

СОДЕРЖАНИЕ

| | |
|--|----|
| АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА | 3 |
| Описание структуры базы данных..... | 4 |
| СТРУКТУРА И ОСНОВНЫЕ АЛГОРИТМЫ | 8 |
| Функциональные возможности и последовательность обработки информации.. | 8 |
| Структура программного комплекса | 10 |
| Работа с <i>LLM</i> в программном комплексе | 12 |
| ГРАФИЧЕСКИЙ ИНТЕРФЕЙС И ВЕРИФИКАЦИЯ | 16 |
| Графическая реализация функций при работе с программным комплексом | 16 |
| Результаты тестирования и верификации программного комплекса | 22 |
| Руководство системного программиста | 27 |
| Руководство программиста..... | 29 |
| Руководство пользователя | 31 |

АРХИТЕКТУРА ПРОГРАММНОГО КОМПЛЕКСА

Приложение имеет клиент-серверную архитектуру взаимодействия. Серверная часть обрабатывает запросы от пользователя, для каждой из сущности выделены различные маршруты для проведения операций.

Для работы с сущностью «Ключ» предусмотрены следующие действия:

- создание ключа;
- редактирование ключа;
- удаление ключа;
- получение всех ключей;
- получение ключа по коду.

Для работы с сущностью «Провайдер» предусмотрены следующие действия:

- создание провайдера;
- редактирование провайдера;
- удаление провайдера;
- получение всех провайдеров;
- получение провайдера по коду.

Для работы с сущностью «Модель» предусмотрены следующие действия:

- создание модели;
- редактирование модели;
- удаление модели;
- получение всех моделей;
- получение модели по коду.

Для работы с сущностью «Курс» предусмотрены следующие действия:

- создание курса;
- редактирование курса;
- удаление курса;
- получение всех курсов;
- получение курса по коду;
- поиск курса по названию;
- получение лабораторных работ курса.

Для работы с сущностью «Промпт» предусмотрены создание и редактирования содержимого, привязанного к конкретному курсу.

Для работы с сущностью «Лабораторная работа» предусмотрены следующие действия:

- создание лабораторной работы;
- редактирование лабораторной работы;
- удаление лабораторной работы по коду.

Для работы с сущностью группа предусмотрены следующие действия:

- создание группы;
- редактирование группы;
- удаление группы;
- получение списка групп;
- получение группы по коду.

Для работы с сущностью «Студент» предусмотрены следующие действия:

- создание студента;
- редактирование студента;
- удаление студента;

- получение студентов по коду группы;
- получение студента по коду.

Для выполнения проверок отчетов по лабораторным работам предусмотрены следующие действия:

- парсинг файла с отчетами – маршрут, который выполняет парсинг архива с отчетами студентов, на выходе формирует список студентов из архива;
- выполнение проверки – маршрут, который выполняет проверку отчетов по лабораторным работам.

Для работы с проверками отчетов предусмотрены следующие действия:

- получение проверки по коду лабораторной работы;
- получение проверок по студенту в рамках заданной лабораторной работы.

Клиентская часть представляет из себя веб-приложение, с которым пользователь может взаимодействовать посредством браузера. Для удобной работы пользователя необходимо правильно проработать содержимое веб-страниц приложения.

Для управления ключевыми параметрами системы предусмотрена отдельная страница настроек. На ней пользователь получает возможность гибко конфигурировать список доступных моделей больших языковых моделей, а также провайдеров, через которых осуществляется доступ к этим моделям. Кроме того, на этой же странице происходит управление *API*-ключами. Эти настройки позволяют пользователю задать список моделей

Функционал для работы с курсами реализован на странице «Курсы». Здесь отображается полный список созданных пользователем курсов. Для каждого отдельного курса реализована своя страница, где пользователь может просмотреть список всех привязанных к нему лабораторных работ и промпт, который может использоваться при проверке лабораторных работ данного курса.

Основной процесс выполнения проверок отчетов происходит на странице «Проверки отчетов». На этой странице пользователь выбирает одну или несколько моделей для анализа, а затем загружает файл, представляющий из себя архив с отчетами студентов. Интерфейс страницы позволяет запускать проверку и отслеживать её ход.

Для анализа и контроля над уже выполненными проверками предусмотрена страница «История проверок». На данной странице пользователь должен иметь возможность просматривать результаты прошлых проверок.

Описание структуры базы данных

Для представления провайдера реализована сущность «Провайдер», которая содержит поля «Код», «Название», «Адрес». Описание сущности «Провайдер» представлено в таблице 1.

Таблица 1 – Таблица с описанием полей сущности «Провайдер»

| Поле | Тип | Пояснение |
|----------|-----------|--------------------------------|
| Код | Числовой | Код для идентификации сущности |
| Название | Строковой | Название провайдера |
| Адрес | Строковой | URL-адрес провайдера |

Для представления ключа представлена сущность «Ключ», которая содержит поля «Код», «Название», «Значение». Описание сущности «Ключ» представлено в таблице 2.

Таблица 2 – Таблица с описанием полей сущности «Ключ»

| Поле | Тип | Пояснение |
|----------|-----------|--------------------------------|
| Код | Числовой | Код для идентификации сущности |
| Название | Строковой | Название ключа |
| Значение | Строковой | Значение API-ключа |

Для представления модели представлена сущность «Модель», которая содержит поля «Код», «Название», «Значение», «Температура», «Максимальный размер», «Интерфейс», «Баланс креативности», «Код ключа», «Код провайдера». Описание сущности «Ключ» представлено в таблице 3.

Таблица 3 – Таблица с описанием полей сущности «Модель»

| Поле | Тип | Пояснение |
|---------------------|-----------|-----------------------------------|
| Код | Числовой | Код для идентификации сущности |
| Название | Строковой | Название модели |
| Значение | Строковой | Значение модели |
| Температура | Числовой | Случайность ответов |
| Баланс креативности | Числовой | Отвечает за «креативность» модели |
| Интерфейс | Строковой | Интерфейс для работы модели |
| Максимальный размер | Числовой | Максимальный размер ответа |
| Код ключа | Числовой | Код ключа |
| Код провайдера | Числовой | Код провайдера |

Для представления студента представлена сущность «Студент», которая содержит поля «Код», «Имя», «Фамилия», «Отчество». Описание сущности «Студент» представлено в таблице 4.

Таблица 4 – Таблица с описанием полей сущности «Студент»

| Поле | Тип | Пояснение |
|----------|-----------|--------------------------------|
| Код | Числовой | Код для идентификации сущности |
| Имя | Строковой | Имя студента |
| Фамилия | Строковой | Фамилия студента |
| Отчество | Строковой | Отчество студента |

Для представления промпта представлена сущность «Промпт», которая содержит поля «Код», «Значение», «Код курса». Описание сущности «Промпт» представлено в таблице 5

Таблица 5 – Таблица с описанием полей сущности «Промпт»

| Поле | Тип | Пояснение |
|-----------|-----------|--------------------------------|
| Код | Числовой | Код для идентификации сущности |
| Значение | Строковой | Текст промпта |
| Код курса | Числовой | Код курса |

Для представления курса представлена сущность «Курс», которая содержит поля «Код», «Название», «Описание». Описание сущности «Курс» представлено в таблице 6.

Таблица 6 – Таблица с описанием полей сущности «Курс»

| Поле | Тип | Пояснение |
|----------|-----------|--------------------------------|
| Код | Числовой | Код для идентификации сущности |
| Название | Строковой | Название курса |
| Описание | Строковой | Описание курса |

Для представления группы представлена сущность «Группа», которая содержит поля «Код», «Название». Описание сущности «Группа» представлено в таблице 7.

Таблица 7 – Таблица с описанием полей сущности «Группа»

| Поле | Тип | Пояснение |
|----------|-----------|--------------------------------|
| Код | Числовой | Код для идентификации сущности |
| Название | Строковой | Название группы |

Для представления лабораторной работы представлена сущность «Лабораторная работа», которая содержит поля «Код», «Название», «Описание», «Название файла», «Размер файла», «Содержимое», «Код курса». Описание сущности «Лабораторная работа» представлено в таблице 8.

Таблица 8 – Таблица с описанием полей сущности «Лабораторная работа»

| Поле | Тип | Пояснение |
|----------------|-----------|---|
| Код | Числовой | Код для идентификации сущности |
| Название | Строковой | Название лабораторной работы |
| Описание | Строковой | Описание лабораторной работы |
| Название файла | Строковой | Название файле лабораторной работы |
| Размер файла | Строковой | Размер файла лабораторной работы |
| Содержимое | Строковой | Текст лабораторной работы |
| Код курса | Числовой | Код курса, к которому привязана лабораторная работа |

Для представления проверки представлена сущность «Проверка», которая содержит поля «Код», «Плюсы», «Минусы», «Оценка», «Отзыв», «Отчет», «Дата», «Код лабораторной», «Код студента», «Код модели». Описание сущности «Проверка» представлено в таблице 9.

Таблица 9 – Таблица с описанием полей сущности «Проверка»

| Поле | Тип | Пояснение |
|--------|-----------|--|
| Код | Числовой | Код для идентификации сущности |
| Плюсы | Строковой | Положительные моменты, выявленные при проверке работы студента |
| Минусы | Строковой | Отрицательные моменты, выявленные при проверке работы студента |
| Оценка | Строковой | Оценка работы |
| Отзыв | Строковой | Краткий отзыв на работу студента |

| Поле | Тип | Пояснение |
|------------------|-----------|---|
| Отчет | Строковой | Текст отчета студента |
| Дата | Дата | Дата проверки работы |
| Код студента | Числовой | Код студента, работа которого проверялась |
| Код лабораторной | Числовой | Код проверенной лабораторной работы |
| Код модели | Числовой | Код модели, которая выполняла проверку |

Таким образом, описание сущностей полностью соответствует предметной области и в полной мере описывает поля сущностей. Схема базы данных представлена на рисунке 2.4.

