Филиал Московского Государственного Университета
имени М.В. Ломоносова в городе Ташкенте

Факультет прикладной математики и информатики
Кафедра прикладной математики и информатики

Абдуллаева Евгения Гасановна

КУРСОВАЯ РАБОТА
на тему: «Перенос сообщений из системы сопровождения „Практикума на ЭВМ“ „Форум МГУ“ в систему сопровождения „МГУ Контест“»

по направлению 01.03.02 «Прикладная математика и информатика»

Научный руководитель
к.ф.-м.н., в.н.с. Алисейчик Павел Александрович

«_____» _____ 2020 г.

Ташкент 2020 г.

Перенос сообщений из системы сопровождения «Практикума на ЭВМ» «Форум МГУ» в систему сопровождения «МГУ Контест»

Цель настоящей работы – реализовать автоматическую систему переноса базы сообщений из системы «Форум МГУ» в систему «МГУ Контест». Обе упомянутые системы являются инструментами поддержки курса «Практикум на ЭВМ» в филиале МГУ имени М.В. Ломоносова в городе Ташкенте. Осуществлена подготовка базы сообщений из системы «Форум МГУ» к переносу, а также реализованы все необходимые непосредственно для переноса сообщений в систему «МГУ Контест» процедуры.

Содержание

1 Введение

За годы сопровождения курса «Практикум на ЭВМ» в филиале МГУ им. М.В. Ломоносова в Ташкенте системой «Форум МГУ» накоплена база сообщений, включающая вопросы студентов и ответы преподавателей по различным этапам и темам данного курса, являющаяся ценным образовательным ресурсом. В связи с созданием системы «МГУ Контест» [2] возникла необходимость переноса базы сообщений в новую систему. Сохранение накопленной базы сообщений ведет как к экономии времени преподавателей, освобождая их от необходимости многократно отвечать на повторяющиеся вопросы, так и студентов, которым не придется ожидать ответа на заданный вопрос, а следовательно способствует развитию обучающей системы.

В настоящий момент качество образования является одним из ключевых факторов определяющих развитие общества, именно благодаря образованию человек приобретает способность свободно и независимо мыслить, имплементировать собственные идеи и эффективно использовать интеллектуальные ресурсы для решения множества практических задач. Преподавание в наши дни представляет собой нить, по которой ученик продвигается посредством исследований и открытий. Следовательно, актуальной задачей является повышение эффективности процесса обучения, а в частности помощь преподавателям в предоставлении студентам обучающего материала.

2 Общая структура и организация работы

Настоящая работа содержит две основные части, описываемые в главах 3 и 4. В главе 3 в основном ведется работа с базой данных системы «Форум МГУ», в главе 4 – с базой данных системы «МГУ Контест». Обе системы представлены проектами, построенными с использованием фреймворка для веб-приложений Django [3], написанного на языке Python [4], поэтому при работе с базами данных проектов используются в основном средства, предоставляемые Django и лишь отчасти реляционной системой управления базами данных MySQL [5].

Проект на Django строится из одного или нескольких приложений. Django использует шаблон проектирования Model-View-Controller [6] – схему разделения данных приложения, пользовательского интерфейса и управляющей логики на три отдельных компонента: модель, представление и контроллер. Данная схема предоставляет возможность независимой модификации каждого компонента.

Модели в Django являются источником информации о данных, содержат поля и описание поведения данных, которые мы храним, и реализованы как классы Python. Каждая модель отображается в одну таблицу базы данных.

В проекте «Форум МГУ» в процессе работы мы будем взаимодействовать со следующими моделями:

- TashkentBoard (доска) – наиболее общая сущность, соответствующая этапу курса «Практикум на ЭВМ» или иной связанной с программированием дисциплине;
- TashkentTopic (тема) – сущность, соответствующая подразделу доски;
- TashkentMessage (сообщение) – сущность, соответствующая отдельному сообщению, написанному пользователем системы;
- TashkentMember (пользователь) – сущность, соответствующая аккаунту пользователя системы.

В проекте «МГУ Контест» работа будет осуществляться с моделями:

- Course (курс) – наиболее общая сущность, соответствующая этапу курса «Практикум на ЭВМ»;
- Contest (раздел) – сущность, соответствующая подразделу курса;
- Problem (задача) – сущность, соответствующая отдельной задаче раздела;

- Comment (комментарий) – сущность, соответствующая отдельному комментарию, написанному пользователем системы;
- Account (аккаунт) – сущность, соответствующая аккаунту пользователя системы.

При упоминании экземпляров моделей будет использован термин *объект*. При упоминании объектов сообщений при работе в проекте «Форум МГУ» для этих объектов будет использовано название *сообщение*, при упоминании же объектов сообщений при работе в проекте «МГУ Контест» будет использовано название *комментарий*.

3 Подготовка к переносу сообщений

3.1 Выделение сообщений необходимых для переноса

Первая задача, которую необходимо выполнить для подготовки к переносу сообщений – определить, какие сообщения следует переносить, так как обязательно соответствие между темами, обсуждаемыми в сообщениях, и содержанием курсов системы «МГУ Контест». Решение задачи осуществляется в два этапа: выделим доски, необходимые для переноса; затем выделим темы и принадлежащие им сообщения, которые будут перенесены. Для этого удобно иметь возможность просматривать сообщения, относящиеся к различным темам и доскам.

Средства управления администратора, предоставляемые Django, не позволяют просматривать текст нескольких сообщений, сгруппированных по какому-либо признаку, поэтому для просмотра сообщений из базы данных системы «Форум МГУ» с указанием их принадлежности к определенной доске и теме используется следующее решение.

