Содержание

1 Аннотация

Идея данного проекта заключается в создании программы, собирающей и обрабатывающей данные оценочных ведомостей факультета компьютерных наук таким образом, чтобы статистические данные этих ведомостей вычислялись автоматически. С помощью данной программы планируется выявлять завышение оценок, отсутствие разброса (случаи, когда оценки ставятся преимущественно одинаковые), отслеживать неуспевающих студентов.

Неотъемлемой частью программы должен стать веб-интерфейс, с помощью которого можно будет наглядно демонстрировать результаты обработчика - просматривать инфографику с несколькими видами графиков или же информацию о ведомостях или успеваемости студентов в исходном виде.

2 Ключевые слова

Ведомость, метрика, парсинг, база данных, веб-интерфейс, клиент-серверное приложение, инфографика, SmartLMS

3 Введение

Ежегодно Факультет Компьютерных Наук оперирует сотнями учебных ведомостей. В основном в конце каждого учебного года эти ведомости подвергаются тщательному анализу. Цели анализа ведомостей могут быть различны. Во-первых, этот процесс позволяет идентифицировать студентов, испытывающих трудности с учебой, выявить причины их академической неуспеваемости и разработать стратегии для устранения препятствий на пути их обучения. Во-вторых, анализ ведомостей помогает определить предметы, где выставляются либо чрезвычайно высокие, либо, наоборот, чрезвычайно низкие оценки. Эта информация может служить основой для корректировки сложности учебной программы с целью улучшения ее соответствия уровню студентов.

Тем не менее, на текущий момент этот процесс столкнулся с двумя крупными проблемами. Первая заключается в том, что анализ зачастую проводится уже после выставления итоговых оценок. Такой подход значительно ограничивает полезность проводимого анализа, поскольку ни студенты, ни преподаватели не в состоянии каким-либо образом повлиять на уже выставленные оценки. Вторая проблема связана с большим объемом ручной работы, который требуется для проведения анализа. По этой причине он часто откладывается до конца учебного года, когда преподаватели и административный персонал могут уделить ему достаточно времени.

В рамках данного проекта предлагается разработать систему, которая в течение учебного года сможет в автоматическом режиме выявлять закономерности в получении и выставлении оценок. Безусловно, это довольно важно для студентов, ведь они смогут заранее узнать о том, что отстают и не усваивают программу на должном уровне, и таким образом смогут начать корректировать подход к учебе. Более того, это так же важно для преподавателей, ведь они смогут увидеть статистику успеваемости студентов и на основе этого упростить или, наоборот, усложнить программу для большего соответствия уровню своих студентов.

Проект включает в себя разработку двух основных компонент: системы сбора, систематизации и анализа ведомостей и веб-интерфейса для удобного взаимодействия с данной системой.

Система сбора будет основываться на парсере, способном проходить по ведомостям, извлекать из них информацию в соответствии с форматом их хранения и сохранять эту информацию в базе данных. После этого информация будет дополнительно обрабатываться для получения возможности ее анализа и построения инфографики. Затем будет проводиться анализ данных с использованием специально разработанных метрик, которые также будут

созданы в рамках этого проекта.

Веб-интерфейс обеспечит пользователей доступом к базе данных, текущей инфографике и различным метрикам. Вся эта информация будет представлена в удобной и понятной форме, включая графики и другие визуализации.

В дополнение к этому, мы планируем разработать функционал для выполнения пользовательских детализированных запросов. Например, пользователь сможет получить специфическую инфографику для студентов, получивших более 100 оценок «отлично» или статистическую информацию о конкретной группе.

Задачи каждого из участников проекта выглядят слеудующим образом:

Тимофеева Юлия Владиславовна:

1. Автоматизация процесса сбора данных из собранных текущих ведомостей;
2. Автоматический анализ данных из собранных текущих ведомостей;
3. Разработка метрик для анализа оценок в текущих ведомостях;
4. Создание базы данных всех оценок из текущих ведомостей;
5. Разработка функционала получения детализированной информации по конкретным запросам.

Бекян Артём Сергеевич:

1. Отображение в веб-интерфейсе инфографики по оценкам текущих ведомостей;
2. Разработка способов графического представления распределения оценок текущих ведомостей;
3. Разработка интерфейса взаимодействия с базой данных оценок текущих ведосотей;
4. Разработка интерфейса получения детализированной информации по конкретным запросам.

4 Обзор литературы

Похожими сервисами, предоставляющими аналитические инструменты для работы с данными, можно назвать Google-таблицы и Excel. В действительности они могут дать возможность пользователю сделать любой фильтр, в том числе пользовательский, и затем вывести это все в удобный формат - например, показать аналитику на графике. Тем не менее, они довольно сильно ограничены в типах данных и не могут обрабатывать все те типы, которые будет обрабатывать наша программа. К тому же, немало ведомостей хранятся в других форматах, в частности в формате HTML-страниц, и для сбора аналитики из такого фор-

мата данных с помощью перечисленных инструментов необходимо было бы предварительно перенести информацию вручную в Google или Excel-таблицу или изначально ограничить преподавателей до этих форматов. Первый вариант не избавляет от ручной работы, второй накладывает дополнительные рамки на преподавателей. Однако, даже если рассмотреть одно из предложенных решений, в любом случае не будет возможности получать информацию по запросу - например, для получения информации об успеваемости студента нужно будет просмотреть все таблицы, в которых он есть.

Другим похожим инструментом можно назвать SQL с подключенным к нему веб-интерфейсом. Там можно формировать различные запросы, а также строить различные дешборды с инфографикой, однако информацию, которую он будет обрабатывать, нужно сначала собрать в базу данных - значит, такой инструмент не является хорошим решением по тем же причинам, что и предыдущие варианты.

Нельзя не упомянуть SmartLMS - его функционал более остальных аналогов подходит для хранения и аналитики ведомостей. Однако, он отличается недружелюбным интерфейсом, низкой стабильностью, а также большинство преподавателей не используют его в качестве системы для ведения текущих ведомостей.

В целом, можно перечислить и другие инструменты, которые в той или иной степени могут предоставить аналогичные аналитические функции и удобный интерфейс для работы с ними и их визуализации. Тем не менее, все они либо приводят искусственному ограничению преподавателей в свободе формата отслеживания успеваемости или не избавляют от ручной обработки ведомостей, что и является базовыми проблемами текущего процесса анализа ведомостей.

Веб-интерфейс же, который предлагается реализовать в рамках данного проекта, будет использовать именно идею надстройки над базой данных, предоставляя дешборды, что будет удобнее для пользователей, чем табличные обработчики, в том числе благодаря простой и понятной настройке.

Подводя итог, наш продукт остается актуальным, поскольку объединяет в себе несколько инструментов. Он будет использовать преимущества уже существующих решений, но при этом будет иметь гораздо большую гибкость за счет возможности обработки большего количества возможных источников данных, сборщика, который будет собирать все и конвертировать в удобный для хранения вид, и самописного веб-интерфейса, который будет легко настраиваться под конкретные задачи и нужды за счет доступа к исходному коду.

5 Разработка бэкенда

Этот раздел выполнялся Тимофеевой Юлией.

Было решено начать выполнение работ с более высокоуровневых задач, постепенно двигаясь к более низкоуровневым - т.е. начать с идей, в дальнейшем спускаясь ко все более прикладным задачам.

5.1 Разработка метрик

Изначально мы определись, что у нас будет три модели данных: данные одной единицы, т.е. конкретной, ведомости, данные студентов, собранные из всех обработанных ведомостей и совокупные данные всех обработанных ведомостей. Поэтому перед тем, как приступать к разработке хранилищ или парсинга, нужно было определиться с дополнительной информацией, метриками, которую мы бы хотели хранить рядом с уже имеющимися в ведомостях данными. Эти метрики должны бы были согласовываться с поставленными целями и приближать нас к их достижению.

