Государственное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н. Э. Баумана»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»
КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Р А С Ч Ё Т Н О - П О Я С Н И Т Е Л Ь Н А Я З А П И С К А

к курсовой работе на тему:

«База данных для системы тестирования»

Студент _____ Анисимов Н.С.
(Подпись, дата)

Руководитель курсового проекта _____ Строганов Ю.В.
(Подпись, дата)

Москва 2018

Содержание

Введение . . . 3
1 Аналитический раздел . . . 4
1.1 Базы данных . . . 4
1.1.1 Реляционная модель . . . 6
1.1.2 Нереляционная модель . . . 7
1.1.3 Сравнение моделей . . . 8
1.2 Задача автоматизации тестирования . . . 8
1.3 Выводы . . . 10
2 Конструкторский раздел . . . 12
2.1 Общая структура системы . . . 12
2.2 Разработка модели данных . . . 21
2.3 Взаимодействие компонентов системы . . . 23
2.4 Вывод . . . 29
3 Технологический раздел . . . 30
3.1 Сервер . . . 30
3.1.1 База данных . . . 30
3.1.2 Язык . . . 30
3.1.3 Servant . . . 32
3.1.4 Persistent . . . 33
3.1.5 Создание коллекций . . . 34
3.1.6 Создание индексов коллекции `user` . . . 35
3.1.7 Создание индексов коллекции `quiz` . . . 36
3.1.8 Описание методов API . . . 36
3.2 Клиент . . . 40
3.3 Интерфейс . . . 41
3.4 Выводы . . . 44
Заключение . . . 45
Список использованных источников . . . 46

Введение

В настоящие время все чаще для проведения тестирования используются автоматические системы. Многочисленные олимпиады проводятся на базе онлайн-тестов. Такие организации как Яндекс, Mail.ru, Stepik, предоставляют свои средства автоматического тестирования, что подтверждает актуальность разработки подобных систем.

1 Аналитический раздел

1.1 Базы данных

Согласно концепции баз данных, основой информационных технологий являются данные, которые должны быть организованы в базы данных в целях адекватного отображения изменяющегося реального мира и удовлетворения информационных потребностей пользователей. Одним из важнейших понятий в теории баз данных является понятие информации. Информация - это любые сведения о каком-либо событии, процессе, объекте. К информации может относиться все, что может интересовать пользователя любого уровня.

Данные – это информация, представленная в определенном виде, позволяющем автоматизировать её сбор, хранение и дальнейшую обработку человеком или информационным средством. Для компьютерных технологий данные – это информация в дискретном, фиксированном виде, удобная для хранения, обработки на ЭВМ, а также для передачи по каналам связи.

База данных (БД) – именованная совокупность данных, отражающая состояние объектов и их отношений в рассматриваемой предметной области, иными словами БД - это совокупность взаимосвязанных данных при такой минимальной избыточности, которая допускает их использование оптимальным образом для одного или нескольких приложений в определенной предметной области.

Важнейшие свойства баз данных:

— Целостность. В каждый момент времени существования БД сведения, содержащиеся в ней, должны быть непротиворечивы. Например, необходимо отслеживать диапазон допустимых значений, соотношения между значениями в полях, особенности написания формата. Существуют ограничения, работающие только при удалении записей. Например, нельзя удалять запись, связанную с другой неудаляемой записью.

— Восстанавливаемость предполагает возможность восстановления БД после сбоя системы или отдельных видов порчи системы. Сюда относится проверка наличия файлов, составляющих приложение. В основном свойство восстанавливаемости обеспечивается дублированием БД и использованием техники повышенной надежности.

— Безопасность БД предполагает защиту данных от преднамеренного и непреднамеренного доступа, модификации или разрушения. Применяется запрещение несанкционированного доступа, защита от копирования и криптографическая защита. Также необходимы и чисто административные меры, например ограничение доступа к носителям информации.

— Эффективность обычно понимается как:

минимальное время реакции на запрос пользователя;

минимальные потребности в памяти;

сочетание этих параметров.

— Предельные размеры и эксплуатационные ограничения, накладываемые использованием данной БД, могут существенно повлиять на проектное решение.

Создание баз данных, поддержка их в целостном, непротиворечивом состоянии, обеспечение безопасности их использования и сохранности информации вплоть до восстановления её после различных сбоев, предоставление различных информационных услуг пользователям и многое другое обеспечивается СУБД.

Термин СУБД включает в себя довольно большое количество сильно отличающихся друг от друга инструментов для работы с базами данных (отдельные программы и подключаемые библиотеки). Так как данные бывают различных видов и типов, начиная со второй половины 20 века было разработано огромное количество разных СУБД и других приложений для работы с БД.

СУБД основываются на модели базы данных - это специальные структуры предназначенные для работы с данными. Все СУБД сильно отличаются в том, каким образом они хранят и обрабатывают свои данные.

В модели данных различают три главные составляющие:

— структурная часть, определяющая правила порождения допустимых для данной СУБД видов структур данных.

— управляющая часть, определяющая возможные операции над такими структурами.

— классы ограничений целостности данных, которые могут быть реализованы средствами этой системы.

Каждая система поддерживает различные модели и структуры баз данных. Эта модель и определяет, как создаваемая СУБД будет оперировать данными. Существует довольно немного моделей БД, которые предоставляют способы четкого структурирования данных, самая популярная из таких моделей - реляционная модель. Реляционная модель и реляционные БД могут быть очень мощным инструментом, но только если программист знает как с ними обращаться. Недавно, стали набирать популярность NoSQL системы с обещанием избавиться от старых проблем БД и добавить новый функционал. Исключая жесткую структуру данных, при этом сохранив реляционный стиль, эти СУБД предлагают более свободный способ работы с ними и гораздо большие возможности для их настройки. Хотя не обходится и без возникновения новых проблем.

1.1.1 Реляционная модель

Реляционный (SQL) подход обозначает определенную идеологию создания баз данных.

Во-первых, БД представляется на внешнем, не зависящем от структуры ЭВМ уровне в виде совокупности двумерных таблиц, повседневно встречающихся в человеческой практике. Работа с таблицами привычна и понятна каждому пользователю. Поиск и обработка информации, хранящейся в таблицах, не зависит от организации хранения данных в памяти ЭВМ, что значительно упрощает взаимодействие пользователя с БД и существенно повышает производительность его труда.

Во-вторых, манипулирование данными реляционной базы данных, которая с математической точки зрения представляет собой конечный набор конечных отношений. Над отношениями модели можно осуществлять различные алгебраические операции. Теория РБД как раз и определяет, какие операции и каким образом необходимо выполнять над отношениями, чтобы достичь заданной цели.

В настоящее время реляционный подход к построению информационных систем является наиболее распространенным. К числу достоинств реляционного подхода можно отнести:

1. Наличие небольшого набора абстракций, которые позволяют сравнительно просто моделировать большую часть распространенных предметных областей и допускают точные формальные определения, оставаясь интуитивно понятными.
2. Наличие простого и в то же время мощного математического аппарата, опирающегося главным образом на теорию множеств и математическую логику и обеспечивающего теоретический базис реляционного подхода к организации БД.
3. Возможность ненавигационного манипулирования данными без необходимости знания конкретной физической организации баз данных во внешней памяти.

Благодаря десятилетиям разработки, СУБД достигли довольно высокого уровня в производительности и отказоустойчивости. Опытом разработчиков и сетевых администраторов было доказано, что все эти инструменты отлично справляются со своими функциями в приложениях любой сложности, не теряют данных даже при некорректных завершениях работы.

Несмотря на большие ограничения в формировании и управлении данными, реляционные базы данных сохраняют широкие возможности по настройке и предлагают довольно большой функционал.

Реляционные базы данных хранят структурированные данные. Это могут быть сведения о человеке, или о поставщиках и поставках, сгруппированные в таблицах, которые были заранее спроектированы.

Причины использовать SQL следующие:

— Необходимость соответствия базы данных требованиям ACID (Atomicity, Consistency, Isolation, Durability – атомарность, непротиворечивость, изолированность, долговечность). Это позволяет уменьшить вероятность неожиданного поведения системы и обеспечить целостность базы данных. Достигается подобное путём жёсткого определения того, как именно транзакции взаимодействуют с базой данных. Это отличается от подхода, используемого в NoSQL-базах, которые ставят во главу угла гибкость и скорость, а не 100% целостность данных.

— Данные, с которыми выполняется работа, структурированы, при этом структура не подвержена частым изменением. Если организация не находится в стадии экспоненциального роста, вероятно, не найдётся убедительных причин использовать базу данных, которая позволяет достаточно вольно обращаться с типами данных и нацелена на обработку огромных объёмов информации.

1.1.2 Нереляционная модель

Нереляционные (NoSQL) базы данных устроены иначе, чем реляционные. Способ структуризации данных заключается в избавлении от ограничений при хранении и использовании информации. Базы данных NoSQL, используя неструктуризированный подход, предлагают много эффективных способов обработки данных в отдельных случаях (например, при работе с хранилищем текстовых документов). Например, документоориентированные базы хранят информацию в виде иерархических структур данных. Речь может идти об объектах с произвольным набором атрибутов. То, что в реляционной БД будет разбито на несколько взаимосвязанных таблиц, в нереляционной может храниться в виде целостной сущности.

NoSQL базы данных не используют общий формат запроса (как SQL в реляционных базах данных). Каждое решение использует собственную систему запросов.

NoSQL следует использовать в следующих случаях:

— Хранение больших объёмов неструктурированной информации. База данных NoSQL не накладывает ограничений на типы хранимых данных. Более того, при необходимости в процессе работы можно добавлять новые типы данных.

— Использование облачных вычислений и хранилищ. Облачные хранилища — отличное решение, но они требуют, чтобы данные можно было легко распределить между несколькими серверами для обеспечения масштабирования. Использование, для тестирования и разработки, локального оборудования, а затем перенос системы в облако, где она и работает — это именно то, для чего созданы NoSQL базы данных.

— Быстрая разработка. Если вы разрабатываете систему, используя agile-методы, применение реляционной БД способно замедлить работу. NoSQL базы данных не нуждаются в том же объёме подготовительных действий, которые обычно нужны для реляционных баз.

Документоориентированная база данных. Документоориентированная база данных предназначена для хранения иерархических структур данных (документов) и обычно реализуемая с помощью подхода NoSQL. В основе документоориентированных баз данных лежат документные хранилища, имеющие структуру дерева (иногда леса).

Документы могут быть организованы в коллекции. Их можно считать отдалённым аналогом таблиц реляционных СУБД, но коллекции могут содержать другие коллекции. Хотя документы коллекции могут быть произвольными, для более эффективного индексирования лучше объединять в коллекцию документы с похожей структурой

1.1.3 Сравнение моделей

Таблица 1.1 — Сравнение моделей

SQL	NoSQL
Структура и тип хранящихся данных	
Требуется наличие однозначно определенной структуры хранения данных	Нет ограничений на структуру данных
Запросы	
Вне зависимости от лицензии, РСУБД реализуют SQL-стандарты, поэтому из них можно получать данные при помощи языка SQL.	Каждая NoSQL база данных реализует свой способ работы с данными.
Масштабируемость	
Вертикальное масштабирование выполняется за счет увеличения числа системных ресурсов. NoSQL обычно предоставляют более простые способы горизонтального масштабирования.	

1.2 Задача автоматизации тестирования

В настоящее время системы автоматического тестирования пользуются все большим спросом. Многие компании просят соискателей пройти специальные тесты, которые помогают определить уровень владения профессиональными навыками. Данное тестирование не редко проводится при помощи систем автоматического тестирования.

Яндекс.Контест – это сервис для онлайн-проверки заданий по математике и программированию. Он предназначен для проведения состязаний любого уровня –

от школьных олимпиад до соревнований международного класса. Его можно использовать также для подготовки к турнирам и приёма экзаменов.

На базе Яндекс.Контеста проходит ежегодный международный конкурс Яндекс.Алгоритм, тренировочные сборы для спортивных программистов, состязания в рамках программы Факультета компьютерных наук НИУ ВШЭ. Сервис позволяет устраивать как командные, так и личные соревнования. Они могут быть доступны всем желающим или только определенному кругу лиц.

Яндекс.Контест поддерживает более двадцати языков программирования и позволяет использовать разные схемы проведения состязаний. Правила игры задают организаторы турнира, они же готовят и размещают задания. Участники соревнований могут их выполнять, находясь где угодно. Единственное, что им нужно, это интернет.

Решения проверяются автоматически — с помощью набора тестов, составленных авторами заданий. Участники отправляют свои решения в тестирующую систему, а та выдает результат.

Для проведения тестирования на базе Яндекс.Контест необходимо связаться с поддержкой.

Сертификация Mail.Ru – это инструмент проверки знаний программистов. Тесты помогут определить уровень знаний для наиболее популярных сред разработки и языков программирования.

Выбор направлений тестирования программистов проводился совместно с сотрудниками компании HH.ru на основании наиболее популярных запросов от работодателей в сфере IT.

По результатам сдачи тестов выдается сертификат от компании Mail.Ru Group. Полученный сертификат будет доступен по статической ссылке.

Авторизированные пользователи могут создавать свои вопросы.

Stepik – образовательная платформа и конструктор онлайн-курсов. Тестирование может быть пройдено на сайте Stepik.org или в мобильных приложениях для iOS и Android. Система предоставляет открытый доступ к онлайн-курсам и возможность создания собственного образовательного материала.

На основе данной платформы проводятся отборочный этап олимпиады НТИ, онлайн-этап акции Тотальный диктант, международная олимпиада по биоинформатике.

