestions*, и первый элемент данного массива берется только в случае, если элементов массива один или больше.

2. При прохождении классического вида теста, если по истечении времени на таймере, или досрочном завершении теста не были даны ответы на все вопросы (рисунок 3.17), то в результате будет показано, что в тесте вопросов ровно столько, сколько было пройдено во время прохождения (рисунок 3.18).

Таймер 00:00:01

Вопрос №3 (из 5)

Что отмечают 31 декабря?

Варианты ответов: (несколько)

☐ День рождения

☒ Новый год

☐ Масленицу

Ответить

Рисунок 3.17 – Прохождение классического вида тестов, дача ответов не на все вопросы

Правильных ответов: 1/2

Рисунок 3.18 – Выдача результатов при прохождении классического вида тестов

Ошибка была найдена в методе *update* файла *ResultController.php* (листинг 3.24).

```
return ['all' => count($resultArray), 'right' => $countRight];
```

Листинг 3.24 – Строка с ошибкой в количестве вопросов теста при выдаче результатов

Ошибка в том, что алгоритм считает количество ответов теста из массива, содержащего ответы пользователя. Чтобы исправить это, нужно вести подсчет количества вопросов на основе массива, содержащего весь тест [17].

Исправленный возвращаемый код представлен в листинге 3.25.

```
$countQuestions=count($testQuestions);  
return ['all' => $countQuestions, 'right' => $countRight];
```

Листинг 3.25 – Исправление ошибки в количестве вопросов теста при выдаче результатов

В данном листинге переменная *\$countQuestions* формируется в зависимости от массива *\$testQuestions*, содержащего все вопросы теста.

3. При публикации классического вида теста не считается за ошибку отсутствие ответа у вопроса вида слово/число. Ситуация воспроизведена на рисунках 3.19 и 3.20.

The screenshot shows a web interface for editing a test question. At the top, there are navigation arrows and the title "Тест 'Название теста'". Below the title is the subtitle "Изменение вопроса". The main editing area contains a rich text editor with a toolbar (bold, italic, underline, link, quote, code, image, list, indent, normal, text color, background color, font family) and a text area containing the question "Чему равен X?" and the formula "4+4=X". Below the editor, there are two input fields: "Вес вопроса (от 1 до 10)" with the value "1", and "Вид вопроса" with a dropdown menu set to "Слово/число". A green button "Изменить вопрос" is located below these fields. At the bottom, there is a section titled "Слово/число" with an empty text input field, a green "Сохранить" button, and two blue buttons: "Добавить следующий вопрос" and "Удалить вопрос".

Рисунок 3.19 – Отсутствие ответа на вопрос вида слово/число.

## Публикация теста "Название теста"

### Предупреждения:

#### Вопрос:

Современные системы тестирования программных компонент должны обеспечить следующие возможности:

Предупреждение: В вопросе правильны все варианты ответа

---

Перед публикацией проверьте данные о тесте еще раз!

Рисунок 3.20 – Отсутствие ошибки об отсутствии ответа на вопрос вида слово/число.

Недоработка обнаружена в компоненте *test\_publish* (листинг 3.26).

```
for (let key in questions){
  if (questions[key].word==0){
    if (questions[key].answers.length>0){
      let count = this.countRightAnswers(questions[key]);
      if (count==0){
        let error = "В вопросе не выбран
          ни один правильный вариант ответа";
        this.addQuestionError(1,questions[key],error);
      } else if (count==questions[key].answers.length){
        let error= "В вопросе правильны все варианты ответа";
        this.addQuestionError(0,questions[key],error);
      }
    } else{
      let error= "В вопросе не добавлено
        ни одного варианта ответа";
      this.addQuestionError(1,questions[key],error);
    } } }
```

Листинг 3.26 – Выдача ошибок в вопросах теста

В данном листинге происходит перебор всех вопросов теста и в каждом проходе цикла идет проверка лишь для вопросов с вариантами ответов [16]. Для исправления недоработки стоит добавить альтернативное условие, когда текущий вопрос имеет тип слово/число (листинг 3.27).

```
else if (questions[key].word==1){
  console.log(questions[key]);
  if (questions[key].answers.length==0){
    let error= "В вопросе не добавлено ни одного варианта ответа";
    this.addQuestionError(1,questions[key],error);
  } }
```

Листинг 3.27 – Исправление выдачи ошибок в вопросах теста

Таким образом, добавлена проверка теста и для вопросов типа слово/число.