Реализована функция `get_messages_view`, которая позволяет получить представление списка сообщений, разделенных по доскам и темам в виде текста с HTML-форматированием. Данная функция принимает в качестве аргументов список досок `all_boards`, список тем `all_topics` и список сообщений `all_messages`, которые необходимо отобразить. В теле функции объявляется переменная `html` типа `str`, изначально пустая. Для каждой доски из списка досок `all_boards` к переменной `html` прибавляется идентификатор и имя доски, заключенные в HTML-теги заголовка первого уровня; переменной `topics` присваивается список тем из `all_topics`, принадлежащих рассматриваемой в текущий момент доске. Для каждой темы из `topics` к переменной `html` прибавляется идентификатор и название темы (определяется как заголовок первого сообщения в теме), заключенные в HTML-теги заголовка второго уровня; переменной `messages` присваивается список сообщений из `all_messages`, относящихся к рассматриваемой в текущий момент теме; сообщения в списке `messages` сортируются по возрастанию даты их написания, определяемой полем `postertime` модели сообщения. Для каждого сообщения из `messages` к переменной `html` прибавляется идентификатор, заголовок, имя автора сообщения, имя изменявшего текст сообщения пользователя, дата написания и непосредственно текст сообщения. Функция возвращает переменную `html`.

Описанная функция вызывается как возвращаемое значение function-based view (представления в функциональном виде) [7] `all_messages`. В качестве аргументов в функцию передаются списки всех существующих в базе досок, всех

тем и всех сообщений. Таким образом данное view предоставляет возможность просмотра сообщений, сгруппированных по принадлежности к доскам и темам в браузере по URL-адресу `<host>:<port>/all_messages`.

База данных системы «Форум МГУ» содержит следующие доски:

1. Практикум на ЭВМ: общие положения
2. Практикум по программированию, 1 курс, 1 семестр
3. Практикум по программированию, 1 курс, 2 семестр
4. Практикум по программированию, 2 курс, 1 семестр
5. Практикум по программированию, 2 курс, 2 семестр
6. Практикум по программированию, 3 курс, 1 семестр
7. Практикум по программированию, 3 курс, 2 семестр
8. Практикум по программированию, 4 курс, 1 семестр
9. 4 курс, 2 семестр
10. Магистратура мех-мат ф-та, практикум
11. Прочее
12. Ассемблер
13. Семинар «Программирование интеллектуальных систем»
14. Системы программирования
15. Спортивное программирование
16. Курсовые и дипломные работы
17. Служебный раздел

База данных системы «МГУ Контест» содержит курсы, соответствующие этапам «Практикума на ЭВМ» начиная от первого семестра первого курса до первого семестра четвертого курса, перечисленные ниже:

1. Простые алгоритмы
2. Дискретная математика
3. Операционные системы

4. Основы ООП
5. Дискретная оптимизация
6. Численные методы, часть 1
7. Численные методы, часть 2

Перенос осуществляется лишь для тех сообщений из базы данных системы «Форум МГУ», которые принадлежат доскам, соответствующим этапам курса «Практикум на ЭВМ» или содержат общие положения проведения «Практикума на ЭВМ». Данному условию соответствуют сообщения, относящиеся к доскам: «Практикум на ЭВМ: общие положения», «Практикум по программированию, 1 курс, 1 семестр», «Практикум по программированию, 1 курс, 2 семестр», «Практикум по программированию, 2 курс, 1 семестр», «Практикум по программированию, 2 курс, 2 семестр», «Практикум по программированию, 3 курс, 1 семестр», «Практикум по программированию, 3 курс, 2 семестр», «Практикум по программированию, 4 курс, 1 семестр», «Прочее». Присвоим выделенное множество досок константе `NECESSARY_BOARDS`. Далее необходимо определить, какие темы, принадлежащие выделенным доскам, будут перенесены.

В результате просмотра сообщений определяются темы, для которых перенос осуществлен не будет. Таковыми являются темы с условиями задач, так как условия задач уже присутствуют в системе «МГУ Контест»; темы, содержащие устаревшую информацию: объявления и распределение задач. Множество тем, включающих сообщения с полезной информацией, присвоим константе `NECESSARY_TOPICS`. Множество сообщений, принадлежащих темам из `NECESSARY_TOPICS` запишем в константу `NECESSARY_MESSAGES`. Таким образом получены необходимые для переноса доски, темы и сообщения.

3.2 Разделение совмещенных сообщений

В системе «Форум МГУ» преподаватели вставляли некоторые ответы на вопросы студентов непосредственно в текст вопроса и выделяли текст ответа одним из следующих цветов: голубым, синим или зеленым. При этом форматирование текста осуществлялось с помощью тегов языка разметки BBCode [8], имеющих следующий вид: `[color=<color_name>]`. В системе «МГУ Контест» существует возможность привязывать ответ к конкретному комментарию, соответственно, возникает задача выделить отдельные сообщения из совмещенных, сохраняя при этом цепочки вопросов и ответов.

Чтобы приступить к решению данной задачи, найдем для начала все совмещенные сообщения. Воспользуемся для поиска модулем re [9] языка Python, предоставляющим функции для работы с регулярными выражениями. Составим шаблон на языке регулярных выражений для поиска вхождений тегов языка разметки BBCode, отвечающих за голубой, синий или зеленый цвет текста:

```
\[color=blue]|\[color=navy]|\[color=green]
```

В константу с именем `COMBINED_MESSAGES` сохраним множество сообщений из `NECESSARY_MESSAGES`, удовлетворяющих условию:

```
if re.search(r'\[color=blue]|\[color=navy]|\[color=green]',
             message.body))
```

Множество же несовмещенных сообщений, полученное вычитанием множества совмещенных сообщений из множества всех необходимых для переноса сообщений, присвоим константе `UNCOMBINED_MESSAGES`.

