Министерство образования и науки Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Робототехника и комплексная автоматизация
КАФЕДРА Системы автоматизированного проектирования

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

Исследование и разработка методов гибкой оценки результатов тестирования

Студент РК6-41М _____ Собенин Г.И.
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР _____ Берчун Ю.В.
(Подпись, дата) (И.О.Фамилия)

Нормоконтролер _____ Грошев С.В.
(Подпись, дата) (И.О.Фамилия)

2021 г.

РЕФЕРАТ

Расчетно-пояснительная записка к выпускной квалификационной работе магистра «Исследование и разработка методов гибкой оценки результатов тестирования» содержит 47 страниц машинописного текста, 3 рисунка.

Целью магистерской квалификационной работы является исследование различных методов оценки знаний, их применимости и способов оценки результатов, а также разработка программного средства автоматизированного тестирования студентов, позволяющего упростить работу преподавателя за счёт оценки сложности вопроса и проверки ответа студента.

В данной работе рассмотрены вопросы, связанные с проведением оценки знаний. Также проанализированы различные программные средства, позволяющие производить тестирование обучающихся. На основе данного анализа написано веб-приложение, позволяющее как создавать тесты, так и проходить их. Система была апробирована на группе студентов. В научно-исследовательской части работы проведен анализ различных методов оценки знаний, а также предложен алгоритм автоматизации оценки результатов тестирования.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Для обеспечения системы образования достоверной информацией о соответствии процессов и результатов образования нормативным требованиям, проводится сбор данных о наиболее значимых характеристиках качества образования, их обработка, анализ и интерпретация. Полученная при анализе информация помогает выявлять проблемы в обучении, основными причинами которых могут быть недостатки методов преподавания, неверно составленные учебные планы или несоответствие используемых учебных пособий. Для достоверности полученной информации важно использовать качественные инструменты или методы контроля знаний учащихся.

В настоящий момент в образовании существуют разнообразные формы и методы контроля: устные опросы, письменные контрольные, рефераты, курсовые, лабораторные и проектные работы, коллоквиумы, семинары, зачеты и экзамены [1]. Каждая из форм имеет свои особенности, достоинства и недостатки. К примеру, на традиционном экзамене фактические знания оцениваются по умению их преподнести. Здесь сказывается уровень развития устной речи, словарный запас, общий кругозор, коммуникабельность, стрессоустойчивость и многие другие личностные качества учащегося. Кроме того, влияние оказывает и субъективное мнение преподавателя о работоспособности студента в течение учебного семестра. Не следует исключать и случайность выбора теоретических вопросов и практических задач на экзамене.

В 20 веке к этим формам добавились тестовые методы, которые активно применяются в разных странах. С развитием информационных технологий тесты приобрели широкую популярность. Актуальность применения тестирования обусловлена, во-первых, необходимостью повышения эффективности использования времени, выделенного на обучение, во-вторых, высокой унификацией процесса оценки знаний и умений объекта тестирования и, в-третьих, уменьшением субъективности этой оценки. Особенно часто этот метод

оценки знаний используется в дистанционных системах образования, к примеру, в системе Moodle [2].

В вузах на сегодняшний день все более востребованными средствами оценивания знаний студентов являются тесты. В отличие от традиционных методов оценки, именно тесты имеют научно обоснованные критерии качества. Кроме того, тесты обладают очевидными преимуществами перед другими методами педагогического контроля [3]. В первую очередь, это эффективность и мобильность контроля при помощи емких тестовых заданий, что выражается в минимальных затратах времени, усилий и средств, при этом индивидуализация контроля позволяет справедливо оценить каждого отдельного испытуемого по единым критериям. Однако, одним из сложных и противоречивых вопросов при проведении тестирования является проблема оценивания знаний [4]. Решению данной проблемы и посвящена данная работа.

1. НАУЧНО-ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

1.1. Обзор методов оценки знаний

В настоящее время существует достаточно большое число различных методик проведения оценки знаний. Все из них имеют свои достоинства и недостатки, однако большая часть из них являются плохо формализуемыми (а, следовательно, и автоматизируемыми) – к примеру, можно перенести написание экзаменов в цифровую среду, однако проверка работ остаётся за преподавателем. Поэтому в данной работе рассматриваются методы, позволяющие уменьшить нагрузку на персонал ВУЗов.

1.1.1. Тесты

Рассмотрим один из самых популярных методов контроля знаний учащихся - тесты. В результате тестирования получают совокупную информацию о результатах обучения. Но для выяснения степени прогресса при обучении и для того, чтобы была возможность при необходимости внести своевременные коррективы, необходим систематический контроль обучения на различных уровнях. Этот вид тестирования необходим преподавателю для того, чтобы иметь информацию об уровне усвоения различных тем учебного материала.

К факторам, влияющим на результаты тестирования, относятся следующие умения учащегося: способность четко выполнять задания, понимать их формулировку, грамотно оформлять свою работу, сосредоточенность и внимание, так как некоторые тесты не дают возможности исправлять допущенные ошибки [5].

1.1.2. Эссе

Другой достаточно эффективной формой контроля и проверки знаний учащегося, активизирующей учебный процесс и позволяющей оценить его способности по анализу, синтезу и творческому применению знаний, является эссе, которое в настоящее время ещё не получило широкого распространения

при изучении технических наук [6]. Эссе — это сочинение-рассуждение небольшого объема со свободной композицией, выражающее индивидуальные соображения по конкретному вопросу, проблеме, которое дает возможность творческого описания материала. В отличие от других форм контроля и проверки знаний, эссе позволяет диагностировать продуктивность творческой составляющей познавательной деятельности обучающихся, заключающейся в анализе информации, её интерпретации, построении рассуждений, сравнении фактов, в формулировке выводов и т.п.

В настоящее время разработано и предлагается для использования большое количество программных средств, предназначенных для компьютерного тестирования, таких как SharePointLMS [7] и упомянутый ранее Moodle. Большинство из них имеют режим ввода ответа на естественном языке, так называемый – «ответ в форме эссе», оценка полноты и правильности на который возлагается на экзаменатора. Но, как правило, этот режим не применяют, поскольку оценка ответа тестируемого, во-первых, требует участия преподавателя, во-вторых, по своей сути сводит такую форму тестирования к письменному экзамену, но без возможности задать студенту уточняющие вопросы для выяснения глубины знаний испытуемого и, в-третьих, повышает уровень субъективности оценки. Поэтому поиск путей автоматизированной оценки знаний студента, выражаемых естественным языком, с минимальным привлечением преподавателя является актуальной задачей. Один из путей решения этой задачи напрямую связан с автоматизированным выявлением и оценкой смыслового (семантического) содержания ответа на тестовый вопрос.

Известные варианты решения этой задачи используют разные подходы и не лишены недостатков. В частности, предлагается использовать статистические подходы на основе векторно-пространственной модели текста [8]. В этой модели текст представляется вектором частот входящих в него слов, а оценка близости текстов определяется косинусом угла между векторами текстов. Такой подход не учитывает содержательные морфологические и синтаксические характеристики естественного языка, посредством которых и проявляются

семантические связи объектов. Другая методика при оценке семантического подобия текстов использует вероятностно-статистическую модель представления текста, в основу которой положен некоторый эталонный текст [9]. Используя аппарат теории информации, оценивается количество общей информации в текстах и количество информации, отличающей сравниваемые тексты. Указанные подходы ориентированы на реализацию компьютерной обработки, в основном атрибутивной внешней стороны документов, и позволяют лишь поверхностно оценить семантическое содержание текстов без определения их семантической, ассоциативной и понятийной связности. Поэтому практически невозможно использовать эти подходы для решения основной задачи экзамена или зачета – оценка уровня теоретических знаний обучаемых.