Stepik — многофункциональная и гибкая платформа для создания образовательных материалов. Она позволяет создавать онлайн курсы, интерактивные уроки с видео и различными типами заданий для учащихся, приватные курсы для ограниченной аудитории, проводить олимпиады и конкурсы, запускать программы про-

фессиональной переподготовки и повышения квалификации, а также обучать своих сотрудников и клиентов.

1.3 Выводы

На основе анализа уже существующих систем тестирование, поставлена задача разработки приложения, позволяющего создавать и проходить тесты. Необходимо реализовать регистрацию пользователя, возможность изменения личной информации.

Необходимо реализовать возможность создания новых тестов, и удаление ранее созданных пользователем тестов. При создании нового теста требуется указывать название, описание. Необходима возможность редактирования списка вопросов.

Должна быть реализована возможность прохождения теста пользователем, просмотр ранее пройденных тестов, и поиск тестов для прохождения. Один и тот же тест может быть пройдем более одного раза. Информация о всех попытках должна быть сохранена.

Каждый вопрос теста представляет собой задание и несколько вариантов ответа.

Приложение должно обладать веб-интерфейсом, как наиболее универсальным средством взаимодействия с пользователем.

Определение бизнес-правил

— обязательной информацией о пользователе является логин, электронная почта, пароль;

— необязательная информация о пользователе это имя, фамилия, пол, дата рождения, аватар;

— логин и электронная почта для каждого пользователя должны быть уникальны;

— логин не может быть пустым;

— электронная почта должна быть валидной;

— пароль должен содержать не менее 6 символов;

— пароль хранится в зашифрованном виде;

— аватар представляет собой путь до графического изображения;

— пол принимает одно из следующих значений: не указано, мужской, женский;

— пользователь должен иметь возможность изменения своих личных данных;

— информация о каждом пройденном тесте должна быть сохранена;

— пользователь может создавать новые тесты и удалять собственные тесты;

— сведения о тесте должны содержать имя, описание до 200 символов, и список вопросов;

— каждый тест должен содержать не менее 1 вопроса;

— результат теста может быть получен, если есть ответ на каждый вопрос;
— каждый вопрос содержит от 1 до 6 вариантов ответа.

2 Конструкторский раздел

2.1 Общая структура системы

Каждый пользователь может быть как создавать тесты, так и проходить их. Следовательно, могут быть выделены две роли: администратор, и испытуемый. Диаграмма вариантов использование представлена на рисунке (2.1).

Система предполагает регистрацию и авторизацию пользователей, создание и прохождение тестов.

Для регистрации пользователю необходимо ввести логин, пароль и электронную почту. Регистрация пройдет успешно в случае, если введенные логин и электронная почта не использованы ранее. В случае успеха, будет создан новый аккаунт, и пользователь сразу получит к нему доступ, иначе – сообщение об ошибке. Диаграмма данного процесса представлена на рисунках (2.2) – (2.3).

Для авторизации пользователь должен ввести свой логин и пароль. Если данные верные, пользователь получит доступ к своему аккаунту, иначе – сообщение об ошибке. Диаграмма процесса авторизации представлена на рисунках (2.4) – (2.5).

Чтобы создать новые тест, пользователю необходимо ввести название теста и его описание. Далее предоставляется ввод вопросов. После того, как новый вопрос добавлен, пользователь может добавить следующий. В результате будет получен новый тест. Диаграмма данного процесса представлена на рисунках (2.6) – (2.7).

Для прохождения теста, пользователю необходимо ответить на все предложенные вопросы. После заполнения формы, пользователь отправляет ее на сервер для обработки и получает результат. Диаграмма процесса прохождения теста представлена на рисунках (2.8) – (2.9).

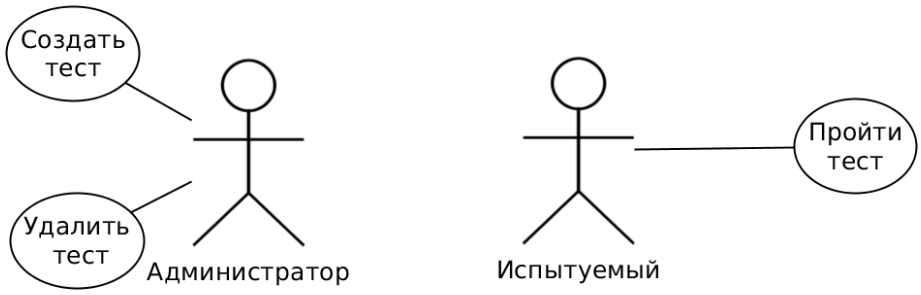

Рисунок 2.1 — Диаграмма вариантов использования

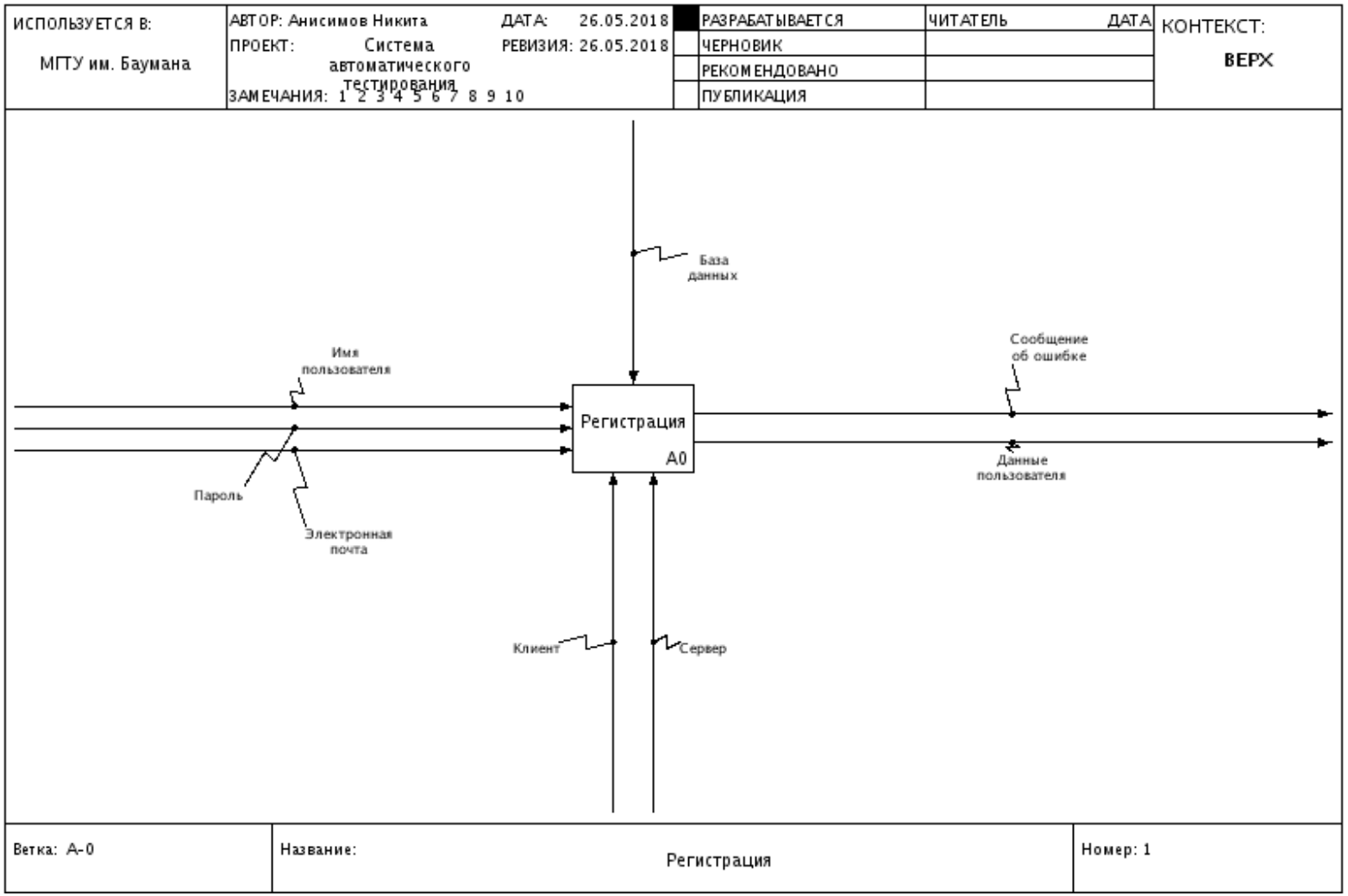

Рисунок 2.2 — Регистрация пользователя (уровень A0)

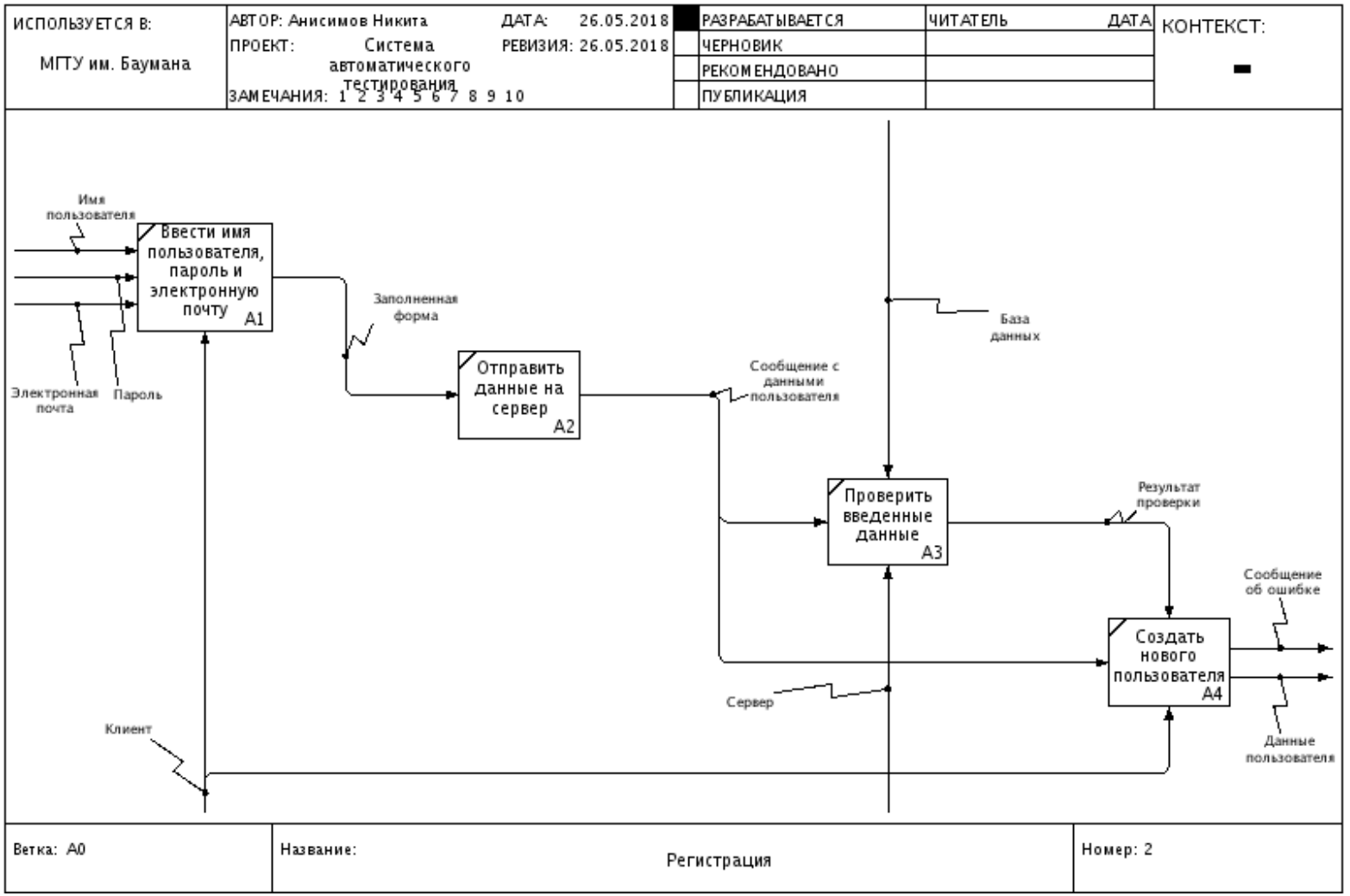

Рисунок 2.3 — Регистрация пользователя (ветка A0)