Для сообщений из `NECESSARY_MESSAGES`, полученных в результате разделения, а также для сообщений, которые в разделении не нуждаются, создадим новую модель данных Message со следующими полями:

- id – содержит положительное целое число – уникальный идентификатор сообщения, является первичным ключом модели;
- parent_msg_id – содержит положительное целое число – id первого сообщения в цепочке, к которой относится данное сообщение;
- author – содержит внешний ключ, указывающий на автора сообщения – объект типа User, где User является встроенной моделью Django;
- text – содержит текст сообщения;
- date_created – содержит положительное целое число – время создания сообщения в формате POSIX-времени, где время определяется как количество секунд, прошедших с полуночи (00:00:00 UTC) 1 января 1970 года;
- subject – текстовое поле, содержащее заголовок сообщения;
- topic – содержит внешний ключ, указывающий на тему, к которой относится данное сообщение;
- board – содержит внешний ключ, указывающий на доску, к которой относится данное сообщение.

Создание промежуточной модели для сообщений позволяет включить в модель сообщения только те поля, которые будут сохранены при переносе, а также сохранить изначальные сообщения.

Реализуем функцию `create_new_messages`, позволяющую разделить совмещенные сообщения и создать разделенные и не нуждавшиеся в разделении сообщения из `NECESSARY_MESSAGES`, как объекты модели Message. Объявим переменную `new_messages` типа `set`, изначально пустую. В это множество будем помещать создаваемые сообщения. Для задания значений уникальных идентификаторов сообщений введем переменную `id_counter` типа `int` с начальным значением равным `1`.

Для каждого сообщения из `UNCOMBINED_MESSAGES` создадим новый объект модели Message, присваивая значения полям создаваемого сообщения следующим образом:

```
Message(
    id=id_counter,
    parent_msg_id=id_counter,
    author=message.id_member,
    text=message.body,
    date_created=message.postertime,
    subject=message.subject,
    topic=message.id_topic,
    board=message.id_board
)
```

и добавим созданное сообщение во множество `new_messages`. При каждой итерации к значению переменной `id_counter` будем прибавлять `1`.

Теперь необходимо разделить совмещенные сообщения `COMBINED_MESSAGES`. Введем специальную переменную-маркер, куда сохраним шаблон на языке регулярных выражений для поиска ответов преподавателей (текста, выделенного одним из упомянутых выше цветов):

```
teacher_message_marker = r'(?<=\[color=blue]).+?(?=\[/color])|' \
                         r'(?<=\[color=navy]).+?(?=\[/color])|' \
                         r'(?<=\[color=green]).+?(?=\[/color])'
```

Для каждого сообщения из `COMBINED_MESSAGES` выясним, начинается ли текст совмещенного сообщения со слов студента (текста, не выделенного цветом). Для этого введем переменную `student_begins` булевского типа, которая принимает значение `True` в случае, если начало сообщения совпадает с шаблоном:

```
\[color=blue]|\[color=navy]|\[color=green]
```

и принимает значение `False` иначе. Затем получим из текста сообщения все части, соответствующие сообщениям студента (таких частей может быть больше одной), используя функцию `split` из модуля re следующим образом:

```
re.split(teacher_message_marker, message.body)
```

В данном случае разделителями для функции `split` являются сообщения преподавателя. Результирующий список сообщений студента сохраним в переменную `student_messages`. В текст сообщений студента, помимо самого сообщения будут записаны так же закрывающие и открывающие теги языка разметки BBCode, заключавшие в себя текст сообщений преподавателя. Для каждого сообщения в `student_messages` удалим из текста сообщения эти теги. Список сообщений преподавателя получим используя функцию `findall` модуля re следующим образом:

```
re.findall(teacher_message_marker, message.body)
```

Сохраним список сообщений преподавателя в переменную `teacher_messages`. Теперь необходимо объединить сообщения студента и преподавателя в один список, сохраняя изначальную последовательность сообщений. Введем переменную `splitted_messages` типа `list`, изначально пустую и переменную `last_index` типа `int` с начальным значением равным `0`. Для каждого индекса списка сообщений студента или списка сообщений преподавателя (выбор списка осуществляется в зависимости от длины списков, выбирается список с минимальной длиной) добавим в конец списка `splitted_messages` сначала сообщение студента с текущим индексом в списке `student_messages`, затем сообщение преподавателя из `teacher_messages` с тем же индексом, если значение переменной `student_begins` равно `True`, иначе добавим сначала сообщение преподавателя из `teacher_messages` с текущим индексом, а затем сообщение студента из `student_messages` с тем же индексом. После добавления двух сообщений в список `splitted_messages` переменной `last_index` присвоим значение текущего индекса. По завершении этого цикла, если в каком-либо из списков `student_messages` или `teacher_messages` есть сообщения с индексами больше `last_index`, то добавим все эти сообщения в конец списка `splitted_messages`. Удалим из списка `splitted_messages` все сообщения, текстом которых является пустая строка, если такие присутствуют. Для каждого индекса из полученного списка сообщений `splitted_messages` создадим объект модели Message для сообщения из `splitted_messages` с этим индексом. Для этого получим уникальный идентификатор автора сообщения и сохраним в переменную `id_member`. Если сообщение с текущим индексом является сообщением преподавателя и поле `modifiedname` данного сообщения не является пустой строкой, то значение `id_member` будет получено как уникальный идентификатор объекта модели TashkentMember, у которого значение поля `realname` равно значению поля

`modifiedname` данного сообщения. Иначе переменной `id_member` будет присвоено значение поля `id_member` текущего сообщения. Создавать новые объекты типа Message будем следующим образом:

```
Message(
    id=id_counter,
    parent_msg_id=id_counter - i,
    author=id_member,
    text=splitted_messages[i],
    date_created=message.postertime + i,
    subject=message.subject,
    topic=message.id_topic,
    board=message.id_board
)
```

Созданное сообщение добавим во множество `new_messages`. При каждой итерации к значению переменной `id_counter` будем прибавлять `1`.

