**Тема занятия № 41: Понятие Токен. JSON Web Token Authentication**

1. Токены. Авторизация и аутентификация. Базовая авторизация

Аутентификация Django обеспечивает как аутентификацию, так и авторизацию вместе и обычно называется системой аутентификации, поскольку эти функции в некоторой степени связаны.

**User объекты ¶**

User объекты являются ядром системы аутентификации. Обычно они представляют людей, взаимодействующих с вашим сайтом, и используются для включения таких вещей, как ограничение доступа, регистрация профилей пользователей, связывание контента с создателями и т. д. В структуре аутентификации Django существует только один класс пользователей, т. е. пользователи- 'superusers' администраторы 'staff'— это просто пользовательские объекты с установлены специальные атрибуты, а не разные классы пользовательских объектов.

Основные атрибуты пользователя по умолчанию:

username

password

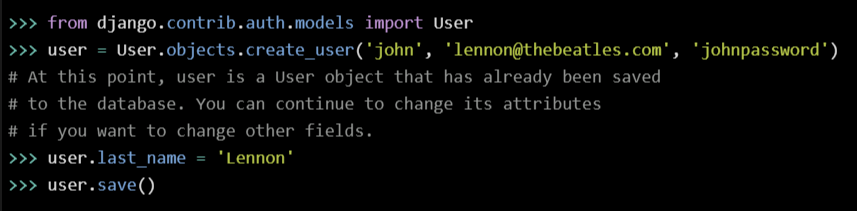
email

first\_name

last\_name

**Создание пользователей**

Самый прямой способ создать пользователей — использовать create\_user()встроенную вспомогательную функцию:

****

Если у вас установлен администратор Django, вы также можете создавать пользователей в интерактивном режиме.

Создайте суперпользователей с помощью createsuperuser команды:



Вам будет предложено ввести пароль. После того, как вы введете его, пользователь будет создан немедленно. Если вы пропустите опции --usernameили --email, вам будет предложено ввести эти значения.

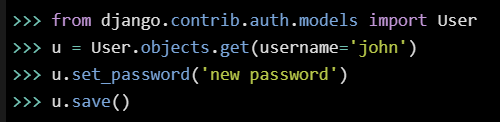
**Смена паролей**

Django не хранит необработанные (открытым текстом) пароли в пользовательской модели, а только хэш. По этой причине не пытайтесь напрямую манипулировать атрибутом пароля пользователя. Вот почему при создании пользователя используется вспомогательная функция.

Чтобы изменить пароль пользователя, у вас есть несколько вариантов:

manage.py changepassword \*username\*предлагает метод изменения пароля пользователя из командной строки. Он предлагает вам изменить пароль данного пользователя, который вы должны ввести дважды. Если они оба совпадают, новый пароль будет немедленно изменен. Если вы не укажете пользователя, команда попытается изменить пароль, имя пользователя которого совпадает с текущим системным пользователем.

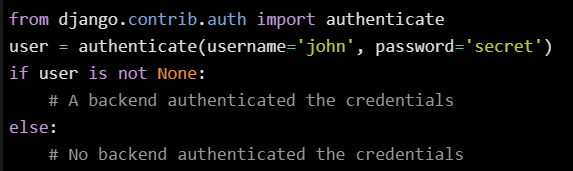
Вы также можете изменить пароль программно, используя set\_password():

****

**Аутентификация пользователей**

authenticate( request = None , \*\* учетные данные ) ¶

Используйте authenticate()для проверки набора учетных данных. Он принимает учетные данные в качестве аргументов ключевого слова usernameи passwordв случае по умолчанию проверяет их на соответствие каждому серверу аутентификации и возвращает Userобъект, если учетные данные действительны для сервера. Если учетные данные недействительны для какого-либо бэкэнда или если бэкенд вызывает PermissionDenied, он возвращает None. Например:

****

Request является необязательным HttpRequest, который передается в authenticate()методе серверной части аутентификации.

Примечание!