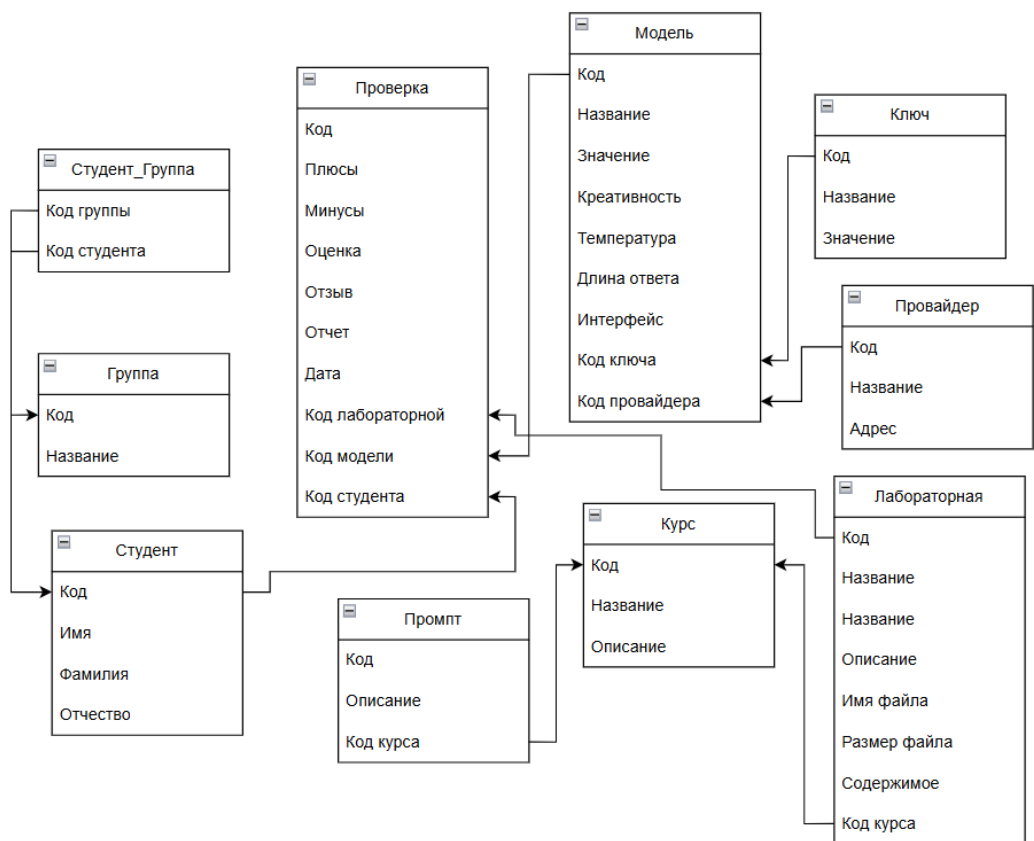


Рисунок 2.4 – Схема базы данных

Таким образом, спроектированная логическая схема базы данных в полной мере отображает сущности предметной области разрабатываемого программного комплекса.

Функциональные возможности и последовательность обработки информации

Основная цель программного комплекса – автоматизированный процесс проверки отчетов студентов по лабораторным работам с помощью больших языковых моделей. Современные языковые модели способны анализировать текст и выявлять в нем ошибки и недочеты.

Стоит понимать, что *LLM* способны принимать данные исключительно в текстовом формате, так что перед проверкой отчет студента по лабораторной работе должен быть преобразован в текстовый формат. Для проверки отчета пишется специальный запрос определенного формата. Данный промпт содержит в себе разделы с заданием и отчетом студента.

После представления промпта модели модель выполняет проверку отчета студента. Модель предоставляет ответ со следующими полями:

- оценка – оценка отчета студента по десятибалльной шкале;
- положительные моменты в работе студента;
- отрицательные моменты и недочеты в работе студента;
- отзыв – краткий отзыв на работу студента;

Схема процесса разбора и проверки отчета моделью представлена на рисунке 1.



Рисунок 1 – Схема преобразование отчета и формирования промпта

Затруднения могут возникнуть, если в отчете содержится графический материал, так как модели не могут анализировать содержимое изображений и соотносить его с представленным текстом. Но это не значит, что модели не могут проверять такие работы, при проверке подобного отчета потеряется лишь небольшая часть информации.

Сервисы, предоставляющие доступ к *LLM*-моделям, могут быть недоступны по разным причинам или иметь ограничения на количество запросов за период времени, поэтому при неудачной проверке, если пришел пустой ответ или же запрос не прошел, выполняется повторный запрос.

Модели также имеют различные интерфейсы взаимодействия. Если модель локальная, нужен один интерфейс взаимодействия, если удаленная, предоставляемая некоторым сервисом, то иной интерфейс.

Схема работы запросов к модели представлена на рисунке 2.

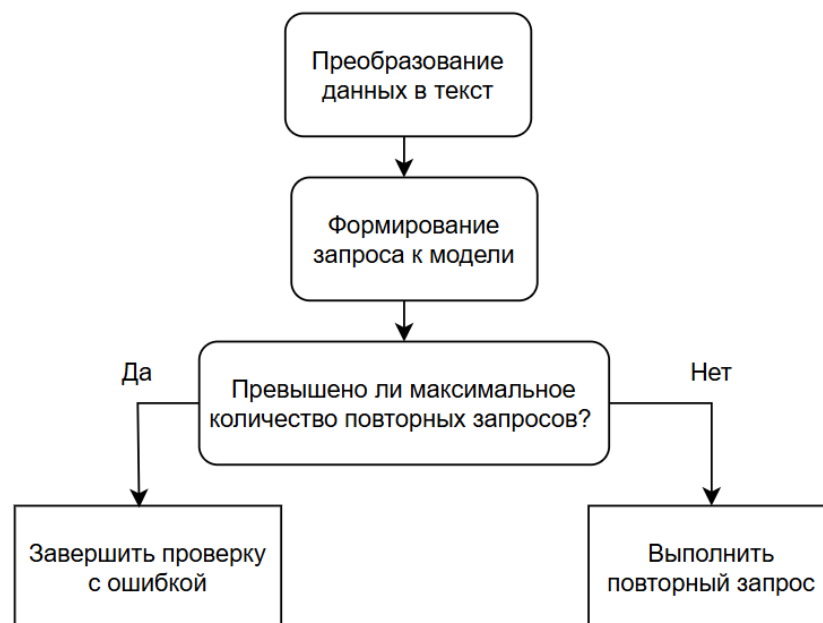


Рисунок 2 – Схема отправки запросов к модели

Если работа студента была ранее проверена и имела выявленные недостатки, то выполняется повторная проверка, в которой берется результат предыдущей проверки, если такой существует, и формируется промпт, который содержит старую версию отчета и новую, затем выполняется проверка отчетов моделью и сравнение отчетов, если были исправлены недочеты, модель повышает оценку, если же нет, то указывает недостатки и требует от студента повторной переработки отчета.

Для более эффективной проверки отчетов также по выбору выполняется проверка с использованием нескольких моделей. Один и тот же отчет проверяется разными моделями, и по итогу проверки формируется сводная рецензия на отчет заранее выбранной моделью.

Такой подход имеет свои плюсы, так как модели довольно часто акцентируют внимание на разных проблемах даже в рамках одного и того же отчета, также стоит иметь в виду, что от степени сложности модели зависит и результат проверки. Так как отчет проверяется несколько раз разными моделями, то оценка будет более объективной и затронет разные стороны работы студента. Схема проверки отчета студента несколькими моделями представлена на рисунке 3

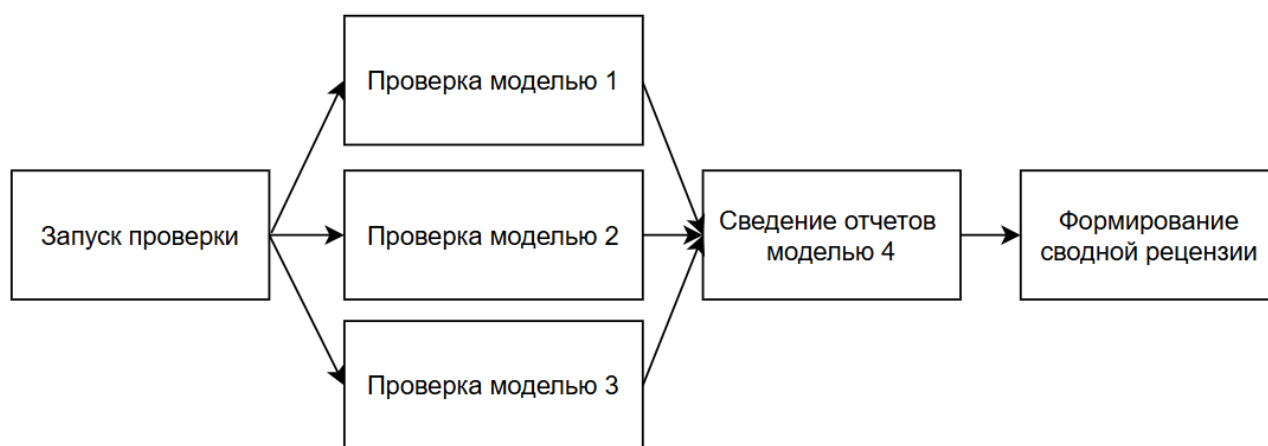


Рисунок 3 – Схема проверки несколькими моделями

Структура программного комплекса

Класс *ProviderService* предназначен для управления сущностями провайдеров в базе данных. Данный класс содержит следующие методы:

- *findMany* – осуществляет поиск и возвращает все записи провайдеров;
- *findOne* – находит и возвращает одну запись провайдера по указанному уникальному идентификатору;
- *create* – создает новую запись провайдера на основе переданного объекта *dto* (Data Transfer Object) и сохраняет её в базе данных;
- *update* – обновляет существующую запись провайдера;

Класс *KeyService* отвечает за выполнение операций с сущностями ключей *API* в системе. Данный класс содержит следующие методы:

- *findOne* – находит ключ по уникальному идентификатору;
- *findMany* – возвращает все записи ключей;
- *create* – создает новую запись ключа;
- *update* – обновляет существующую запись ключа;
- *delete* – асинхронно удаляет запись ключа.

Класс *ModelService* предоставляет функционал для управления сущностями моделей, которые используются для взаимодействия с различными интерфейсами *LLM*. Данный класс содержит следующие методы:

- *findByIds* – находит и возвращает список моделей по заданному массиву идентификаторов;
- *findOne* – извлекает одну запись модели по её уникальному идентификатору;
- *findMany* – возвращает все записи моделей, доступные в репозитории, включая связанные данные о ключах и провайдерах;
- *create* – является универсальным методом для создания моделей, создает модель на основе ее типа;
- *update* – универсальный метод для обновления моделей, обновляет модель на основе ее типа;
- *delete* – удаляет запись модели из базы данных.

Класс *GroupService* занимается управлением сущностями групп. Данный класс содержит следующие методы:

- *findOne* – находит и возвращает одну запись группы по указанному идентификатору;
- *findMany* – возвращает все записи групп;
- *create* – создает новую запись группы;
- *update* – обновляет существующую запись группы;
- *delete* – асинхронно удаляет запись группы из базы данных;
- *isMember* – проверяет, является ли студент членом указанной группы;
- *addMember* – добавляет студента в группу;
- *removeMember* – удаляет студента из группы;
- *getGroupStudents* – возвращает список студентов, принадлежащих указанной группе.

Класс *StudentService* управляет всеми операциями, связанными с сущностями студентов. Данный класс содержит следующие методы:

- *findByIds* – находит и возвращает список студентов по заданному массиву идентификаторов;

- *findRawStudent* – ищет и возвращает одного студента по полному совпадению имени, фамилии и отчества;
- *findOne* – находит и возвращает одну запись студента по его уникальному идентификатору;
- *findMany* – возвращает все записи студентов;
- *searchStudents* – осуществляет поиск студентов с пагинацией и фильтрацией;
- *create* – создает новую запись студента;
- *update* – обновляет существующую запись студента;
- *delete* – удаляет запись студента из базы данных.