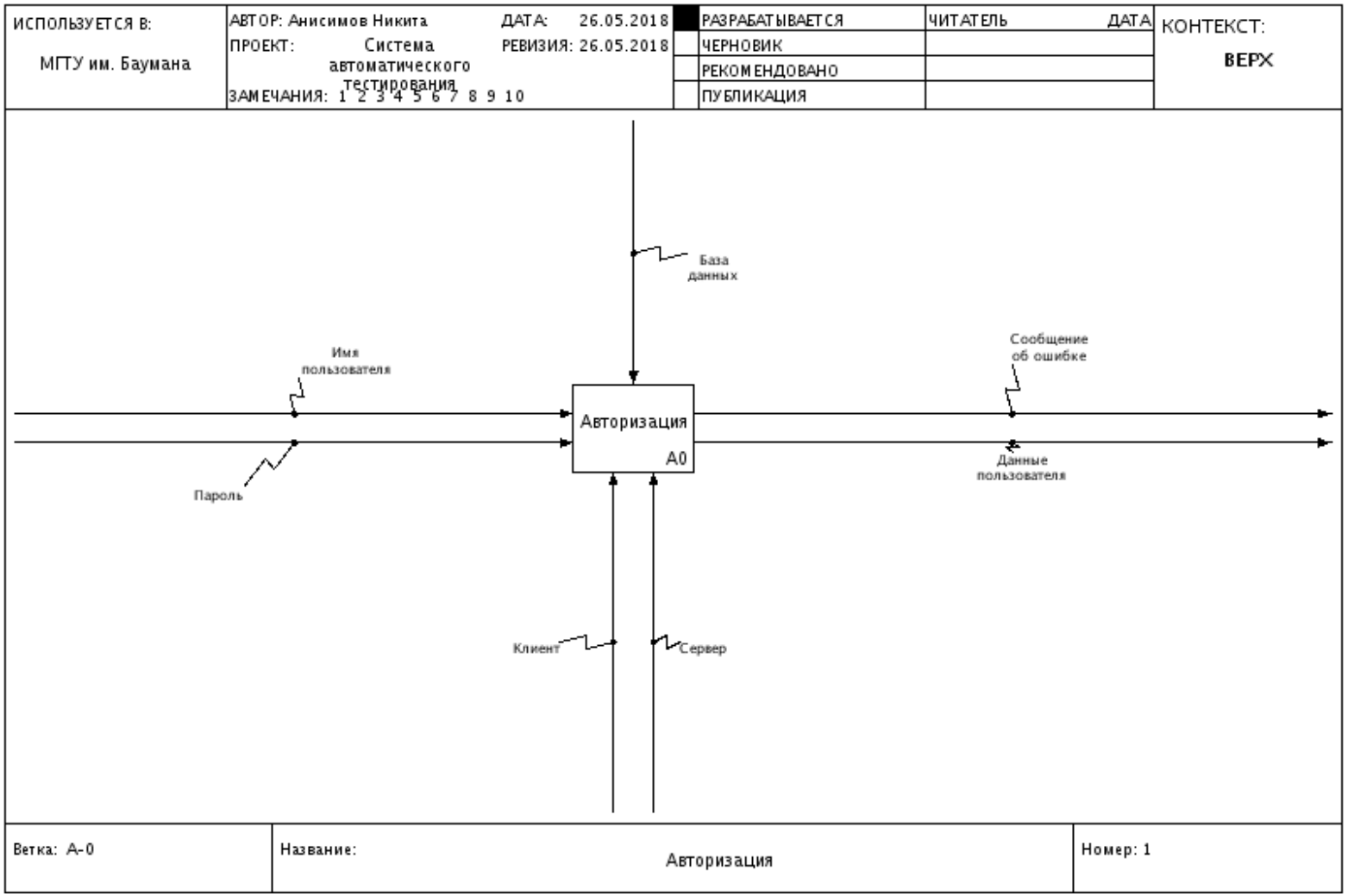

Рисунок 2.4 — Авторизация пользователя (уровень A0)

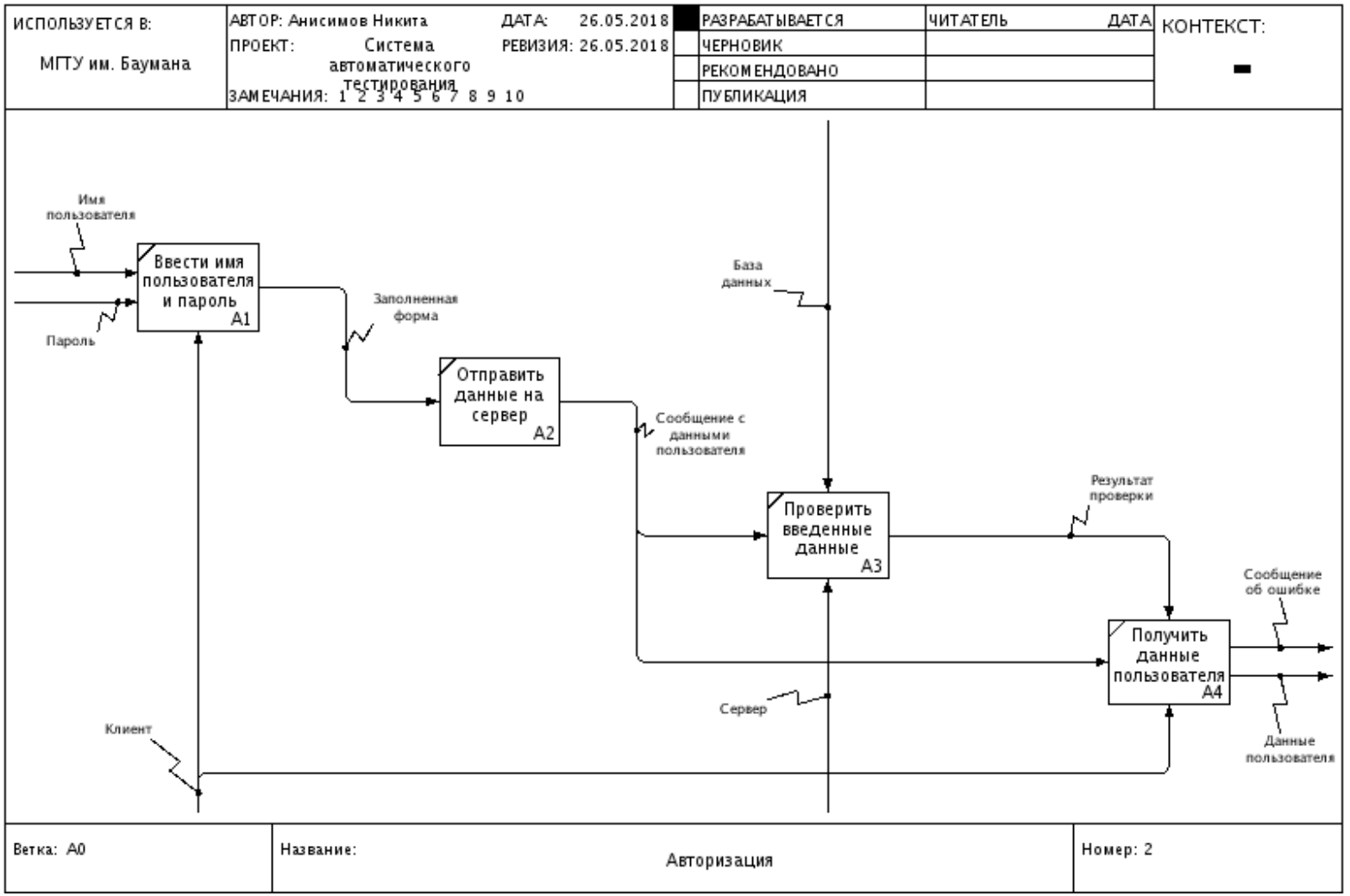

Рисунок 2.5 — Авторизация пользователя (ветка A0)

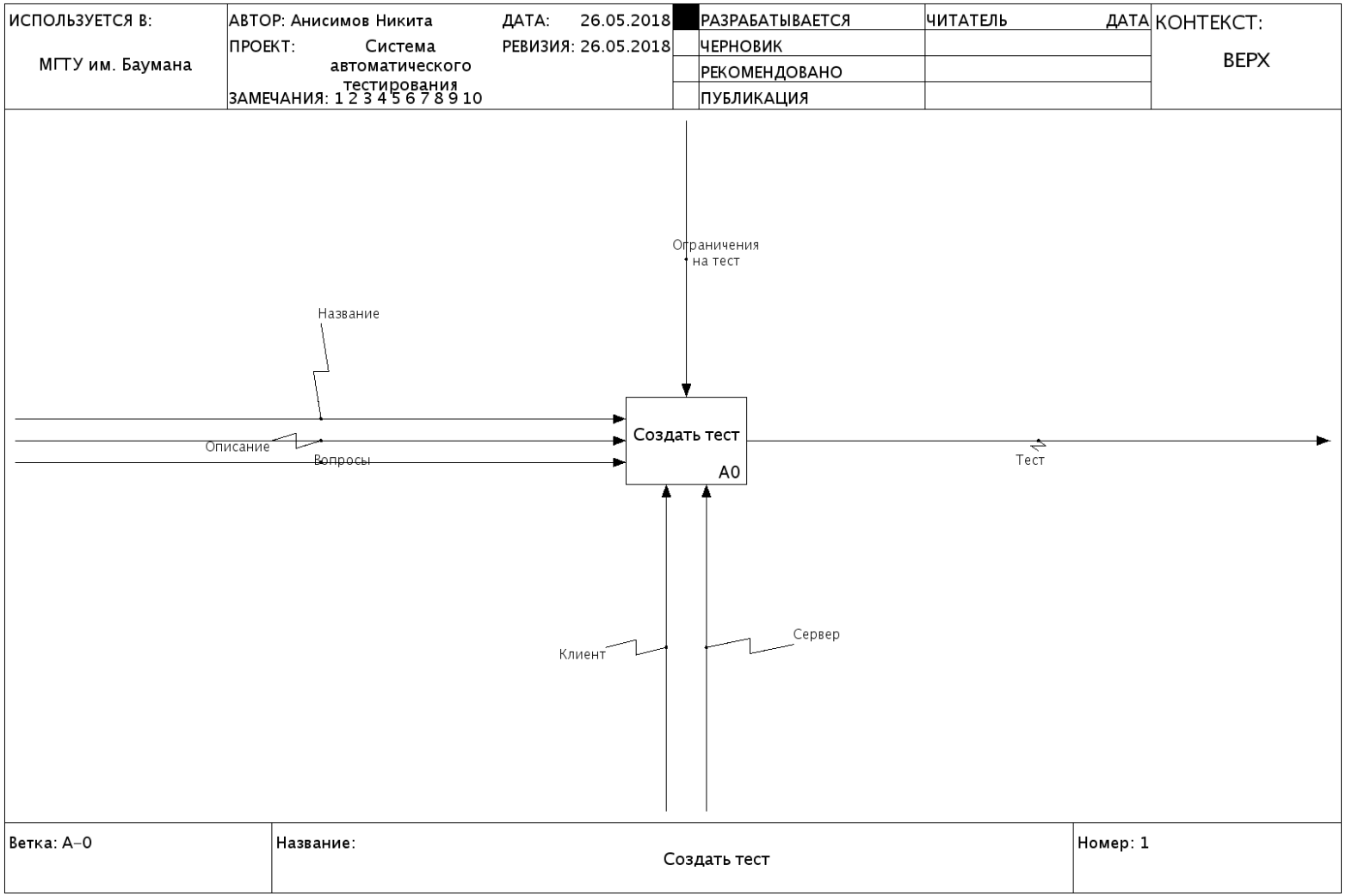

Рисунок 2.6 — Создание теста (уровень A0)

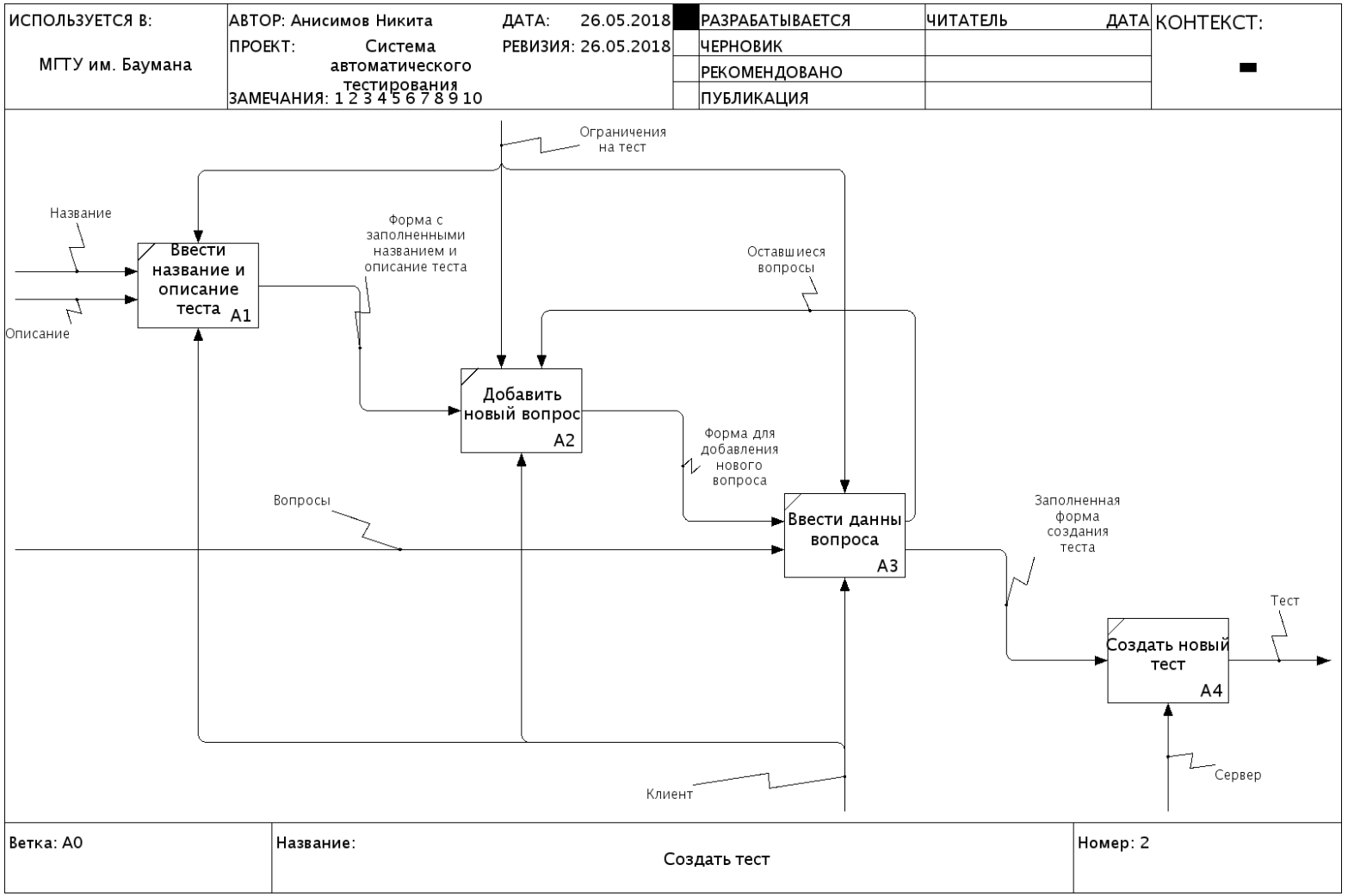

Рисунок 2.7 — Создание теста (ветка A0)