Это низкоуровневый способ аутентификации набора учетных данных; например, он используется RemoteUserMiddleware. Если вы не пишете свою собственную систему аутентификации, вы, вероятно, не будете ее использовать. Скорее, если вы ищете способ входа пользователя в систему, используйте расширение LoginView.

**Разрешения и авторизация**

Django поставляется со встроенной системой разрешений. Он позволяет назначать разрешения определенным пользователям и группам пользователей.

Он используется административным сайтом Django, но вы можете использовать его в своем собственном коде.

**Сайт администратора Django использует разрешения следующим образом:**

Доступ к объектам просмотра ограничен пользователями с разрешением «просмотр» или «изменение» для этого типа объекта.

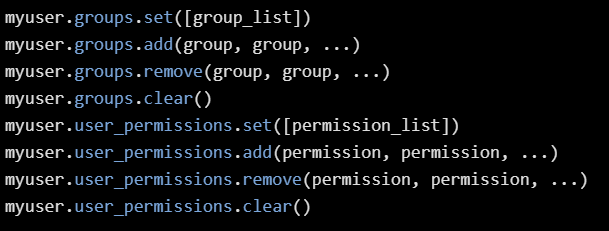
Доступ к просмотру формы «добавить» и добавлению объекта ограничен пользователями с правом «добавлять» для этого типа объекта.

Доступ к просмотру списка изменений, просмотру формы «изменения» и изменению объекта ограничен пользователями с правом «изменения» для этого типа объекта.

Доступ к удалению объекта ограничен пользователями с правом «удалить» для этого типа объекта.

Разрешения могут быть установлены не только для типа объекта, но и для конкретного экземпляра объекта. С помощью методов has\_view\_permission(), и has\_add\_permission(), предоставляемых классом, можно настроить разрешения для разных экземпляров объектов одного и того же типа.has\_change\_permission()has\_delete\_permission()ModelAdmin

User объекты имеют два поля «многие ко многим»: groupsи user\_permissions. Userобъекты могут обращаться к своим связанным объектам так же, как и любая другая модель Django:

****

**Разрешения по умолчанию**

Когда django.contrib.authэто указано в ваших INSTALLED\_APPS настройках, это гарантирует, что четыре разрешения по умолчанию — добавление, изменение, удаление и просмотр — будут созданы для каждой модели Django, определенной в одном из ваших установленных приложений.

Эти разрешения будут созданы при запуске; при первом запуске после добавления в , разрешения по умолчанию будут созданы для всех ранее установленных моделей, а также для любых новых моделей, устанавливаемых в это время. После этого он будет создавать разрешения по умолчанию для новых моделей при каждом запуске (функция, создающая разрешения, связана с сигналом).manage.py migratemigratedjango.contrib.authINSTALLED\_APPSmanage.py migratepost\_migrate

Предполагая, что у вас есть приложение с моделью и с именем, для проверки основных разрешений вы должны использовать:app\_label fooBar

добавлять:user.has\_perm('foo.add\_bar')

сдача:user.has\_perm('foo.change\_bar')

Удалить:user.has\_perm('foo.delete\_bar')

Посмотреть:user.has\_perm('foo.view\_bar')

К Permission модели редко обращаются напрямую.

**Группы**

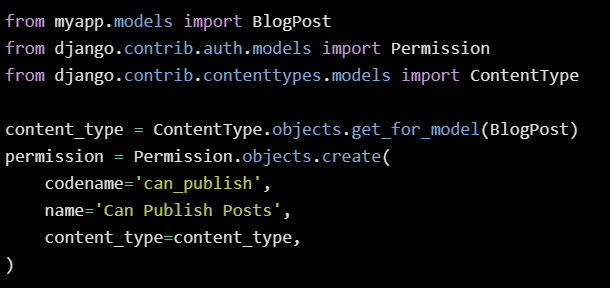
django.contrib.auth.models.Group модели — это общий способ категоризации пользователей, поэтому вы можете применять разрешения или другие метки к этим пользователям. Пользователь может принадлежать к любому количеству групп.