Класс *LabService* управляет сущностями лабораторных работ. Данный класс содержит следующие методы:

- *findOne* – находит и возвращает одну запись лабораторной работы;
- *findMany* – возвращает все записи лабораторных работ;
- *prepareFile* – обрабатывает загруженный файл;
- *create* – создает новую запись лабораторной работы;
- *update* – обновляет существующую запись лабораторной работы;
- *delete* – удаляет запись лабораторной работы из базы данных;

Класс *CheckService* предназначен для управления всеми операциями, связанными с проверками отчетов. Данный класс содержит следующие методы:

- *getStudentChecks* – получает все записи проверок студентов;
- *getByIds* – находит и возвращает список проверок по заданному массиву идентификаторов;
- *findLastCheck* – находит и возвращает последнюю проверку для указанного студента;
- *create* – создает новую запись проверки;
- *getLabChecks* – получает и группирует все проверки для конкретной лабораторной работы;
- *findByLabs* – находит все проверки, связанные с конкретной лабораторной работой;
- *findByStudent* – находит проверки по идентификатору лабораторной работы и идентификатору студента.

Класс *CourseService* предназначен для управления сущностями курсов. Данный класс содержит следующие методы:

- *findOne* – находит и возвращает одну запись курса по указанному идентификатору;
- *findMany* – осуществляет поиск курсов с пагинацией и фильтрацией по имени;
- *create* – создает новую запись курса;
- *update* – обновляет существующую запись курса;
- *delete* – удаляет запись курса из базы данных;
- *getLabs* – возвращает список всех лабораторных работ.

Класс *FileService* предоставляет функционал для работы с файловой системой, включая запись файлов, парсинг содержимого архивов и извлечение текста из различных форматов документов. Данный класс содержит следующие методы:

- *writeFile* – записывает строковое содержимое в файл по указанному пути;
- *parseStudentsFromFile* – обрабатывает ZIP-архив, извлекая из него информацию о студентах;
- *parseArchive* – распаковывает ZIP-архив во временную директорию, вызывает *processFolders* для дальнейшей обработки содержимого;

– *processFolders* – обрабатывает содержимое заданной корневой директории. Метод сканирует поддиректории, и для каждой из них, если имя папки соответствует определенному формату, извлекает информацию о студенте и содержимое файлов внутри папки;

– *parseFolderName* – парсит имя папки (строку) для извлечения фамилии, имени, отчества и номера;

– *extractFolderContent* – извлекает текстовое содержимое из первого найденного *PDF*, *DOCX* или *DOC* файла в указанной папке;

– *parseFile* – извлекает текстовое содержимое из буфера файла в зависимости от его расширения. Поддерживает форматы *PDF*, *DOCX*, а для других типов файлов возвращает содержимое буфера как *UTF-8* строку;

– *extractPDF* – извлекает текст из *PDF*-файла, представленного в виде буфера;

– *extractDOCX* – извлекает необработанный текст из *DOCX*-файла, представленного в виде буфера.

Класс *ReportsService* является сервисом для управления процессами проверки отчетов студентов с использованием больших языковых моделей. Данный класс содержит следующие методы:

– *parseStudentsFile* – выполняет парсинг архива с отчетами студентов;

– *getLabChecks* – получает все проверки, связанные с указанной лабораторной работой;

– *getChecks* – получает записи проверок по списку их идентификаторов;

– *handleCheckReports* – определяет, каким образом будет производиться проверка отчетов: одной моделью или несколькими;

Работа с *LLM* в программном комплексе

Провайдер представляет собой сущность, которая является удаленным сервисом, который предоставляет доступ к *LLM*. Каждый провайдер имеет свои параметры подключения, а именно название и *URL*-адрес провайдера.

Доступ к большинству удаленных *LLM*-сервисов защищен с помощью ключей. Эти ключи подтверждают, что запрос пользователя имеет право на использование ресурсов провайдера. Ключ *API* применяется для удаленных провайдеров, поскольку именно для доступа и работы к удаленному сервису требуется такого рода работа.

За управление ключами отвечает сервис *KeyService*. Он хранит ключи в безопасном месте, привязывает их к конкретным моделям и провайдерам. Без правильно настроенного ключа многие модели не смогут работать, поэтому при создании ключей нужно убедиться, что ключи рабочие.

Каждая модель обладает определенными характеристиками, такими как её название, значение, а также параметры температуры и *top_p*. Эти параметры влияют на то, насколько «креативным» или, наоборот, предсказуемым будет ответ модели. Чем выше температура, тем более разнообразными могут быть ответы.

Также каждая модель имеет свой специфический интерфейс взаимодействия *llmInterface*, который определяет, как именно система будет с ней взаимодействовать. Все операции, связанные с моделями, находятся в сервисе *ModelService*. Есть два интерфейса для работы с моделями. Это интерфейсы *OpenAI* и *Ollama*.

OpenAI-совместимый интерфейс используется для моделей, которые доступны через удаленные сервисы и требуют *API*-ключей и провайдеров для доступа. Примерами могут служить модели от самого *OpenAI*, а также другие модели, которые предоставляют совместимые *API*. Для моделей с этим интерфейсом строго требуется, чтобы

были указаны и ключ *API*, и провайдер, ведь без них связь с моделью попросту невозможна.

Ollama-интерфейс разработан для взаимодействия с моделями, которые развернуты локально, то есть работают на устройстве пользователя через платформу *Ollama*. Отличительной особенностью таких моделей является то, что им не требуются отдельные *API*-ключи или сторонние провайдеры для доступа. Для их настройки достаточно указать лишь базовые параметры модели.

Сервис *ModelService* обрабатывает эти различия. Когда пользователь создает или обновляет модель, проверяется указанный интерфейс. Если это *OpenAI*-совместимая модель, произойдет проверка того, что был предоставлен и ключ, и провайдер. Если же это *Ollama*-модель, то любая попытка привязать ключ или провайдер будет отклонена, чтобы избежать путаницы и ошибок.

Основной сервис, отвечающий за взаимодействие с *LLM*, это *LlmService*. Его главная задача управление процессом получения ответов и их обработка.

LlmService занимается непосредственным взаимодействием с конкретными провайдерами *LLM* через специализированных обработчиков, реализующим интерфейс *ILlmProviderHandler*. Для создания нужного обработчика используется *LlmProviderFactory*. Этот фабричный сервис, в зависимости от типа интерфейса, предоставляет соответствующий обработчик: *OpenAiHandler* для работы с *API OpenAI* и *OllamaHandler* для взаимодействия с локально развернутыми моделями через *Ollama*.

OpenAiHandler отвечает за формирование запросов к *OpenAI*, используя базовый *URL* провайдера и *API*-ключ, привязанные к конкретной модели. Он также обрабатывает специфичные для *OpenAI* ошибки, преобразуя *APIError* со статусом 400 в стандартную ошибку.

OllamaHandler использует библиотеку *Ollama* для взаимодействия с локальными моделями и имеет свою логику обработки ошибок, специфичных для этого провайдера.

Метод *query* сервиса *LlmService* инкапсулирует логику повторных попыток и задержек. Он использует *LlmProviderFactory* для получения нужного обработчика и затем вызывает его метод *completion*. Если вызов *completion* завершается ошибкой или возвращает пустой ответ, *LlmService* не прекращает работу немедленно. Вместо этого он выполняет следующие шаги:

- вызывает метод *processError* у соответствующего обработчика для обработки ошибки;
- логирует предупреждение;
- вызывает задержку перед повторным выполнением запроса;
- повторяет запрос.

Этот процесс повторяется до тех пор, пока не будет достигнуто максимальное количество попыток *maxRetries*, заданное в настройках модели.

После успешного получения непустого ответа от *LLM* через один из обработчиков, *LlmService* использует метод *extractData* для извлечения данных. Этот метод ищет в текстовом ответе специальный маркер – блок *JSON*, заключенный в теги «*<JSON>*» и «*</JSON>*».

Если блок найден, он извлекается, парсится и проходит валидацию с использованием *class-transformer* и *class-validator* на соответствие заранее определенному формату *DTO*. В случае, если *JSON*-блок отсутствует, некорректно сформирован или не проходит валидацию, метод *extractData* выбрасывает ошибку.

Когда преподаватель начинает проверку, он загружает *ZIP*-архив, содержащий отчеты студентов. Приложение принимает этот архив и параметры проверки. Эти параметры включают идентификатор лабораторной работы, выбранные *LLM*-модели для проверки (может быть одна или несколько), а также список конкретных студентов для проверки и параметр, указывающий, нужно ли учитывать предыдущие проверки этого студента.

Метод *handleCheckReports* построен так, что выполняет проверку в фоновом режиме с помощью вызова специальной функции *setImmediate*. Перед началом основной работы отправляется уведомление о том, что процесс проверки запущен. Если в ходе всего процесса возникнет непредвиденная ошибка, система также отправит уведомление о том, что не удалось проверить отчеты студентов.

FileService занимается парсингом загруженного *ZIP*-архива. Ожидается, что внутри архива каждый отчет студента находится в отдельной папке, название которой содержит информацию о студенте (фамилия, имя, отчество). *FileService* извлекает эти данные из имен папок.

Далее, из каждой такой папки извлекается файл отчета. *FileService* читает содержимое этих файлов, преобразуя его в обычный текст. Для *PDF* используется библиотека *pdf-parse*, а для *DOCX* – *mammoth*. Если файл имеет другое расширение, его содержимое просто читается как текстовый файл в кодировке *UTF-8*.

Используя извлеченные из имен папок ФИО, *StudentService* ищет совпадения в базе данных. Если студент не найден, для него создается новая запись в системе.

При проверке сначала подготавливается промпт – текстовый запрос к *LLM*. *PromptService* формирует промпт на основе содержимого отчета студента, текста задания лабораторной работы, текста промпта, привязанного к курсу, к которому относится лабораторная работа. Если был установлен флаг *checkPrev* (учитывать предыдущие проверки) и для студента существует предыдущая оценка по этой работе, *PromptService* изменяет промпт, добавив в него информацию о предыдущей проверке.

Сформированный промпт вместе с информацией о выбранной *LLM*-модели передается в *LlmService*. Этот сервис отвечает за непосредственное взаимодействие с *LLM*. Он поддерживает разные типы интерфейсов *LLM*, такие как *OpenAI* и *Ollama*. В зависимости от модели *LlmService* отправляет запрос к соответствующему *API*.

Если *LLM* не отвечает или возвращает пустой результат, *LlmService* делает несколько попыток с задержками (10-20 секунд), прежде чем окончательно признать запрос неудачным. Каждый ответ от *LLM* записывается в лог-файл и также выводит сообщения об ошибках, успешного выполнения запросов, предупреждениях и так далее.

После получения ответа от *LLM*, который представляет собой текстовую строку, *LlmService* извлекает из него данные. В промпте указывается, что *LLM* должна возвращать результат в формате *JSON*, обернутом в теги «<JSON>». *LlmService* находит этот блок, парсит его и преобразует в объект с предопределенными полями:

- оценка по десятибалльной шкале (минимум 4 балла);
- краткий отзыв на работу;
- положительные работы;
- отрицательные моменты работы.