Мы получили множество сообщений `new_messages`, содержащее все сообщения из `NECESSARY_MESSAGES`, где совмещенные сообщения разделены и сохранены как цепочки нескольких последовательных сообщений. Вставим объекты из множества `new_messages` в базу данных, используя стандартный метод Django QuerySet API [10] `bulk_create` следующим образом:

```
Message.objects.bulk_create(new_messages)
```

3.3 Получение промежуточного представления пользователей

Чтобы привести оставлявших сообщения пользователей к промежуточному состоянию предшествующему их переносу создадим новую модель данных User со следующими полями:

- id – содержит положительное целое число – уникальный идентификатор пользователя, является первичным ключом модели;
- username – содержит короткое имя пользователя;
- full_name – содержит реальные имя и фамилию пользователя;
- first_name – содержит фамилию пользователя;

- last_name – содержит имя пользователя;
- email – содержит адрес электронной почты пользователя;
- admission_year – содержит положительное целое число – год поступления пользователя.

Для создания пользователей как объектов модели User реализуем функцию `create_users`, принимающую в качестве аргумента `necessary_members` множество пользователей. Введем переменную `users` типа `set`, изначально пустую. Для каждого пользователя `member` из `necessary_members` создадим объект модели User, присваивая значения полям объекта следующим образом:

```
User(
    id=member.id_member,
    username=member.membername,
    full_name=member.realname,
    email=member.emailaddress,
    admission_year=
        datetime.fromtimestamp(member.dateregistered).year
)
```

Здесь при получении года поступления пользователя из формата времени POSIX используется метод `fromtimestamp` объекта datetime модуля datetime [13] языка Python, предоставляющего классы для работы с датами и временем. Вставим объекты из множества `users` в базу данных, используя стандартный метод Django QuerySet API `bulk_create` следующим образом:

```
User.objects.bulk_create(users)
```

Свяжем теперь созданных пользователей с сообщениями, получение которых описано в предыдущем разделе. Данное действие необходимо для установления авторства пользователей новой модели. Для этого реализуем функцию `link_messages_with_users`, которая для каждого `message` – объекта типа Message из базы данных – полю `user` данного объекта присваивает объект типа User, у которого значение поля `id` совпадает со значением уникального идентификатора автора данного сообщения, то есть со значением `message.author.id_member` и по завершении итераций по сообщениям обновим поле `user` во всех экземплярах модели Message, чтобы произвести данное действие одним запросом, используем стандартный метод Django QuerySet API `bulk_update` следующим образом:

```
Message.objects.bulk_update(messages, ['user'])
```

3.4 Стандартизация имен пользователей

Каждый пользователь в системе «Форум МГУ» обладает собственным, возможно, не уникальным именем, наличие которого обеспечивается CharField-полем `full_name` модели User (ранее полем `realname` модели TashkentMember). Это имя является строкой и определяется самим пользователем-человеком при регистрации в системе (на веб-сайте). При этом на формат имени не накладываются ограничения: это может быть одно или несколько слов, для записи которых могут использоваться алфавиты разных языков и некоторые специальные знаки. Полю `full_name` модели User системы «Форум МГУ» неформально соответствуют поля-строки `firstname` и `lastname` модели User в системе «МГУ Контест». Это означает, что имя пользователя в системе «МГУ Контест» разделено на непосредственно имя и фамилию. Ещё два ограничения, накладываемых на имя и фамилию пользователя в системе «МГУ Контест», обозначены требованием использовать для записи имени и фамилии исключительно кириллический алфавит и требованием записывать первые буквы имени и фамилии заглавными, а остальные – строчными.

В связи с вышеописанными ограничениями возникает задача: привести имена пользователей в системе «Форум МГУ» в формат, соответствующий требованиям к имени и фамилии в системе «МГУ Контест». Возможно было бы несложно автоматизировать это приведение, если бы задача состояла только в транслитерации букв имени латинского алфавита в буквы кириллического алфавита, а также в изменении первых букв имени и фамилии на заглавные, при условии, что имя и фамилия находятся в поле `full_name` в определённом порядке (то есть, например, сначала имя, затем фамилия). Однако исходный вид имён пользователей не удовлетворяет описанным условиям, поэтому приведение осуществляется вручную. Для этого выводится список имён пользователей и с помощью инструмента администрирования Django точечно выполняются переименования.

Для ускорения ручной работы реализуем function-based view `users`, предоставляющее возможность просмотра по URL-адресу `<host>:<port>/users` полей экземпляров модели User системы «Форум МГУ» в едином списке со ссылками на страницы инструмента администрирования Django для редактирования каждого пользователя. В теле функции `users` объявим переменную `html` типа `str`, изначально имеющую значение пустой строки. Введем счетчик `i` с начальным значением равным `1`. Для всех объектов `user` модели User системы «Форум МГУ» прибавим к переменной `html` заключенные в необходимые HTML-теги форматирования текста значение счетчика `i`, уникальный идентификатор пользователя, значение поля `full_name` пользователя, значение поля `username` пользователя и ссылку на страницу для редактирования данного пользователя, полученную следующим образом:

```
<host>:<port>/admin/board/user/{user.id}/change
```

При каждой итерации будем увеличивать значение счетчика `i` на `1`. Возвращаемым значением функции `users` будет предоставляемый Django объект HttpResponse [11], в конструктор которого будет передано содержимое веб-страницы в виде строки `html`.