Так, для задачи выявления ведомостей с завышенными оценками по результатам консультации с руководителем было решено ввести показатели количества оценок определенного значения - другими словами количество «десяток», «девяток», ..., «нулей» после перевода в 10-ти балльную шкалу и округления исходных оценок. Их названия - соответственно «10», «9», ..., «0» (см. рис. 5.1). Для описанной задачи они применяются следующим образом: считаем ведомость «с завышением», если 50% ее оценок - отличные (10, 9, 8) и 35% оценок - «десятки».

Еще один вариант использования этого класса метрик - выявление неуспевающих студентов. К этой категории относим 5% студентов, имеющих наибольшую долю неудовлетворительных оценок (3, 2, 1, 0). Под долей понимается суммарное число перечисленных оценок в отношении к числу всех оценок.

При знакомстве с ведомостями было замечено следующее: в абсолютном большинстве случаев оценка 0 ставится студентам, в принципе не сдавшим работу. Поэтому в рамках следующего класса метрик было интересно выделить не только стандартную среднюю оценку, но и среднюю ненулевую оценку с целью получить информацию о том, на сколько в среднем преподаватель оценивает сданную работу. Метрики называются MeanMark и MeanPositiveMark соответственно (см. 5.1).

Хочется отметить также, что в действительности есть еще метрики MarkSum, MarkCount и MarkPositiveCount, означающие сумму оценок в неокругленном виде (но переведенных

в 10-ти балльную шкалу), количество оценок и количество ненулевых оценок. Эти метрики существуют исключительно для технических целей - для вычисления MeanMark и MeanPositiveMark.

	Name	MeanMark	MeanPositiveMark	10	9	8	7	6	5	4	3	2	1	0
1	Студент 1	3.097794	7.902537	27	4	2	3	2	3	1	1	4	2	76
3	Студент 3	5.645730	9.047644	61	0	3	4	4	3	1	0	0	2	47
4	Студент 4	3.769585	8.628160	29	4	2	2	0	4	2	1	0	1	58
5	Студент 5	3.063519	6.509979	8	3	8	6	2	0	6	2	3	2	45
6	Студент 6	4.529208	8.710016	33	14	4	6	3	3	0	1	1	0	60
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3202	Студент 3202	8.537194	8.537194	1	2	2	1	0	0	0	0	0	0	0
3208	Студент 3208	8.574634	8.574634	2	1	1	2	0	0	0	0	0	0	0
3209	Студент 3209	10.000000	10.000000	24	0	0	0	0	0	0	0	0	0	0
3210	Студент 3210	8.333333	9.722222	17	0	0	0	0	1	0	0	0	0	3
3215	Студент 3215	8.079922	8.079922	3	1	1	1	3	0	0	0	0	0	0

(a) Таблица студентов

	Discipline	MeanMark	MeanPositiveMark	10	9	8	7	6	5	4	3	2	1	0
1	Ведомость 1	5.564930	7.635431	741.0	457.0	1479.0	251.0	410.0	103.0	160.0	24.0	26.0	83.0	1462.0
2	Ведомость 2	6.427968	7.754106	271.0	171.0	613.0	122.0	202.0	44.0	80.0	9.0	9.0	1.0	314.0
4	Ведомость 4	5.938996	7.325322	1253.0	436.0	400.0	193.0	318.0	226.0	222.0	131.0	215.0	136.0	824.0
8	Ведомость 8	6.895958	7.784940	215.0	15.0	175.0	6.0	40.0	7.0	30.0	1.0	54.0	0.0	70.0
12	Ведомость 12	6.506039	9.274202	14429.0	303.0	383.0	383.0	93.0	769.0	51.0	126.0	171.0	366.0	7266.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
324	Ведомость 324	6.415479	8.103763	8.0	1.0	4.0	1.0	2.0	3.0	0.0	0.0	0.0	0.0	5.0
328	Ведомость 328	7.085199	8.811292	256.0	50.0	48.0	23.0	23.0	11.0	7.0	3.0	7.0	3.0	105.0
338	Ведомость 338	5.399661	6.999560	44.0	5.0	6.0	8.0	5.0	14.0	7.0	7.0	6.0	6.0	32.0
340	Ведомость 340	3.974733	7.096588	69.0	20.0	39.0	23.0	21.0	34.0	15.0	7.0	13.0	5.0	195.0
346	Ведомость 346	7.786846	8.683948	245.0	60.0	51.0	17.0	16.0	19.0	12.0	4.0	8.0	2.0	50.0

(b) Таблица ведомостей

Рис. 5.1: Метрики

Можно заметить, что в такой модели хранения данных нарушена нормализация [12], т.к. MeanMark и MeanPositiveMark только от неключевых атрибутов MarkSum, MarkCount и MarkPositiveCount. Это сознательное решение - во-первых, имея перед глазами эти метрики легче отлаживать и тестировать код. Например, если у какой-то ведомости будут нетипичные для предмета показатели - это повод перепроверить алгоритмы парсинга. Во-вторых, наша база данных не поддерживает автоматическое обновление или частичные корректировки - на данном этапе, если в базе что-то нужно обновить, придется заново обрабатывать всю базу целиком. А значит, мы не столкнемся с аномалиями изменений данных.

Таким образом, были разработаны агрегирующие метрики MeanMark, MeanPositiveMark, число 10-к, число 9-к, ..., число 0-лей, а также технические показатели MarkSum, MarkCount, MarkPositiveCount.

5.2 Алгоритмические решения

Как уже было отмечено ранее, имеем три модели данных. Первый вид - совокупные данные всех обработанных ведомостей, иначе root-таблица. Подробности реализации обсудим в разделе 5.4, а сейчас тезисно обсудим главный смысл присутствия данной сущности в проекте.

Для начала напомним, откуда она берется. Перед тем, как человек, анализирующий ведомости, приступает к своей работе, он собирает информацию о них - о названии предмета, имени преподавателя, модулях и курсе, в которых читается предмет, и др. Далее какое-то время он посвящает добыванию ссылок на эти ведомости. И только после этого в выделенное под это время приступает к анализу. Таблица, заполненная таким человеком-аналитиком, и есть root-таблица в нашем понимании - в ней есть вся предварительная информация о совокупности ведомостей определенного учебного года (см. 5.2).

						Преподаватель	
Уровень	Программа	Курс	Модуль	Дисциплина	ФИО преподавателя	Ссылка на ведомость	Ссылка Wiki / GIT /
Бакалавриат	Программная инженерия	1	1,2,3,4	Предмет 1	Преподаватель 1	url1	
Бакалавриат	Программная инженерия	1	2,3	Предмет 2	Преподаватель 2	url2	
Бакалавриат	Программная инженерия	1	2,3	Предмет 3	Преподаватель 3	url3	
Бакалавриат	Программная инженерия	1	1,2,3,4	Предмет 4	Преподаватель 4	url4	
Бакалавриат	Программная инженерия	1	2,3	Предмет 5	Преподаватель 5	url5 url6	
Бакалавриат	Программная инженерия	1	2,3	Предмет 6	Преподаватель 6	url7	
Бакалавриат	Программная инженерия	1	1,2,3	Предмет 7	Преподаватель 7	url8 (1 модуль) url9 (итоговая)	url10

Рис. 5.2: Root-таблица в формате Google таблицы

После же обрабатывает исходную root-таблицу так, чтобы каждая предоставленная ссылка на ведомость была в отдельной строке root-таблицы. У одного предмета может быть несколько ведомостей и вестись они могут в разных форматах. Например, на предмете может быть два ассистента, один из которых предпочитает вести ведомости в Google таблицах, а другой - в SmartLMS. Google таблицы - общедоступны, в SmartLMS нужно иметь доступ. Так, одна ведомость «делится» на несколько дочерних в разных форматах с разными доступами. Поэтому в обработанной root-таблице было решено выделить каждой предоставленной ссылке отдельную строку для того, чтобы отслеживать необходимую для парсера информацию, в частности, как отмечалось, о доступности ведомости, ее формате или ошибке, которую вернула попытка обращения по ссылке. Таким образом root-таблица создается и наполняется первой информацией, а также заполняется нулями в полях всех метрик.

Далее создается таблица студентов. Для этого пользователь должен передать через веб-интерфейс excel-файлы с актуальными списками студентов (см. рис. 5.3). После этого парсер

обрабатывает переданные файлы, выделяя идентификационную информацию студентов - ФИО, номер группы, образовательную программу и др. Эта информацию так же дополняется метриками, их значения изначально обнулены.