4. При публикации теста, если выставить дату окончания публикации меньше, чем дату начала, то публикация теста происходит успешно. Ситуация воспроизведена на рисунке 3.21.

The screenshot shows a web form for publishing a test. It contains the following elements:

- A dropdown menu labeled "Публикация" with the text "Выбрать дату".
- A text input field labeled "Дата публикации (формат ГГГГ-ММ-ДД ЧЧ:ММ:СС):" containing the value "2017-07-02 13:07:50".
- A dropdown menu labeled "Окончание публикации" with the text "Выбрать дату".
- A text input field labeled "Дата публикации (формат ГГГГ-ММ-ДД ЧЧ:ММ:СС):" containing the value "2017-06-29 13:07:49".
- A dropdown menu labeled "Назначаемые группы" with the value "13ИВ" and a blue "OK" button next to it.
- A green button labeled "13ПИ" below the group selection.
- A large green button labeled "Опубликовать" at the bottom.

Рисунок 3.21 – Дата начала публикации позже даты ее окончания

Ошибка возникает в результате отсутствия соответствующей проверки сравнения дат.

```
if (makeDateFromString(this.dateEnd) <
    makeDateFromString(this.dateStart))
{
    alert('Дата окончания публикации раньше чем дата начала');
    return false;
}
```

Листинг 3.28 – Ошибка сравнения дат

Для правки ошибки требовалась проверка, представленная в листинге выше.

5. Среди найденных недоработок важной можно выделить то, что при отсутствии настройки теста «произвольное перемещение по вопросам» доступна функция пропуска ответов (рисунок 3.22).

Название теста

✕ Закончить тест

Времени осталось:

00:19:49

04 Текст вопроса

Варианты ответов:

☐ Вариант ответа №2

☐ Вариант ответа №3

☐ Вариант ответа №1

Ответить

Пропустить

Рисунок 3.22 – Наличие кнопки пропустить при отсутствии настройки «произвольное перемещение по вопросам»

Решение проблемы представлено в листинге 3.29.

```
<a class="btn-alternative"
v-if="tests.pass_other_questions"
@click="incCurrentQuestionNumber">
    Пропустить
</a>
```

Листинг 3.29 – Добавление директивы *v-if* для сокрытия кнопки «пропустить»

Для своевременного сокрытия кнопки используется директива *v-if*.

### 3.6 Расчет себестоимости от внедрения результатов ВКРБ

### 3.7 Охрана труда

### 3.8 Выводы

## ЗАКЛЮЧЕНИЕ

В ходе работы над выпускной квалификационной работой были выполнены следующие действия:

1. Проанализированы существующие виды тестов и предметная область по заданной теме.

2. Предъявлены требования к формированию тестов и их применению для тестирования студентов.

3. Спроектирован программный продукт

4. Разработан программный продукт для проведения тестирования двух различных видов:

- классическое тестирования;
- тесты-сопоставления.

5. Проведено тестирование программного продукта. Тестирование проводилось как лично автором, так и посторонними людьми. Результаты тестирования подтвердили работоспособность программы.