После того, как для всех пользователей поля `full_name` будут приведены в требуемый формат, получим отдельно фамилии и имена пользователей и добавим их в соответствующие поля `first_name` и `last_name` модели пользователя. Для этого реализуем функцию `add_users_first_name_last_name`, которая для всех объектов `user` модели User системы «Форум МГУ» полю `last_name` присваивает имя пользователя, а полю `first_name` присваивает фамилию пользователя, где имя и фамилия пользователя получаются как элементы списка возвращаемого стандартным методом `split` языка Python для разбиения строк следующим образом:

```
user.full_name.split()
```

Обновим поля `first_name` и `last_name` во всех экземплярах модели User системы «Форум МГУ», используя стандартный метод Django QuerySet API `bulk_update` следующим образом:

```
User.objects.bulk_update(users, ['first_name', 'last_name'])
```

Кроме имени и фамилии каждый пользователь в системе «Форум МГУ» обладает коротким именем, наличие которого обеспечивается CharField-полем `username` модели User (ранее полем `membername` модели TashkentMember). Короткое имя также как и имя с фамилией определяется пользователем-человеком при регистрации в системе (на веб-сайте) и на формат короткого имени не накладываются ограничения. Полю `username` модели User системы «Форум МГУ» неформально соответствует поле `username` модели User системы «МГУ Контест». Короткое имя пользователя в системе «МГУ Контест» создается автоматически при добавлении пользователя в систему и имеет следующий формат:

```
msu_<год поступления>_<двузначный порядковый номер студента>
```

В связи с этим возникает задача: заменить короткие имена пользователей в системе «Форум МГУ» на короткие имена требуемого в системе «МГУ Контест» формата. Для решения данной задачи реализуем функцию `update_usernames`. В теле функции введем переменную `usernames` типа `set`. Для всех объектов `user` модели User системы «Форум МГУ» полю `username` присвоим новое значение, полученное с помощью следующей конструкции:

```
i = 1
while 'msu_' + str(user.admission_year) + '_' + str(i).zfill(2) \
        in usernames:
    i += 1
```

Добавим значение, присвоенное полю `username` во множество `usernames`. По завершении итераций по пользователям обновим поле `username` во всех экземплярах модели User при помощи стандартного метода Django QuerySet API `bulk_update`.

3.5 Получение идентификаторов курсов, разделов и задач

В системе «МГУ Контест» каждый комментарий имеет поля `object_type` (содержит внешний ключ, указывающий на тип объекта, к которому принадлежит комментарий) и `object_id` (содержит целое положительное число – уникальный идентификатор объекта, к которому принадлежит комментарий), однозначно определяющие, к какому объекту (курсу, разделу или задаче) относится данный комментарий. Чтобы сообщения из системы «Форум МГУ» при переносе были корректно отнесены к соответствующим их содержанию объектам в системе «МГУ Контест», необходимо предварительно задать их отношение к данным объектам, другими словами до осуществления переноса необходимо распределить сообщения из системы «Форум МГУ» по курсам, разделам и задачам из системы «МГУ Контест». Для решения данной задачи необходимо экспортировать в каком-либо виде список объектов (курсов, разделов и задач) из системы «МГУ Контест» и импортировать его в систему «Форум МГУ». Для сохранения списка объектов в файле вне базы данных выбран текстовый формат обмена данными JSON [12], так как он позволяет использовать те же структуры данных, что и язык Python, в частности набор пар ключ-значение, реализованный в Python как структура данных `dict` и упорядоченный набор значений, реализованный в Python как структура данных `list`. Для работы с JSON используется стандартный модуль json языка Python.

Реализуем функцию `get_identifiers_json`, позволяющую экспортировать список курсов (представлены моделью данных Course), разделов (представлены моделью данных Contest) и задач (представлены моделью данных Problem) из системы «МГУ Контест» в файл формата JSON. Для каждого объекта сохраним его тип, уникальный идентификатор и название. Данных свойств объекта достаточно для распределения сообщений по объектам. Типы объектов пронумеруем следующим образом: 1 – курс, 2 – раздел, 3 – задача, и сохранять будем номер типа объекта. Введем переменную `identifiers` типа `list`. Для каждого

объекта `course` из всех объектов типа Course в список `identifiers` добавим словарь вида:

```
{
    'type': 1,
    'id_in_contest': course.id,
    'name': course.title
}
```

Так же для каждого объекта `contest` из всех объектов типа Contest в список `identifiers` добавим словарь следующего вида:

```
{
    'type': 2,
    'id_in_contest': contest.id,
    'name': contest.course.title + '>>>' + contest.title
}
```

И аналогично для каждого объекта `problem` из всех объектов типа Problem в список `identifiers` добавим словарь следующего вида:

```
{
    'type': 3,
    'id_in_contest': problem.id,
    'name': problem.contest.course.title + '>>>'
            + problem.contest.title + '>>>'
            + problem.title
}
```

Сериализуем полученный список `identifiers` с помощью функции `dumps` модуля json и сохраним в файл identifiers.json.