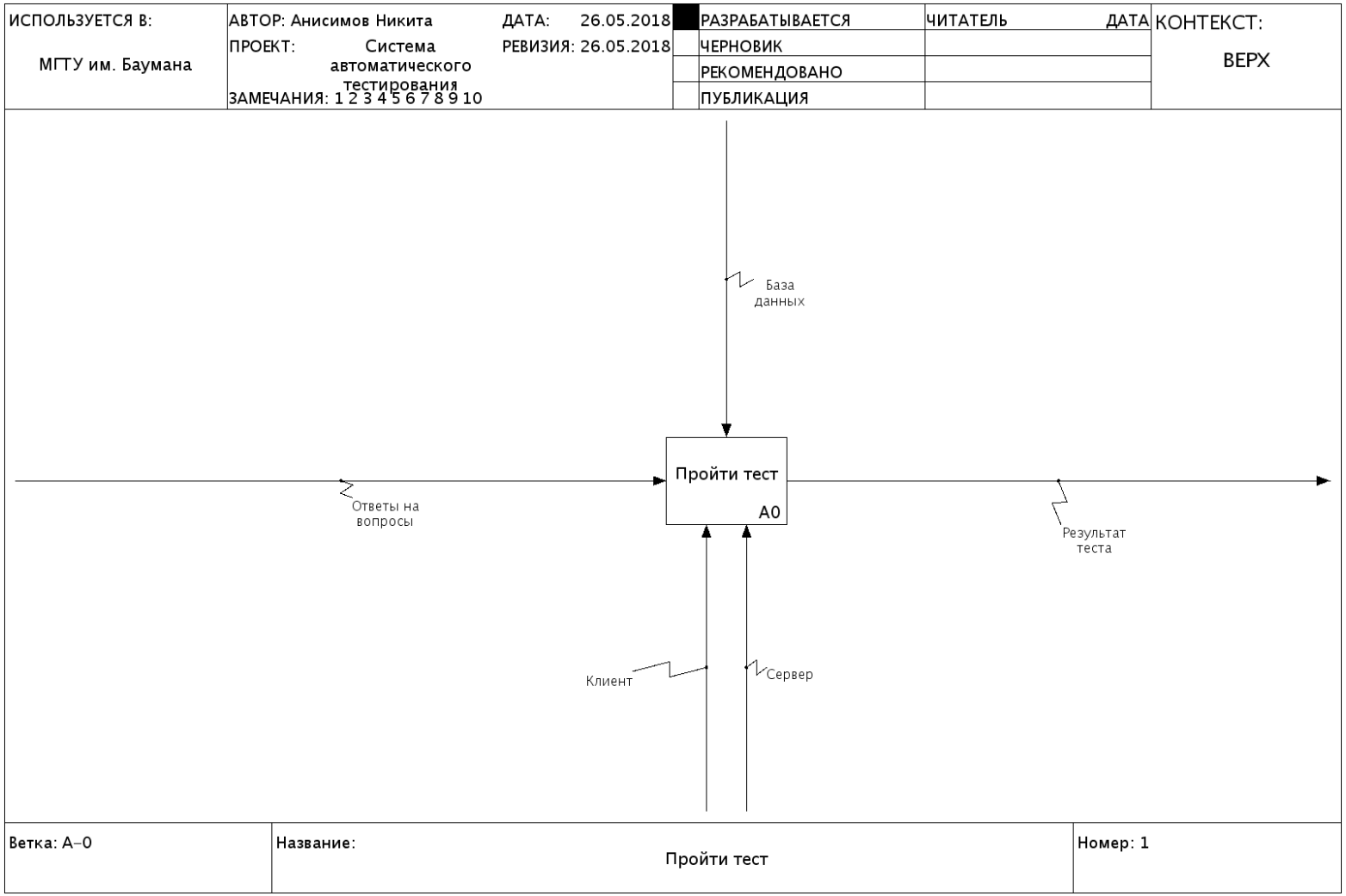

Рисунок 2.8 — Прохождение теста (уровень A0)

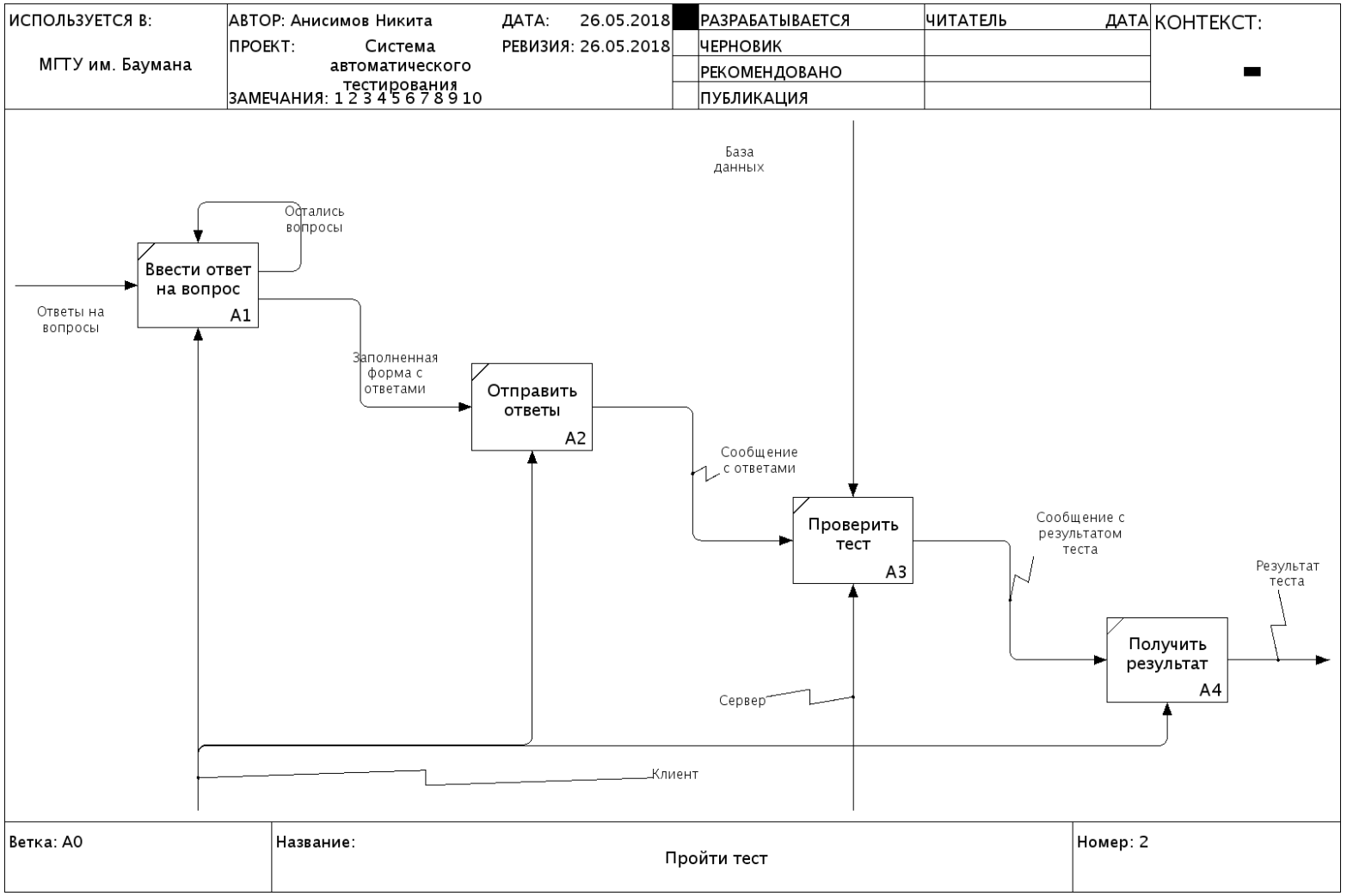

Рисунок 2.9 — Прохождение теста (ветка A0)

2.2 Разработка модели данных

В результате анализа было выявлено 4 сущности: `Пользователь`, `Тест`, `Результат теста`, `Вопрос`. `Пользователь` может создавать и проходить `Тесты`. После прохождения `Теста`, `Пользователь` получает новый `Результат теста`. В `Результате теста` хранится ключ на пройденный `Тест`. `Тест` содержит список `Вопросов`. ER-диаграмма данной модели представлена на рисунке 2.10.

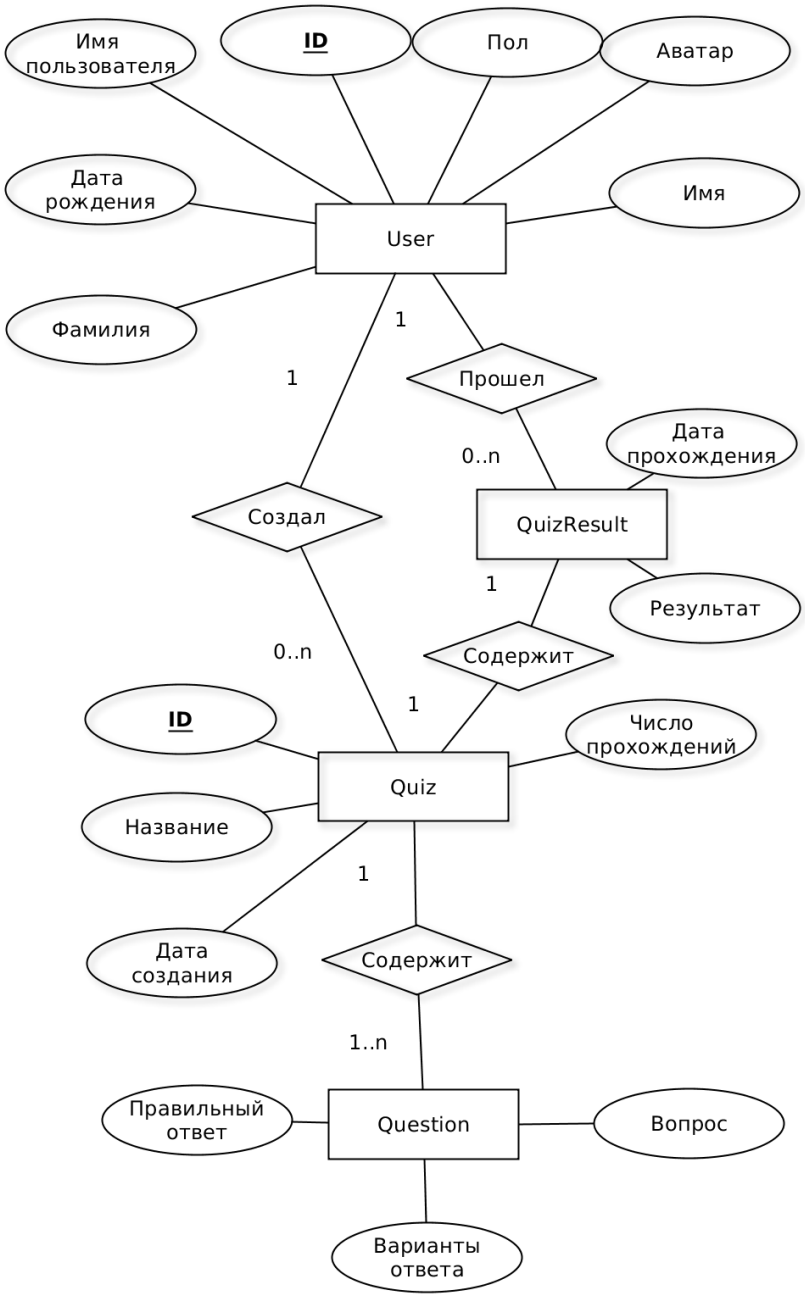

Рисунок 2.10 — ER-диаграмма (нотация Чена)

В качестве базы данных выбрана документоориентированная СУБД, благодаря чему становится возможным использование иерархических структур.

В таблицах 2.1 – 2.4 представлено описание каждой сущности.