1.1.3. Формальные онтологии

Ещё одной формой контроля знаний обучающихся является тестирование с применением формальных онтологий [10]. Первоначальным шагом является формирование онтологии понятий из предметной области, в которой планируется тестирование. То есть необходимо ввести семантическую сеть (СС) онтологии О, которую определяет кортеж

$$SS = \langle C, R, W_C, W_R \rangle,$$

где $C = \{c_i\}$ - множество понятий СС;

$R = \{r_i\}$ - множество отношений между понятиями набора С;

$W_C = \{w_{C,i}\}$, $W_R = \{w_{R,i}\}$ - значения мер важности понятий набора C и отношений R соответственно.

Затем вводятся модели локальной когнитивной карты обучающегося (ККО) и расширенной ККО. Тест состоит из локальных тестовых заданий. Степень покрытия тестом семантической сети можно оценить как

$$n_i(h) = \frac{|T_i(h)|}{|C_i(h)|} \in (0; 1].$$

Оценку тестирования при использовании онтологий проводят с помощью следующих метрик:

- метрика $\rho_1(c_i)$ - оценивается соотношение множеств ответов к множествам онтологий (как понятий, так и отношений). Метрика не зависит от сложности таковых в онтологии;
- метрика $\rho_2(c_i)$ - представляет собой метрику $\rho_1(c_i)$, но учитывается сложность понятий и важность их отношений;
- метрики $\rho_3(c_i, h)$ и $\rho_4(c_i, h)$ - аналогичны метрикам $\rho_1(c_i)$ и $\rho_2(c_i)$, но дополнительно оценивается фрагмент СС с расстоянием между понятиями, превышающем 1.

К достоинствам данного подхода можно отнести формализованность метода, возможность легко оценить степень покрытия предметной области. Недостатками использования онтологического подхода являются ограниченный формат тестов и необходимость ручного составления онтологии.

1.2. Оценка качества теста

Важным аспектом выбора метода или инструмента контроля знаний является его техническое качество. Чем лучше качество инструмента, тем полезнее он будет, тем больше уверенность в оценках и тем больше уверенность в принятии решений на основе этих результатов. Поэтому крайне важно использовать высококачественные инструменты для проведения контроля знаний. Двумя основными элементами, которые определяют качество теста, являются его достоверность и надежность [1].

Достоверность теста отражает его способность получать результаты, соответствующие поставленной цели, и обосновывающая адекватность принимаемых решений (определение уровня знаний студента). Проверить достоверность теста можно несколькими способами.

Оценка достоверности теста может носить количественный и качественный характер. Достоверность теста, полученную на основании экспертных оценок, невозможно измерить: она специфична для каждого конкретного случая использования и выражается такими степенями, как высокая, средняя и низкая. Качественная оценка производится описательно, на основании логических выводов.

Для вычисления количественного показателя – коэффициента валидности − сопоставляются результаты, полученные при применении диагностической методики. Используются разные виды линейной корреляции (по Спирмену, по Пирсону).

Надежность отражает точность и корректность изучаемого явления. Тест называется надежным, если он дает одни и те же показатели при повторном тестировании. Качественный тест не может быть создан без тщательного изучения этого важного аспекта измерения. Использование ненадёжных тестов, может стать причиной педагогических и административных ошибок, последствия которых трудно исправить.

Н. Гронлунд отмечает: «Тесты по оценке результатов должны быть надёжными, и, в связи с этим, их обработка должна осуществляться очень тщательно. Если балл, полученный учеником в результате теста по оценке результатов, будет соответствовать той оценке, которую он получил бы при повторном прохождении того же теста или идентичного с ним по форме, то данная оценка считается высоко надежной. Чем длиннее тест, тем более надежными и адекватными будут результаты» [11]. Исходя из этого утверждения, можно сделать вывод, что надежность возрастает с величиной теста. Поскольку истинные показатели определены как показатели генеральной совокупности заданий, должно выполняться предположение, что чем больше величина теста, тем выше корреляция с истинным показателем. Таким образом, если преподаватель сможет сформулировать большой набор однородных заданий, это позволит ему создать надежный тест.

1.3. Методы оценки результатов тестирования

Одним из сложных и противоречивых вопросов при проведении тестирования является проблема оценивания знаний. Самым распространенным способом решения данной проблемы является использование дихотомической системы оценивания тестовых заданий, в которой за каждое задание можно получить 0 или 1 балл. Данная система удобна при оценивании заданий с выбором одного правильного ответа из многих, т.е. заданий закрытого типа. В то же время существует многообразие типов тестовых заданий: закрытые (многоальтернативные и одноальтернативные), открытые, на установление соответствия между элементами, на установление правильной последовательности, ситуационные тестовые задания. Дихотомическую систему оценки выполнения целесообразнее применять, если тест содержит большое количество заданий и на его выполнение отводится достаточно много времени. В педагогической практике данную систему оценки используют при итоговом контроле знаний.

Политомическую систему оценки используется для тестов, которые контролируют понимание каких-то крупных элементов содержания − (определений, событий, фактов, процессов и т.п.), и эти элементы могут быть структурированы или разбиты на более мелкие части. В тестах для текущего и тематического контроля тест может состоять только из нескольких заданий с политомической оценкой [1].

Выбор системы оценки прежде всего зависит от цели тестирования и от типа и вида формы тестового задания. Объективность и качественность оценивания результатов достигаются за счет стандартизации процедуры тестирования, а также стандартизации и проверки показателей качеств заданий и тестов в целом.

Однако, после получения оценки результатов тестирования требуется понять, как интерпретировать полученные результаты. В настоящее время существует два основных подхода [12].

Нормативно-ориентированный подход позволяет сравнивать учебные достижения отдельных учащихся друг с другом. Критериально-ориентированный подход позволяет оценивать, в какой степени студенты овладели необходимым учебным материалом.

Подход к интерпретации тестового балла является основным критерием для разделения тестов на критериально- и нормативно-ориентированные, различающиеся по методам конструирования и особенностям применения.

Система оценивания результатов тестирования является сильно субъективной и у разных преподавателей разные подходы к ней. К примеру, одни считают, что отличную оценку можно ставить только в том случае, когда испытуемый справился со всеми заданиями, а другие – что для получения положительной оценки достаточно ответить более чем на половину заданий в тесте.

В этой связи следует отметить, что педагогические тесты имеют основания для сравнения. Для критериально-ориентированных тестов – это полученный на основе экспертных оценок критерий значимости, превысив который, как считается, тестируемый справился, с тестированием. Для нормативно-ориентированных тестов основанием для сравнения служат статистические нормы.

1.4. Предлагаемая реализация метода тестирования

1.4.1. Требования к реализации

При определении тематики данной работы был поставлен ряд функциональных требований как к вырабатываемой методике анализа результатов тестирования, так и к программной реализации:

- вся работа с тестами должна проходить в веб-браузере;
- должна присутствовать функциональность создания и прохождения тестов;
- тесты должны состоять из нескольких заданий с выбором одного или нескольких ответов;
- при прохождении теста каждое задание находится на отдельной странице, при этом остаётся возможность навигации между заданиями;
- система должна автоматически определять сложность заданий;
- система должна автоматически выставлять оценку ответа студента.

1.4.2. Определение сложности задания

Для определения сложности задания и, следовательно, максимального числа баллов за него, было предложено внедрение различных категорий вариантов ответов: правильные, частично правильные, неправильные и полностью неправильные.

Увеличение числа как суммарного числа ответов, так и числа ответов каждой категории приводит к увеличению сложности задания. Было решено оценивать максимальное число баллов за задание путём увеличения базовой стоимости задания S_b. Увеличение стоимости задания за вопросы какого-либо класса не зависит от других классов, поэтому применяется следующая формула:

$$S_0 = k^c(n_0^c) \cdot k^p(n_0^p) \cdot k^i(n_0^i) \cdot k^t(n_0^t) \cdot k^\Sigma(n_0^c, n_0^p, n_0^i, n_0^t) \cdot S_b,$$

где k^c, k^p, k^i, k^t – коэффициенты, зависящие от числа ответов соответствующих классов;

k^Σ – коэффициент, зависящий от суммарного количества вариантов ответов;

S_b зависит от типа задания (выбор одного или нескольких правильных ответов).