Чтобы сохранить полученные идентификаторы объектов в системе «Форум МГУ» создадим новую модель данных Identifier со следующими полями:

- object_type – содержит положительное целое число – номер типа объекта;
- object_id_in_contest – содержит положительное целое число – уникальный идентификатор объекта в системе «МГУ Контест»;
- object_name – текстовое поле, содержащее название объекта.

Для импорта идентификаторов из файла формата JSON и создания их как объектов модели Identifier реализуем функцию `create_identifiers`. В теле функции десериализуем данные из файла identifiers.json с помощью функции `loads` модуля json в объект-список `identifiers`. Для каждого объекта `identifier` из списка `identifiers` создадим объект модели Identifier, присваивая значения полям объекта следующим образом:

```
object_type=identifier['type'],
object_id_in_contest=identifier['id_in_contest'],
object_name=identifier['name']
```

Теперь в системе «Форум МГУ» мы имеем список объектов из «МГУ Контест» и можем приступить к распределению сообщений, то есть указанию их принадлежности к какому-либо объекту системы «МГУ Контест». Для определения принадлежности сообщения к какому-либо объекту, добавим к модели Message поле `identifier`, содержащее внешний ключ, указывающий на объект модели Identifier.

3.6 Распределение сообщений по курсам, разделам и задачам

Распределение сообщений осуществляется частично с использованием SQL-команд для задания идентификаторов сразу многих сообщений, частично вручную через инструмент администрирования, предоставляемый Django, так как для корректного определения отношения сообщения к какому-либо идентификатору, необходимо просмотреть текст данного сообщения.

Для просмотра объектов сообщений модели Messages в удобном для распределения виде по URL-адресу `<host>:<port>/new_messages` реализуем function-based view `new_messages`. В теле функции `new_messages` объявим переменную `html` типа `str`, изначально пустую. Для каждой доски из `NECESSARY_BOARDS` добавим к переменной `html` уникальный идентификатор и имя рассматриваемой доски, заключенные в необходимые HTML-теги форматирования. Переменной `topics` присвоим множество всех тем из `NECESSARY_TOPICS`, относящихся к рассматриваемой доске. Для каждой темы из `topics` к переменной `html` добавим уникальный идентификатор и название (определяемое, как заголовок первого сообщения в теме) рассматриваемой темы, заключенные в необходимые HTML-теги форматирования. Переменной `messages` присвоим список всех сообщений (объектов модели Message), относящихся к рассматриваемой теме

и отсортируем полученный список `messages` по возрастанию времени написания сообщения. Для каждого сообщения из `messages` к переменной `html` добавим уникальный идентификатор рассматриваемого сообщения, уникальный идентификатор первого сообщения в цепочке, в которой находится рассматриваемое сообщение, имя автора, заголовок и непосредственно текст рассматриваемого сообщения, заключенные в необходимые HTML-теги форматирования. Возвращаемым значением функции `new_messages` будет предоставляемый Django объект HttpResponse, в конструктор которого будет передано содержимое веб-страницы в виде строки `html`.

По завершении просмотра текста сообщений определяются темы, для которых известно отношение содержащихся в них сообщений к определенным объектам системы «МГУ Контест». Для распределения таких сообщений, применяются SQL-команды следующего вида:

```
update <база данных системы «Форум МГУ»>.<таблица модели Message>
    set identifier=<ID идентификатора> where topic=<ID темы>;
```

Гарантируется, что сообщения, находящиеся в одной цепочке обязательно относятся к одному и тому же объекту в системе «МГУ Контест». Следовательно, для ускорения ручного распределения сообщений разумным решением является указание идентификаторов вручную только для первых сообщений в цепочках, а для всех остальных сообщений в цепочке копирование значения идентификатора первого сообщения в цепочке, в которой находится рассматриваемое сообщение, осуществляемое автоматически. Множество первых сообщений в цепочках получим следующим образом:

```
set(message for message in Message.objects.all()
    if message.id == message.parent_msg_id)
```

и присвоим константе `PARENT_MESSAGES`.

По завершении распределения первых сообщений в цепочках вручную с использованием инструмента администрирования, предоставляемого Django, реализуем функцию `get_replies_identifiers_sql` для генерации SQL-команд, позволяющих скопировать значения идентификаторов первых сообщений в цепочках для всех сообщений, которые не являются первыми в цепочке. В теле функции введем переменную `replies` и присвоим ей множество сообщений, полученное вычитанием из множества всех объектов модели Message множества `PARENT_MESSAGES`. Для каждого сообщения `message` из множества `replies` в файл define_replies_identifiers.sql запишем SQL-команду следующего вида:

```
update <база данных системы «Форум МГУ»>.<таблица модели Message>
    set identifier=<ID идентификатора> where id=<message.id>;
```

где ID идентификатора получается таким образом:

```
Message.objects.get(id=message.parent_msg_id).identifier.id
```

После применения сгенерированных SQL-команд для всех объектов модели Message будет заполнено значение поля `identifier`. Значит, сообщения готовы к переносу.

3.7 Экспорт сообщений

Для экспорта сообщений из базы данных системы «Форум МГУ» и сохранения их в JSON-файл вне базы данных реализуем функцию `get_comments_json`. Введем переменную `comments` типа `list`. Для каждого объекта `message` из всех объектов типа Message добавим в список `comments` словарь следующего вида:

```
{
    'old_id': message.id,
    'author': message.author.id,
    'parent_id': message.parent_msg_id,
    'object_type': message.identifier.object_type,
    'object_id': message.identifier.object_id_in_contest,
    'text': message.text,
    'date_created': message.date_created
}
```

Сериализуем полученный список `comments` с помощью функции `dumps` модуля json и сохраним в файл comments.json. Теперь в файле comments.json мы имеем необходимый для переноса сериализованный список сообщений.