На каждом этапе проверки одного отчета через *NotificationService* приложение отправляет пользователю информацию о ходе прогресса проверки отчетов студентов. Данный сервис содержит следующие методы для отправки уведомлений:

- *reportOneChecked* – уведомляет пользователя о том, что один отчет успешно проверен;

- *reportOneStarted* – уведомляет пользователя о начале обработки одного отчета;
- *reportOneFailed* – уведомляет пользователя о неудачной обработке одного отчета;
- *reportsChecked* – уведомляет пользователя о том, что несколько отчетов были проверены;
- *checkStarted* – уведомляет пользователя о начале процесса проверки отчетов;
- *checkFailed* – уведомляет пользователя о неудаче общего процесса проверки.

ReportsService является сервисом, который занимается проверкой отчетов по лабораторным работам. Методом, выполняющим проверку, является метод *handleCheckReports*. Он получает *CheckReportDto* со всеми необходимыми данными и выполняет проверку.

Первым шагом он обращается к *ReportStrategy*, чтобы получить конкретную стратегию проверки, основываясь на количестве переданных моделей. Затем он использует *setImmediate* для асинхронного запуска метода *check* выбранной стратегии.

Внутри этой асинхронной функции он управляет уведомлениями через *NotificationService*, сообщая о начале проверки, ее успешном завершении или о возникшей ошибке. Кроме того, *ReportsService* предоставляет вспомогательные методы, такие как *parseStudentsFile* для разбора файла со студентами и *getLabChecks*, *getChecks* для получения информации о ранее выполненных проверках, делегируя эту работу *FileService* и *CheckService* соответственно.

Сервис *ReportStrategy* выполняет роль фабрики, реализуя паттерн «Фабричный метод» для выбора способа проверки. Его единственный публичный метод, *getStrategy*, принимает число – количество моделей. Если это число равно или больше двух, он, используя *ModuleRef* для получения зависимостей, возвращает экземпляр *MultipleModelStrategy*. В противном случае, если модель всего одна, возвращается *OneModelStrategy*.

OneModelStrategy предназначена для сценария с одной моделью. Ее метод *prepareCheckData* загружает все необходимые данные: лабораторную, модель, задание и отчеты. Затем метод *check* создает массив промисов, где каждый промис – это вызов *reportCheck.checkOneReport* для одного отчета. Эти промисы выполняются параллельно с помощью *Promise.allSettled*. Результаты обрабатываются *reportCheck.filterSuccessResults*, и успешные проверки сохраняются через *reportCheck.createChecks*.

MultipleModelStrategy реализует проверку через несколько моделей. Сначала ее метод *prepareCheckData* загружает данные лабораторной, всех моделей и отчеты. Затем метод *check* организует первый этап: для каждой модели создается массив промисов *reportCheck.checkOneReport* для всех отчетов.

Эти массивы промисов также выполняются параллельно. После завершения первого этапа метод *prepareMultipleData* собирает все успешные результаты и группирует их по студентам, формируя для каждого студента массив его проверок разными моделями.

На втором этапе метод *combineCheckResults* итерирует по этим сгруппированным данным и для каждого студента вызывает *combineCheckResult*. Этот метод, в свою очередь, формирует новый, «сводный» промис через *promptService.prepareMultiplePrompt*. Полученный итоговый ответ извлекается и формируется финальный *CheckResult*. Наконец, эти финальные результаты сохраняются в базе данных через *reportCheck.createChecks*.

ГРАФИЧЕСКИЙ ИНТЕРФЕЙС И ВЕРИФИКАЦИЯ

Графическая реализация функций при работе с программным комплексом

Пользователь может задавать настройки для моделей, провайдеров, ключей на странице «Настройки». Пользователю представлено меню навигации в левом углу экрана. Для того, чтобы перейти на страницу настроек, пользователь должен нажать на кнопку «Настройки», после чего произойдет переход на страницу настроек. Скриншот вида меню навигации представлен на рисунке 1.

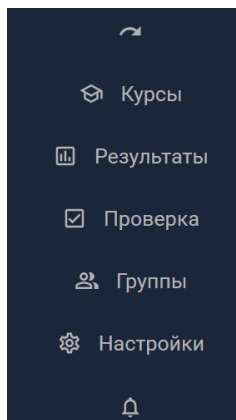


Рисунок 1 – Меню навигации

На странице настроек пользователю представлен функционал для задания настроек, пользователь задает модели, ключа, провайдеров для дальнейшей работы в приложении. Скриншот вида страницы настроек представлен на рисунке 2.

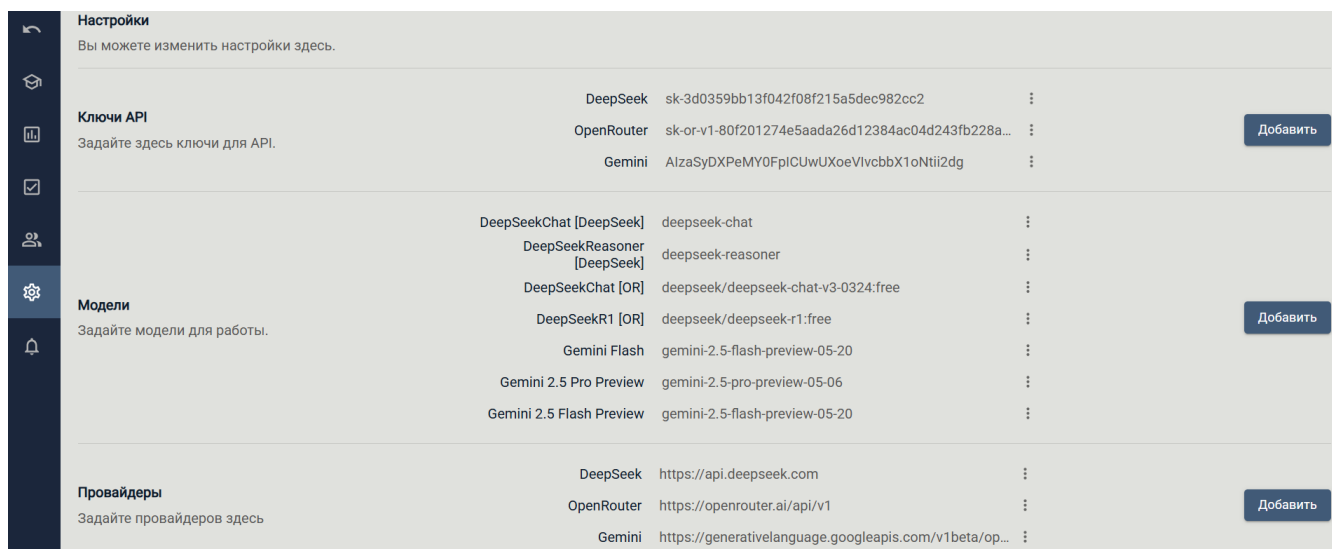


Рисунок 2 – Страница настроек

Пользователь может задать провайдера (настройки удаленного сервиса, который предоставляет работу с *LLM*). Он может изменить как существующий, так и добавить новый. Для того, чтобы задать провайдера, пользователь должен нажать кнопку «Добавить», после чего откроется окно для создания провайдера. Внешний вид окна для добавления провайдера представлен на рисунке 3.

Рисунок 3 – Добавление провайдера

Настройки ключа *API* пользователь задает в секции «Ключи *API*». Новый ключ пользователь может создать, нажав на кнопку «Добавить» в секции «Ключи *API*». Внешний вид окна для добавления ключа представлен на рисунке 4.

Рисунок 4 – Добавление ключа

Новую модель пользователь может добавить в секции «Модели», для этого он должен нажать кнопку «Добавить», после чего откроется окно для добавления модели. В данном окне пользователь должен задать название, значение, температуру, максимальный размер, креативность, провайдера для модели. Также дополнительно может задать ключ и провайдера, если интерфейс модели *OpenAI*, если же модель локальная (работа с интерфейсом *Ollama*), то задавать ключ и провайдера нет нужды. Внешний вид окна для создания модели представлен на рисунке 5.

Рисунок 5 – Добавление модели

На странице курсов пользователь может создавать учебные дисциплины (курсы), в рамках которых он может создавать лабораторные работы и выполнять проверки лабораторных работ. Для того, чтобы перейти на страницу курсов, пользователь должен нажать

кнопку «Курсы» в меню навигации, после чего произойдет переход на страницу курсов. Внешний вид страницы курсов представлен на рисунке 6.

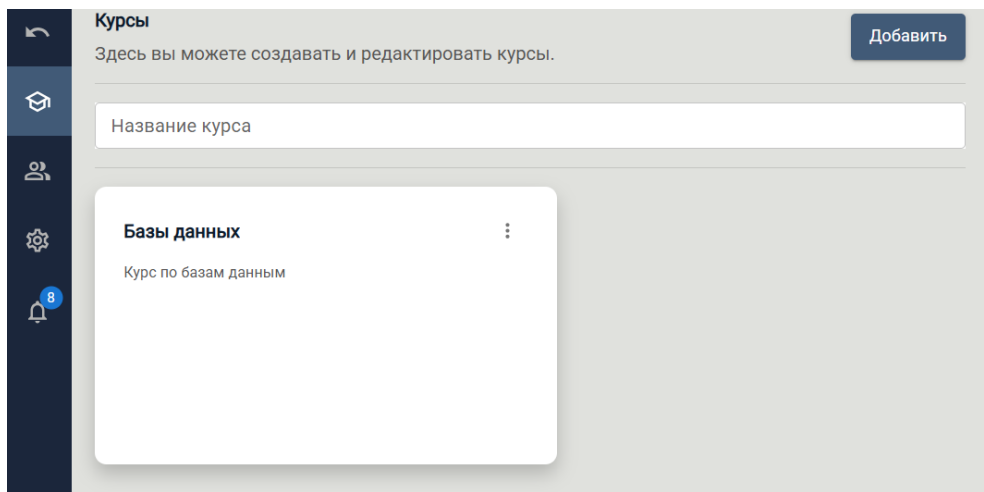


Рисунок 6 – Страница курсов

Для того, чтобы создать курс, пользователь должен нажать кнопку «Добавить», после чего откроется окно для добавления курса. В данном окне пользователь должен ввести название курса и его описание, после чего нажать кнопку «Создать», после чего произойдет создание курса. Внешний вид окна для создания курса представлен на рисунке 7.