Конкретные формулы для получения значений коэффициентов, а также значения S_b приведены в п. 2.2.1.

1.4.3. Оценка ответа на задание

Оценка ответа на вопрос является важнейшим элементом в обработке результатов тестирования. В рамках этой работы было принято решение по возможности отказаться от строго линейных методов оценки. Таким образом, алгоритмом в какой-то мере воспроизводится процесс мышления преподавателя, который, в свою очередь, может как повышать, так и понижать оценку относительно линейных значений правильности ответа в зависимости от выбранных студеном вариантов.

Результатом алгоритма является коэффициент $W \in [0; 1]$, на который домножается максимальное число баллов за задание S_0.

Оценка ответа на вопрос зависит от типа задания. Поскольку для заданий с выбором одного правильного ответа не представляется возможным нелинейно анализировать выбранные варианты ответов, было решено использовать фиксированные значения коэффициента W. В свою очередь, для заданий с выбором нескольких вариантов ответов, такая возможность имеется.

При оценке таких заданий используются значения начальной стоимости задания S_0, количества вариантов ответов различных классов: n_0^c, n_0^p, n_0^i, n_0^t, числа ответов студента различных классов: n^c, n^p, n^i и n^t, а также коэффициента увеличения вклада полностью неправильных вариантов k^t.

Было решено выделять из ответа студента его «правильность» и «неправильность». Для этого он разбивается на положительную и отрицательную компоненты, которые рассчитываются независимо друг от друга.

Положительная компонента W^+ отражает, сколько правильных и частично правильных вариантов ответов были выбраны студентом. Отрицательная компонента W^- в свою очередь показывает, сколько неправильных и полностью неправильных вариантов ответов не были выбраны.

К примеру, в случае наличия только правильных и неправильных вариантов ответов при линейном анализе результата испытуемого определение коэффициента правдивости W сводилось бы к формуле

$$W = \frac{n^c}{n_0^c} \cdot \left(1 - \frac{n_0^i}{n^i}\right),$$

где $\frac{n^c}{n_0^c}$ – положительная компонента ответа;

$1 - \frac{n_0^i}{n^i}$ – отрицательная.

Определённые сложности в процесс оценки работы студента вносят дополнительные классы вариантов ответов – частично правильный и полностью неправильный.

Выбранные частично правильные варианты ответов было решено вносить в положительную компоненту, причём таким образом, чтобы при большом числе выбранных правильных вариантов ответов частично правильные снижали итоговую оценку, а при малом – повышали.

Полностью неправильные варианты ответов вносятся в отрицательную компоненту, при этом имеют больший вклад в неё по сравнению с неправильными ответами, это увеличение задаётся коэффициентом k^t.

Конкретные формулы для расчёта, как обеих компонент ответа, так и итогового коэффициента, а также значение k^t приведены в п. 2.2.2.

1.4.4. Статистическая обработка результатов

Кроме формальной обработки ответа студента, было предложено добавить механику поощрения и наказания сильно выделяющихся работ. Для этого по окончании тестирования производится статистический анализ результатов всей группы испытуемых по каждому заданию.

В случае, если ответ студента получил оценку значительно выше, чем среднее по группе, то он вознаграждается повышением своей оценки (также возможна ситуация, когда он получает больше баллов, чем задано максимумом). Аналогично производится и понижение оценки в случае плохого результата по сравнению с группой.

Формулы для расчёта приведены в п. 2.2.3.

1.4.5. Учёт параданных прохождения теста

Благодаря цифровизации, во время тестирования мы можем собирать и обрабатывать не только формальные параметры ответа (то есть выбранные варианты), но и ряд поведенческих аспектов прохождения теста студентом, к примеру, движение мышью, время выполнения задания, порядок выбора вариантов ответов, переключение между вкладками браузера и т.д.

На основе подобных событий, можно оценить вероятность того, что испытуемый списывал или пытался угадать правильный ответ. В итоге мы получаем коэффициент, который может уменьшить количество баллов за задание.

Реализация учёта параданных описана в п. 2.2.4.

1.4.6. Подсчёт результатов

В итоге испытуемый должен получить оценку всего теста. Она высчитывается как

$$S_{\Sigma} = \sum_{i=1}^{N} k_s^i k_b^i W^i S_0^i,$$

где N – число заданий в тесте;

k_s^i – статистический коэффициент i-го задания;

k_b^i – поведенческий коэффициент i-го задания;

W^i – коэффициент правдивости i-го задания;

S_0^i – стоимость i-го задания.

Однако стоит отметить, что при выводе результатов необходимо также указывать и баллы по конкретным заданиям, поскольку преподавателю нужно знать, в каких вопросах теста не только конкретный студент, но и вся группа показывает плохие результаты.

2. КОНСТРУКТОРСКАЯ ЧАСТЬ

2.1. Программная реализация

Разработанная система является клиент-серверным приложением. Клиентская часть исполняется веб-браузером, что удовлетворяет поставленным требованиям. Серверная часть приложения развёрнута на облачном сервере, там же расположена и база данных.

Вся логика работы приложения завязана на взаимодействии различных сущностей между собой. К примеру, у сущности типа «ТЕСТ» существует массив, указывающий на сущности типа «ЗАДАНИЕ». Более подробно сущности описаны в п. 2.1.3.

2.1.1. Клиентская часть

Клиентская часть приложения реализована на языке TypeScript [13], который при подготовке к исполнению в браузере компилируется в JavaScript. Основным фреймворком для вёрстки является React, хранилище клиентских данных представлено фреймворком Flexis-Redux.

Клиент представлен двумя контроллерами – Edit и Run – отвечающими, соответственно, за создание/редактирование и прохождение теста. Для доступа к редактированию теста необходимо знать его уникальный ключ, который генерируется при создании.

Контроллер Edit располагается по URL */create* и */edit/<testId>*. Для создания теста пользователю необходимо перейти по первому URL, система автоматически сгенерирует шаблон теста, ключ к нему, и перенаправит пользователя на второй URL. Находясь в режиме редактирования теста, пользователь может:

1. изменять название теста;
2. просматривать список заданий;
3. добавлять новый задания;
4. редактировать задания;

5. удалять задания.

Форма редактирования теста изображена на рис. 1.