Фамилия	Имя	Отчество	на факультете		
			Курс	Группа	Образовательная программа
Фамилия1	Имя1	Отчество1	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия2	Имя2	Отчество2	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия3	Имя3	Отчество3	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия4	Имя4	Отчество4	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия5	Имя5	Отчество5	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия6	Имя6	Отчество6	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия7	Имя7	Отчество7	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия8	Имя8	Отчество8	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия9	Имя9	Отчество9	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия10	Имя10	Отчество10	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия11	Имя11	Отчество11	2 курс	Группа	Компьютерные науки и анализ данных
Фамилия12	Имя12	Отчество12	2 курс	Группа	Компьютерные науки и анализ данных

Рис. 5.3: Исходные файлы со списками студентов

Следующим шагом парсер проходиться по root-таблице, переходит по ссылкам и начинает обрабатывать информацию части (или всей) ведомости.

Если у предыдущих моделей, таблицы студентов и root-таблицы, были фиксированные исходные форматы (парсер знал, в каких колонках что находится), то теперь парсеру предстояло работать с неструктурированными данными [13]. Нужно было понять, как решать следующие задачи:

1. Если в ведомости присутствуют несколько страниц (как, например, в Google таблицах или в Microsoft Sharepoint), как понять, какие обрабатывать?
2. Какие колонки будут для нас содержательными?
3. Какие, наоборот, будут несодержательными?

Для начала ответим на эти вопросы идейно, не углубляясь в реализацию. Для решения задачи, поднятой в первом вопросе, мы выделили несколько подходов, которые собирались проверить опытным путем и по результатам остановиться на том решении, которые дает наибольшую точность. На показателях точности парсинга останавливаться не будем - подробно об этом написано в разделе 7.

Подход 1: Парсить только ту страницу, на которую предоставлена ссылка.

Данный подход не выдержал проверки и показал худшие результаты (при визуальной оценке парсинга - подробнее в разделе 7). Действительно - зачастую ведомость разделена на несколько подведомостей по группам, где каждая находится на своей отдельной странице. Тогда обрабатывая лишь одну, теряем информацию об успеваемости в других группах.

Подход 2: Парсить только те страницы, названия которых похожи на названия групп.

Более точное решение. Однако, с ним есть две проблемы. Во-первых, так тоже можем

пропустить страницы с ценной информацией. Например, иногда оценки за контрольную перемещают на отдельную страницу. Во-вторых, точность такого подхода сильно зависит от выбора стратегии выявления названий групп. Стратегии, которые испробовали мы, а именно вычисление названий групп по регулярным выражениям, показали плохие результаты - часто подходящие страницы не распознавались, как подходящие, а неподходящие - наоборот.

Подход 3: Парсить все страницы.

На этом подходе мы решили остановиться. Помимо того, что он интуитивно согласуется с логикой обработки ведомостей (т.е. изначально, не посмотрев «внутрь» ведомости, мы не игнорируем ни одну страницу, исходя исключительно из названия), он еще и показал наилучшие результаты на практике.

Итак, мы ответили на первый вопрос - какие страницы ведомости обрабатывать. Перейдем ко второму - какие нужно обрабатывать колонки, внутри одно страницы? На этот вопрос ответить проще - нам точно нужна информация, идентифицирующая студентов, и информация о «нескорректированных» оценках. Под «нескорректированными» понимаются данные, не являющиеся агрегированной или другой вычисленной информацией на основе поставленных оценок. Примерами «скорректированных» оценочных данных могут являться: средняя, накопленная, итоговая оценка и др.

Информация о студентах должна быть предоставлена обязательно - как минимум ФИО студента есть в любой ведомости. Иногда, особенно при сокращении имени и отчества студентов, при обработке исключительно именных данных могут возникать коллизии - два или более студента могут иметь одну фамилию и инициалы. Поэтому также будем искать на странице ведомости название группы. Проверять данные на соответствие имени или названию группы будет с помощью регулярных выражений (см. рис. 9.1). В дальнейшем при переходе к обработке конкретной строки будем также валидировать имена студентов. Здесь нам впервые понадобиться таблица студентов - если данного имени нет в таблице студентов, то считаем, что либо регулярные выражения подобрали неподходящие данные (технически подходящие, но не логически), либо студент с данными именем не учится на ФКН.

В процессе предварительного исследования ведомостей и правил их заполнения мы обнаружили, что в абсолютном большинстве случаев преподаватели дополняют ведомостей «скорректированными» оценками с помощью формул. Нас не интересуют такие «скорректированные» оценки потому, что в противном случае, если мы учитываем и такие, и исходные оценки, в суммированных данных одни и те же оценки учтутся несколько раз и этим дан-

ным уже нельзя будет доверять. Поэтому валидация оценочных данных может ограничиться рассмотрением двух случаев. Первый - отформатированные данные приводятся к типу чисел с плавающей точкой. Второй - не отформатированные данные не являются формулой (с некоторыми поправками; подробнее см. рис. 9.2).

Плавно переходя к третьему вопросу, мы уже дали ответ на его часть - несодержательными для нас являются «скорректированные» оценки. Но есть еще данные, нас не интересующие, которые часто встречаются в ведомостях. Это, например, индексы, номера вариантов. Так как в большинстве случаев эта информация находится даже до колонки с именами студентов, то избавиться от нее можем, пропустив все колонки перед колонкой с именами. Еще одна не интересующая нас информация - данные о преподавателях, ассистентах. Если они находятся после колонки с именами студентов, что часто и бывает, - парсер просто пропустит ее, так как настроен на обработку только первой попавшейся именной колонки.

Сразу можно выделить небольшую проблему, которой не удалось найти решения, - если имя ассистента будет написано в колонке студентов и в той же строке будут написаны какие-то цифры (например, так иногда обозначают максимальные оценки, см. рис. 5.4), то они будут приняты парсером за оценки. Это значит, что в таблице студентов у данного студента-ассистента появятся лишние «десятки», что испортит статистику этого студента. Однако, по наблюдениям ассистенты - студенты, в основном, успевающие, а значит на задачу выявления неуспевающих студентов их успеваемость не влияет.

	Тесты													Теоретические домашн				
Студенты	1	2	3	4	5	6	7	8	9	10	11	...	Среднее	ДЗ 1	ДЗ 2	ДЗ 3	ДЗ 4	ДЗ 5
Студент 1	-	-	2,5	5	-	-	-	-	-	-			0,83	-	-			
Студент 2	5	5	-	-	-	-	-	-	-	-			1,11	43	97			
Студент 3	5	5	2,5	10	7,5	6	7,5	5	10	7.5			6,50	98	100			
Студент 4	2,5	7,5	7,5	7,5	7,5	6	2,5	7,5	10	7.5			6,50	87	-			
Студент 5	7,5	7,5	7,5	10	7,5	8	7,5	5	10	10			7,83	95	100			
Студент 6	5	5	7,5	7,5	7,5	5	5	7,5	-	-			5,56	94	99			
Ассистент	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10**	**10,00**	**100**	**100**	**100**	**100**	**100**

Рис. 5.4: Пример ведомости, в которой ассистент будет принят за студента

Итак, теперь, когда мы научились парсить нужные данные, распарсенные оценки нужно нормировать, т.е. привести к 10-балльной шкале. Для этого мы придумали простой алгоритм достаточно неплохой точности. Найдем в колонке (а значит, для конкретной контрольной точки) максимальную оценку. Разделим каждую оценку на эту максимальную и умножим на 10. Получим приведенную к 10-балльной шкале оценку. Конечно, не всегда такой подход сто процентно точен - бывают задания, по которым никто из списка не получает наивысшую оценку. Однако, таких случаев относительно немного, будем считать их погрешностью.

После нормировки мы наконец можем вычислить метрики, которые мы продумали в разделе 5.1.

Суммируя, мы разработали идеи и подходы к парсингу ведомостей, валидации данных, а также определились с логикой данных, наполняющих три выделенные модели данных.