Пользователь в группе автоматически получает разрешения, предоставленные этой группе. Например, если группа имеет разрешение, любой пользователь в этой группе будет иметь это разрешение.Site editorscan\_edit\_home\_page

Помимо разрешений, группы — это удобный способ категоризировать пользователей, чтобы дать им какую-либо метку или расширенную функциональность. Например, вы можете создать группу и написать код, который может, скажем, предоставить им доступ к части вашего сайта, предназначенной только для членов, или отправлять им сообщения электронной почты только для членов.'Special users'

**Программное создание разрешений**

Хотя настраиваемые разрешения могут быть определены в Metaклассе модели, вы также можете создавать разрешения напрямую. Например, вы можете создать can\_publishразрешение для BlogPostмодели в myapp:

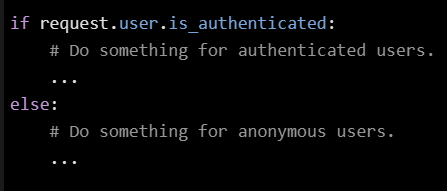
****

**Аутентификация в веб-запросах**

Django использует сеансы и промежуточное ПО для подключения системы аутентификации к .request objects

Они предоставляют request.user атрибут для каждого запроса, который представляет текущего пользователя. Если текущий пользователь не вошел в систему, этот атрибут будет установлен на экземпляр AnonymousUser, в противном случае он будет экземпляром User.

Вы можете отличить их друг от друга с помощью is\_authenticated, например:

****

**Как войти в систему?**

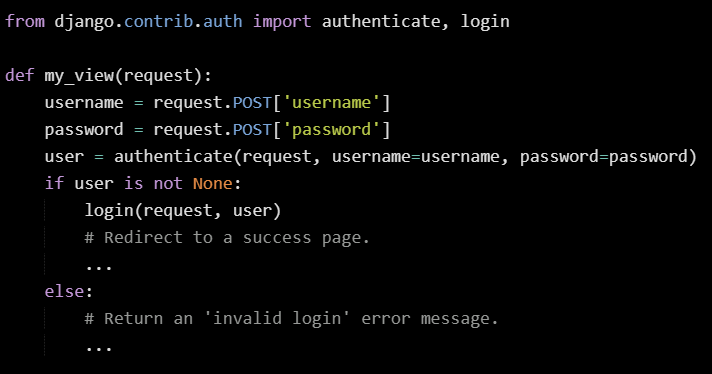
Если у вас есть аутентифицированный пользователь, которого вы хотите подключить к текущему сеансу, это делается с помощью login()функции.

**login(запрос , пользователь , серверная часть = нет ) ¶**

Чтобы войти в систему из представления, используйте login(). Он принимает HttpRequestобъект и Userобъект. login()сохраняет идентификатор пользователя в сеансе, используя структуру сеанса Django.

Обратите внимание, что любые данные, установленные во время анонимного сеанса, сохраняются в сеансе после входа пользователя в систему.

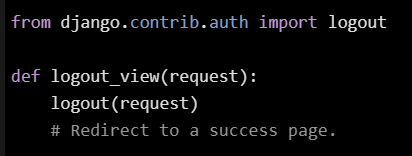
В этом примере показано, как вы можете использовать и , authenticate()и login():

****

**Как выйти из системы**

**logout(запрос)**

Чтобы выйти из системы пользователя, который вошел в систему через django.contrib.auth.login(), используйте django.contrib.auth.logout()в своем представлении. Он принимает HttpRequestобъект и не имеет возвращаемого значения. Пример:

****

Обратите внимание, что logout()не выдает никаких ошибок, если пользователь не вошел в систему.

Когда вы вызываете logout(), данные сеанса для текущего запроса полностью очищаются. Все существующие данные удаляются. Это делается для того, чтобы другой человек не мог использовать тот же веб-браузер для входа в систему и получения доступа к данным сеанса предыдущего пользователя. Если вы хотите поместить в сеанс что-либо, что будет доступно пользователю сразу после выхода из системы, сделайте это после вызова django.contrib.auth.logout().

1. Библиотека JWT – json web token.