Таблица 2.1 — Пользователь

Поле	**Тип**	**Описание**
username	Строка	имя пользователя, уникально, не пусто
email	Строка	электронная почта, уникально, не пусто
password	Строка	пароль, не пусто, не менее 8 символов
avatar	Строка	путь до аватара пользователя, не пусто, если аватара нет, путь до стандартного изображения
first name	Строка	имя, может быть пусто
second name	Строка	фамилия, может быть пусто
birthday	Дата	дата рождения, может быть пусто, не может быть больше сегодняшнего дня
gender	Пол	пол, может быть пусто
passed quiz	Список<Тест>	список тестов, может быть пусто
created quiz	Список<Идентификатор>	список ссылок на созданные тестов, может быть пусто

Таблица 2.2 — Результат теста

Поле	**Тип**	**Описание**
quiz id	Идентификатор	ссылка на пройденный тест, не пусто
result	Строка	результат теста, не пусто
passing date	Дата	дата прохождения теста, не пусто

Таблица 2.3 — Тест

Поле	**Тип**	**Описание**
name	Строка	название теста, не пусто
description	Строка	описание теста, не пусто
creation date	Дата	дата прохождения теста, не пусто
passing number	Целое	число сдач теста, не пусто
question	Список<Вопрос>	список вопрос, не пусто

Таблица 2.4 — Вопрос

Поле	**Тип**	**Описание**
text	Строка	текст вопроса, не пусто
answer	Строка	правильный вариант ответа, не пусто
variants	Список<Строка>	список вариантов ответа, не пусто

2.3 Взаимодействие компонентов системы

Взаимодействие компонентов системы представлено при помощи диаграмм последовательностей, изображенных на рисунках (2.11) – (2.20).