5.3 Использованные технологии

Главным инструментом парсера, несомненно является язык python версии 3.8.10 [10]. Он был выбран по нескольким причинам. Во-первых, на сегодняшний день он является самым популярным языком программирования для задач, связанных с анализом данных. Во-вторых, у него простой, интуитивно понятный сиснтаксис, который делает его код читаемым даже для человека, не знакомого с python. В-третьих, python входит в топ языков, для которых написано большинство библиотек. Это значит, что почти на любую задачу найдется высокоуровневая оболочка (модуль, библиотека), позволяющая ее решать максимально эффективно.

Важным инструментом при обработке Google таблиц является библиотека pygsheets [9]. В числе главных ее преимуществ - простое, удобное и хорошо задокументированное взаимодействие с Google API и наличие функции, способной убирать незначащие (пустые) столбцы и строки из таблицы. Это сильно экономит время как непосредственной разработки, так и время выполнения программы.

Конечно, нельзя не упомянуть библиотеку для анализа данных pandas [8]. Кроме удобного хранения данных и возможности преобразовать их в разные форматы, например, в csv-файл, она также дает возможность понятно фильтровать данные, т.е. по условию разной сложности получать нужный срез. Также есть возможность считывать Excel-файлы в базу и переносить данные в SQL БД.

Особое место в проекте занимает библиотека регулярных выражений re [4]. С помощью re мы валидируем и ищем нужные данные - по поиску шаблона или проверки на соответствие.

Базу данных мы решили использовать из пакета sqllite3 [3]. Она поддерживает многопоточность, что важно для фронтенда, а также это очень простая и удобная SQL СУБД, доступная из среды python.

Библиотека numpy [7] - очень быстрый и эффективный вычислительный инструмент, позволяющий работать с данными как с матрицами. С его помощью очень удобно нормировать оценки.

А также стоит отметить небольшой, но все же вклад в проект модуля collections.Container

[2], позволивший быстро подсчитать количество вхождений элементов в список, что понадобилось нам при вычислении метрик, а также библиотеку requests [11], принять код возврата запроса с сайта и обернуть его в сообщение с ошибкой (если она была).

5.4 Архитектура

Было решено реализовать парсер в ООП подходе - в оболочке класса StatementAnalysis как единой сущности. Так, мы решили, какие функции будут доступны для вызова внешнему пользователю, а какие скрыты. При этом база данных находится в атрибутах этого класса.

Логические блоки мы реализовали в виде отдельных методов класса - приватных, публичных и статических, т.е. таких, что имеют доступ к (см. рис. 5.5).

```python
@staticmethod
def cnt_positive_values(lst):
    cnt = 0
    for value in lst:
        cnt += int(value > 0)
    return cnt
```

(a) Статический метод подсчета положительных значений

```python
def __get_group_names(self):
    group_names = set()
    for group in self.df_students.Group.unique():
        group_names.add(group)
    return group_names
```

(b) Приватный метод получения уникальных названий групп

Рис. 5.5: Примеры методов класса

Аналогично, в соответствии с логическими блоками устроена файловая система проекта:

- *StatementAnalysis.py* - файл с классом парсера
- *Analysis.ipynb* - файл-ноутбук с анализом парсинга
- *data* - директория со всеми данными, которые когда-либо обрабатывал парсер
  - *data.json* - файл, хранящий информацию об обработанных root-таблицах: их ссылки и пути к папкам с обработанными по ним данными
  - *Папки формата* YYYY-MM-DD_hh:mm:ss - папки с обработанными root-таблицами, их ведомостями и списками студентов
    - *root.csv* - обработанная root-таблица в формате csv
    - *students.csv* - обработанные списки студентов в формате одного csv файла
    - *students_src* - директория, хранящая Excel-файлы со списками студентов
    - *statements* - директория, хранящая обработанные ведомости в формате csv
- *client* - директория с информацией о сервисном аккаунте, необходимая для взаимодействия с Google API

- *config* - директория с конфигурациями

Опишем отдельно структуру наших хранилищ. Начнем с root-таблицы. Как мы уже упоминали она хранится в двух формах - в виде SQL-таблицы базы данных sqllite3 и в виде csv-файла, находящегося по адресу /data/[*название папки*]/root.csv. Эта таблица имеет следующую структуру:

1. **ID** (целое число) - внутренний идентификатор ведомости;
2. **Level** (текст) - уровень обучения - бакалавриат или магистратура;
3. **Program** (текст) - образовательная программа;
4. **Year** (целое число) - номер курса;
5. **Module** (текст) - модули, в которых преподается предмет;
6. **Discipline** (текст) - название предмета;
7. **Teacher** (текст) - преподаватель;
8. **URL** (текст) - ссылка на ведомость;
9. **Row** (целое число) - номер строки ведомости в исходной root-таблице;
10. **Type** (текст) - тип ведомости (Google Sheet, Sharepoint и др.);
11. **IsParsed** (булево значение) - обработана ли ведомость;
12. **ErrorName** (текст) - название ошибки (если было);
13. **Text** (текст) - исходное значение ячейки, из которой взят URL;
14. **Path** (текст) - путь к обработанному csv-файлу с ведомостью;

Далее также идут колонки метрик, смысл и содержание которых мы обсудили в разделе 5.1. Значения всех метрик имеют тип числа с плавающей точкой.

Следующую структуру имеет таблица студентов (аналогично, опускаем перечисление метрик):

1. **ID** (целое число) - внутренний идентификатор ведомости;
2. **Name** (текст) - имя студента;
3. **Year** (целое число) - номер курса;
4. **Group** (целое число) - название группы;
5. **Program** (текст) - образовательная программа;

Такую структуру имеют таблицы ведомостей (аналогично, опускаем перечисление метрик):

1. **ID** (целое число) - внутренний идентификатор ведомости;
2. **Student** (текст) - имя студента;

3. **Group** (целое число) - название группы;
4. **MarksDict** (текст) - словарь вида название оценки и колонки - оценка (ненормированная);
5. **MarksNorm** (текст) - список нормированных оценок;

6 Разработка фронтенда

Этот раздел выполнялся Бекяном Артёмом

6.1 Использованные технологии

Фронтенд часть была написана на языке программирования Python3 с использованием библиотеки Django [5]. Конечно, помимо Python3, также использовались стандартные средства веб-разработки, а именно: HTML [16] для отрисовки страниц, CSS [15] для настройки положения, размера и остальных свойств показывающихся элементов, и JavaScript [14] для написания простых скриптов на страницы. Также для упомянутых скриптов использовалась JS-библиотека JQuery [6].

Для отображения графиков используется популярная инфографическая библиотека Chart.js [1]. Выбор именно этой библиотеки обусловлен тем, что она имеет весь необходимый функционал, при этом графики отрисовываются быстро и красиво.

Данные, необходимые для корректного функционирования веб-интерфейса, хранятся в специальной базе данных на SQLite3. SQLite3 была использовано как стандартная библиотека Django и не управлялась напрямую, только через специальные менеджеры из Django. Таким образом, хотя в проекте и используется SQLite3, ее можно легко переделать на другую базу данных, поддерживающую прямую интеграцию с Django.

6.2 Архитектура

Фронтенд отвечает за отображение инфографики по полученной информации. Запросы для отображения могут быть получены от пользователя или от бэкенд части.

Фронтенд часть разделена на несколько модулей. Некоторые из них являются техническими, как, например, модуль, отвечающий за показ подсказок при введении дополнительных фильтров для построения графика. Другие модули представляют собой веб-приложения, использующиеся для предоставления пользователю какой-либо информации. Список необходимой информации может быть получен от пользователя, например, как запрос на отрисовку

какого-либо графика, либо от бэкенд части, как запрос на показ пользователю предобработанной метрики.

Для работы фронтенд части также необходима база данных Django, в которой хранится информация о пользователях и их активности (например, дешборды и графики, построенные пользователем), и некоторая техническая информация (например, элементы навигационной панели).

Помимо этого, фронтенд часть использует бэкенд через специальный класс-обертку. Бэкенд часть используется для доступа к базе данных, полученной из ведомостей путем парсинга, а также для получения значений предобработанных метрик.