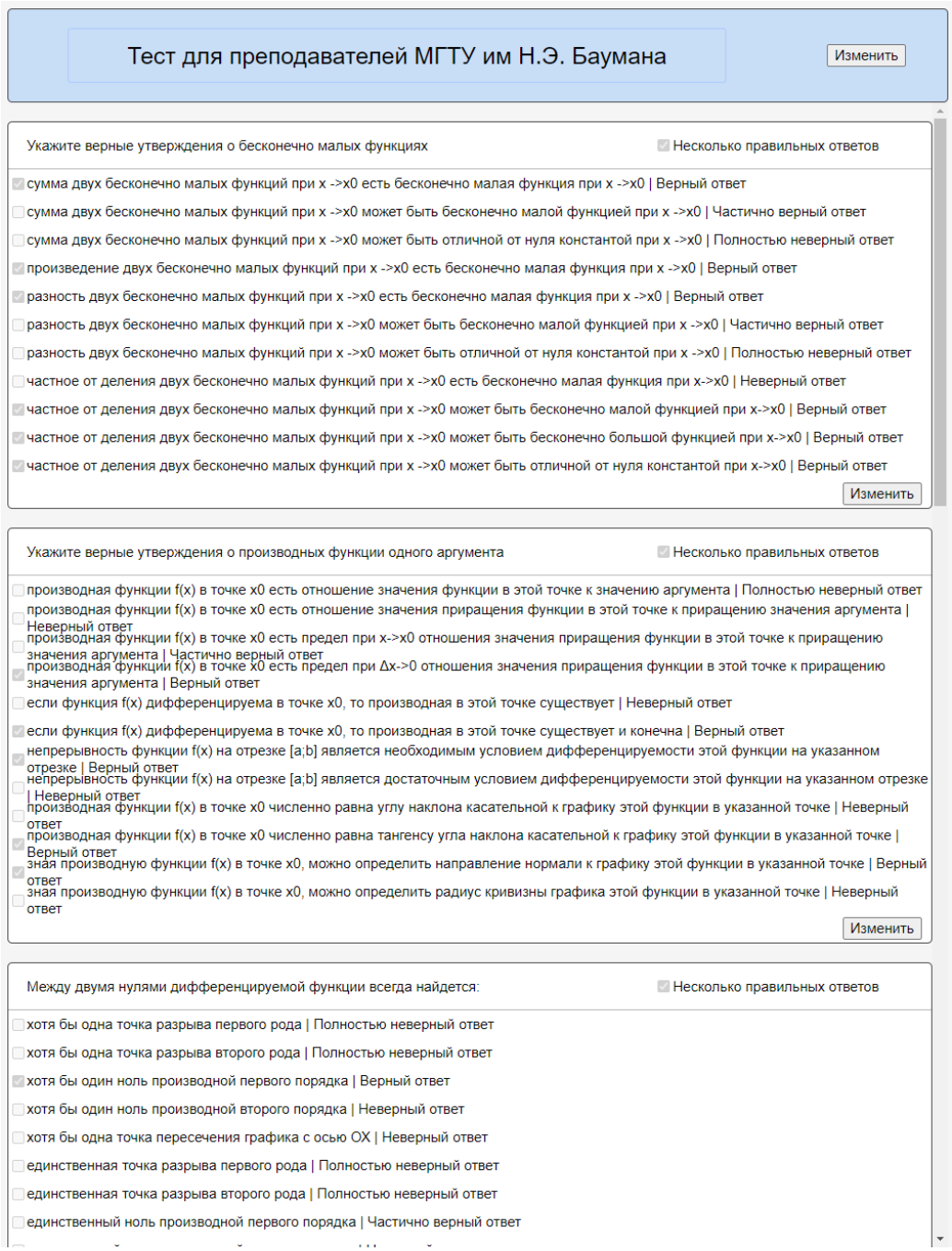

Рис. 1. Общий вид формы редактирования теста

При создании или редактировании задания пользователь может:

1. изменять текст вопроса;

2. изменять тип задания (выбор одного или нескольких вариантов ответов);

3. добавлять новые варианты ответов;
4. изменять текст варианта ответа;
5. изменять класс варианта ответа.

Форма редактирования задания изображена на рис. 2.

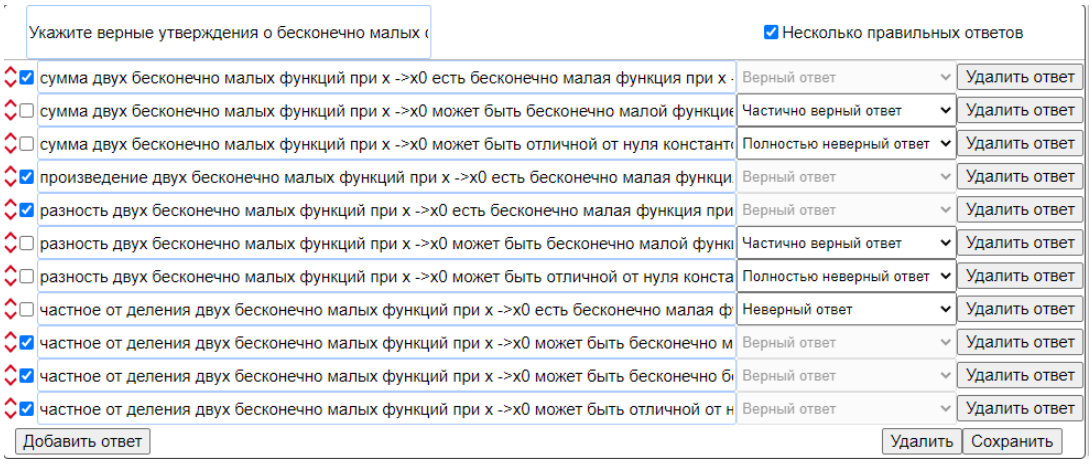

Рис. 2. Форма редактирования задания

Более подробные рисунки, изображающие контроллер Edit, приведены в прил. *А.2*.

Во время редактирования теста данные на сервере обновляются при:

1. создании теста;
2. изменении названия теста;
3. сохранении задания после создания или редактирования.

Контроллер Run располагается по URL */run/<testId>*. Испытуемый, переходя по нему, попадает на страницу приветствия, на которой отображены название теста и кнопка начала. Начав тестирование, студент отвечает на вопросы в заданном порядке, при этом одновременно он видит только одно задание. После ответа на очередное задание, испытуемый получает возможность перейти к следующему, при этом вернуться на предыдущее он может в любой момент. После ответа на последний вопрос студент должен завершить тестирование, чтобы результаты его ответов зафиксировались и обработались. Ознакомиться с прохождением теста можно на рис. 3 и в прил. *А.3*.

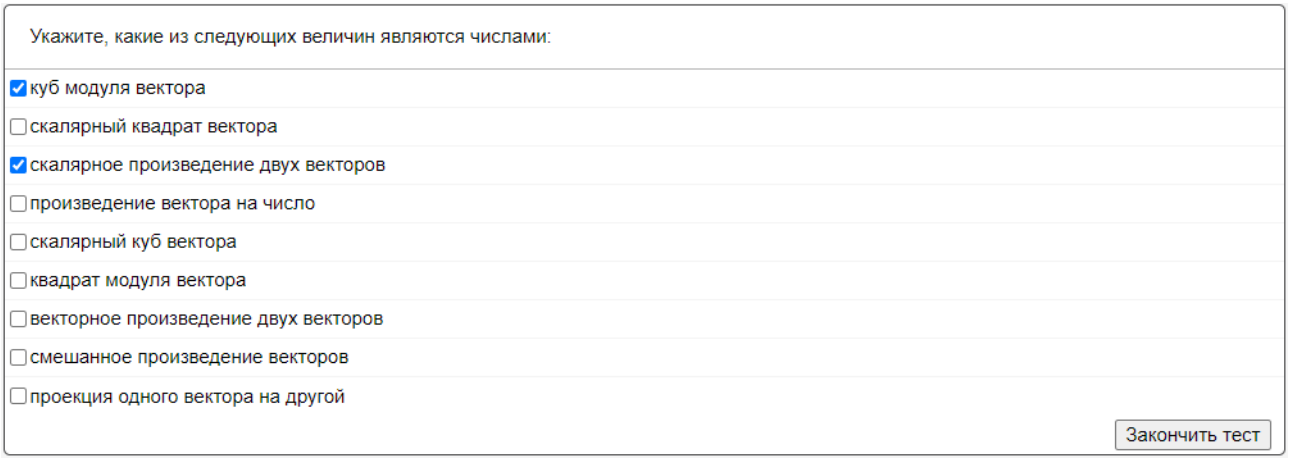

Рис. 3. Прохождение задания.

Приложение загружает с сервера необходимые данные о тесте и заданиях при переходе по URL с тестом, отправка результатов ответов и поведенческой информации происходит при переходе между заданиями и окончании тестирования.

Для оптимизации нагрузки на сервер основная часть объёмных запросов происходит при переключении заданий, причём клиент не ожидает завершения этих запросов, то есть пользователю не приходится ждать пересылки данных для того, чтобы приступить к следующему заданию.