4 Перенос сообщений

4.1 Импорт комментариев

На данном этапе работы импорт комментариев осуществляется в локальную версию веб-сайта «МГУ Контест».

В системе «МГУ Контест» в модели Comment добавим поле `old_id`, которое будет содержать положительное целое число – уникальный идентификатор комментария в системе «Форум МГУ». Наличие данного поля необходимо, чтобы иметь возможность сохранить цепочки комментариев.

Для импорта комментариев из JSON-файла и создания их как объектов модели Comment в системе «МГУ Контест» реализуем функцию `create_comments`. В теле функции десериализуем данные из файла comments.json, полученного на предыдущем этапе работы с помощью функции `loads` модуля json в объект-список `forum_comments`. Введем переменную `comments` типа `list`, изначально пустую. Для каждого объекта `forum_comment` из списка `forum_comments` получим тип объекта следующим образом:

```
if forum_comment['object_type'] == 1:
    object_type = ContentType.objects.get(model='Course')
elif forum_comment['object_type'] == 2:
    object_type = ContentType.objects.get(model='Contest')
else:
    object_type = ContentType.objects.get(model='Problem')
```

и создадим объект модели Comment, присваивая значения полям создаваемого комментария следующим образом:

```
Comment(
    old_id=forum_comment['old_id'],
    author=
        Account.objects.get(old_id=forum_comment['author']).user,
    order=forum_comment['date_created'],
    parent_id=forum_comment['parent_id'],
    object_type=object_type,
    object_id=forum_comment['object_id'],
    text=forum_comment['text'],
    date_created=make_aware(datetime.fromtimestamp(
        forum_comment['date_created']))
)
```

Созданный комментарий добавим в список `comments`. Для получения даты создания комментария из формата представления времени POSIX здесь используется метод `fromtimestamp` объекта datetime модуля datetime языка Python, а также функция `make_aware` из модуля timezone [14], предоставляемого среди django.utils для добавления часового пояса к дате создания комментария. По завершении итераций по объектам из `forum_comments` вставим объекты из списка `comments` в базу данных, используя стандартный метод Django QuerySet API `bulk_create`.

В системе «МГУ Контест» модель Comment содержит кроме прочих следующие поля:

- thread_id – целое положительное число – уникальный идентификатор первого комментария в цепочке;
- parent_id - целое положительное число – уникальный идентификатор комментария, ответом на который является рассматриваемый комментарий;
- order – целое положительное число – порядковый номер комментария в цепочке;
- level – целое положительное число – уровень вложенности комментария.

Для определения значений перечисленных полей объектов модели Comment, добавленных ранее, реализуем функцию `update_comments`. Введем переменную `comments` и присвоим ей список всех объектов модели Comment. Для каждого объекта `comment` из списка `comments` присвоим полям `thread_id` и `parent_id` значение поля `id` комментария со значением поля `old_id`, равным значению поля `parent_id` рассматриваемого комментария. Если рассматриваемый комментарий не является первым в цепочке, то полю `level` рассматриваемого комментария присвоим значение 2. Полям `order` комментариев значения были присвоены автоматически при добавлении и соответствуют порядковому номеру комментария при добавлении. Чтобы обновить значение данного поля для каждого комментария, не являющегося первым в цепочке, используем следующую конструкцию:

```
comment.order = \
    comment.order \
    - Comment.objects.get(id=comment.parent_id).order + 1
```

Затем присвоим полям `order` всех комментариев, являющихся первыми в цепочке, значение 1. По завершении итераций по комментариям обновим поля `thread_id`, `parent_id`, `order` и `level` во всех экземплярах модели Comment при помощи стандартного метода Django QuerySet API `bulk_update`.

4.2 Редактирование комментариев

После импорта комментариев в систему «МГУ Контест» возникают задачи: просмотреть все добавленные комментарии и объединить связанные по смыслу комментарии в цепочки; если необходимо, изменить уровень вложенности комментариев на более глубокий, чем второй уровень; добавить возможность переходить по ссылкам, содержащимся в тексте комментариев, по нажатию. Перечисленные задачи решаются вручную путем редактирования комментариев с использованием инструмента администрирования, предоставляемого Django.

4.3 Экспорт аккаунтов и комментариев

Для добавления комментариев после их редактирования в стабильную версию веб-сайта «МГУ Контест» необходимо сначала экспортировать комментарии и аккаунты авторов данных комментариев из локальной версии веб-сайта и сохранить их в JSON-файлы вне базы данных.

Для экспорта аккаунтов и сохранения их в файле формата JSON реализуем функцию `get_accounts_json`, которая для каждого объекта модели Account добавляет в список `accounts` словарь, в котором ключами являются названия полей модели Account, а значениями – значения этих полей рассматриваемого объекта модели Account, затем сериализует полученный список `accounts` с помощью функции `dumps` модуля json и сохраняет в файл accounts.json.

Для экспорта комментариев и сохранения их в файле формата JSON реализуем функцию `get_comments_json`, которая для каждого объекта модели Comment добавляет в список `comments` словарь, в котором ключами являются названия полей модели Comment, а значениями – значения этих полей рассматриваемого объекта модели Comment, затем список `comments` сортирует по возрастанию даты написания комментариев, сериализует полученный список `comments` с помощью функции `dumps` модуля json и сохраняет в файл comments.json.