6.3 Схема базы данных с описанием

Как было упомянуто выше, для работы фронтенд части необходима база данных Django, в которой хранится различная информация. В рамках данной секции предлагается подробно рассмотреть использующуюся базу данных, данные, хранимые внутри, а также формат их хранения. Для этого рассмотрим Рисунок 6.1

На данной схеме представлены не все таблицы, содержащиеся в базе данных. Тем не менее, оставшиеся таблицы являются стандартными для Django, и никакой отдельной работы с ними не проводилось.

Стоящим отдельно классом является MenuItem. Это технический класс, использующийся для показа меню навигации. При запуске программы туда добавляются все необходимые страницы, которые позже можно наблюдать на каждой странице, поддерживающей навигацию. Тем не менее, также возможно добавление новой страницы в панель навигации по ходу использования сайта, хотя на данный момент это не используется.

Остальные классы на данной схеме составляют единую структуру. Класс CustomUser является, пожалуй, самым важным во всей базе данных, поскольку именно он отвечает за всех пользователей. С помощью его базовых функций происходит авторизация, и он же используется при построении дешбордов как ForeignKey.

Помимо класса пользователя, можно заметить класс Dashboard. Этот класс является специальным классом дешбордов, где один объект такого класса обозначает один дешборд. У него всего 3 поля: id, author и name. id - PrimaryKey, по которому отличаются объекты класса. Поле author содержит пользователя, создавшего данный дешборд, а поле name - имя, которое дали данному дешборду.

Следующим классом является Chart, класс графиков. Он содержит уже намного больше

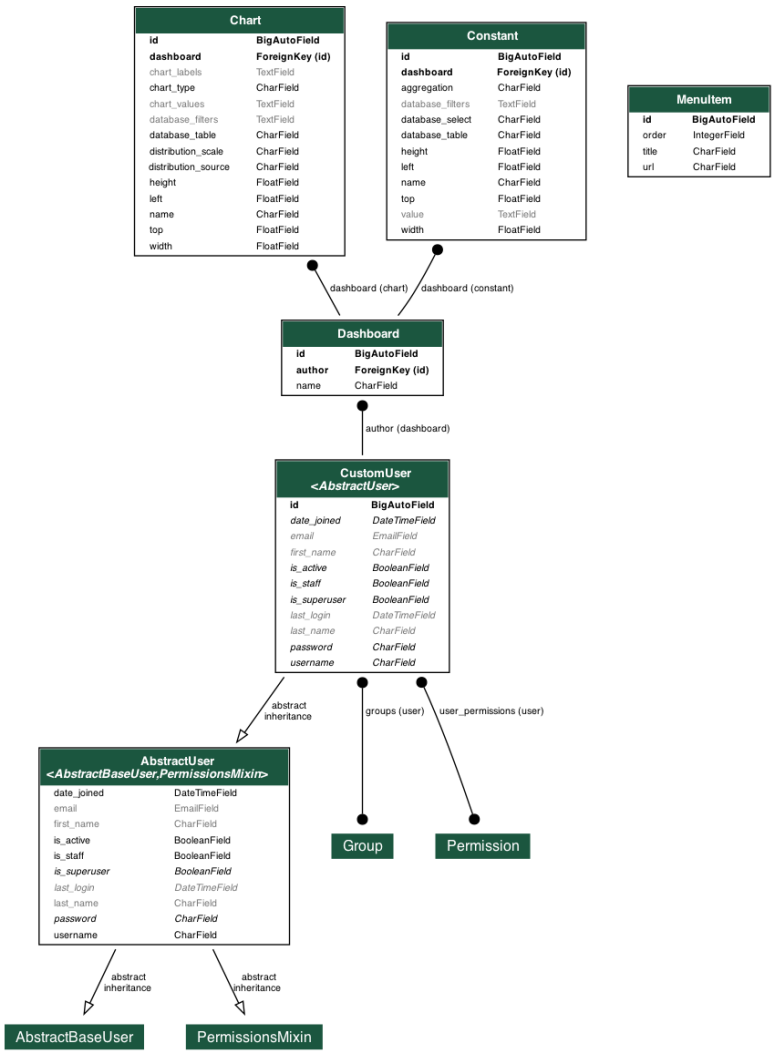


Рис. 6.1: Схема базы данных Django.

полей, ведь по объекту такого класса должно быть возможно построить график, правильно расположив его на правильном дешборде. Рассмотрим поля данного класса подробнее:

1. **id** - PrimaryKey, по которому отличаются объекты класса;
2. **dashboard** - ForeignKey, дешборд, на котором был создан данный график;
3. **name** - Название графика, введенное пользователем;
4. **chart_type** - Тип графика, один из 2 доступных: гистограмма или круговая диаграмма. Доступно всего 2 вида графиков, поскольку в большинстве случаев этого будет достаточно для получения желаемой информации, ведь основное предназначение графиков - оценивать распределения, для чего лучше всего подходят как раз гистограмма и круговая диаграмма;
5. **database_table** - Таблицы базы данных, по которой строится график. root для таблицы ведомостей, students для таблицы студентов;
6. **distribution_source** - Распределение чего строится, одна из 3 опций: оценка, средняя оценка или средняя ненулевая оценка;

7. **distribution_scale** - Шкала оценивания, 5-балльная или 10-балльная. Перевод происходит в соответствии со стандартной шкалой перевода ВШЭ;
8. **database_filters** - Дополнительные фильтры. Текстовое поле для пользователей, которые хотят сделать более детализированный запрос. Представляет собой длинную строку, например, Средняя_оценка $>$ 5;
9. **chart_labels** - Названия меток графика. Не хранятся в самой базе данных, используются исключительно для удобной передачи графика в HTML шаблон;
10. **chart_values** - Данные, соответствующие каждой метке графа. Не хранятся в самой базе данных, используются исключительно для удобной передачи графика в HTML шаблон;
11. **height** - Высота графика на дешборде;
12. **width** - Ширина графика на дешборде;
13. **left** - Координата левой стороны графика на дешборде;
14. **top** - Координата верхней стороны графика на дешборде.

Большинство этих полей используется для получения информации или отображения графика на странице дешборда. О том, как конкретно это происходит, детально написано в секции 6.4.5.

Остается еще один неразобранный класс, а именно Constant. Этот класс используется также для отображения на дешборде, однако он представляет не графики, а константные числа, которые пользователи могут захотеть добавить. Во многом он схож с классом Chart, но есть некоторые отличия. Говоря конкретно, у класса Constant отсутствуют такие поля, как chart_labels, chart_type, chart_values, distribution_scale и distribution_source. Отсутствие каждого их них довольно очевидно, поскольку все они служат для уточнения данных, которые важны для графика, но не относятся к константным числам. Вместо этого, добавляется несколько новых полей, рассмотрим их внимательно:

1. **database_select** - По какой величине мы хотим вычислять константу, может принимать одно из следующих значений: средняя оценка, средняя ненулевая оценка, сумма оценок, количество оценок, количество ненулевых оценок, количество какой-либо конкретной оценки (от 0 до 10);
2. **aggregation** - Как мы хотим вычислять константу, может принимать одно из следующих значений: как среднее арифметическое, как медиану, как максимальное значение, как минимальное значение, как стандартное отклонение;
3. **value** - Значение. Не хранится в самой базе данных, используется исключительно для удобной передачи константы в HTML шаблон.

О конкретном принципе работы этих полей написано в секции 6.4.5.

6.4 Подробный обзор модулей

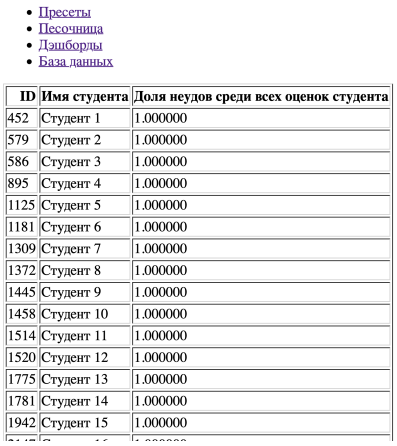
Как уже упоминалось выше, фронтенд часть представляет собой набор различных модулей. В данной секции все написанные модули будут детально разобраны.