Все сущности (тесты, задания), загружаемые с сервера, хранятся в едином хранилище, к которому у обоих контроллеров есть полный доступ. В случае обновления сущности в базе данных копия, хранящаяся на клиенте, заменяется обновлённой сущностью.

2.1.2. Серверная часть

Серверная часть приложения также реализована на языке TypeScript, который при исполнении кода динамически компилируется в JavaScript. Средой исполнения кода является Node.js [14]. Сервер состоит из 3 компонентов: балансировщика Nginx, микросервиса *Layouts* и микросервиса *Tests*.

Балансировщик отвечает за выдачу статики (файлы форматов .js, .css, .svg) и за перенаправление запросов в микросервисы. Он реализован при помощи фреймворка Openresty, который генерирует конфигурационные файлы Nginx[15] из заранее подготовленных шаблонов, используя переменные окружения.

Микросервис *Layouts* отвечает за выдачу «точки входа» - html-страницы, которая создаётся из шаблона и включает в себя необходимые данные для старта работы клиентского приложения.

Микросервис *Tests* выполняет основную логику работы серверной части приложения. Он отвечает за обработку следующих «ручек»:

- *GET /:testId* – загрузка всех данных о конкретном тесте. Если с запросом передаётся ключ, то возвращается информация, скрытая от студентов (сам ключ);
- *POST /* - создание новой сущности теста. Возвращает всю информацию о созданном тесте, включая ключ;
- *PATCH /:testId* – обновление сущности теста (названия). Возвращает обновлённую сущность теста;
- *GET /task/:*taskId – получение данных о задании. Если с запросом передаётся ключ, то также возвращается скрытая информация (к примеру, правильные ответы);
- *POST /task/:*testId – создание нового задания. Микросервис создаёт новую сущность задачи, а также подвязывает её к тесту. Ответ – обновлённая сущность теста и созданная – задачи;
- *PATCH /task/:*taskId – изменение задания. Возвращаемое значение – обновлённая сущность задачи;
- *DELETE /task/:*taskId – удаление задания. Микросервис удаляет связь задачи и теста, при этом сама сущность задачи остаётся в базе данных. Возвращает обновлённую сущность теста;
- *POST /result/:*testId – создание сущности прохождения теста. Возвращает созданную сущность;
- *PATCH /result/:*resultId – добавление событий, относящихся к прохождению. Микросервис создаёт сущность события и подвязывает её к прохождению. Также в рамках этой «ручки» сервер обрабатывает результаты тестирования;

- *POST /answer/:*resultId – создание сущности ответа на конкретное задание. Кроме того, тут создаются сущности событий, относящихся к данному ответу.

2.1.3. База данных

В качестве базы данных было решено использовать нереляционную базу данных MongoDB. Вся информация в ней для избежания коллизий хранится в одной коллекции. Каждая сущность представляет собой JSON-объект, содержащий как ряд общих полей, так и уникальные для каждого типа сущности.

Универсальными полями являются:

- *_id* – уникальный идентификатор сущности в БД, генерируемый автоматически;
- *type* – строка, обозначающая тип сущности.

В базе данных представлены сущности следующих типов:

- *TEST* – сущность теста, содержащая общую информацию о нём, а также ссылки на его задания;
- *TASK* – сущность задания, содержащая тексты вопроса и вариантов ответов, правильные варианты ответа, классы ответов, максимальное число баллов за задание;
- *RESULT* – однократное прохождение теста, содержит ссылки на глобальные события и на ответы в рамках этого прохождения. По окончании тестирования в эту сущность также записывается результат;
- *ANSWER* – сущность ответа на конкретный вопрос, содержащая информацию о выбранных вариантах ответа, а также ссылки на события, произошедшие в рамках этого ответа;
- *GLOBAL_EVENT* – глобальное событие, к примеру, открытие консоли или изменение фокуса окна;
- *TASK_EVENT* – локальное событие, к примеру, траектория движения мыши при ответе на вопрос.

2.2. Реализация алгоритмов оценки результатов тестирования

2.2.1. Определение сложности задания

Как упоминалось в п. 1.4.2., для определения максимального числа баллов за задание определяется из набора коэффициентов, зависящих от числа вариантов ответов различных классов, и базовой стоимости задания конкретного типа.

Логично было предположить, что увеличение числа вариантов ответов должно приводить к увеличению сложности задания, однако были предложены два дополнительных условия:

1. коэффициенты должны быть ограничены, то есть при росте числа вариантов ответов коэффициенты должны доходить до какой-либо фиксированной величины;
2. при малом числе вариантов ответов рост коэффициентов должен быть медленным, поскольку в таком случае сложность задания не должна сильно отличаться от базовой.

Идеальным кандидатом под эти условия оказались функции сигмоидального типа. В данной работе было принято, что коэффициенты k^c, k^p, k^i вычисляются по следующей сигмоидальной функции (рис. 4):

$$k = \frac{1}{1 + e^{-b \cdot (n-c)}} + 1,$$

где n – число вариантов ответов соответствующего класса;

коэффициенты b и c соответственно равны 0,4 и 10.

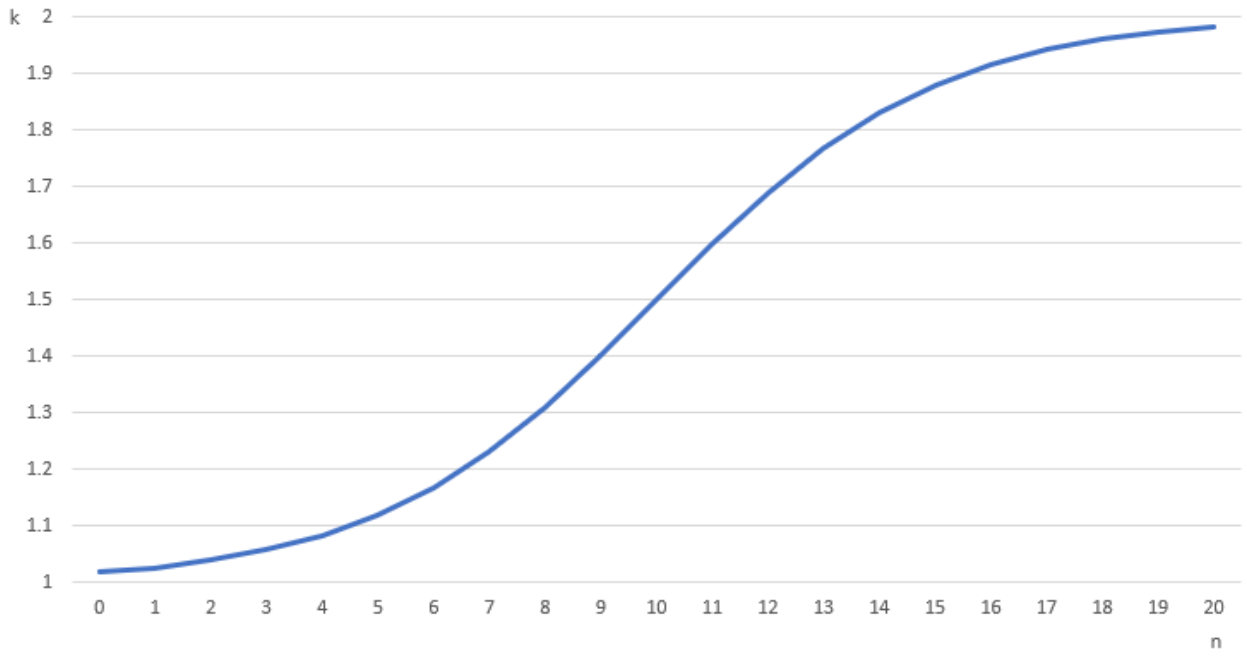

Рис. 4. График зависимости коэффициентов k^c, k^p, k^i от числа вариантов ответов соответствующих классов

Коэффициент k^t было решено оставить фиксированным и равным 1 вне зависимости от числа вариантов ответов.