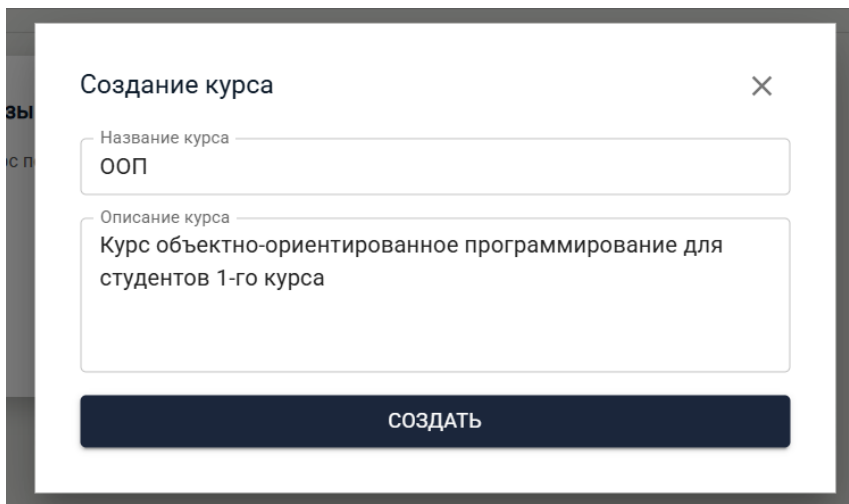


Рисунок 7 – Окно создания курса

Пользователь может перейти на страницу курса, на которой отображается информация о курсе, промпт, привязанный к курсу, список лабораторных работ, относящихся к курсу. Для этого пользователь должен нажать на название курса, после чего произойдет переход на страницу курса. Внешний вид страницы курса представлен на рисунке 8.

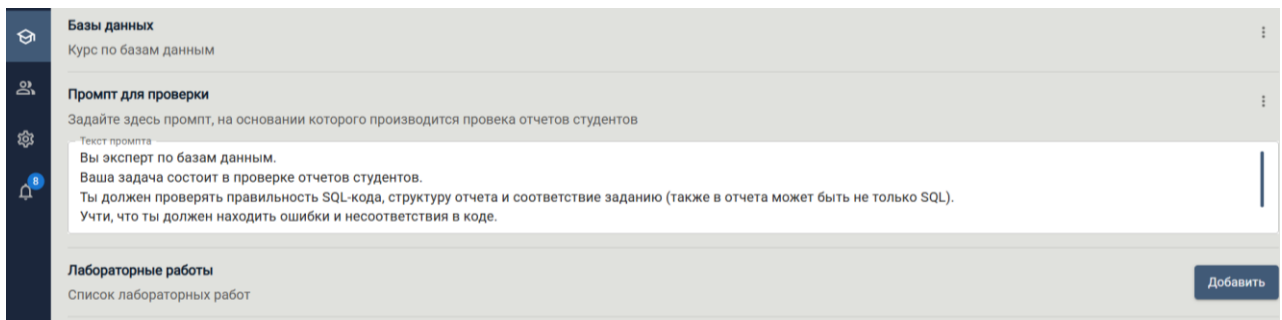


Рисунок 8 – Страница курса

Пользователь также может задать промпт, для этого он должен нажать на кнопку с иконкой трех точек, после чего откроется контекстное меню, в котором он должен нажать кнопку «Редактировать», после чего промпт перейдет в режим редактирования. Внешний вид секции с редактированием промпта представлен на рисунке 9.

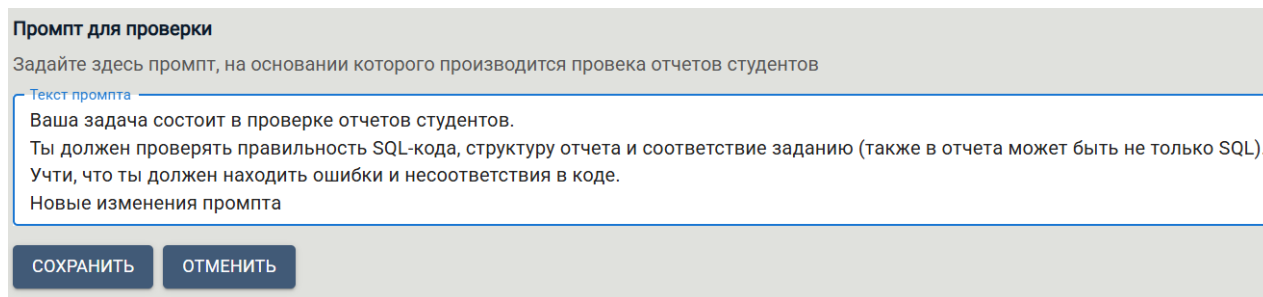


Рисунок 9 – Редактирование промпта

Проверка отчетов в системе выполняется тогда, когда заданы модели, имеется созданный курс и лабораторные работы с заданиями, так что перед началом проверки пользователь должен убедиться в том, что он выполнил все действия, указанные выше.

Для того, чтобы добавить лабораторную работу, пользователь должен нажать кнопку «Добавить» в секции лабораторных работ на странице курса, после чего откроется окно для добавления лабораторной работы. В данном окне пользователь должен задать название и описание лабораторной работы, а также выбрать файл в *docx*, *pdf* или *txt* формате. Внешний вид окна для добавления лабораторной работы представлен на рисунке 10.

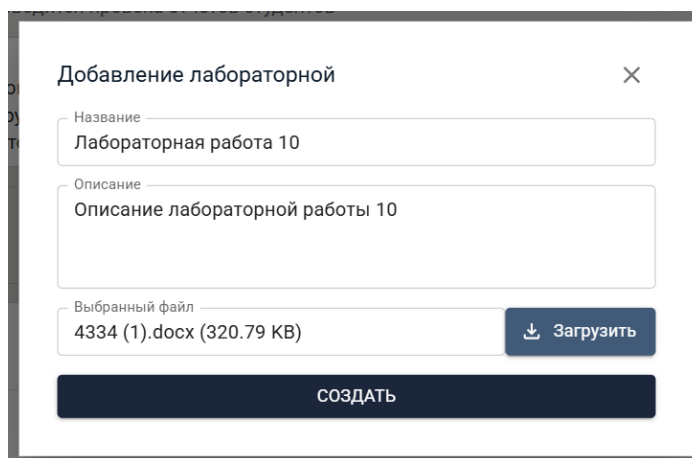


Рисунок 10 – Окно добавления лабораторной работы

Пользователь может просмотреть загруженное задание по лабораторной работе, для этого он должен нажать кнопку с иконкой троеточия, после чего откроется контекстное меню, в котором он должен нажать кнопку «Задание», после чего откроется окно с заданием по лабораторной работе. Внешний вид окна с заданием по лабораторной работе представлен на рисунке 11.

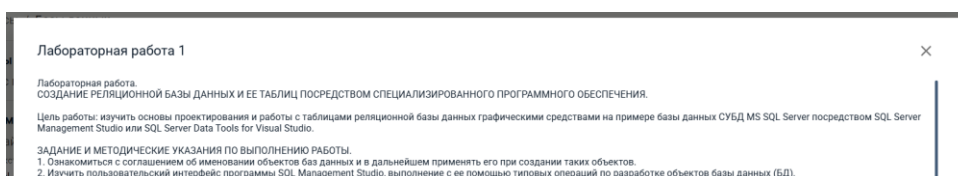


Рисунок 11 – Окно с заданием лабораторной работы

После того, как пользователь в курсе задал промпт, добавил лабораторные работы, он может выполнять проверки лабораторных работ. Для этого он должен нажать на кнопку с иконкой троеточия, после чего откроется контекстное меню, в котором пользователь должен нажать кнопку «Проверка», после чего произойдет переход на страницу для проверки. Внешний вид страницы для выполнения проверки представлен на рисунке 12.

Рисунок 12 – Страница проверки

На данной странице представлена информация о лабораторной работе, элементы для настройки проверки, а именно задание моделей, группа, к которой будут причислены студенты из Zip архива, если ранее не было создано студентов с такими данными, поле для выбора архива, содержащего отчеты студентов. Архив скачивается из системы Moodle, в которой студенты публиковали свои отчеты, и имеет определенную структуру. Стоит отметить, что если пользователь выбрал архив неверного формата, то выведется сообщения об ошибке. Пример такого сообщения представлен на рисунке 13.

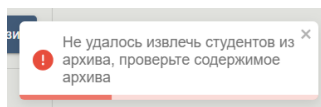


Рисунок 13 – Пример сообщения об неправильном парсинге архива

Если же архив имел правильный формат, то поле для выбора пользователей станет активным и в нем появится возможность выбора списка студентов, чьи отчеты будут проверены в ходе проверки. Внешний вид такого поля представлен на рисунке 14.

Рисунок 14 – Поле для выбора студентов на проверку

В поле «Модели» пользователь задает модель, которая выполняет проверку, для этого он должен нажать на само поле, после чего появится выпадающий список, в котором будет список моделей, которые он может выбрать для проверки. Пример такого заполненного поля представлен на рисунке 15.

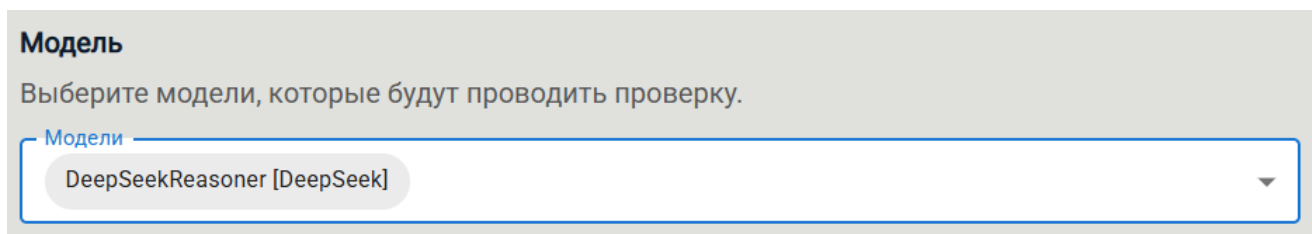


Рисунок 15 – Поле с выбранной моделью

После того, как пользователь задал все параметры, он может начинать процесс проверки отчетов. Для этого он должен нажать кнопку «Отправить», после чего начнется процесс проверки отчетов. Пользователю отобразится сообщение о том, что процесс проверки начался. Пример такого сообщения представлен на рисунке 16.

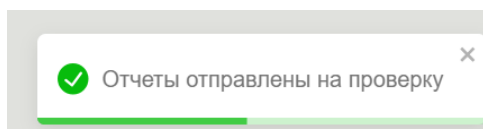


Рисунок 16 – Сообщение о начале проверки

Помимо отображения сообщения о начале проверки, пользователю для удобства в реальном времени показывается прогресс проверки отчетов в секции «Ход проверки». В данной секции отображается список студентов, который отвечает за информацию о ходе проверки отчетов. Если отображается индикатор загрузки, то это обозначает, что проверка еще идет, если же появилась зеленая галочка, то это обозначает, что отчет успешно проверен, если отображается красный крестик, то это обозначает, что проверка завершилась с ошибкой. Внешний вид такой секции при проверке представлен на рисунке 17.