4.4 Импорт аккаунтов и комментариев с использованием миграций

Импорт комментариев и аккаунтов авторов комментариев в стабильную версию веб-сайта «МГУ Контест» осуществляется с использованием миграций дан-

ных [15]. Миграции представляют собой предоставляемый Django способ внесения изменений моделей в схему базы данных, а также и изменения данных в базе. Миграции, изменяющие данные, называются миграциями данных. Основной операцией, используемой в миграциях данных является операция RunPython, которая запускает пользовательский Python-код – функцию, которая передана в качестве первого аргумента `code`. Вторым аргументом `reverse_code` в RunPython можно передать функцию, которая обращает все действия, произведенные функцией, переданной в качестве первого аргумента, другими словами отменяет миграцию.

Реализуем миграцию данных `accounts_import` для импорта аккаунтов авторов комментариев, где в операцию RunPython передаются два аргумента: функция `import_accounts` для импорта аккаунтов из файла формата JSON и функция `delete_users` для обращения данной миграции. В функции `import_accounts` производится десериализация данных из файла accounts.json, полученного ранее, с помощью функции `loads` модуля json в объект-список `accounts`. Для каждого объекта `account` из списка `accounts` создается новый объект `user` модели User со значениями полей, получаемыми из рассматриваемого объекта `account`; затем создается объект модели Account, полю `user` которого присваивается полученный только что объект `user`, а значения других полей получаются из рассматриваемого объекта `account`; созданный объект модели Account добавляется в список `new_accounts`. По завершении итераций по списку `accounts` объекты из списка `new_accounts` вставляются в базу данных при помощи стандартного метода Django QuerySet API `bulk_create`. Функция `delete_users` удаляет всех созданных предыдущей функцией пользователей, аккаунты же, связанные с этими пользователями удаляются автоматически.

Реализуем миграцию данных `comments_import` для импорта комментариев. В данной миграции в операцию RunPython передаются два аргумента: функция `import_comments` для импорта комментариев из файла формата JSON и функция `delete_comments`, позволяющая обратить данную миграцию данных. В функции `import_comments` в случае, если в базе данных уже существуют комментарии, эти комментарии сохраняются как объекты типа `dict` в список `existing_comments` и временно удаляются из базы данных. Данное действие необходимо для того, чтобы более новые комментарии отображались выше старых, добавляемых из файла comments.json. Затем производится десериализация данных из файла comments.json, полученного ранее, с помощью функции `loads` модуля json в объект-список `comments`. Для каждого объекта `comment` из списка `comments` создается новый объект модели Comment со значениями полей, получаемыми из рассматриваемого объекта `comment`; созданный объект модели Comment добавляется в список `new_comments`. По завершении итераций по списку `comments` объекты из списка `new_comments` вставляются в базу данных

при помощи стандартного метода Django QuerySet API `bulk_create`. В случае, если список `existing_comments` не пуст, комментарии из этого списка так же добавляются в базу данных. Затем производится обновление полей `thread_id` и `parent_id` для всех комментариев для восстановления цепочек комментариев. Функция `delete_comments` удаляет все импортированные предыдущей функцией из файла comments.json комментарии.

Таким образом комментарии могут быть добавлены в базу данных стабильной версии веб-сайта «МГУ Контест».

5 Заключение

В рамках настоящей курсовой работы было осуществлено изменение структуры базы сообщений системы «Форум МГУ», сообщения были приведены к требуемому для переноса виду, произведен экспорт сообщений, а также реализованы все необходимые процедуры для импорта базы сообщений в систему «МГУ Контест». Таким образом реализована автоматическая система переноса базы сообщений из системы «Форум МГУ» в систему «МГУ Контест», используемую для поддержки курса «Практикум на ЭВМ» в филиале МГУ имени М.В. Ломоносова в городе Ташкенте.

6 Приложения

- Инструменты для работы с проектом «Форум МГУ»
  https://github.com/eugeuie/forum-tools
- Инструменты для работы с проектом «МГУ Контест»
  https://github.com/eugeuie/contest-tools

7 Список использованных источников и литературы

[1] Моделирование процесса обучения / В. Б. Кудрявцев, П. А. Алисейчик, К. Вашик и др. // Интеллектуальные системы. Теория и приложения (ранее: Интеллектуальные системы по 2014, № 2, ISSN 2075-9460). – 2006. – Т. 10. – С. 189-270.

[2] Репозиторий проекта «МГУ Контест»
https://github.com/ruslanbektashev/contest

[3] Документация фреймворка Django
https://docs.djangoproject.com/en/3.0

[4] Документация языка Python
https://docs.python.org/3

[5] Документация СУБД MySQL
https://dev.mysql.com/doc/

[6] Страница схемы MVC на сайте «Википедия»
https://ru.wikipedia.org/wiki/Model-View-Controller

[7] О написании представлений Django
https://docs.djangoproject.com/en/3.0/topics/http/views/

[8] Страница языка разметки BBCode на сайте «Википедия»
https://ru.wikipedia.org/wiki/BBCode

[9] Документация модуля re языка Python
https://docs.python.org/3/library/re.html

[10] Справочкик по Django QuerySet API
https://docs.djangoproject.com/en/3.0/ref/models/querysets/

[11] Об объекте Django HttpResponse
https://docs.djangoproject.com/en/3.0/ref/request-response/#httpresponse-objects

[12] Сайт формата обмена данными JSON
https://www.json.org/json-en.html

[13] Документация модуля datetime языка Python
https://docs.python.org/3/library/datetime.html

[14] О часовых поясах в Django
https://docs.djangoproject.com/en/3.0/topics/i18n/timezones/

[15] О миграциях в Django
https://docs.djangoproject.com/en/3.0/topics/migrations/