## СПИСОК ЛИТЕРАТУРЫ

1. Астахова И. Ф. SQL в примерах и задачах: Учеб. Пособие / И.Ф. Астахова, В.М. Мельников. – Мн.: Новое знание, 2002. – 176 с.
2. Дж. Рамбо, М. Блаха. Объектно-ориентированное моделирование и разработка. – СПб.: Питер. 2007. – 544 с.
3. Дэвид Мак Фарланд. JavaScript. Подробное руководство. Изд.:Эксмо. 2009. – 608 с.
4. Дэвид Мак Фарланд. JavaScript. Сильные стороны. СПб.:Питер. 2012. – 176 с.
5. Зандстра М. PHP. Объекты, шаблоны и методики программирования. Изд.: Вильямс. 2011. – 560с.
6. Колесниченко Д. PHP и MySQL. Разработка Web-приложений. СПб.: БХВПетербург, 2015. – 593с.
7. К. Дж. Дейт. Введение в системы баз данных, 8-е издание. – М.: Вильямс. 2017. – 1328 с.
8. Равен Дж. Педагогическое тестирование: Проблемы, заблуждения, перспективы. Пер. с англ. - М.: «Когито-Центр», 1999. - 144 с.
9. Томас Х. Кормен. Алгоритмы. Вводный курс. – Москва: Вильямс. 2016. – 208 с.
10. Шапошников И.Ф. Web-сервисы Microsoft .NET. СПб.: БХВПетербург, 2002. – 327с.
11. Груздева М.Л., Кошелев И.А. Педагогическое оценивание результатов образовательного процесса в вузе // Современные наукоемкие технологии. – 2015. – № 12–1. – С. 70–72.
12. Крепова С.Н. Тестирование как форма организации и контроля самостоятельной работы студентов // Вестник ААЭП. – 2010. – № 15. – С. 111–113.
13. Кручинина Г.А., Дарьенкова Н.Н. Применение информационных и коммуникационных технологий в творческой деятельности студентов технического вуза / Приволжский научный журнал. – 2015. – № 1 (33). – С. 193–199.

14. Маматова О. Г. Формы контроля знаний студентов педагогических вузов // Молодой ученый. — 2012. — №8. — С. 353-355.

15. Laravel [Электронный ресурс] – Режим доступа:  
<https://www.laravel.ru/docs/v5>

16. Learn.javascript.ru [Электронный ресурс] – Режим доступа:  
<http://learn.javascript.ru/>

17. PHP [Электронный ресурс] – Режим доступа:  
<http://www.php.net/manual/ru/>

18. StackOverflow [Электронный ресурс] – Режим доступа:  
<http://www.stackoverflow.com>

19. Vue.js [Электронный ресурс] – Режим доступа:  
<https://www.ru.vuejs.org/v2/guide>

20. Wikipedia [Электронный ресурс] – Режим доступа:  
<https://www.ru.wikipedia.org>

21. Мастер-тест [Электронный ресурс] – Режим доступа:  
<http://master-test.net>



**Листинг алгоритма обработки классических тестов**

```

foreach ($test['questions'] as $key=>$question){
    $test['questions'][$key]['weightAnswer']=0;
    $countRightAnswers=0;
    $countUserRightAnswers=0;
    $curQ=ResultController::getArrayIndex
($questionsAnswers,'id_question',$question['id']);
    foreach ($test['questions'][$key]['answers'] as
        $k => $answer) {
        $f=false;
        if($answer['incorrect']) $countRightAnswers++;
        if ($curQ>-1) {
            foreach ($questionsAnswers[$curQ]['answers'] as
                $kq => $answerq) {
                if ($answer['id'] == $answerq['id_answer']){
                    if (array_key_exists('answer',$answerq)){
                        if ($answerq['incorrect']){
                            $test['questions'][$key]
                                ['answers'][$k]['right'] = 2;
                            $f = true; break;
                        } else {
                            $test['questions'][$key]
                                ['answers'][$k]['right'] = 1;
                            $test['questions'][$key]['answers'][] =
                                $answerq;
                            $test['questions'][$key]['answers']
                                [count($test['questions']
                                    [$key]['answers'])-1]['right']=0;
                            break; }
                        } else
                            if ($answer['incorrect']) {
                                $test['questions'][$key]['answers'][$k]['right'] = 2;
                                $f = true;
                                break;
                            } else if (!$answer['incorrect']) {
                                $test['questions'][$key]['answers'][$k]['right'] = 1;
                                $f = false;
                                break; }
                    } } }
                if (!$f&&$answer['incorrect'])
                    $test['questions'][$key]['answers'][$k]['right']=0;
                if ($f) $countUserRightAnswers++;
            }
            $result['allWeight']+=$question['weight'];
            if ($countRightAnswers==$countUserRightAnswers){
                $result['weight']+=$question['weight'];
            }
            $test['questions'][$key]['weightAnswer']=
                $question['weight']/$countRightAnswers; }