6.4.1 Модуль авторизации

Первым модулем, который видит пользователь при заходе на сайт, является модуль авторизации. Для пользователя он проявляется в виде формы авторизации с полями "логин"и "пароль". При вводе правильных данных пользователь перенаправляется на страницу, где ему нужно указать, с какой Root-таблицей он хочет работать, путем ввода ее url-адреса.

Если введенная таблица уже обработана, то пользователя просто направляет на главную страницу, где он может выбрать следующий модуль. Если же такая таблица еще не была обработана, то пользователю необходимо дополнительно загрузить файл с данными студентов, и затем запускается процесс обработки. По его окончанию пользователь может приступить к работе с данными. Url-адрес таблицы при этом записывается в сессии, и может быть получен в любом модуле, где он необходим.

6.4.2 Модуль пресетов



- Пресеты
- Песочница
- Дэшборды
- База данных

ID	Имя студента	Доля неудов среди всех оценок студента
452	Студент 1	1.000000
579	Студент 2	1.000000
586	Студент 3	1.000000
895	Студент 4	1.000000
1125	Студент 5	1.000000
1181	Студент 6	1.000000
1309	Студент 7	1.000000
1372	Студент 8	1.000000
1445	Студент 9	1.000000
1458	Студент 10	1.000000
1514	Студент 11	1.000000
1520	Студент 12	1.000000
1775	Студент 13	1.000000
1781	Студент 14	1.000000
1942	Студент 15	1.000000

Рис. 6.2: Пример вывода метрики "Неуспевающие"модуля пресетов фронтенд части.

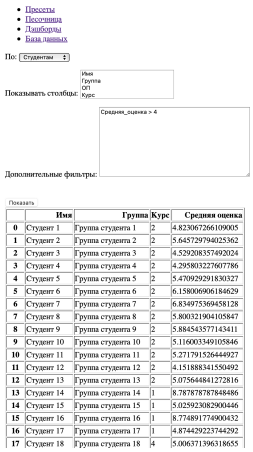
Модуль пресетов является модулем, который использует специально разработанные метрики и выводит информацию на их основе.

Сначала данный модуль выглядит как меню выбора, где написаны различные названия метрик. Эти названия кликабельны, так что при нажатии на них пользователя перенаправляет на специальную страницу с выводом этой метрики.

Для вывода метрики фронтенд должен запросить данные у бэкенда. Для этого создается объект класса бэкенда, куда отправляется запрос с названием метрики, которую хочет увидеть пользователь. Обратно фронтенд часть получает данные, которые нужно вывести, и выводит их пользователю.

Пример работы модуля можно увидеть на рисунке 6.2.

6.4.3 Модуль базы данных



	Имя	Группа	Курс	Средняя оценка
0	Студент 1	Группа студента 1	2	4.823067266109005
1	Студент 2	Группа студента 2	2	5.645729794025362
2	Студент 3	Группа студента 3	2	4.52920835749204
3	Студент 4	Группа студента 4	2	4.295803227607786
4	Студент 5	Группа студента 5	2	5.470929291830327
5	Студент 6	Группа студента 6	2	6.158006906184629
6	Студент 7	Группа студента 7	2	6.834975369458128
7	Студент 8	Группа студента 8	2	5.800321904105847
8	Студент 9	Группа студента 9	2	5.884543577143411
9	Студент 10	Группа студента 10	2	5.116003349105846
10	Студент 11	Группа студента 11	2	5.271791526444927
11	Студент 12	Группа студента 12	2	4.151883415924492
12	Студент 13	Группа студента 13	2	5.075644841272816
13	Студент 14	Группа студента 14	1	8.787878787848486
14	Студент 15	Группа студента 15	1	5.025923082900446
15	Студент 16	Группа студента 16	1	8.774891774900432
16	Студент 17	Группа студента 17	1	4.874429223744292
17	Студент 18	Группа студента 18	4	5.006371396318655

Рис. 6.3: Пример использования модуля базы данных фронтенд части. Показаны результаты вывода модуля по запросу: вывести имя, группу, курс и среднюю оценку студентов при условии, что средняя оценка > 4 .

Модуль базы данных является модулем, который позволяет пользователю взаимодействовать с базой данных с помощью различных запросов.

Самая важная задача данного модуля - понятность пользователям. Например, можно было бы просто сделать поле для ввода SQL команд, однако пользователям это было бы непонятно и не интуитивно. Поэтому было выбрано решение сделать интерфейс подробным, пусть и с ограниченным функционалом.

Пользователю предлагается ввести следующие данные:

1. На что он хочет смотреть, на ведомости или на студентов;
2. Какие столбцы он хочет увидеть в итоге. Возможные столбцы меняются в зависимости от выбранной таблицы (студенты или ведомости), так что возможность запросить что-то, что невозможно вывести, у пользователей отсутствует;
3. Дополнительные фильтры. Наверное, самая сложная часть данного модуля. Перед нами стояла задача сделать так, чтобы пользователи могли задавать детальные запросы, при этом синтаксис таких запросов был интуитивно понятен, и отсутствовала возможность что-либо сломать. Нами было принято решение сделать "перевод"языка SQL: на самом деле все фильтры после получения программой переводятся в SQL и посылаются в базу данных. Такое решение не совсем безопасно,

однако за пользователей уже написано начало команды, а именно SELECT <что-то> FROM <что-то> WHERE <свободный ввод пользователя>;. Таким образом, пользователь может влиять только на часть после WHERE, что сильно ограничивает его способность к нанесению вреда базе данных. Дополнительно был запрещен символ ';', поскольку он мог означать возможность ввести полностью новую команду. Тем не менее, предпринятые меры защиты не являются сильными, так что при наличии желания злоумышленник сможет навредить базе данных. Выходом из данной ситуации является только фильтрация пользователей, чтобы такого не происходило.
Также для интуитивности были введены подсказки. О точном алгоритме их работы написано в секции 6.4.6.

После получения запроса от пользователя, он обрабатывается и превращается в SQL-запрос. Затем создается объект класса бэкенда, с помощью которого выполняется запрос к базе данных ведомостей. Полученные результаты преобразовываются в табличку и выводятся на страницу.

Пример работы модуля можно увидеть на рисунке 6.3.

6.4.4 Модуль песочницы

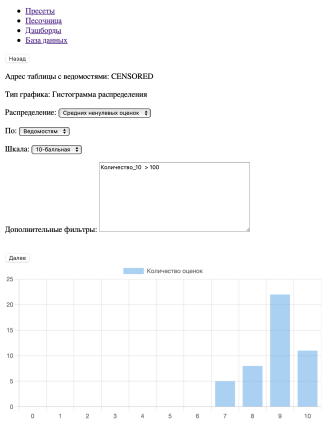


Рис. 6.4: Пример использования модуля песочницы фронтенд части. Показаны результаты вывода модуля по запросу: построить гистограмму распределения средних ненулевых оценок по ведомостям, где количество 10 больше 100. Вывести данные в 10-балльной системе оценивания.

Модуль песочницы является модулем, который позволяет пользователю построить различные графики без необходимости их сохранения на какой-либо дешборд.

В данном модуле пользователь отправляет запрос на создание объекта класса Chart или Constant, которые никуда не сохранятся. Этот модуль подразумевается для использования при необходимости посмотреть какую-то редко необходимую, и поэтому не сохраненную на дешбордах, инфографику, или для того, чтобы разобраться, как работает построение графи-

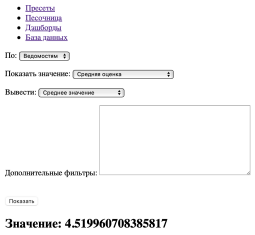


Рис. 6.5: Пример использования модуля песочницы фронтенд части. Показаны результаты вывода модуля по запросу: Вывести число, являющееся средним арифметическим среди средних оценок всех ведомостей.

ков.

Данный модуль состоит из нескольких страниц. На первой его просят ввести тип графика, который он хочет получить. Это может быть гистограмма распределения, круговая диаграмма распределения или константное число.