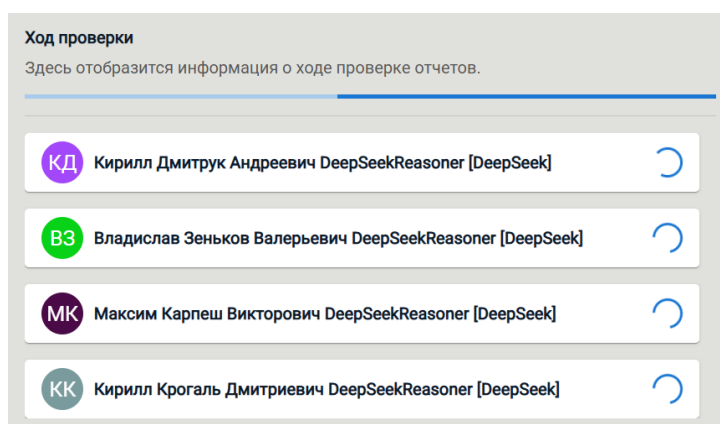


Рисунок 17 – Секция хода проверки

Когда отчеты студентов будут проверены, то индикатор загрузки исчезнет и рядом с элементом списка появится зеленая галочка, которая символизирует о том, что проверка

успешно выполнена. Внешний вид секции с успешно проверенными отчетами представлен на рисунке 18.

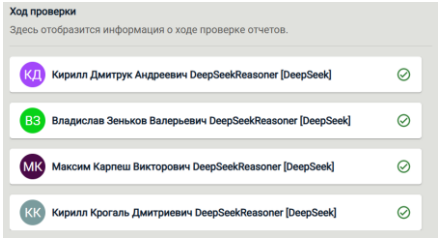


Рисунок 18 – Секций хода проверки с успешной проверкой

Также, когда отчеты будут проверены, пользователю отобразится уведомление о том, что отчеты успешно были проверены. Пример такого сообщения представлен на рисунке 19.

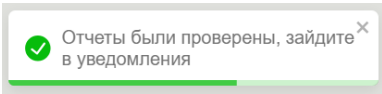


Рисунок 19 – Сообщение об успешной проверке

После того, как пользователь получил сообщение об успешной проверке, он должен перейти на в уведомления и открыть окно с результатами проверки, для этого пользователь в меню навигации должен нажать кнопку с иконкой уведомлений, после чего откроется окно с уведомлениями. После чего пользователь должен нажать кнопку «Перейти», чтобы открыть результаты проверки. Внешний вид окна с результатами проверок представлен на рисунке 20.

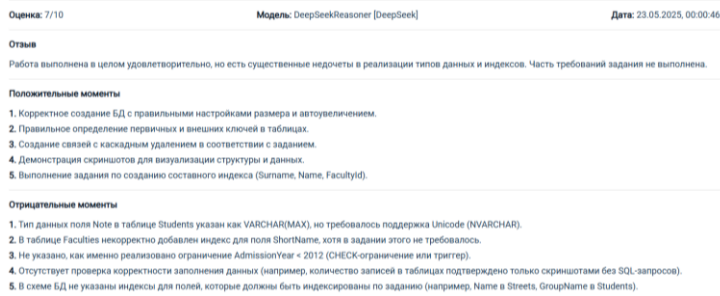


Рисунок 20 – Пример результата проверки отчета

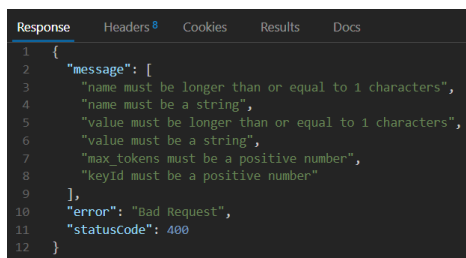
В данном окне отображается информация о проверке, а именно оценка по десяти-балльной шкале, модель, выполнившая проверку, дата проверки. Также отображается краткий отзыв на работу, положительные моменты, недостатки и моменты, требующие доработки.

Результаты тестирования и верификации программного комплекса

В качестве мероприятий тестирования происходила проверка основных эндпоинтов приложения с использованием программы для выполнения *HTTP* запросов *Thunder Client*.

Проверка конечной точки на работу с моделями включало в себя проверку создание модели, так как модели могут иметь разных провайдеров и важно, чтобы все данные были корректно переданы.

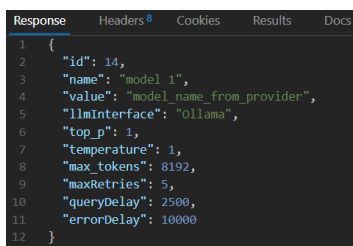
Проверен запрос на создание модели без указания всех необходимых данных, ожидается появление ошибки при запросе. Результат выполнения запроса представлен на рисунке 23.



```
Response Headers Cookies Results Docs
1 {
2   "message": [
3     "name must be longer than or equal to 1 characters",
4     "name must be a string",
5     "value must be longer than or equal to 1 characters",
6     "value must be a string",
7     "max_tokens must be a positive number",
8     "keyId must be a positive number"
9   ],
10  "error": "Bad Request",
11  "statusCode": 400
12 }
```

Рисунок 23 – Запрос на создание модели

Как и ожидалось, в ответ на неправильные данные в результате выполнения запроса произошла ошибка. В дальнейшем произведен запрос на создание модели, но уже с правильными данными. От такого запроса ожидается ответ без ошибки с созданной моделью. Результат выполнения запроса представлен на рисунке 24.

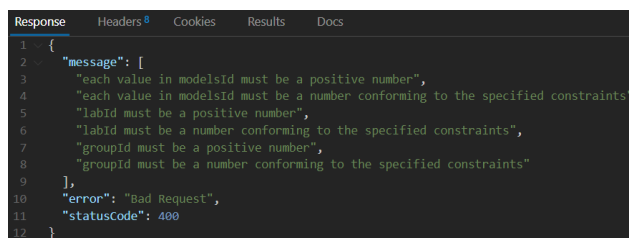


```
Response Headers Cookies Results Docs
1 {
2   "id": 14,
3   "name": "model 1",
4   "value": "model_name_from_provider",
5   "llmInterface": "Ollama",
6   "top_p": 1,
7   "temperature": 1,
8   "max_tokens": 8192,
9   "maxRetries": 5,
10  "queryDelay": 2500,
11  "errorDelay": 10000
12 }
```

Рисунок 24 – Запрос на создание модели

При проверке конечных точек на выполнение проверок проверялся процесс создания отчетов, разбор файла с отчетами, получение отчетов по проверенным работам.

При выполнении запроса на проверку отчетов ожидается, чтобы пользователь передал все необходимые данные. Если же пользователь не передал все данные, то запрос завершится с ошибкой. От запроса с неполными данными ожидается ошибка. Результат выполнения запроса с неполными данными представлен на рисунке 25.



```
Response Headers Cookies Results Docs
1 {
2   "message": [
3     "each value in modelsId must be a positive number",
4     "each value in modelsId must be a number conforming to the specified constraints",
5     "labId must be a positive number",
6     "labId must be a number conforming to the specified constraints",
7     "groupId must be a positive number",
8     "groupId must be a number conforming to the specified constraints"
9   ],
10  "error": "Bad Request",
11  "statusCode": 400
12 }
```

Рисунок 25 – Запрос на выполнение проверки

Запрос завершился с ошибкой, это значит, что система работает корректно. Если пользователь задал все данные верно, то в ответе он получит пустой ответ с статус-кодом 201. Результат выполнения запроса со всеми данными представлен на рисунке 26.

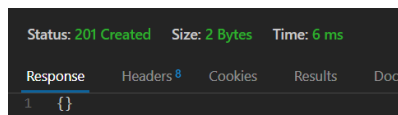


Рисунок 26 – Запрос на выполнение проверки

При отправке запроса на разбор архива с отчетами если файл не представляет из себя архив, то запрос завершится с ошибкой, если же запрос содержит корректный архив, то пользователь получи список студентов из архива. Результат выполнения запроса с некорректным файлом представлен на рисунке 27.

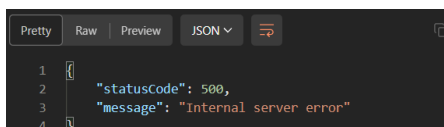


Рисунок 27 – Запрос на разбор файла

Если же архив корректный, то в отчете будет архив со студентами. Результат выполнения корректного запроса представлен на рисунке 28.

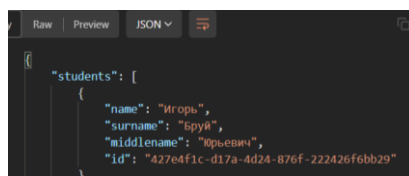


Рисунок 28 – Результат запроса с корректным архивом

При отправке запроса на получение результатов проверки если пользователь задал корректный код лабораторной работы, то в ответе он получит список проверок, иначе ошибку. Результат выполнения запроса с неправильным кодом представлен на рисунке 29

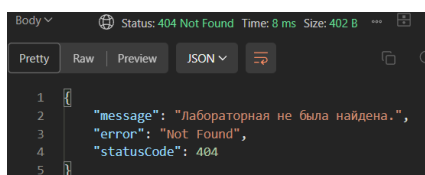


Рисунок 29 – Результат запроса с некорректным кодом

При запросе с корректным кодом пользователь получит в ответ список проверок. Результат выполнения запроса с корректным кодом представлен на рисунке 30.

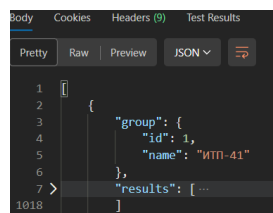


Рисунок 30 – Результат запроса с корректным кодом

В качестве мероприятий по верификации разработанного программного комплекса выполнялась проверка способности приложения по написанию рецензий на отчеты студентов большого размера по сложным работам, включающим в себя написания кода и составление таблиц. Также наиболее важным фактором при проверке будет соответствие реальному результату.

Выполним проверку реальной работы, которая содержит определенные недостатки. Проверяемая работа содержит следующие недостатки:

- нарушение условия задания (не создано нужное количество записей);
- нарушение нормальных форм таблиц;
- наличие заранее заданных данных для запросов в коде (вместо подстановке параметров).