```

## Листинг обработки результатов для тестов-сопоставлений

```

foreach ($questionsAnswers as $key => $value) {
    $questionId = $value['question'];
    $questionsAnswers[$key]['question'] = $allQuestions[$key];
    $answers = $value['answers'];
    $countRight = ResultController::checkCountRightAnswers
        ($allAnswers, $questionId);
    $equivalentAnswers = 0;
    $count = count($answers);
    if ($count > 0) {
        foreach ($allAnswers as $k => $answer) {
            $f = false;
            for ($i = 0; $i < $count; $i++) {
                if ($questionsAnswers[$key]['answers'][$i]['id']!=0) {
                    if ($answers[$i]['id'] == $answer['id'] &&
                        $questionId == $answer['id_question']) {
                        $equivalentAnswers++;
                        $f = true;
                        $questionsAnswers[$key]['answers'][$i]['right']
                            = 2;
                        $result['countRight']++;
                        $result['trueWeight'] +=
                            $allQuestions[$key]['weightAnswer'];
                        continue;
                    } else if ($answers[$i]['id'] == $answer['id'] &&
                        $questionId != $answer['id_question']) {
                        $questionsAnswers[$key]['answers'][$i]['right']=1;
                        $result['countFalse']++;
                        $result['falseWeight'] +=
                            $allQuestions[$key]['weightAnswer'];
                        $f = true; continue;
                    }
                }
            }
        } else {
            if (!array_key_exists('right',
                $questionsAnswers[$key]['answers'][$i])) {
                if ($countRight == 0) {
                    $equivalentAnswers++;
                    $f = true;
                    $questionsAnswers[$key]['answers'][$i]['right']
                        = 2;
                    $result['trueWeight'] +=
                        $allQuestions[$key]['weightAnswer'];
                } else {
                    $questionsAnswers[$key]['answers'][$i]['right']
                        = 1;
                    $result['countFalse']++;
                    $result['falseWeight'] +=
                        $allQuestions[$key]['weightAnswer'];
                }
            }
        }
        continue;
    }
}

```

```

    }
  }
}
if ($questionId == $answer['id_question'] && !$f
    && ResultController::getArrayIndex($answers, 'id',
    $answer['id']) === -1) {
    $questionsAnswers[$key]['answers']
        [count($questionsAnswers[$key]['answers'])]
        = $answer;
    $questionsAnswers[$key]['answers']
        [count($questionsAnswers[$key]['answers'])-1]
        ['right'] = 0;
    }
}
} else {
    if (count($allQuestions[$key]['answers'])==0) {
        $questionsAnswers[$key]['answers'][count($answers)]
            = ResultController::makeNullableAnswer();
        $questionsAnswers[$key]['answers']
            [count($answers)]['right'] = 0;
    } else {
        foreach ($allQuestions[$key]['answers'] as
            $keyR=>$answerR) {
            $questionsAnswers[$key]['answers']
                [count($questionsAnswers[$key]['answers'])]
                = $answerR;
            $questionsAnswers[$key]['answers']
                [count($questionsAnswers[$key]['answers'])-1]
                ['right'] = 0;
        }
    }
}
}
if ($equivalentAnswers > 0) {
    $result['atLeast1']++;
    $result['atLeastWeight'] += $allQuestions[$key]['weight'];
}
if (!$results['passed']) {
    foreach ($answers as $k => $answer) {
        if ($answer['id'] != 0) {
            $qaItem = [
                'id_test' => $idT,
                'id_user' => $idU,
                'id_answer' => $answer['id'],
                'id_question' => $questionId,
                "created_at" => \Carbon\Carbon::now(),
                "updated_at" => \Carbon\Carbon::now()
            ];
            $testQA->insert($qaItem);
        }
    }
}
}
}

```