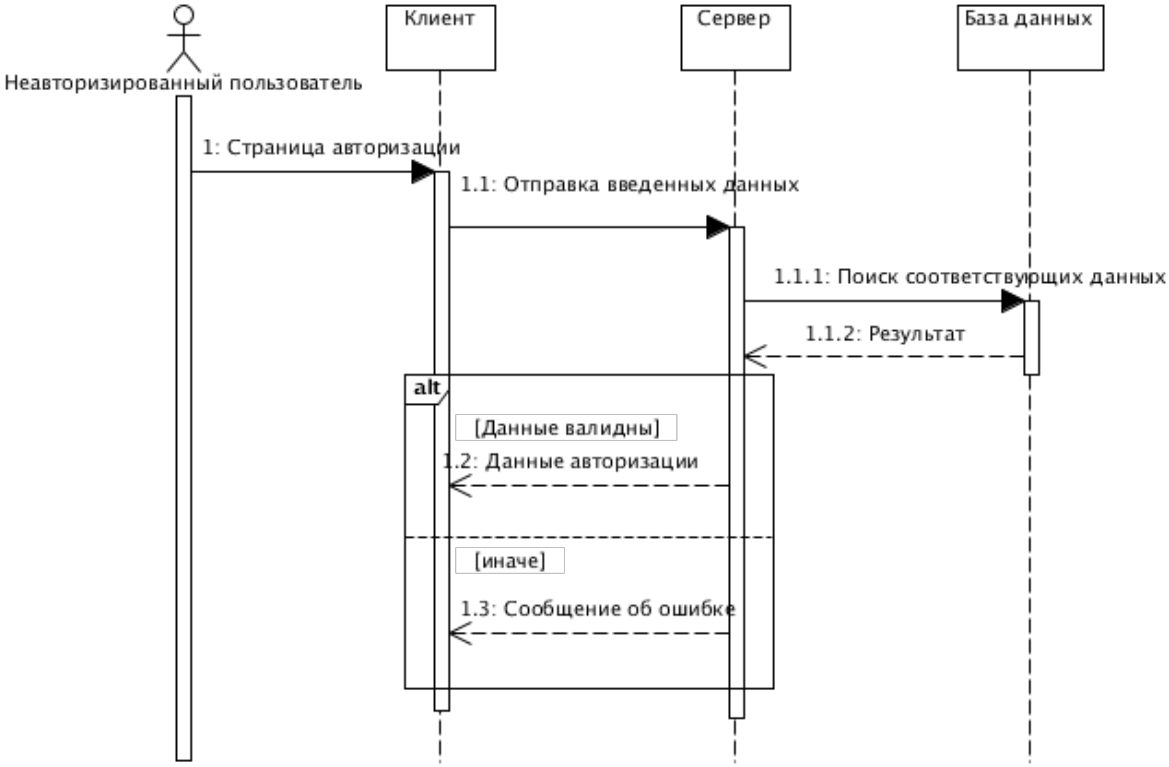

Рисунок 2.11 — Взаимодействие компонентов при авторизации пользователей

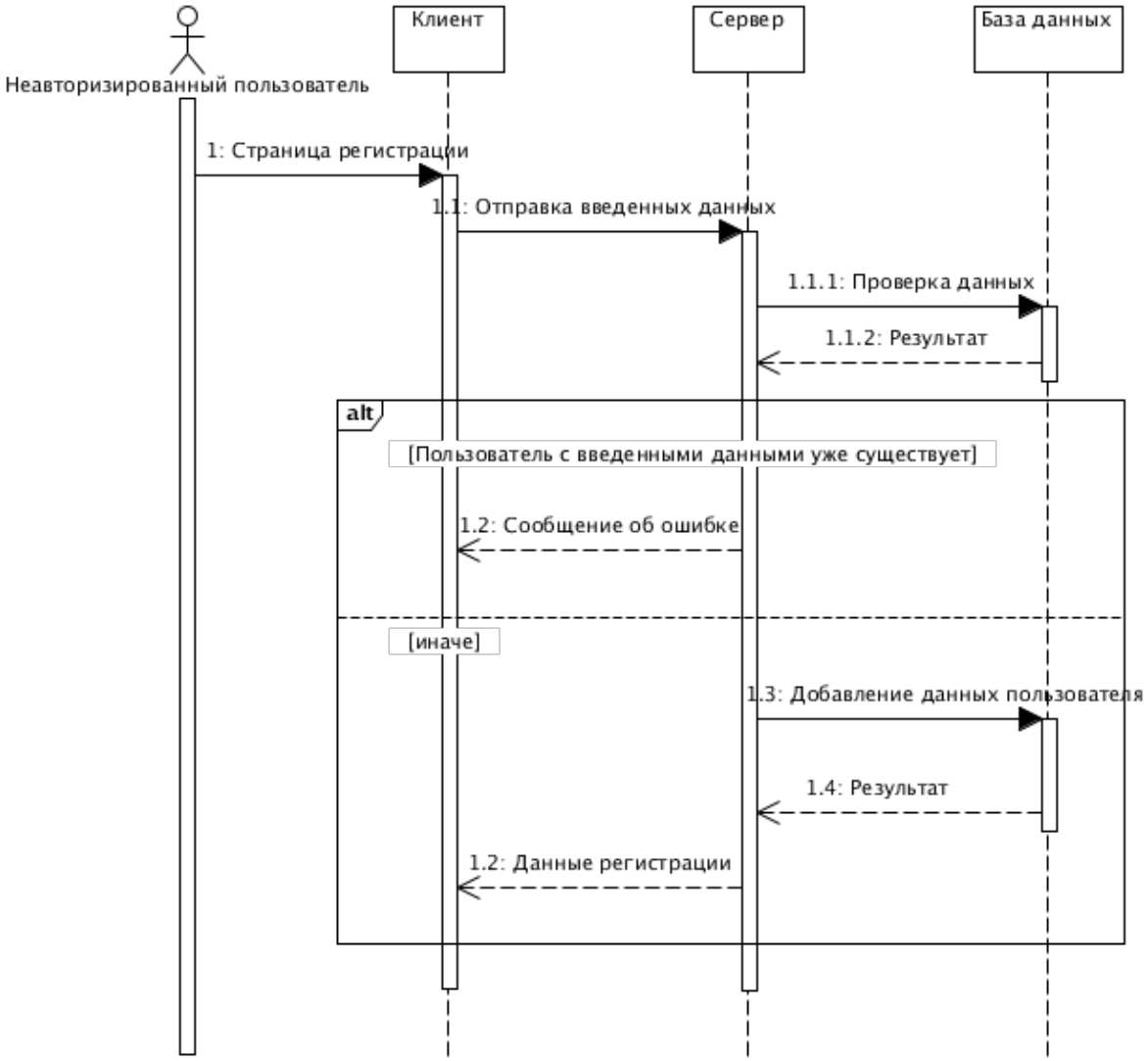

Рисунок 2.12 — Взаимодействие компонентов при регистрации пользователей

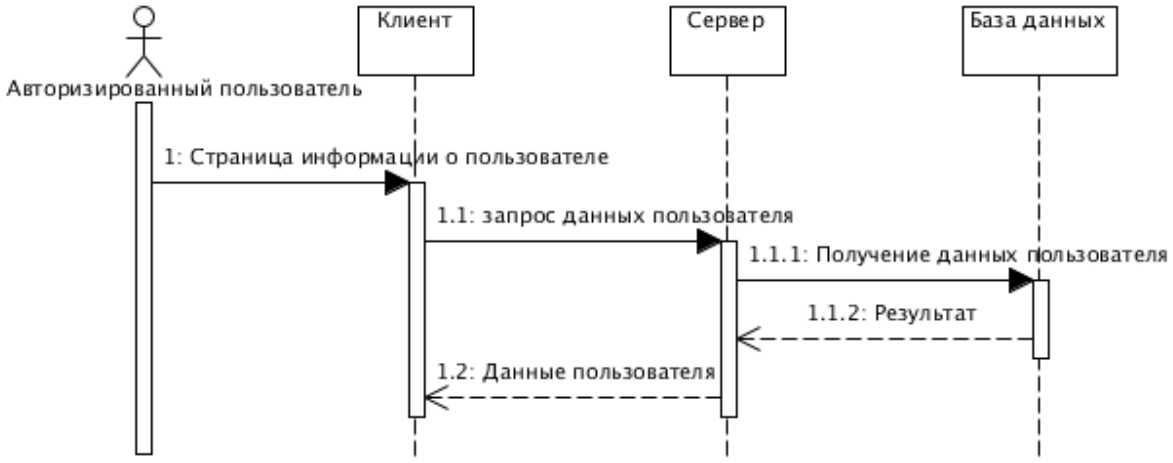

Рисунок 2.13 — Взаимодействие компонентов при запросе данных пользователя

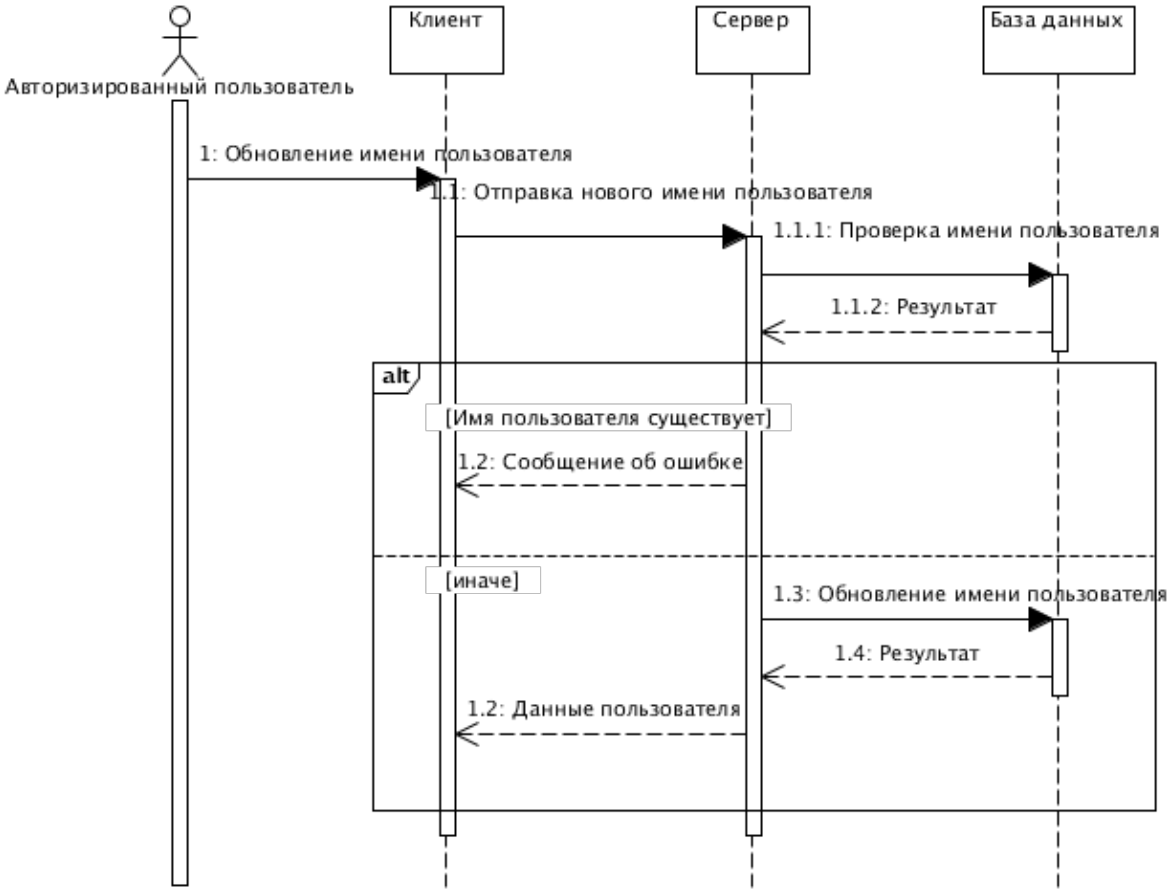

Рисунок 2.14 — Взаимодействие компонентов при изменении имени пользователя

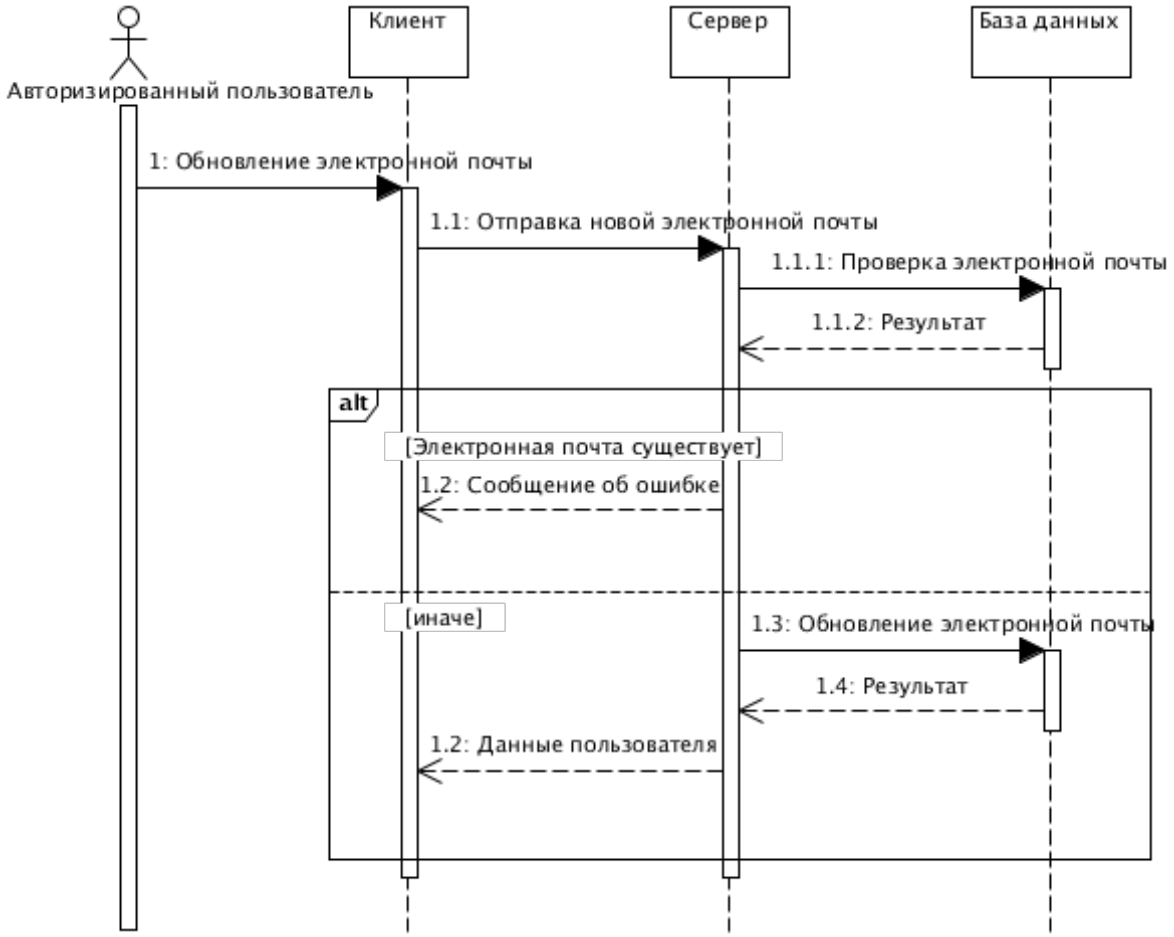

Рисунок 2.15 — Взаимодействие компонентов при изменении электронной почты

Рисунок 2.16 — Взаимодействие компонентов при добавлении нового теста

Рисунок 2.17 — Взаимодействие компонентов при прохождении теста

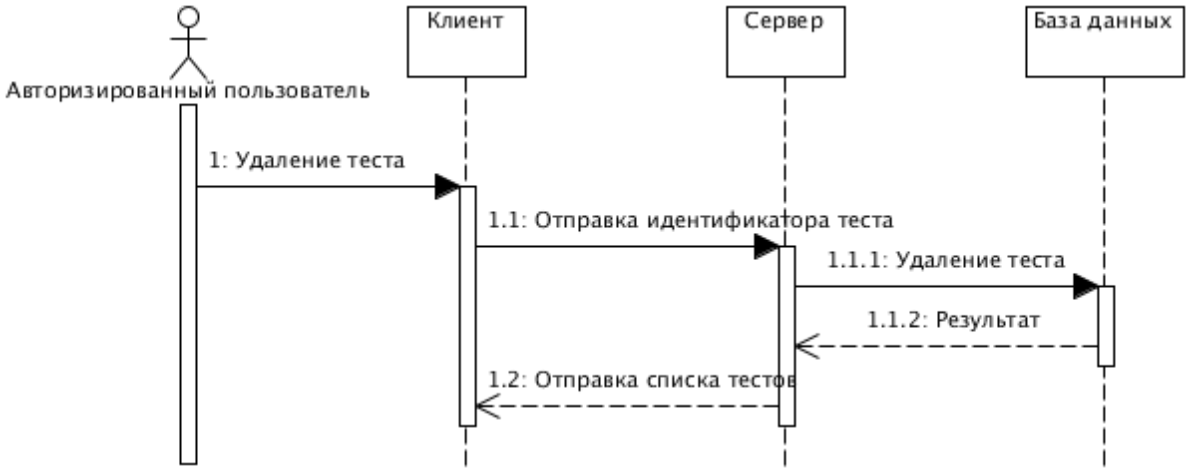

Рисунок 2.18 — Взаимодействие компонентов при удалении теста

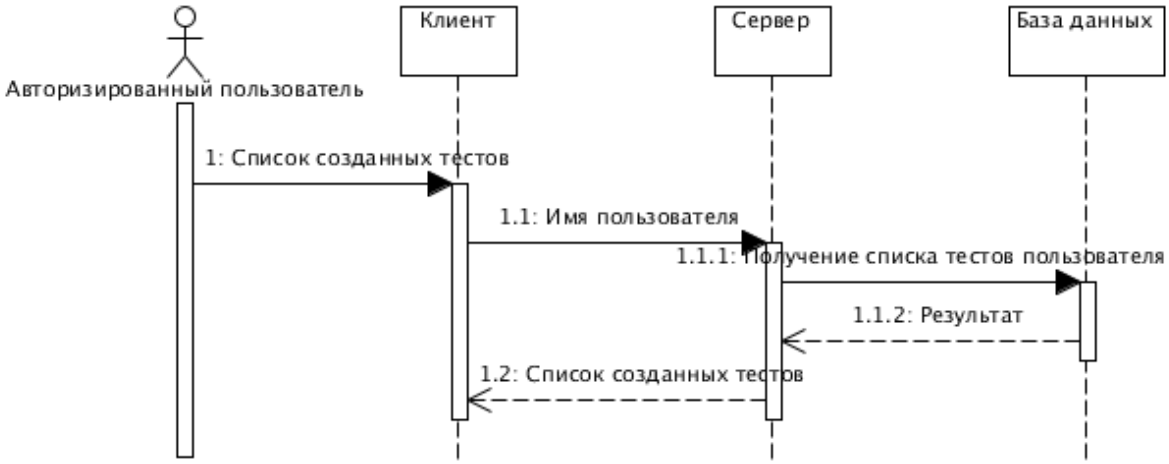

Рисунок 2.19 — Взаимодействие компонентов при получении списка созданных пользователем тестов

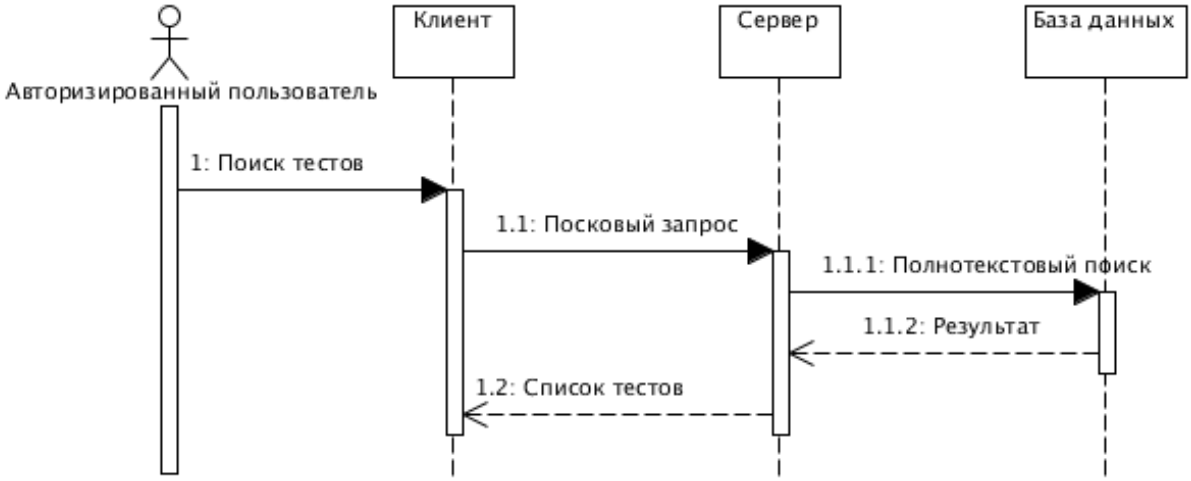

Рисунок 2.20 — Взаимодействие компонентов при поиске тестов

2.4 Вывод

Разрабатываемая система предназначена для проведения автоматического тестирования пользователей. Были определены две роли: администратор, испытуемый.

Процессы авторизации, регистрации, добавления и прохождения тестов детально документированы при помощи IDEF0 диаграмм.

В ходе анализа выявлены следующие сущности: `Пользователь`, `Тест`, `Результат теста`, `Вопрос`. Каждая сущность описана и документирована. Была построена ER-диаграмма для данного набора сущностей.

Определены варианты взаимодействия подсистем для следующих действий: авторизация, регистрация, запрос данных пользователя, изменение имени пользователя, изменение пароля пользователя, изменение электронной почты пользователя, добавление нового теста, прохождение теста, удаление теста, получение списка созданных тестов, поиск тестов.

3 Технологический раздел

На сегодняшний день существует большое количество языков и технологий, используемых как для разработки серверной части, так и для клиентской части.

3.1 Сервер

3.1.1 База данных

В качестве базы данных была выбрана NoSQL база данных MongoDB.

MongoDB это кросс-платформенная, документоориентированная база данных, которая обеспечивает высокую производительность и лёгкую масштабируемость. В основе данной СУБД лежит концепция коллекций и документов.

Коллекция – это группа документов MongoDB. Является эквивалентом простой таблицы в реляционной базе данных. Коллекция помещена внутри одной БД. Документ в коллекции может иметь различные поля. Чаще всего, все документы в коллекции созданы для одной, либо относящихся друг ко другу целей.

Документ – это набор пар «ключ – значение». Документ имеет динамическую схему. Это означает, что документ в одной и той же коллекции не обязан иметь один одинаковый набор полей или структуру, а общие поля в коллекции могут иметь различные типы данных.

Любая реляционная БД имеет стандартную схему, которая показывает количество таблиц и связи между ними. В MongoDB такой схемы с связи между таблицами нет.

Основные особенности MongoDB:

— Отсутствие схемы.

— Данная БД основана на коллекциях различных документов. Количество полей, содержание и размер этих документов может отличаться. Т.е. различные сущности не должны быть идентичны по структуре.

— Легко масштабируется.

— Для хранения используемых в данный момент данных используется внутренняя память, что позволяет получать более быстрый доступ.

— Данные хранятся в виде JSON документов.

— MongoDB поддерживает динамические запросы документов (document-based query).

— Отсутствие сложных JOIN запросов.

3.1.2 Язык

В качестве языка программирование для разработки серверной части был выбран язык Haskell.

Haskell – это функциональный язык. Он воплощает понятие чистоты, модель его вычислений основана на концепции “лени”, обладает параметрическим полиморфизмом.

Среди особенностей данного языка можно отметить следующие:

— **Автоматическое управление памятью.** Память выделяется неявно, автоматически, а специальный сборщик мусора (garbage collector) возвращает системе неиспользуемые куски памяти. Оптимальные алгоритмы управления памятью сложны, но сегодня уже достаточно хорошо проработаны. Использование этих алгоритмов не сильно увеличивают время работы программы в целом (в сравнении с тем, когда программист сам, по-умному, занимается выделением и освобождением памяти).

— **Чистые функции.** Функция называется детерминированной, если возвращаемое ею значение зависит только от аргументов. Говорят, что функция не имеет побочных эффектов, если при ее вызове не производится запись в файл, чтение из сокета, изменение глобальных переменных и так далее. Функция называется чистой, если она является детерминированной и не имеет побочных эффектов. Все функции в Haskell представляют собой выражения, подобные математическим. Они не имеют побочных эффектов (за некоторыми исключениями). Работа с сетью и файлами производится посредством «грязных» функций. В Haskell функции поделены на чистые и «грязные», то есть недетерминированные и имеющие побочные эффекты. «Грязные» функции используются для ввода данных, передачи их в чистые функции и вывода результата.

— **Ленивая модель вычислений.** Haskell реализует ленивую модель вычислений. Выражение на языке Haskell – это лишь обещание того, что оно будет вычислено при необходимости. Одной из проблем ленивых вычислений является использование большого количества памяти, так как необходимо хранить целое выражение для последующего вычисления. Поскольку большинство функций в Haskell являются чистыми, значения, возвращаемые функцией для заданных аргументов, кэшируются. Если функция вызывается многократно с одними и теми же аргументами, реальная работа выполняется только один раз. При втором или третьем вызове из кэша берется уже посчитанное значение.