Если пользователь выбирает гистограмму распределения или круговую диаграмму распределения, то его переадресует на страницу с вводом различной информации, необходимой для построения такого графика. Конкретно его просят ввести, распределение какой величины он хочет увидеть, по какой таблице, по какой шкале и с какими дополнительными фильтрами. Говоря в терминах класса Chart, пользователь вводит переменные chart_type, distribution_source, database_table, distribution_scale и database_filters.

После этого происходит перевод данных, введенных пользователем, в SQL-запрос, который затем отправляется в объект класса бэкенда. Полученные данные, в зависимости от distribution_scale, переводятся в 5-балльную систему или остаются в 10-балльной. Также от этой переменной зависят и метки графика. После выполнения всех этих действий, полученный график выводится на экран пользователя.

В случае, если пользователь выбирает, что хочет получить число, пайплайн несколько меняется. Пользователя также переводит на страницу с формой, в которой нужно заполнить поля, однако сами поля отличаются. В этом случае пользователю нужно ввести величину, по которой он хочет считать, как считать итоговое значение, по какой таблице его считать и дополнительные фильтры. Говоря в терминах класса Constant, пользователь вводит переменные database_table, database_select, aggregation и database_filters.

После этого происходит перевод данных, введенных пользователем, в SQL-запрос, ко-

торый затем отправляется в объект класса бэкенда. Полученные данные обрабатываются в соответствие со значением поля aggregation, и выводятся на экран пользователя.

Пример работы модуля можно увидеть на рисунках 6.4 и 6.5.

6.4.5 Модуль дешбордов

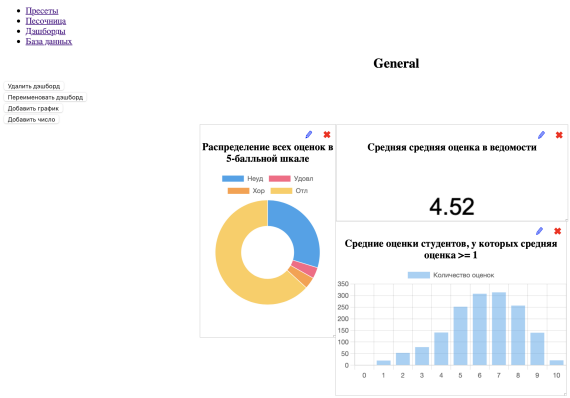


Рис. 6.6: Пример использования модуля дешбордов фронтенд части. Показан дешборд с 3 типами графиков. Пояснения к построению каждого графика написаны в их названиях.

Модуль дешбордов является модулем, который позволяет пользователю построить различные дешборды, на каждом из которых построить различные графики, сохранить их, их размер и расположение.

Первое, что нужно сделать пользователю - создать новый дешборд. После того, как он введет его название, дешборд сохранится и будет доступен к просмотру. Если зайти в него, то можно увидеть кнопки удаления и переименования дешборда, а также создания графика или числа. Кнопки удаления и переименования работают в соответствии с их названием.

Кнопки же создания графика или числа работают во многом как в модуле песочницы. Основным отличием является то, что здесь также требуется ввести название графика или числа. После этого пользователь сможет использовать кнопку "предпросмотр чтобы посмотреть, какой график получился, либо кнопку "добавить чтобы сохранить его на дешборде. Сами графики можно двигать и менять их размер. Их размеры и положение записаны в специальных полях класса Chart или Constant, и сохраняются туда после каждого изменения. При первом открытии страницы они, наоборот, подгружаются из базы данных Django.

Также существует возможность изменить график или удалить его, это делается путем нажатия на соответствующие иконки около графиков.

Перед удалением графика или дешборда появляется всплывающее окно с вопросом, точно ли пользователь хочет удалить данный элемент. При нажатии на кнопку "да"элемент

удалятся.

Пример дешборда можно увидеть на рисунке 6.6.

6.4.6 Вспомогательные модули

Модуль подсказок Модуль дешбордов является вспомогательным модулем, который позволяет пользователю видеть подсказки при вводе текста в поле дополнительных фильтров.

Для показа подсказок сначала определяется, из какой таблицы пользователь хочет видеть информацию. В зависимости от этого текст подсказок будет разниться. После этого происходит определение, что сейчас вводит пользователь. Если пользователь еще не начал что-то вводить или начал вводить какое-то слово, соответствующее названию столбца в таблице, то он будет видеть как подсказки столбцы, начинающиеся так же. Также, если пользователь только что ввел операнд, то ему так же предлагаются столбцы как подсказки. В иных случаях, например, если пользователь только что закончил ввод названия столбца и нажал пробел, ему предлагаются операнды, такие как +, -, /, * и так далее.

Модуль навигации Модуль навигации является вспомогательным модулем, который отвечает за формирование панели навигации на каждой странице, где это необходимо.

При запуске веб-интерфейса запускается скрипт, который создает нужные количество объектов класса MenuItem и добавляет их в базу данных Django. Таким образом, панель навигации доступна на любой странице путем написания всего одной строчки в HTML шаблоне.

7 Тестирование и результаты

Этот раздел выполнялся совместно Бекяном Артёмом и Тимофеевой Юлией.

Тестировали качество парсинга мы двумя способами - визуальным и ручным. Осуществлялось оно на данных 2021/2022 учебного года, так как для тестирования парсинга нужны были наиболее полные данные.

Визуальная проверка заключалась в следующем: сначала мы открывали исходную ведомость и искали оценочные колонки. Т.е. такие колонки, значения которых нужно обязательно учесть в качестве оценки. Далее мы открывали обработанную парсером копию и по полю MarksDict проверяли, какие колонки парсер посчитал оценочными. Если множества колонок совпадают или отличаются незначительно ($\pm 5\%$), считали, что исходная ведомость обработана корректно с точностью близкой к 100%. Иначе, применяли ручную проверку.

Ручная проверка заключалась в полностью ручном подсчете всех метрик исходной таблицы. Т.е. вычислялись все метрики для конкретного студента, а после - вычисленная информация агрегировалась уже по всем студентам (см. рис. 7.1 и рис. 7.2)

ФИО	Средняя	Сумма	Число неотр	Учитывали?	Отлы	Хоры	Уды	Неуды
Студент1	8,205555556	147,7	18	1	18	0	0	0
Студент2	7,755555556	139,6	17	1	15	2	0	1
Студент3	7,533333333	135,6	16	1	16	0	0	2
Студент4	8,002777778	144,05	18	1	15	3	0	0
Студент5	6,983333333	125,7	17	1	13	1	2	2
Студент6	7,422222222	133,6	18	1	11	7	0	0
Студент7	6,383333333	114,9	16	1	10	4	1	3
Студент8	7,527777778	135,5	17	1	17	0	0	1
Студент9	7,483333333	134,7	17	1	15	2	0	1

Рис. 7.1: Ручной анализ успеваемости студентов

SUM	SUMMean	CntPositive	Cnt		СУМ ОТЛ	СУМ ХОР	СУМ УД	СУМ НЕУД
13634,55	757,475	1780	2124		1276	306	109	433
	MeanMark	MeanMean	MeanPosit	Excellent	Good	Satisfactory	Fail	
	6,412314501	6,082023905	7,694376757	2284	600	211	746	

Рис. 7.2: Агрегирование данных студентов

Далее уже точные метрики (вычисленные вручную) сравнивались с метриками, получившимися у парсера, вычислялась «похожесть» показателей в процентах и делался финальный вывод о качестве парсинга на конкретной ведомости (см. рис. 7.3). Если точность получалась «хорошая» (более 85%), то переходили к анализу обработки других ведомостей. Иначе - возвращались к отладке кода и пересмотру алгоритмов.

