ТЕХНИЧЕСКИЕ ТРЕБОВАНИЯ

Название проекта: Portal

База данных: postgresql 10.\* и выше.

СТРУКТУРА САЙТА

* Управление пользователями
* Курсы
* Тесты
* Статистика

# УПРАВЛЕНИЕ ПОЛЬЗОВАТЕЛЯМИ

## Авторизация

Должна проходить по полям username и password. Авторизованный пользователь получает токен, который используется в дальнейших запросах.

## Аутентификация

Должна проходить по токену с ограниченным сроком жизни (3 часа).

КУРСЫ

Внутри данного раздела присутствуют папки, внутри которых находятся документы.

Папка имеет следующие поля:

* название,
* превью,
* видимость (да/нет).

Документ имеет следующие поля:

* папка, в которой он находится,
* название,
* описание,
* превью,
* дата создания,
* видимость (да/нет).

Кроме того, к документу крепятся блоки с контентом. Каждый блок имеет следующие поля:

* документ, которому он принадлежит,
* тип (один из вариантов: html, изображение, файл),
* текст,
* порядок следования,
* файл.

Необходимо фиксировать просмотры и лайки документов каждым пользователем. Отмена лайка не предусмотрена.

# ТЕСТЫ

Внутри данного раздела присутствуют тесты с вопросами.

Тест имеет поля:

* название
* превью
* минимальный результат для прохождения
* время в минутах, которое дается одну на попытку
* видимость

Вопрос имеет поля:

* тип, который может быть одним из следующих:
  + выбор одного варианта,
  + выбор нескольких вариантов,
  + сортировка
* текст
* порядок следования

Результатом прохождения теста является процент верных ответов.

# СТАТИСТИКА

Отображение данного раздела зависит от роли пользователя.

Пользователь с ролью «Сотрудник» видит список всех тестов. При каждом тесте присутствует информация о результате и успешности сдачи.

Пользователь с ролью «Админ» имеет возможность получить статистику в двух срезах:

* выбрав пользователя с ролью «Сотрудник» и получив его результаты;
* выбрав тест и получив результаты всех пользователей с ролью «Сотрудник» по нему.

# Задачи

29.07.2020

1. Создать пустой проект. В settings.py настроить доступ к БД, а также к файлам контента и статики. (<https://docs.djangoproject.com/en/2.2/ref/settings/>, параметры STATUC\_ROOT, MEDIA\_ROOT)
2. Создать суперпользователя, чтобы можно было работать с сайтом Django Admin.
3. Создать приложение courses. В models.py определить модели папок и документов, а в admin.py оформить доступ к ним через Django Admin. (<https://docs.djangoproject.com/en/2.2/ref/models/fields/>, поля моделей)

31.07.2020

1. Удалить модель UIDModel, а модель Element наследовать от models.Model.
2. Сделать поля моделей по ТЗ.
3. В admin.py настроить поиск по таблицам.
4. Исключить из индекса папки migrations, .idea и файл db.sqlite3 и удалить их из репозитория. db.sqlite3 удалить вообще.

20.08.2020

В приложение courses добавить следующие запросы:

1. folder/get/

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| folder\_id | int or None | id папки или None для корня |
| **Выход** | | |
| folders | list of dicts | массив папок с полями id, title, preview |
| documents | list of dicts | массив документов с полями id, title, description, preview, created (timestamp) |

1. document/get/

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| doc\_id | int | id документа |
| **Выход** | | |
| title | str | название |
| content | list of dicts | массив элементов с полями text, type, file в порядке возрастания поля order |

21.08.2020

1. Создать приложение authentication, в котором определить модель CustomUser, унаследованную от django.contrib.auth.models.User. В модель CustomUser добавить два поля:
   1. token типа models.UUIDField(default=uuid.uuid1),
   2. date\_token\_renewed типа models.DateTimeField(default=timezone.now).
2. В файле settings.py указать
   1. модель CustomUser в качестве модели авторизации,
   2. параметр TOKEN\_EXPIRE = 3, обозначающий количество часов валидности токена.
3. В приложение authentication добавить запрос signin/ для авторизации.

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| username | str | username пользователя |
| password | str | пароль пользователя |
| **Выход** | | |
| token | str | временный токен пользователя |

Этот запрос находит пользователя по username, проверяет активен ли он и если да, то создает новый токен, сохраняет его в поле user.token и дату его создания. Если пользователь с таким username не существует или заблокирован, то возвращать пустое тело и статус 401.

25.08.2020

В приложении authentication создать модуль decorators.py, а в нем определить декоратор auth\_and\_parse, который будет навешиваться на представления в views, требующие аутентификации.

Декоратор выполняет следующие функции:

* Аутентификация пользователя по токену. Токен будет приходить в заголовке Authorization в формате «Token токен», например «Token 693f8d98-e69f-11ea-a955-00505603340c». По присланному токену нужно найти пользователя в БД и проверить возраст этого токена. Если токен устарел или пользователь заблокирован, то вернуть пустое тело со статусом 401.
* Распределение входящих параметров в request. Входящие параметры должны быть распределены по двум словарям:
  + system – словарь, в который попадают query параметры, а также объект пользователя под ключом «\_\_user»;
  + data – словарь в который попадает содержимое тела запроса.
* Декорируемая функция должна принимать system и data, а возвращать словарь с ответом и статус в виде числа.