Коэффициент k^Σ вычисляется как

$$k^\Sigma = \frac{1}{1+ e^{-b\cdot(\sum n - c)}} + 1,$$

где $\sum n$ – суммарное число вариантов ответов;

коэффициенты b и c соответственно равны 0,25 и 20.

Базовая стоимость задания S_b была принята 1 для заданий с выбором одного варианта ответа, 2 для заданий с выбором нескольких.

К примеру, система оценила задание с выбором нескольких вариантов ответов, содержащее по 2 варианта каждого класса в 2 балла. Если увеличить число вариантов каждого класса до 4, то задание уже оценивается в 3 балла.

Далее указан фрагмент программного кода, отвечающий за оценку сложности задания:

```
calculateMaxTaskPoints(task: ITask) {
```

```
        const {
            answersGroups,
            multipleCorrectAnswers,
        } = task;

        const basePoints = multipleCorrectAnswers ?
baseTaskPoints.multiOption : baseTaskPoints.singleOption;
        const answersCount = answersGroups.length;
        const answersGroupsCount: Partial<Record<AnswersGroups, number>>
= {};
        answersGroups.forEach((group) => {
            if (answersGroupsCount[group]) {
                answersGroupsCount[group]++;
            } else {
                answersGroupsCount[group] = 1;
            }
        });

        return Math.round(Object.values(AnswersGroups).reduce((points,
group) => {
            if (!answersGroupsCount[group]) {
                return points;
            }
            return points *
taskPointsModifiers[group](answersGroupsCount[group]);
        }, basePoints)
            * taskPointsModifiers.total(answersCount));
    }
```

2.2.2. Оценка ответа на задание

В случае задания с выбором одного правильного ответа система выдаёт следующие значения коэффициента W:

- 1 для правильного ответа студента;
- 0.25 для частично правильного ответа;
- 0 для неправильного и полностью неправильного ответов.

Для заданий с выбором нескольких правильных ответов, согласно п. 1.4.3., требуется получить положительную и отрицательную компоненты ответа испытуемого, и использовать их для расчёта получаемых за задание баллов. Он производится в четыре этапа:

Этап 1. Расчёт линейной положительной компоненты W_l^+

$$W_l^+ = \frac{n^c}{n_0^c},$$

где n^c – число выбранных студентом правильных вариантов ответов;

n_0^c – число правильных вариантов ответов в задании.

Этап 2. Расчёт смещения линейной положительной компоненты

Достаточно трудно оценить влияние частично правильных ответов на итоговый результат. При удовлетворительном ответе выбор частично правильных ответов должен приводить к ухудшению итогового результата, а при неудовлетворительном – наоборот, к улучшению. Для расчёта этого смещения было принято решение об использовании границы поправки линейной положительной компоненты ответа, которая вычисляется при помощи функции сигмоидального типа:

$$W^b = -\frac{arctg(a \cdot W_l^+ - b)}{c},$$

где a = 100;

b = 0,5;

c = 2π.

Таким образом, граница составляет -0,25 при $W^+ = 1$ и 0,25 при $W^+ = 0$.

Само же смещение считается по формуле

$$W_l^p = \frac{n^p}{n_0^p} \cdot W^b,$$

где n^p - число выбранных студентом частично правильных вариантов ответов;

n_0^p – число частично правильных вариантов ответов в задании.

Этап 3. Расчёт отрицательной компоненты W_l^-

$$W_l^- = 1 - \frac{n^i + k^t \cdot n^t}{n_0^i + k^t \cdot n_0^t},$$

где n^i – число выбранных студентом неправильных вариантов ответов;

n^t – число выбранных студентом полностью неправильных вариантов ответов;

n_0^i – число неправильных вариантов ответов в задании;

n_0^t – число полностью неправильных вариантов ответов в задании;

k^t - коэффициент увеличения вклада полностью неправильных вариантов.

Этап 4. Расчёт оценки ответа

Для симуляции принятия решения преподавателем, в данной работе было принято решение использовать нелинейное вычисление оценки. Поэтому для получения нелинейных компонент их линейные аналоги пропускаются через сигмоидальную функцию:

$$W^+ = \frac{1}{1 + e^{-b(W_l^+ + W_l^p - c)}},$$

$$W^- = \frac{1}{1 + e^{-b(W_l^- - c)}},$$

где коэффициенты b и c соответственно равны 7,5 и 0,5.

Оценка ответа W рассчитывается как

$$W = W^+ \cdot W^-.$$

Далее представлен фрагмент программного кода, отвечающий за выставление оценки ответа:

```
calculateAnswerPointsModifier(answer: IAnswer, task: ITask) {
        const {
            answersGroups,
            multipleCorrectAnswers,
        } = task;
        const {
            answers,
        } = answer;

        if (!multipleCorrectAnswers) {
            return
userAnswerPointsModifiers.singleOption[answersGroups[answers[0]]];
        }

        const answersCount = {
            [AnswersGroups.Correct]: answersGroups.filter((v) => v ===
AnswersGroups.Correct).length,
            [AnswersGroups.PartiallyCorrect]: answersGroups.filter((v) =>
v === AnswersGroups.PartiallyCorrect).length,
            [AnswersGroups.Incorrect]: answersGroups.filter((v) => v ===
AnswersGroups.Incorrect).length,
            [AnswersGroups.TotallyIncorrect]: answersGroups.filter((v) =>
v === AnswersGroups.TotallyIncorrect).length,
        };
        const userAnswersCount = {
            [AnswersGroups.Correct]: answersGroups.filter((v, i) => v ===
AnswersGroups.Correct && answers.includes(i)).length,
```

```
            [AnswersGroups.PartiallyCorrect]: answersGroups.filter((v, i)
=> v === AnswersGroups.PartiallyCorrect && answers.includes(i)).length,
            [AnswersGroups.Incorrect]: answersGroups.filter((v, i) => v
=== AnswersGroups.Incorrect && answers.includes(i)).length,
            [AnswersGroups.TotallyIncorrect]: answersGroups.filter((v, i)
=> v === AnswersGroups.TotallyIncorrect && answers.includes(i)).length,
        };

        const correct = userAnswersCount[AnswersGroups.Correct] /
answersCount[AnswersGroups.Correct];

        const partialCorrectBound = -Math.atan(100 * (correct - 0.5)) /
(2 * Math.PI);

        const partialCorrect =
userAnswersCount[AnswersGroups.PartiallyCorrect] * partialCorrectBound /
(answersCount[AnswersGroups.PartiallyCorrect] || 1);

        const incorrect = 1 -
            (
                userAnswersCount[AnswersGroups.Incorrect]
                + 2 * userAnswersCount[AnswersGroups.TotallyIncorrect]
            )
            / ((
                answersCount[AnswersGroups.Incorrect]
                + 2 * answersCount[AnswersGroups.TotallyIncorrect]
            ) || 1);

        const result = (1 / (1 + Math.exp(-7.5 * ((correct +
partialCorrect) - 0.5)))) * (1 / (1 + Math.exp(-7.5 * (incorrect -
0.5))));

        return round(result, 1);
    }
```

2.2.3. Статистическая обработка оценки

Как и было предложено в п. 1.4.4., приложение производит обработку результатов тестирования со сравнением конкретного ответа со средними значениями по всем прохождениям данного теста. Расчёт статистического коэффициента k_s^i для *i*-го задания теста производится по следующей формуле:

$$k_s^i = \begin{cases} \dfrac{W^i + \left(W^i - (W_{\text{ср}}^i + s^i)\right) \cdot k_s}{W^i} \text{ при } W^i > W_{\text{ср}}^i + s^i, \\ 1 \text{ при } W_{\text{ср}}^i - s^i \leq W^i \leq W_{\text{ср}}^i + s^i \\ \dfrac{W^i - \left((W_{\text{ср}}^i + s^i) - W^i\right) \cdot k_s}{W^i} \text{ при } W^i < W_{\text{ср}}^i - s^i, \end{cases}$$

где W^i - коэффициент правдивости *i*-го задания данного прохождения;

$W_{\text{ср}}^i$ – среднее арифметическое коэффициента правдивости *i*-го задания по всем прохождениям теста;

s^i - среднеквадратическое отклонения коэффициента правдивости *i*-го задания по всем прохождениям теста;

k_s - коэффициент влияния статистики, в данной работе принятый 0,25.