Качество парсинга (Mean+Mean Positive)	MeanMark	MeanMarkAccuracy	MeanPositiveMark	MeanPositiveAccuracy	MeanMeanMark	MeanMeanAccuracy	Excellent	ExcellentAccuracy	Good	GoodAccuracy	Satisfactory	SatisfactoryAccuracy	Fail	FailAccuracy
93,01%	5,56	86,79%	7,64	0,99	5,73	0,94	2677	1	661	1	263	1	1595	46,77%

Рис. 7.3: Сопоставление метрик, вычисленных вручную и автоматически

Такой подход существенно помогал корректировать алгоритмы парсера во время разработки и в конце концов мы добились высокой точности парсинга вне зависимости от конкретных ведомостей.

Из-за временных ограничений нам удалось отладить обработку только ведомостей в формате Google Sheet. Однако, среди все предоставленных ведомостей, которые были корректны, доступны и которые можно было обработать (не были удалены, не были закрыты доступы, не было ошибок в адресах ссылок и др.), их число составляет почти 80% (см. рис. 7.4).

В тип Google Sheet также попадают Google таблицы, сохраненные на диске в другом формате, например, в формате xlsx-файла. Google API не обрабатывает такие форматы.

```
print("Среди доступных ведомостей\n")
for url_type in config["URL_TYPE"].values():
    cnt = len(df_root[(df_root.Type == url_type) & (pd.isna(df_root.ErrorName))])
    print("{} ~ {} %".format(url_type, math.ceil(100 * cnt / len(df_root[pd.isna(df_root.ErrorName)]))))

cnt = len(df_root[pd.isna(df_root.Type) & (pd.isna(df_root.ErrorName))])
print("\nOther types ~", math.ceil(100 * cnt / len(df_root[pd.isna(df_root.ErrorName)])), "%")
```

```
Среди доступных ведомостей

Google Sheet ~ 79 %
Drive Directory ~ 2 %
Drive PDF ~ 5 %
Drive Unknown File ~ 0 %
Sharepoint ~ 2 %
MS Teams ~ 0 %
Dropbox ~ 4 %

Other types ~ 10 %
```

Рис. 7.4: Анализ количества ведомостей по типу: Analysis.ipynb

Эмпирические наблюдения показали, что таких файлов - около 10% от числа Google Sheet ведомостей. Поэтому можем считать, что StatementAnalysis способен обрабатывать около 70% всех корректно подающихся ведомостей ФКН.

Одной из важнейших задач данного проекта была задача минимизации ручной обработки ведомостей. В среднем, по мнению нашего руководителя, на анализ ведомостей уходит от одной до двух недель. В среднем в зависимости от операционной системы и прочих характеристик сервера на обработку ведомостей уходят сутки. Таким образом, нам удалось теоретически сильно ускорить процесс анализа ведомостей и, возможно, освободить много времени человека, этим занимавшегося.

8 Заключение

В результате получилась удобная система, способная автоматически собирать данные из ведомостей и выводить ее в удобном для пользователей формате. Помимо того, что пользователь имеет возможность самому генерировать дешборды, у него также есть возможность использовать специально созданные метрики для более точного анализа и понимания ситуации.

Тем не менее, существует еще несколько вещей, которые доработать в рамках этого проекта. Относительно фронтенд части, сильно не хватает дизайна, а также хромает безопасность запросов с дополнительными фильтрами.

В части бэкенда, очевидно, хотелось бы обрабатывать и другие форматы ведомостей, отличные от Google Sheet. Так же можно было бы попытаться сделать результаты парсинга еще более точными, например, за счет подключения машинного обучения к алгоритмам. Ну и наконец, можно также поработать над ускорением времени работы парсера.

Список литературы

[1] Chart.js. *Chart.js Documentation*. URL: https://www.chartjs.org/docs/latest/ (дата обр. 13.12.2022).

[2] docs.python.org. *Container datatypes*. URL: https://docs.python.org/3/library/collections.html (дата обр. 29.01.2023).

[3] docs.python.org. *DB-API 2.0 interface for SQLite databases*. URL: https://docs.python.org/3/library/sqlite3.html (дата обр. 05.05.2023).

[4] docs.python.org. *Regular expression operations*. URL: https://docs.python.org/3/library/re.html (дата обр. 15.11.2022).

[5] Django Software Foundation. *Django documentation*. URL: https://docs.djangoproject.com/en/4.1/ (дата обр. 08.02.2023).

[6] OpenJS Foundation. *Jquery documentation*. URL: https://api.jquery.com (дата обр. 14.05.2023).

[7] numpy.org. *NumPy documentation*. URL: https://numpy.org/doc/stable/user/index.html#user (дата обр. 22.01.2023).

[8] pandas.pydata.org. *pandas documentation*. URL: https://pandas.pydata.org/docs/user_guide/index.html#user-guide (дата обр. 15.11.2022).

[9] pygsheets.readthedocs.io. *pygsheets 1.1 documentation*. URL: https://pygsheets.readthedocs.io/en/latest/ (дата обр. 11.11.2022).

[10] Python.org. *Python Release Python 3.8.1*. URL: https://www.python.org/downloads/release/python-381/ (дата обр. 10.05.2023).

[11] requests.readthedocs.io. *Requests: HTTP for Humans*. URL: https://requests.readthedocs.io/en/latest/user/quickstart/ (дата обр. 30.10.2022).

[12] Википедия. *Нормальная форма*. URL: https://ru.wikipedia.org/wiki/%D0%9D%D0%BE%D1%80%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B0%D1%8F_%D1%84%D0%BE%D1%80%D0%BC%D0%B0 (дата обр. 01.05.2023).

[13] Бессмертный А. И. *Виды ведомостей*. URL: https://telegra.ph/Vidy-vedomostej-10-15 (дата обр. 15.10.2022).

[14] Илья Кантор. *Справочник JavaScript*. URL: https://javascript.ru/manual.

[15] Влад Мержевич. *Справочник CSS*. URL: http://htmlbook.ru/css (дата обр. 02.05.2023).

[16] Влад Мержевич. *Справочник по HTML*. URL: http://htmlbook.ru/html (дата обр. 19.01.2023).

9 Приложения

9.1 Терминологический словарь

Парсинг ведомостей - автоматизированный сбор и систематизация информации из ведомостей с помощью программ на одном или нескольких языках программирования.

Метрика ведомости - статистическая функция, применяемая к каждой оценке каждого студента, учтенного в конкретной ведомости, или функция, агрегирующая данные конкретной ведомости.

Root-таблица - Google-таблица, в которую представители образовательных программ заносят ссылки на ведомости предметов.

Бэкендом будем называть часть работ, относящуюся обработке и хранению данных - парсинг root-таблицы и ведомостей, запись обработанных данных в базу, дополнение метриками, предоставление срезов данных по запросу пользователя и другое.

Фронтендом - будем называть часть работ, относящуюся к веб-интерфейсу и взаимодействию с пользователем - разработка инфографики, разработка клиент-серверного приложения, разработка передачи данных от пользователя бэкенду и наоборот и другое.

9.2 Примеры кода

```
def __is_name(self, s):
    r1 = re.match(r"^(\s*[a-zA-Za-яА-Я]+.*\s+[a-zA-Za-яА-Я]+.*\s+[a-zA-Za-яА-Я]+.*\s+[a-zA-Za-яА-Я]+.*\s*)", s)
    r2 = re.match(r"^(\s*[a-zA-Za-яА-Я]+.*\s+[a-zA-Za-яА-Я]+.*\s+[a-zA-Za-яА-Я]+.*\s*)", s)
    r3 = re.match(r"^(\s*[a-zA-Za-яА-Я]+.*\s+[a-zA-Za-яА-Я]+.*\s*)", s)
    r4 = re.search(r"([0-9])", s)
    return ((r1 is not None) or (r2 is not None) or (r3 is not None)) and (r4 is None)
```

Рис. 9.1: Код функции, выявляющей имена

```
def __is_formula(self, value, title, titles):
    if type(value) != str:
        return (False, [])
    if value != '' and value[0] == '=':
        regex = re.compile("(" + "|".join([t for t in titles]) + ")")
        found_titles = re.findall(regex, value)
        if len(found_titles) != 0:
            return (True, found_titles)
        if len(self.cells_from_formula(value)) == 0:
            return (False, [])
        return (True, [])
    return (False, [])
```

Рис. 9.2: Код функции, выявляющей данные, вычисленные на основе других данных