— **Параметрический полиморфизм.** Параметрический полиморфизм позволяет давать участку кода обобщенный тип, используя переменные вместо настоящих типов, а затем конкретизировать, замещая переменные типами. Параметрические определения однородны: все экземпляры данного фрагмента кода ведут себя одинаково.

— **Параллельные вычисления.** Haskell обеспечивает программиста средствами детерминированного параллельного программирования. Они позволяют ускорить чистое вычисление, не теряя при этом самой чистоты.

— **Широкая область применения.** Существует большое количество программ, написанных на языке Haskell, от приложений с графическим интерфейсом, до серверов.

3.1.3 Servant

Фреймворк – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

Servant – это веб-фреймворк, для языка Haskell. Одним из основных преимуществ данного фреймворка заключается в том, что API сервера описывается как тип. Таким образом на этапе компиляции могут быть отловлены все несоответствия со спецификацией API.

Описание API выглядит следующим образом:

Листинг 3.1 — Определение API

```
1  type UserAPI =
2         "login"
3      :> ReqBody '[JSON] Login
4      :> Post '[JSON] Tokens
5    :<|> "register"
6      :> ReqBody '[JSON] UserRegister
7      :> Post '[JSON] Tokens
8    :<|> "username"
9      :> Get '[JSON] Text
10   :<|> "profile"
11     :> Get '[JSON] Profile
12   :<|> "edit"
13     :> ( "username"
14        :> ReqBody '[JSON] Text
15        :> Post '[JSON] Tokens
16     :<|> "password"
17        :> ReqBody '[JSON] Text
18        :> Post '[JSON] NoContent
19     :<|> "email"
20        :> ReqBody '[JSON] Text
21        :> Post '[JSON] NoContent
22     :<|> "avatar"
23        :> MultipartForm Mem (MultipartData Mem)
24        :> Post '[JSON] Text
25     :<|> "profile"
26        :> ReqBody '[JSON] UserInfo
27        :> Post '[JSON] NoContent
28         )
```

3.1.4 Persistent

Haskell предлагает множество различных библиотек к базам данных. Однако большинство из них имеют малое представление о схеме базы данных и потому не обеспечивают полезных статических проверок. Кроме того, они вынуждают программиста использовать API и типы данных, зависящие от конкретной базы данных. Чтобы избавиться от этих проблем, программистами на Haskell была предпринята попытка пойти более революционным путем и создать хранилище данных, специфичное для Haskell, тем самым получив возможность с легкостью хранить любой тип данных Haskell. Эта возможность действительно прекрасна в некоторых случаях, но она делает программиста зависимым от техники хранения данных и используемой библиотеки, плохо взаимодействует с другими языками, а также для обеспечения гибкости может требовать от программиста написания большого количества кода, запрашивающего данные. В отличие от Persistent, который предоставляет выбор среди множества баз данных, каждая из которых оптимизирована для различных случаев, позволяет взаимодействовать с другими языками, а также использовать безопасный и производительный интерфейс запросов.

Persistent следует принципам безопасности типов и краткого, декларативного синтаксиса. Среди других возможностей следует отметить:

— Независимость от базы данных. Имеется первоклассная поддержка PostgreSQL, SQLite и MongoDB, а также экспериментальная поддержка CouchDB и находящаяся в разработке поддержка MySQL.

— Будучи нереляционным по своей природе, Persistent позволяет одновременно поддерживать множество слоев хранения данных и не обременен проблемами производительности, связанными с использованием JOIN'ов.

— Основной проблемой при использовании SQL баз данных является попытка изменения схемы базы данных. Persistent позволяет автоматически выполнять обновление схемы базы данных.

Пакет Persistent активно использует расширение языка Template Haskell. Template Haskell (TH) — это расширение Haskell, добавляющее в язык шаблоны. Шаблоны в Haskell представляют собой подобие макросов Lisp, только со строгой статической типизацией. Другими словами, TH добавляет в язык возможность метапрограммирования, то есть, написания программ, которые генерируют код программы на этапе компиляции.

3.1.5 Создание коллекций

Коллекции создаются описанием моделей данных на специальном синтаксисе, предоставляемом библиотекой Persistent. Код создания моделей представлен в листинге 3.2.

В качестве типа при описании поля модели используется тип языка Haskell. В таблице 3.1 представлено соответствие между типами языка Haskell и базы данных MongoDB.

Для указания того, что поле может содержать значение `null` или отсутствовать, используется слово `Maybe`. Чтобы использовать внешний ключ на какую либо объявленную модель, в качестве типа данных указывается тип `{Модель}Id`, где `{Модель}` – это одна имя одной из объявленных моделей. Например, модели `QuizResult` и `User` имеют внешний ключ на модель `Quiz`, и в качестве типа поля используют `QuizId`.

Для реализации иерархической структуры в качестве типа указывается тип Модели. Модель `Quiz` включает в себя список вопросов `Question`, а модель `User` включает в себя список пройденных тестов `QuizResult`.

Таблица 3.1 — Соответствие между типами языка Haskell и базы данных MongoD

Haskell	**MongoDB**
Text	String
ByteString	BinData
Int	NumberLong
Double	Double
Rational	*Unsupported*
Bool	Boolean
Day	NumberLong
TimeOfDay	*Unsupported*
UTCTime	Date

Листинг 3.2 — Определение моделей

```
1  share[mkPersist (mkPersistSettings (ConT ''MongoContext))]
       [persistLowerCase|
2  Question json
3    text      Text
4    answer    Text
5    variants [Text]
6    deriving Eq Read Show Generic
7
8  Quiz json
9    name              Text
```

```
10    description     Text
11    creationDate    UTCTime
12    passingNumber   Int
13    questions       [Question]
14    deriving        Eq Read Show Generic
15
16  QuizResult json
17    quizid          QuizId
18    result          Text
19    passing         UTCTime
20    deriving        Eq Read Show Generic
21
22  User
23    username        Text
24    password        ByteString
25    email           Text
26    avatar          Text
27    firstName       Text Maybe
28    secondName      Text Maybe
29    birthday        UTCTime Maybe
30    gender          Gender Maybe
31    createdQuizzes  [QuizId]
32    passedQuizzes   [QuizResult]
33    UniqueUsername  username
34    UniqueEmail     email
35    deriving        Eq Read Show Generic
36  |]
```

Поле `_id` создается автоматически базой данных MongoDB для каждой коллекции и имеет индекс Primary Key.

3.1.6 Создание индексов коллекции `user`

Поля `username` и `email` коллекции `user` должны быть уникальными, поэтому для них необходимо создать ограничение уникальности. Библиотека Persistent не содержит функции создания индексов, поэтому в данном случае необходимо воспользоваться функцией библиотеки mongodb.

Листинг 3.3 — Создание уникальных индексов

```
1  createUserIndexes :: MonadIO m => Action m ()
2  createUserIndexes = do
3    Driver.ensureIndex $ (Driver.index "user" ["username" =: (1 :: Int)])
       {Driver.iUnique = True}
4    Driver.ensureIndex $ (Driver.index "user" ["email" =: (1 :: Int)])
       {Driver.iUnique = True}
```

3.1.7 Создание индексов коллекции `quiz`

Полнотекстовый поиск должен производиться по полям `name` и `description`. Для этого необходимо создать соответствующий индекс. В данном случае, ни одна из используемых библиотек не поддерживает создание индекса полнотекстового поиска, поэтому необходимо создать документ с необходимыми параметрами индекса, и включить его в системную таблицу `system.indexes`.

Листинг 3.4 — Создание индекса для полнотекстового поиска

```
1  createQuizIndexes :: MonadIO m => Action m ()
2  createQuizIndexes = createTextIndex "quiz" "nameText"
       ["name","description"]
3
4
5  createTextIndex :: MonadIO m => Driver.Collection -> Text ->
       [Driver.Label] -> Action m ()
6  createTextIndex col name keys = do
7      db <- Driver.thisDatabase
8      let doc = [ "ns"    =: concat [db, ".", col]
9                , "key"   =: [key =: ("text" :: String) | key <- keys]
10               , "name"  =: name
11               , "default_language" =: ("russian" :: Text)
12               ]
13     Driver.insert_ "system.indexes" doc
```

3.1.8 Описание методов API

Метод:	POST `/user/login`
	Авторизация пользователя
Запрос:	<объект авторизации>
Ответ:	<JWT токен>
Статус:	200 OK
	404 Not Found
	500 Internal Server Error

Метод:	POST `/user/register`
	Регистрация пользователя
Запрос:	<объект регистрации>
Ответ:	<JWT токен>
Статус:	200 OK
	400 Bad Request
	500 Internal Server Error

Метод:	GET `/user/username`
	Получение имени пользователя
Запрос:	
Ответ:	<Строка>
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	GET `/user/profile`
	Получение информации о пользователе
Запрос:	
Ответ:	<объект профиля пользователя>
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	POST `/user/edit/username`
	Изменение имени пользователя
Запрос:	строка
Ответ:	<JWT токен>
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	POST `/user/edit/password`
	Изменение пароля
Запрос:	<строка>
Ответ:	
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	POST `/user/edit/email`
	Изменение электронной почты пользователя
Запрос:	<строка>
Ответ:	
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	POST `/user/edit/avatar`
	Изменение картинки пользователя
Запрос:	<изображение>
Ответ:	<строка>
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	POST `/user/edit/profile`
	Изменение информации о пользователе
Запрос:	<объект профиля пользователя>
Ответ:	
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	POST `/quiz/new`
	Добавление нового теста
Запрос:	<объект теста>
Ответ:	
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	GET `/quiz/get/user/{offset}/{count}`
	Получение {count} тестов, созданных пользователем, начиная с {offset}
Запрос:	
Ответ:	<список объектов теста>
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	GET `/quiz/get/{offset}/{count}`
	Получение {count} тестов, начиная с {offset}
Запрос:	
Ответ:	<список объектов теста>
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	GET `/quiz/get/{id}`
	Получение конкретного теста, с заданным {id}
Запрос:	
Ответ:	<объекта теста>
Статус:	200 OK
	401 Unauthorized
	404 Not Found
	500 Internal Server Error

Метод:	GET `/quiz/search/{query}/{offset}/{count}`
	Получение {count} тестов, начиная с {offset}, удовлетворяющих запросу {query}
Запрос:	
Ответ:	<список объектов теста>
Статус:	200 OK
	401 Unauthorized
	500 Internal Server Error

Метод:	POST `/quiz/remove/{id}`
	Удаление теста с заданным {id}
Запрос:	
Ответ:	
Статус:	200 OK
	401 Unauthorized
	404 Not Found
	500 Internal Server Error

Метод:	POST `/quiz/result/{id}`
	Добавление результатов теста, с заданным {id}
Запрос:	<объект результата теста>
Ответ:	
Статус:	200 OK
	401 Unauthorized
	404 Not Found
	500 Internal Server Error

3.2 Клиент

Клиент был разработан на языке Elm.

Elm — функциональный язык, предназначенный для декларативного создания графических интерфейсов, основанных на браузере. Elm предоставляет возможность описывать графические интерфейсы, не выходя за рамки функциональной парадигмы, используя функционально-реактивный стиль программирования.

Изначальная реализация компилировала Elm в HTML, CSS и JavaScript. В следующих выпусках набор инструментов был расширен: добавлен REPL, пакетный менеджер, отладчик и установщики для Mac OS и Windows. На официальном сайте ведётся репозиторий библиотек, разрабатываемых для языка.

Язык обладает следующими особенностями:

— Код на языке Elm компилируется в JavaScript код.

— Elm код не производит ошибок во время выполнения. Elm использует вывод типов для обнаружения проблем во время компиляции.

— Высокая производительность по сравнению с другими фронт-энд фреймворками.

Логика любой программы на Elm, делится на три части:

— Модель – состояние приложения.

— Обновление – способ обновления состояния.

— Представление – способ отображения состояния как HTML.

Каждое приложение имеет следующий общий вид:

```
1  import Html exposing (..)
2
3  -- Модель
4
5  type alias Model = { ... }
6
7  -- Обновление
8
9  type Msg = Reset | ...
10
11 update : Msg -> Model -> Model
12 update msg model =
13 case msg of
14 Reset -> ...
15 ...
16
17 -- Представление
18
19 view : Model -> Html Msg
20 view model =
21 ...
```

3.3 Интерфейс

На рисунках (3.1) – (3.4) представлен интерфейс некоторых страниц системы.