Далее представлен фрагмент программного кода, отвечающий за проведение статистической обработки оценки:

```
await Promise.all(currentResultAnswers.map(async (answer) => {
            const {
                initialPoints: currentPoints,
                taskId,
            } = answer;

            const taskAnswers = allAnswers.filter((a) => a.taskId ===
taskId);
            const m = taskAnswers.reduce((sum, a) => sum +
a.initialPoints, 0) / taskAnswers.length;

            let s;
            if (taskAnswers.length > 1) {
```

```
                s = Math.sqrt(taskAnswers.reduce((sum, a) => sum +
Math.pow(a.initialPoints - m, 2), 0) / (taskAnswers.length - 1));
            } else {
                s = 0;
            }

            let points = currentPoints;
            if (currentPoints > m + s) {
                points = currentPoints + (currentPoints - (m + s)) *
0.25;
            }
            if (currentPoints < m - s) {
                points = currentPoints - ((m - s) - currentPoints) *
0.25;
            }
            points = round(points);
            if (points < 0) {
                points = 0;
            }
            return this.entityService.update(answer._id, {
                $set: {
                    points,
                },
            });
        }));
```

2.2.4. Учёт параданных

Для оценки вероятности списывания или простановки ответов наугад потребовалось собрать статистику прохождения тестов в системе. Было проведено апробирование, в котором приняли участие студенты факультета РК МГТУ им. Н.Э. Баумана. Всего было начато 737 уникальных прохождения теста, однако только 59 из них были завершены.

Собранной статистики оказалось недостаточно для того, чтобы можно было снижать оценки из-за нечестного прохождения теста, поэтому поведенческий коэффициент k_b был принят равным 1 для всех заданий.

Подробно сбор поведенческой статистики и её анализ рассмотрен в работе [16].

3. ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

3.1. Обоснование выбора технологического стека

Основной аргументацией выбора стека разработки был опыт работы автора с TypeScript, React, Node.js. Однако, можно привести и ряд объективных причин выбора того или иного решения. Поскольку приложение состоит из нескольких частей, то и разбирать их следует по отдельности.

Выбор JavaScript является вполне обычным для реализации клиентского слоя веб-приложений. Однако, как упоминалось в п.2.1.1., в данном случае было принято решение об использовании TypeScript. Он предлагает ряд возможностей, которых не хватает в обычном JavaScript при разработке комплексных приложений, к примеру типизации переменных или интерфейсов. Также популярной альтернативой для реализации типизации в JavaScript является Flow, однако его функциональность ограничена статичной типизацией [17]. Компиляция TypeScript и сборка клиентского слоя производится при помощи библиотеки Webpack [18].

Фреймворком для реализации клиентской вёрстки был выбран React. Он позволяет производить динамическую сборку приложения прямо в браузере. При помощи ряда оптимизаций данный фреймворк позволяет не только упростить работу разработчика, но и увеличить производительность самого приложения. Также существуют ряд фреймворков со схожей функциональностью, к примеру, Vue или Angular. В среде разработчиков нет единого мнения, какой из вышеперечисленных фреймворков является наилучшим, так как у каждого из них есть свои преимущества и недостатки [19].

Существует большое количество подходов к разработке серверного слоя веб-приложений, однако в большинстве своём они сводятся к выбору между удобством реализации приложения и быстродействием сервера. Node.js в значительной мере сохраняет и то, и то: с одной стороны, писать программный код на JavaScript значительно проще, чем на более низкоуровневых языках, а с другой стороны, приложения на Node.js несильно уступают в

производительности по сравнению с C++ [20]. Кроме того, за счёт асинхронного и неблокирующего исполнения кода, а также JIT-компиляции, Node.js намного эффективнее использует процессорные мощности компьютера [21]. Минусом использования Node.js является значительно возрастающие требования к доступной оперативной памяти.

Выбор нереляционной базы данных MongoDB был в первую очередь обоснован простотой взаимодействия с ней при помощи Node.js, а также хранением данных в формате JSON. На текущем этапе работы выигрыш в удобстве разработки покрывал проигрыш в скорости работы нереляционной БД по сравнению с реляционными.

3.2. Использованные библиотеки и фреймворки

3.2.1. React

Как уже упоминалось выше, React является библиотекой для динамической отрисовки веб-страницы. В основном он применяется для написания одностраничных и мобильных приложений [22]. В основе React лежит использование компонентов – обособленных единиц вёрстки веб-страницы. Его основными особенностями являются:

- однонаправленная передача данных – изменения свойств родительских компонентов вызывают перерисовку дочерних;
- виртуальный DOM – React хранит виртуальную копию объектной модели документа, в которой просчитываются необходимые изменения реального DOM, что даёт значительный прирост в производительности;
- использование JSX – возможности использовать HTML-подобный синтаксис при написании JavaScript-кода, что повышает его читаемость и удобство разработки;
- методы жизненного цикла – функционал изменения свойств компонентов.

3.2.2. NestJS

NestJS является фреймворком для реализации серверного слоя веб-приложений [23]. Исторически сложилось, что в Node.js не было поддержки написания серверных приложений «из коробки». NestJS использует стандартные библиотеки Node.js для реализации сервера, но при этом предлагает готовую архитектуру приложения.

NestJS предоставляет разработчику набор возможных декораторов, каждый из которых показывает фреймворку, какую функциональность выполняет какой-либо класс:

- *Controllers* – слой контроллеров отвечает за обработку входящих клиентских запросов и возврат ответа. Контроллер обозначается путём добавления декоратора *@Controller*;
- *Providers* – почти любой класс, используемый в NestJS, является провайдером. Они отвечают за внутреннюю логику серверного слоя. Для обозначения провайдеров используется декоратор *@Injectable*;
- *Modules* - NestJS использует модули для организации структуры приложения. Для создания модуля требуется декоратор *@Module*. Каждое приложение NestJS имеет как минимум один модуль - корневой. Это место, где NestJS начинает упорядочивать дерево приложений. Фактически, корневой модуль может быть единственным модулем в вашем приложении, особенно когда приложение маленькое, но это не имеет смысла. В большинстве случаев у вас будет несколько модулей, каждый из которых имеет тесно связанный набор возможностей;
- *Middlewares* – функции, которые вызываются перед обработкой входящих запросов. Они применяются для того, чтобы каким-либо образом изменить запрос или выполнить какую-то логику на его основе (к примеру, логгирование запросов). Для обозначения используется декоратор *@Injectable* и имплементация интерфейса *NestMiddleware*;

- *Exception Filters* – слой обработки исключений, в обязанности которого входит перехват необработанных исключений и возврат соответствующего ответа конечному пользователю. Каждое исключение обрабатывается глобальным фильтром исключений, и когда оно не распознается, пользователь получает стандартный ответ с ошибкой;
- *Pipes* – классы, применяющиеся для преобразования данных запроса. К примеру, их можно применять для валидации тела запроса. Pipe должен реализовывать интерфейс *PipeTransform* и использовать декоратор *@Injectable*;
- *Guards* – классы, имплементирующие интерфейс *CanActivate* и применяющиеся для определения того, должны ли запросы обрабатываться в контроллерах обработчиками.