В итоге после применения декоратора представления будут иметь вид, аналогичный рис. 1.

В приложении courses создать модели:

* ViewDocument с полями user, doc, date,
* LikeDocument с полями user, doc.

В модель Document добавить поле views для счетчика просмотров.

Когда пользователь получает документ, должна создаваться или обновляться соответствующая запись в ViewDocument, а также должен увеличиваться на единицу счетчик views самого документа.

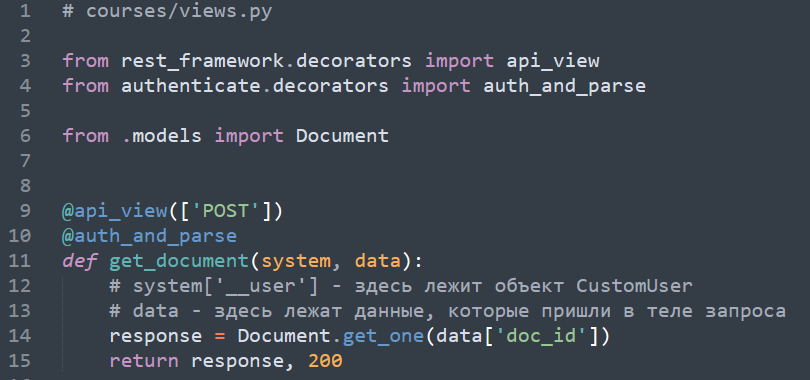
Добавить запрос courses/document/like/ на фиксацию лайка документа, который будет создавать запись в LikeDocument, если таковой нет.

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| doc\_id | str | id документа |
| **Выход** | | |
| status | str |  |

В запрос courses/folder/get/ в списке документов каждый документ должен иметь поля:

* флаг is\_liked (bool), обозначающий, что пользователь уже ставил лайк этому документу,
* likes (int) – общее количество лайков,
* views (int) – обще число просмотров.

Рис.



26.08.2020

1. Создать приложение tests.
2. В tests.models.py создать следующие модели:
   1. Test c полями:
      1. title: str
      2. preview: image
      3. is\_visible: bool
      4. min\_result: decimal
      5. time: int
   2. Question с полями:
      1. type: str с вариантами: radio, checkbox, sort
      2. text: str
      3. order: int
   3. Option с полями:
      1. question: FK to Question
      2. text: str
      3. is\_right: bool
      4. order: int
3. В tests.admin.py оформить управление созданными моделями.

27.08.2020

В модуль tests.models.py добавить следующие модели:

* Модель попытки Attempt с полями:
  + user: FK to CustomUser
  + start: datetime
  + finish: datetime
  + result: decimal
  + is\_over: bool
* Модель ответа Answer с полями:
  + attempt: FK to Attempt
  + is\_right: bool
  + options: MtoM to Option throw Choice
* Choice: промежуточная модель между Answer и Option
  + answer: FK to Answer
  + option: FK to Option

В приложение tests добавить следующее API:

1. Получение списка тестов test/all/

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
|  |  |  |
| **Выход** | | |
| tests | list of dicts | список словарей {‘id’: int, ‘title’: str, ‘preview’: str, ‘available\_attempts’: число доступных попыток для прохождения (0 или 1)} |

1. Получение теста test/get/

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| test\_id | int |  |
| **Выход** | | |
| num\_questions | int | количество вопросов |
| min\_result | decimal | минимальный результат для прохождения |
| title | str | название |
| time | int | время в минутах на одну попытку |
| id | int | id теста |

1. Старт попытки attempt/start/. В представлении должна проходить проверка числа доступных попыток. Если доступных попыток нет, то возвращать {}, 400. После этого нужно создать объект попытки со статусом «не завершена» и собрать вопросы теста.

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| test\_id | int |  |
| **Выход** | | |
| questions | list of dict | список словарей {‘id’: int, ‘text’: str, ‘type’: str, ‘options’: список словарей {‘id’: int, ‘text’: str}} |
| attempt\_id | int | id созданной попытки |

1. Завершение попытки attempt/finish/. Представление должно искать попытку и проверять не устарела ли она. Если устарела, то возвращать {}, 400. Затем сохранять присланные ответы, вычислять результат и закрывать попытку.

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| attempt\_id | int |  |
| answers | dict | ключ: id вопроса  значение: список id вариантов |
| **Выход** | | |
| result | decimal | результат |
| status | bool | пройден или не пройден |

01.09.2020

1. В модель authentication.models.CustomUser добавить поле role со значениями:
   1. admin – Админ
   2. user – Сотрудник
2. Создать приложение statistics.
3. Реализовать запрос statistics/results/

|  |  |  |
| --- | --- | --- |
| **Поле** | **Тип** | **Описание** |
| **Вход** | | |
| user\_id | int or None |  |
| test\_id | int or None |  |
| **Выход** | | |
| results | list | список словарей вида:  {‘user’: str,  ‘test’: str,  ‘result’: decimal,  ‘is\_success’: bool} |