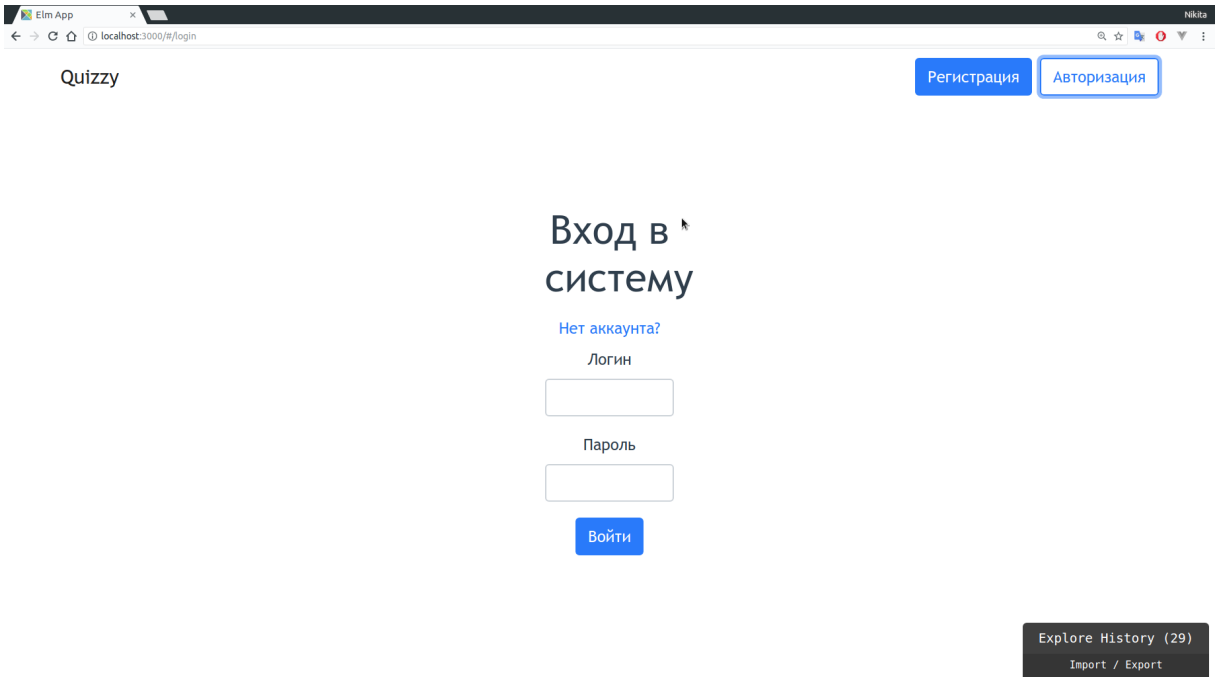

Рисунок 3.1 — Страница авторизации пользователя

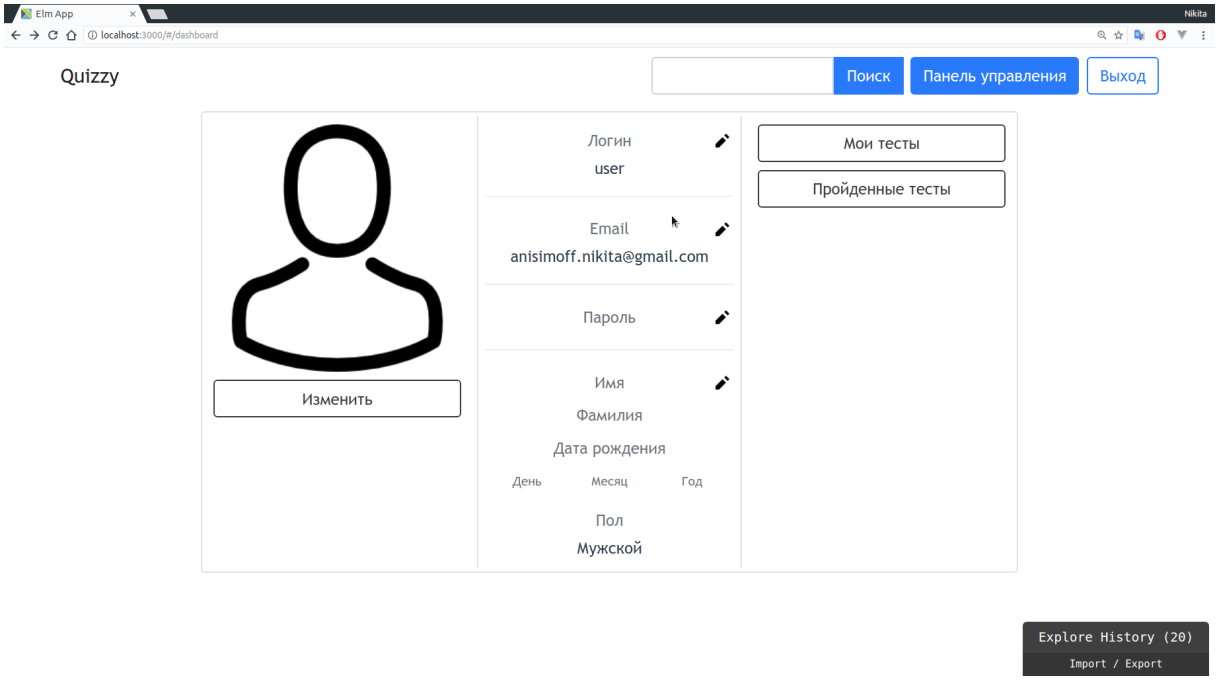

Рисунок 3.2 — Страница пользователя

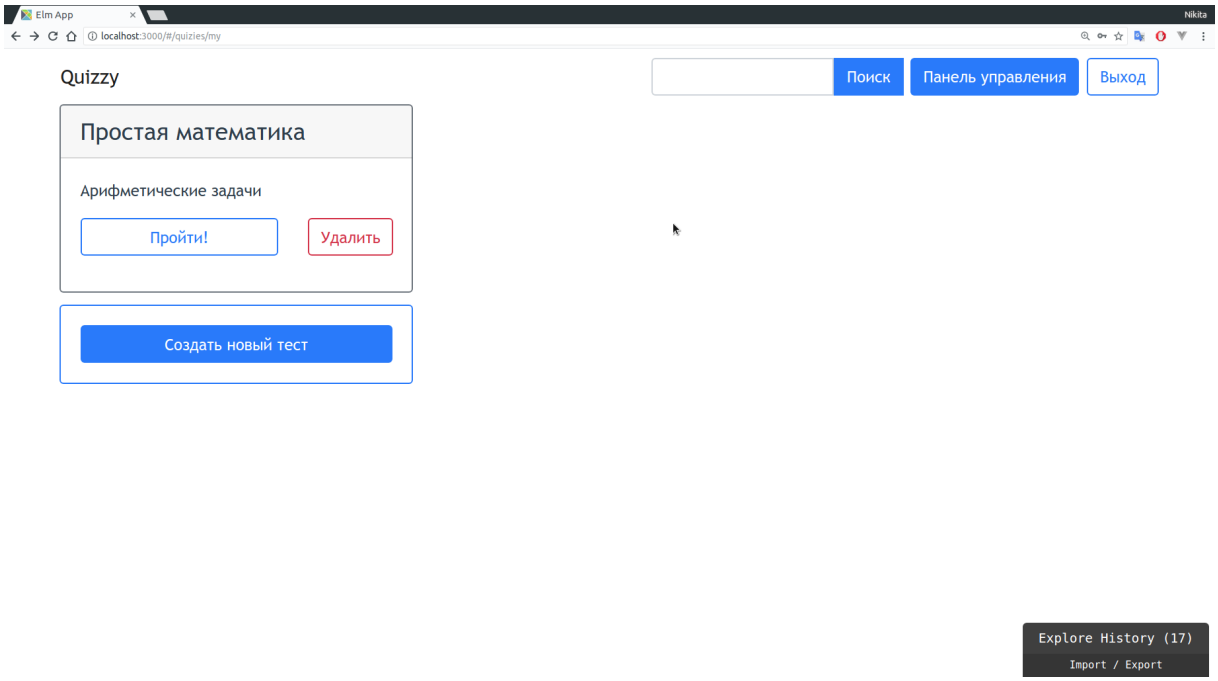

Рисунок 3.3 — Страница управления тестами

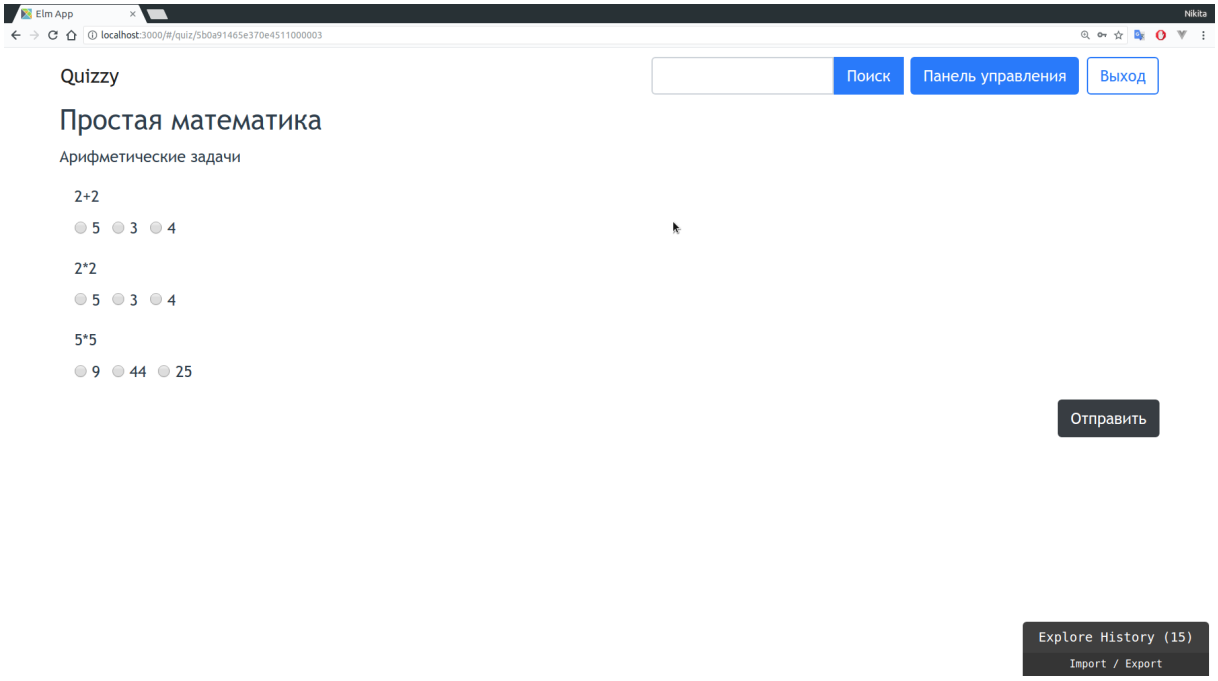

Рисунок 3.4 — Страница прохождения теста

3.4 Выводы

В качестве документоориентированной базы данных была выбрана MongoDB. Серверная часть написана на языке Haskell, с использованием библиотек Servant и Persistent. Клиент написан на языке Elm. Система разработана и протестирована.

Заключение

В ходе данной работы была разработана автоматической тестирующей системы. Пользователь имеет возможность создавать и проходить тесты.

Определены роли пользователей: администратор, создает тесты, и испытуемые, выполняет задания.

Детально были описаны процесса авторизации, регистрации, добавления и прохождения тестов с использование IDEF0 диаграмм.

Выявлены следующие сущности: `Пользователь`, `Тест`, `Результат теста`, `Вопрос`. Пользователь создает и проходит тесты. Тесты содержат вопросы. В результате выполнения теста, пользователь получает результат теста. Для данных сущностей построена ER-диаграмма. Описаны и документированы атрибуты каждой сущности.

С помощью диаграмм последовательностей описаны взаимодействия системы, в следующих действиях: авторизация, регистрация, запрос данных пользователя, изменение имени пользователя, изменение пароля пользователя, изменение электронной почты пользователя, добавление нового теста, прохождение теста, удаление теста, получение списка созданных тестов, поиск тестов.

В качестве базы данных выбрана нерялиционная документоориентированная база данных MongoDB. Были описаны преимуществе данной БД.

Для разработки серверной части использовался язык Haskell. В качестве вебфреймворка использовалась библиотека Servant. Пакет Persistent использовался для взаимодействия с базой данных. Клиентская часть разработана на языке Elm.

Список использованных источников

1. Дейт, К.Дж. Введение в системы баз данных /К.Дж. Дейт. — 8-е изд. — Москва: Вильямс, 2005. — 1328с.

2. Бэнкер Кайл MongoDB в действии /Бэнкер Кайл. — 1-е изд. — Москва: ДМК-пресс, 2017. — 394с.

3. О сервисе Яндекс.Контест. [Электронный ресурс] Режим доступа: https://contest.yandex.ru/about/ , свободный.

4. О сервисе Сертификация Mail.ru. [Электронный ресурс] Режим доступа: https://certification.mail.ru/about/ , свободный.

5. О сервисе Stepik URL. [Электронный ресурс] Режим доступа: https://welcome.stepik.org/ru/about , свободный.

6. Базы данных SQL, NoSQL и различия в моделях баз данных [Электронный ресурс]. Режим доступа: http://devacademy.ru/posts/sql-nosql/, свободный.

7. Миран Липовача Изучай Haskell во имя добра! /Миран Липовача. — ДМК-Пресс, 2017. — 490.

8. Учебник по Haskell [Электронный ресурс]. Режим доступа: https://anton-k.github.io/ru-haskell-book/book/home.html, свободный.

9. servant: A family of combinators for defining webservices APIs [Электронный ресурс]. Режим доступа: https://hackage.haskell.org/package/servant, свободный.

10. servant-server: A family of combinators for defining webservices APIs and serving them [Электронный ресурс]. Режим доступа: https://hackage.haskell.org/package/servant-server, свободный.

11. persistent: Type-safe, multi-backend data serialization. [Электронный ресурс]. Режим доступа: https://hackage.haskell.org/package/persistent, свободный.

12. persistent-mongoDB: Backend for the persistent library using mongoDB. [Электронный ресурс]. Режим доступа: https://hackage.haskell.org/package/persistent-mongoDB, свободный.

13. persistent-template: Type-safe, non-relational, multi-backend persistence. [Электронный ресурс]. Режим доступа: https://hackage.haskell.org/package/persistent-template, свободный.

14. An Introduction to Elm. [Электронный ресурс]. Режим доступа: https://guide.elm-lang.org/, свободный.

15. Elm Tutorial. [Электронный ресурс]. Режим доступа: https://www.elm-tutorial.org/en/, свободный.