Кроме того, NestJS поддерживает функционирование микросервисов. В этом случае это остаётся тем же приложением, однако оно использует другой транспортный уровень – TCP или Redis pub/sub вместо HTTP [24]. Однако, в данной работе для реализации необходимой инфраструктуры микросервисы были представлены обычными приложениями NestJS, изолированными друг от друга. Подробнее об этом – в следующем пункте.

3.3. Инфраструктурные решения

Для упрощения как разработки, так и развёртывания, было принято решение о контейнеризации приложения с помощью программного обеспечения Docker. Таким образом, появилась возможность изолировать различные слои приложения друг от друга, а также настраивать их окружение [25]. При проектировании архитектуры приложения были выделены следующие контейнеры:

- *client* – отвечает за сборку клиентского слоя при помощи библиотеки Webpack. При разработке контейнер запущен постоянно и всё время пересобирает клиент в случае изменений в коде, при работе в рабочем окружении – однократно собирает клиентский код и затем выключается;

- *service-layouts* – в этом контейнере запущен микросервис *Layouts*, который генерирует и отдаёт клиенту корневую html-страницу;
- *service-tests* – в этом контейнере запущен микросервис *Tests*, который генерирует и отдаёт клиенту корневую html-страницу;
- *balancer* – балансировщик, который отвечает за распределение запросов между микросервисами, а также за отдачу собранного клиентского кода. Также балансировщик обеспечивает корректную работу SSL-сертификатов;
- *database* – контейнер, содержащий в себе базу данных.

Контейнеры изолированы от внешней среды, но находятся в одной docker-сети. Доступ из сети Интернет открыт только для контейнера *balancer* по стандартным TCP-портам 80 и 443. Кроме того, был открыт доступ к базе данных по секретному порту с целью работы с данными в ней через графический интерфейс.

Для развёртывания вышеперечисленных контейнеров использовался функционал cli-приложения docker-compose, также для него был разработан конфигурационный файл docker-compose.yml. Для развёртывания приложения в различных средах (разработка, производство) используется уникальный для каждого окружения env-файл.

Для проведения апробирования было необходимо развернуть приложение в рабочем окружении. Для развёртывания был выбран облачный хостинг, предоставляемый компанией Selectel [26]. На момент написания данной работы приложение доступно по адресу

https://rk6sobprotest.ru/run/60ba4ab511ae9a000e284dad

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной магистерской работы был предложен метод гибкой оценки результатов тестирования студентов. Полученный алгоритм удовлетворяет как формальные требования в виде необходимой функциональности, так и неформальные, поскольку подобранные функции и коэффициенты в полной мере удовлетворяют видение автора насчёт оценки ответов студентов.

В ходе выполнения работы было разработано приложение, реализующее как необходимую функциональность составления и прохождение тестов, так и предложенную методику оценки результатов тестирования. Система была развёрнута на облачном сервере, и было проведено её апробирование студентами факультета РК МГТУ им. Н.Э. Баумана.

Стоит отметить, что в рамках работы не был реализован интерфейс просмотра результатов тестирования, и вся аналитика проходила путём импорта результатов из базы данных. Данный функционал является основной задачей для дальнейшей разработки, кроме того, присутствует потребность внедрения в алгоритм оценки результатов тестирования анализа усвоения предметной области на основе понятийного аппарата.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Василевская Д.Л. Тестирование как метод контроля знаний студентов в учебном процессе // Сборник статей II Международной научно-практической конференции, БНТУ, Минск, 2018.
2. Moodle [Электронный ресурс] / https://moodle.org/?lang=ru (дата обращения: 04.03.2021).
3. Кударов Р.С., Сутугинене Ю. Анализ достоверности тестовой формы контроля знаний // Информационно-управляющие системы, Санкт-Петербург, 2016.
4. Бондаренко М., Семенец В., Белоус Н., Борисенко В. Технология оценивания тестов в зависимости от типа и уровня сложности тестовых заданий на основе интегрированной модели // Fourth International Conference "Modern (e-) Learning", 2009.
5. Морев И.А. ОБРАЗОВАТЕЛЬНЫЕ ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ Часть 4. Развивающий измерительный процесс в вузе // Издательство Дальневосточного университета, Владивосток, 2004.
6. Михайлов С.Н., Демьяненко В.Ю. Инфологические технологии в задачах автоматизированного тестирования знаний студентов // Известия Юго-Западного государственного университета, 2017.
7. SharePoint LMS [Электронный ресурс] / https://www.sharepointlms.com/ (дата обращения: 06.03.2021).
8. Прохорович Г.А., Перанцева А.В., Брезицкая В.В., Туева Е.В., Петросян М.О. Векторная модель анализа данных // Решетневские чтения, Красноярск, 2018.
9. Мячина Е. В. Вероятностно-статистическая модель анализа текста правового документа // Исследовано в России, Москва, 2002.
10. Гаврилина Е.А., Захаров М.А., Карпенко А.П., Смирнова Е.В. Онтологический подход к тестированию уровня владения обучающимся метапредметными понятиями // Наука и Образование. МГТУ им. Н.Э. Баумана, Москва, 2015.

11. Gronlund, N.E. How to construct achievement tests (4th ed.) // Eng-lewood Cliffs, NJ: Prentice-Hall, 1988.
12. Тесленко В.И. Методика анализа и оценка результатов тестирования // Вестник Красноярского государственного педагогического университета им. В.П. Астафьева, Красноярск, 2006.
13. TypeScript [Электронный ресурс] / https://www.typescriptlang.org/ (дата посещения: 16.05.2021).
14. Node.js [Электронный ресурс] / https://nodejs.org/en/docs/ (дата посещения: 16.05.2021).
15. Nginx [Электронный ресурс] / https://nginx.org/ru/ (дата посещения: 17.05.2021).
16. Прокопенко А.П. Исследование методов анализа параданных при прохождении онлайн-тестов / ВКР, МГТУ им. Н.Э. Баумана, 2021.
17. Типобезопасность в JavaScript: Flow и TypeScript [Электронный ресурс] / https://habr.com/ru/post/552760/ (дата посещения: 20.05.2021).
18. Webpack [Электронный ресурс] / https://webpack.js.org/ (дата посещения: 20.05.2021).
19. Angular vs React vs Vue: Which Framework to Choose in 2021 [Электронный ресурс] / https://www.codeinwp.com/blog/angular-vs-vue-vs-react/ (дата посещения: 22.05.2021).
20. Stefanoski K., Karadimche A., Dimitrievski I. Performance comparison of C++ and JavaScript (Node.js – V8 Engine) // Computer Science and Technologies, 2019.
21. Знай свой JIT: ближе к машине [Электронный ресурс] / https://habr.com/ru/company/oleg-bunin/blog/417459/ (дата посещения: 24.05.2021).
22. React [Электронный ресурс] / https://en.wikipedia.org/wiki/React (дата посещения: 24.05.2021).
23. NestJS [Электронный ресурс] / https://docs.nestjs.com/ (дата посещения: 24.05.2021)

24. NestJS - тот самый, настоящий бэкенд на nodejs [Электронный ресурс] / https://habr.com/ru/post/439434/ (дата посещения: 25.05.2021);
25. Docker [Электронный ресурс] / https://ru.wikipedia.org/wiki/Docker (дата посещения: 27.05.2021).
26. Selectel [Электронный ресурс] / https://selectel.ru/ (дата посещения: 28.05.2021).

ПРИЛОЖЕНИЕ А

ГРАФИЧЕСКИЕ ЛИСТЫ.

1. Структура клиентского слоя приложения.
2. Изображения приложения при редактировании теста.
3. Изображения приложения при прохождении теста.
4. Структура серверного слоя приложения.
5. Функции, используемые для оценки сложности теста.
6. Функции, используемые для анализа результатов тестирования.
7. Сценарий редактирования теста.
8. Сценарий прохождения теста.
9. Результаты апробирования.
10. Схема разворачивания приложения.