Выполним проверку данной работы, но через обычную не рассуждающую модель. Результаты проверок представлены на рисунке 21.

| | | |
|---|---------------------------------|----------------------------|
| Оценка: 8/10 | Модель: DeepSeekChat [DeepSeek] | Дата: 01.06.2025, 22:46:46 |
| Отзыв Работа выполнена на хорошем уровне, но есть несколько моментов, которые требуют доработки. | | |
| Положительные моменты <ol style="list-style-type: none">1. Полное соответствие заданию по варианту 8 (Радистанция).2. Корректное проектирование базы данных с учетом нормализации до 3NF.3. Наличие всех требуемых хранимых процедур для вставки данных.4. Разработано консольное приложение на C# с использованием ADO.NET.5. Реализованы все запросы и операции, указанные в задании.6. Хорошая структура отчета с выделением разделов. | | |
| Недостатки и моменты, требующие доработки <ol style="list-style-type: none">1. Не указано, выполнены ли требования по количеству записей (100 для таблиц 'idni' и 10000 для 'mknogi').2. В коде C# не показано использование транзакций при заполнении таблиц.3. Нет примеров вывода результатов запросов в полном объеме (только фрагменты).4. Не хватает проверки на соответствие требованиям к параметрическим запросам.5. В отчете не указано, как именно организована работа с ADO.NET (Connection, Command и т.д.).6. Нет информации о том, как обрабатываются ошибки в приложении. | | |

Рисунок 21 – Результат проверки отчета не рассуждающей моделью

Теперь выполним проверку рассуждающей моделью, которая способна анализировать большие и сложные по структуре отчеты. Результат проверки рассуждающей модели представлен на рисунке 22.

| | | |
|---|-------------------------------------|----------------------------|
| Оценка: 7/10 | Модель: DeepSeekReasoner [DeepSeek] | Дата: 01.06.2025, 22:50:49 |
| Отзыв Работа демонстрирует хорошее понимание принципов проектирования БД и работы с ADO.NET. Основные требования задания выполнены: создана нормализованная структура БД, реализованы хранимые процедуры, консольное приложение корректно использует транзакции и параметризованные запросы. Однако выявлены критические нарушения в части генерации тестовых данных, а также несколько концептуальных ошибок в архитектуре приложения. | | |
| Положительные моменты <ol style="list-style-type: none">1. Правильно спроектирована структура БД: соблюдена 3NF, корректно определены первичные/внешние ключи.2. Полностью реализован набор хранимых процедур для вставки данных с параметризацией.3. Консольное приложение использует транзакции для инициализации данных, что соответствует требованию.4. Корректно реализованы параметризованные запросы для операций с данными.5. Верно выполнены все требуемые варианты запросов (выборки, агрегация, параметрические запросы).6. Логика приложения разделена на классы с соблюдением SRP. | | |
| Недостатки и моменты, требующие доработки <ol style="list-style-type: none">1. КРИТИЧЕСКОЕ НАРУШЕНИЕ: Не соблюдены требования к объему тестовых данных (5 записей вместо 100/10000).2. Отсутствует использование созданных хранимых процедур при вставке данных из C#.3. Нарушен порядок заполнения: не обеспечена предварительная загрузка справочников перед оперативными данными.4. Уязвимость в проверке инициализации: проверяется только таблица RefOtperts, а не все справочники.5. Отсутствует обработка возможных дубликатов при генерации тестовых данных.6. Некорректная логика в запросе 3 (не учитывается связь исполнителей и жанров через записи).7. Неэффективное выполнение запроса 6 (использование MAX вместо PIVOT для кросс-таблицы).8. Отсутствует валидация параметров в C# (риск SQL-инъекций в QueryExecutor). | | |

Рисунок 22 – Результат проверки отчета рассуждающей моделью

Как видно из результата, рассуждающая модель нашла выявила недостатки, соответствующие недостаткам, выявленным человеком, притом больше, чем обычная модель.

Первый вариант проверки происходил через модель «DeepSeek-R1», которая способна выявлять логические взаимосвязи в тексте и анализировать код. Выполним также проверку еще одной рассуждающей моделью под названием «Gemini-2.5 Flash». Результат проверки данной моделью представлен на рисунке 23.

| | |
|--|--|
| <p>Положительные моменты</p> <ol style="list-style-type: none"> 1. Корректное проектирование базы данных (3НФ, первичные и внешние ключи, ограничения целостности). 2. Все требуемые таблицы созданы с правильными типами данных и связями. 3. Все требуемые хранимые процедуры для вставки данных созданы и синтаксически корректны. 4. Все запросы и операции по обработке данных (пункт 3.2) реализованы в С# и логически верны. 5. Правильное использование объектов ADO.NET (Connection, Command, Parameter, DataReader, Transaction). 6. Корректная реализация управления транзакциями для инициализации данных. 7. Хорошая структура С#-приложения с разделением на классы (подключение, инициализация, выполнение запросов). 8. Присутствует обработка ошибок в С#-коде. 9. Реализована проверка наличия данных в таблицах перед инициализацией, что предотвращает повторное заполнение. | |
| <p>Недостатки и моменты, требующие доработки</p> <ol style="list-style-type: none"> 1. **Несоответствие объему данных.** В таблицы вставлено только 5 записей, что грубо нарушает требование о 100 записях для таблиц «один» и 10000 записях для таблиц «многие». 2. **Неиспользование хранимых процедур для инициализации.** В С#-коде для инициализации данных используются прямые SQL-запросы 'INSERT' вместо вызова хранимых процедур, созданных в пункте 2 задания. Это является серьезным нарушением требований. 3. **Неоптимальное использование 'Random'. Объект 'Random' создается внутри цикла в методах инициализации, что может привести к генерации повторяющихся последовательностей чисел при быстром выполнении цикла. 4. **Несоответствие имени строки подключения. Название строки подключения в 'App.config' ('RealEstateAgencyConnection') не соответствует предметной области «Риэлтистинг». 5. **Ограниченность перекрестного запроса. Реализация перекрестного запроса (пункт 6) слишком упрощена (показывает только факт присутствия сотрудника, а не детали расписания) и жестко кодирует имена сотрудников, что снижает его гибкость. | |

Рисунок 23 – Положительные моменты и недостатки, выявленные моделью

Как видно, рассуждающая модель «*Gemini-2.5 Flash*» также отлично справилась с задачей, указав недочеты и положительные моменты, которые включали в себя реальные недостатки и ошибки в коде студента. Отзыв о работе студента данной моделью представлен на рисунке 24.

| | |
|---|--|
| <p>Отзыв</p> <p>Студент продемонстрировал хорошее понимание принципов проектирования баз данных, включая нормализацию и использование первичных и внешних ключей. SQL DDL для создания базы данных и таблиц является корректным и хорошо структурированным. Хранимые процедуры для вставки данных также реализованы правильно. Приложение на С# эффективно использует объекты ADO.NET для взаимодействия с базой данных, включая транзакции, и корректно реализует все указанные запросы и операции по обработке данных. Структура отчета ясна и соответствует требованиям задания. Однако, имеются существенные отклонения от требований задания в фазе инициализации данных. Во-первых, приложение вставляет только 5 записей в каждую таблицу, что прямо противоречит требованию о наличии не менее 100 записей для таблиц на стороне «один» и не менее 10000 записей для таблиц на стороне «многие». Это критическое упущение, поскольку оно влияет на масштаб и реалистичность тестовых данных. Во-вторых, приложение на С# использует прямые SQL-операторы 'INSERT' для инициализации данных вместо вызова хранимых процедур, которые были явно созданы в пункте 2 задания. Это указывает на отсутствие интеграции между SQL- и С#-частями решения, как это предполагалось заданием. Кроме того, объект 'Random' в методах инициализации С# создается внутри цикла, что может привести к случайным последовательностям, если цикл выполняется очень быстро (хотя для 5 записей это вряд ли будет проблемой). Название строки подключения в 'App.config' также имеет небольшое несоответствие. Перекрестный запрос, хотя и функционально корректен, предоставляет очень базовое представление графика работы (присутствие/отсутствие), а не подробную информацию о времени, и жестко кодирует имена сотрудников, что ограничивает его гибкость.</p> | |
|---|--|

Рисунок 24 – Отзыв о работе студента

Данная модель также предоставила более детальный и структурированный отзыв о работе студента, так что результаты проверки также зависят и от модели.

Таким образом, программный комплекс способен выполнять проверку отчетов по лабораторным работам студента, так как способен анализировать как большие по объему работы, так и работы, содержащие большое количество кода, на проверку человеком кото-рых ушло бы огромное количество времени. К тому же модель выявила те недостатки, которые были выявлены человеком. Значит, модель способно проверять отчеты студентов уровне человека и решает задачу проверки отчетов.

Общие сведения о программе

Программный комплекс предназначен для рецензирования отчетов по лабораторным работам студентов.

Функции программного комплекса:

- проверка отчетов;
- настройка ключей, моделей, провайдеров;
- навигация по истории проверок;
- создание курсов, групп, студентов;
- выполнение проверок разными моделями;
- создание и редактирование промптов.

Назначение и условия применения. Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

- процессор: 2.2 ГГц или быстрее с двумя или более ядрами;
- более 4 Гб оперативной памяти;
- наличие клавиатуры, мыши;
- операционная система *Windows 10, Windows 11*;
- наличие подключения к сети интернет.

Структура программы

Приложение состоит из двух проектов. Первый проект клиентское приложение на *React.js*, находящиеся в каталоге *client*. Второй проект – серверное приложение в каталоге *server* на *Node.js* с использованием фреймворка *NestJS*.

Настройка программы

Для работы с программным комплексом необходимо:

- установить платформу *Node.js*;
- запустить файл *install.bat* в корне проекта;
- подождать, пока выполнится загрузка всех пакетов;
- после установки запустить файл *start.bat*.

.

Проверка программного комплекса

Проверка работоспособности программного комплекса осуществляется при отправке запросов серверу. В случае успешного выполнения запроса сервер вернет клиенту ответ с результатом. В случае возникновения ошибки – ответ с сообщением об ошибке.

Методы тестирования программы представлены в таблице Б.1.

Таблица 1 – Методы тестирования программы

| Описание метода | Ожидаемый результат |
|---|---|
| Проверка на создание моделей. Пользователь отправляет запрос на создание модели на эндпоинт создания модели | Если пользователь передал все данные верно, то в отчете получит созданную запись, если данные некорректные, то пользователю вернется ответ со статус-кодом ошибки |

| Описание метода | Ожидаемый результат |
|--|--|
| Проверка на работу с курсами и лабораторными работами. Пользователь отправляет запросы на получение, редактирование курсов связанных с ними лабораторных работ | Если пользователь передал все данные верно, то в отчете получит созданные или же запрошенные данные, иначе увидит в ответе статус-код ошибки и сообщение об ошибке |
| Проверка на парсинг архива с отчетами студентов. Пользователь отправляет запрос на разбор архива с отчетами на эндпоинт по работе с отчетами | Если пользователь передал корректные данные, то ему вернется список студентов, который были в архиве, иначе ему вернется сообщение об ошибке |
| Запрос на выполнение проверки отчетов. Пользователь отправляет все необходимые данные для проверки отчетов | Если пользователь передал корректный ответ, то ему вернется сообщение о начале проверки. Если пользователь передал не все данные, то ему будет возвращено сообщение об ошибке. |

Сообщения системному программисту

В ходе работы с системой системному программисту могут выдаваться сообщения, приведенные в таблице Б.2.

Таблица 2 – Список сообщений системному программисту

| Текст сообщения | Описание сообщения | Возможные действия |
|---|--|--|
| Не удалось создать модель | Появляется, если что-то пошло не так при создании модели | Пользователь неправильно передал данные для создания модели или же сервис недоступен. Пользователь должен перепроверить данные |
| Не удалось выполнить парсинг файла | Появляется, когда пользователь выбрал некорректный файл с архива студентов | Пользователь должен выбрать корректный формат архива, после чего ошибка исчезнет |
| Не удалось начать проверку отчетов | Пользователь отправил запрос на начало проверки отчетов | Пользователь должен перепроверить введенные данные |
| Не удалось выполнить проверку отчетов | Пользователь отправил запрос на проверку отчетов, получил сообщение о начале проверки и через время получил сообщение о неудачной проверке | Вероятнее всего, не удалось выполнить проверку из-за того, что превышено максимальное количество запросов к модели. Пользователь должен выполнить проверку через время или же сменить проверяющую модель |
| Не удалось обновить лабораторную работу | Пользователь отправил запрос на обновление лабораторной работы | Пользователь должен перепроверить введенные им данные |

Назначение и условия применения программы

Программный комплекс предназначен для рецензирования отчетов по лабораторным работам студентов.

Функции программного комплекса:

- проверка отчетов;
- настройка ключей, моделей, провайдеров;
- навигация по истории проверок;
- создание курсов, групп, студентов;
- выполнение проверок разными моделями;
- создание и редактирование промптов.

Назначение и условия применения. Для корректной работы приложения необходима следующая конфигурация технических средств аппаратного обеспечения:

- процессор: 2.2 ГГц или быстрее с двумя или более ядрами;
- более 4 Гб оперативной памяти;
- наличие клавиатуры, мыши;
- операционная система *Windows 10*, *Windows 11*;
- наличие подключения к сети интернет.

Характеристика программы

Приложение состоит из двух проектов. Первый проект клиентское приложение на *React.js*, находящиеся в каталоге *client*. Второй проект – серверное приложение в каталоге *server* на *Node.js* с использованием фреймворка *NestJS*.

Обращение к программе

Для работы с программным комплексом необходимо:

- установить платформу *Node.js*;
- запустить файл *install.bat* в корне проекта;
- подождать, пока выполнится загрузка всех пакетов;
- после установки запустить файл *start.bat*.

Сообщения

В ходе работы программисту могут выдаваться сообщения, приведенные в таблице 1.

Таблица 1 – Список сообщений системному программисту

| Текст сообщения | Описание сообщения | Возможные действия |
|---|--|--|
| Не удалось создать модель | Появляется, если что-то пошло не так при создании модели | Пользователь неправильно передал данные для создания модели или же сервис недоступен. Пользователь должен перепроверить данные |
| Не удалось выполнить парсинг файла | Появляется, когда пользователь выбрал некорректный файл с архива студентов | Пользователь должен выбрать корректный формат архива, после чего ошибка исчезнет |
| Не удалось начать проверку отчетов | Пользователь отправил запрос на начало проверки отчетов | Пользователь должен перепроверить введенные данные |
| Не удалось выполнить проверку отчетов | Пользователь отправил запрос на проверку отчетов, получил сообщение о начале проверки и через время получил сообщение о неудачной проверке | Вероятнее всего, не удалось выполнить проверку из-за того, что превышено максимальное количество запросов к модели. Пользователь должен выполнить проверку через время или же сменить проверяющую модель |
| Не удалось обновить лабораторную работу | Пользователь отправил запрос на обновление лабораторной работы | Пользователь должен перепроверить введенные им данные |

Введение

Программный комплекс предназначен для рецензирования отчетов по лабораторным работам студентов.

Функции программного комплекса:

- проверка отчетов;
- настройка ключей, моделей, провайдеров;
- навигация по истории проверок;
- создание курсов, групп, студентов;
- выполнение проверок разными моделями;
- создание и редактирование промптов.

Пользователь должен владеть следующими навыками для начала работы с программным комплексом:

- базовые навыки работы на компьютере;
- базовые навыки использования клиентской программы в среде интернет.

Перед началом работы с программным комплексом пользователю рекомендуется ознакомиться с настоящим документом.

Назначение и условия применения

В системе пользователь может выполнять проверки отчетов студентов с использованием больших языковых моделей. Первым делом перед началом проверок пользователь должен задать все необходимые настройки, а именно задать модели, провайдеры, ключи.

Также пользователь должен задать начальные учебные группы, к которым будут причисляться студенты. Для выполнения проверки пользователь должен перейти на страницу с проверкой лабораторной работы, задать параметры проверки, указав модель, группу, файл с архивом и список студентов, чьи отчеты должны быть проверены.

Подготовка к работе

Для работы с программным комплексом необходимо:

- установить платформу *Node.js*;
- запустить файл *install.bat* в корне проекта;
- подождать, пока выполнится загрузка всех пакетов;
- после установки запустить файл *start.bat*.

Описание операций

Для выполнения операции «Создание модели» необходимо соблюдать следующие условия:

- ввести все правильные данные;
- задать ключ;
- задать провайдера;
- выполнить запрос на создание модели.

Для выполнения операции «Создание провайдера» необходимо соблюдать следующие условия:

- ввести все правильные данные;
- задать все данные (название, адрес);
- выполнить запрос на создание провайдера.

Для выполнения операции «Создание ключа» необходимо соблюдать следующие условия:

- ввести все правильные данные;
- задать все данные (название, значение);
- выполнить запрос на создание ключа.

Для выполнения операции «Создание курса» необходимо соблюдать следующие условия:

- ввести все правильные данные;
- задать все данные (название, описание);
- выполнить запрос на создание курса.

Для выполнения операции «Создание группы» необходимо соблюдать следующие условия:

- ввести все правильные данные;
- задать все данные (название);
- выполнить запрос на создание группы.

Для выполнения операции «Создание студента» необходимо соблюдать следующие условия:

- ввести все правильные данные;
- задать все данные (имя, фамилия, отчество, группу);
- выполнить запрос на создание студента.

Для выполнения операции «Проверка отчетов» необходимо соблюдать следующие условия:

- выбрать файл с корректным архивом с отчетами студентов;
- выбрать модель для проверки;
- задать список студентов;
- выполнить запрос на проверку отчетов.

Аварийные ситуации

Список аварийных ситуаций представлен в таблице 1.

Таблица 1 – Список аварийных ситуаций

| Аварийная ситуация | Описание | Рекомендации |
|--|--|--|
| Сервис, предоставляющий LLM, в данный момент не является доступным | Удаленные сервисы, предоставляющие API для работы с LLM, бывают нагружены и могут быть недоступны | Подождать некоторое время или же выбрать модель с провайдером, который работает без перебоев или же выбрать локальную модель |
| Ошибка при проверке отчетов | Если пользователь задал неправильные настройки для модели, провайдера или же ключа API, то он увидит ошибку при проверке отчетов | Пользователь должен проверить, правильно ли он ввел все данные для работы, валидное ли значение ключа, провайдера, название модели, если пользователь выполнил все эти шаги, то проблема должна быть устранена |

| Аварийная ситуация | Описание | Рекомендации |
|---------------------------|--|--|
| Ошибка при разборе архива | Пользователь в поле ввода файла в качестве архива с работами студентов может задать и некорректный файл, который не содержит нужных файлов | Пользователь должен выбрать правильный формат архива, иначе он не сможет проверить лабораторные работы |

Рекомендации перед эксплуатированием

После того, как пользователь создал модели, провайдеры, ключи, он может выполнять проверку отчетов. Для этого он должен открыть меню навигации и выбрать нужную лабораторную работу. Скриншот меню навигации представлен на рисунке 1.

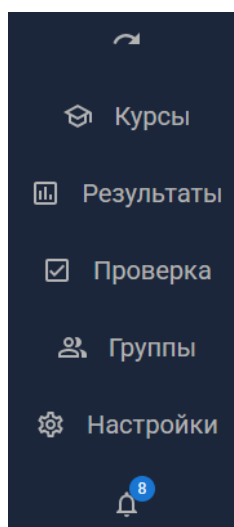


Рисунок 1 – Меню навигации

После нажатия на кнопку «Проверка» под нужной пользователю лабораторной работой произойдет переход на страницу проверки отчетов. На данной странице пользователь должен задать модель для проверки, архив с работами студентов, список студентов для проверки (опциональный параметр). Скриншот страницы проверок со всеми нужными данными представлен на рисунке 2.

Проверка отчетов
Здесь вы можете проверить отчеты студентов.

Лабораторная работа 2
Описание работы 2

task2.txt 13.87 KB

Модель
Выберите модели, которые будут проводить проверку.

Модели
DeepSeekReasoner [DeepSeek]

Студенты
Выберите студентов, отчеты которых будут проверены.

Студенты
Венгура Ганцевич Герад

☐ Учитывать предыдущую проверку

Группа
Выберите группу, к которой причислены студенты.

Выберите группу
ИТП-41

Отчеты
Выберите архив со списком отчетов в формате zip.

Выбранный файл
labs_2.zip (4.04 MB)

Ход проверки
Здесь отобразится информация о ходе проверки отчетов.

Рисунок 2 – Страница проверок

После того, как пользователь задал все необходимые данные, он может начать проверку отчетов, для этого он должен нажать кнопку «Отправить», после чего произойдет отправка отчетов на проверку. Во время проверки пользователю будет отображаться информация о ходе проверки. Скриншот отображения хода проверки представлен на рисунке

Ход проверки
Здесь отобразится информация о ходе проверки отчетов.

АВ Артур Венгура Анатольевич DeepSeekReasoner [DeepSeek]

ЕГ Екатерина Ганцевич Владимировна DeepSeekReasoner [DeepSeek]

АГ Александр Герад Денисович DeepSeekReasoner [DeepSeek]

Рисунок 3 – Информация о ходе проверки

Когда проверка завершится, пользователь увидит сообщение о том, что работы были проверены. Чтобы просмотреть полученные результаты, пользователь должен перейти в раздел уведомления на странице навигации и открыть последнее полученное уведомление, в котором ему отобразится результат проверки отчетов. Скриншот результатов проверки представлен на рисунке 4.

Результаты проверки

AB

Артур Венгура Анатолевич

Оценка: 5/10

Модель: DeepSeekReasoner [DeepSeek]

Дата: 28.05.2025, 10:43:47

Отзыв

Работа содержит значительные ошибки и несоответствия заданию. Основные проблемы: отсутствие выполнения части обязательных заданий (1.1, 1.2), некорректные SQL-запросы, игнорирование требований к созданию представлений, ошибки в логике группировки данных и неверные вычисления.

Положительные моменты

1. Предоставлены тексты SQL-запросов и результаты их выполнения

2. Для большинства заданий есть QBE-бланки (хотя их содержание не проверяется)

3. Правильно выполнены базовые JOIN-операции в простых запросах

4. Верно реализованы запросы на удаление и обновление (3.3.1, 3.3.2)

Недостатки и моменты, требующие доработки

1. Критическая ошибка: полностью отсутствуют задания 1.1 и 1.2

2. Не созданы представления (VIEW) для заданий 2.1-2.3, хотя это прямое требование

3. Некорректная группировка в заданиях 3.1.1-3.1.3: GROUP BY избыточен и искажает данные

4. Ошибка в задании 3.1.3: отсутствует фильтрация по городу, нарушено условие задачи

5. В задании 4.3 неверный выбор групп (1 и 3 вместо требуемых 1 и 2)

6. Фундаментальная ошибка в задании 4.4: использование фиксированного 2011 года вместо текущего

7. Ошибка в вычислении процента (3.2): делитель 100000 не обоснован

8. В задании 4.2 результат не соответствует варианту (должна быть сумма по группам, но без фильтрации по варианту)

9. Отсутствует обработка каскадного удаления в задании 4.4

10. Некорректные названия столбцов (например, 'Цена' вместо 'Сумма')

Рисунок 4 – Результаты проверки отчетов

35