Django поставляется с системой аутентификации, основанной на сеансах, и это работает из коробки. Это включает в себя все модели (models), представления (views) и шаблоны (templates), которые могут быть нужны вам для создания и дальнейшего логина пользователей. Но вот в чем загвоздка: стандартная система аутентификации Django работает только с традиционным 'запрос-ответ' циклом HTML.

Что мы имеем ввиду под «традиционным 'запрос-ответ' циклом HTML»? Исторически, когда пользователь хотел выполнить какое-то действие (например, создать новый аккаунт), он заполнял определенную форму в браузере. Далее, когда он кликал на кнопку «Отправить», браузер формировал запрос - который включал в себя данные, введенные пользователем - и отправлял на сервер, сервер обрабатывал запрос, и отвечал либо HTML страницей, либо редиректом на новую страницу. Это то, что мы имеем ввиду, когда говорим о «полном обновлении страницы».

Почему важно знать, что встроенная система аутентификации Django работает только с традиционным 'запрос-ответ' циклом HTML? Потому что клиент, для которого мы создадим данный API, не придерживается этого цикла. Вместо этого, клиент будет ожидать, что сервер вернет JSON, вместе обычного HTML. Возвращая JSON, мы можем позволить решать клиенту, а не серверу, что делать дальше. В цикле 'запрос-ответ' JSON, сервер получает данные, обрабатывает их и возвращает ответ (пока что как и в цикле 'запрос-ответ' HTML), но ответ не управляет поведением браузера. Ответ просто сообщает браузеру результат запроса.

К счастью, команда разработки Django поняла, что тренды веб разработки движутся именно в этом направлении. Они также знали, что некоторые проекты могут не захотеть использовать встроенные модели, представления и шаблоны. Вместо этого, они могут использовать собственные. Чтобы убедиться, что все усилия, затраченные на создание встроенной системы аутентификации Django, не потрачены зря, они решили сделать возможным использование наиболее важных частей, сохраняя при этом возможность настройки конечного результата.

Аутентификация, основанная на токенах наиболее распространенной альтернативой аутентификации на основе сессий/сеансов, является т.н. аутентификация на основе токенов. Мы будем использовать особую форму такой аутентификации для защиты нашего приложения. При аутентификации на основе токенов сервер предоставляет клиенту токен после успешного запроса на вход. Этот токен уникален для пользователя и хранится в базе данных вместе идентификатором пользователя (если точнее, возможны раные вариации генерации токена, основная же идея в том, чтобы он аутентифицировал пользователя, позволяя знать кто это и давая доступ к апи, и имел время жизни, по истечении которого "протухал"). Ожидается, что клиент отправит токен вместе с будущими запросами, чтобы сервер мог идентифицировать пользователя. Сервер делает это путем поиска в таблице базы данных, содержащей все созданные токены. Если соответствующий токен найден, то сервер продолжает проверять, действителен ли токен. Если не найден, мы говорим, что пользователь не аутентифицирован. Поскольку токены хранятся в базе данных, а не в файлах куки, аутентификация на основе токенов соответствует нашим потребностям.

**Верификация токенов**

Мы всегда имеем возможность сохранить не только идентификатор пользователя (ID) с его токеном. Мы также можем хранить такие вещи, как дата истечения срока действия токена. В данном примере нам необходимо убедиться, что срок действия токена не прошел. Если прошел - считать, что токен недействителен. В таком случае, мы удаляем его из базы данных и просим пользователя снова войти в систему.

**JSON Web Tokens**

JSON Web Token (сокр. JWT) - это открытый стандарт (RFC 7519), который определяет компактный и автономный способ безопасной передачи информации между двумя сторонами. Можно думать о JWT как о токенах аутентификации на стероидах.

Помните, что мы сказали, что будем использовать особую форму аутентификации на основе токенов? JWT это как раз то, что имелось ввиду.

**Почему JSON Web Tokes лучше обычных токенов?**

При переходе с обычных токенов на JWT мы получаем несколько преимуществ:

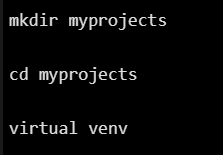
JWT - открытый стандарт. Это означает, что что все реализации JWT должны быть довольно похожими, что является преимуществом при работе с разными языками и технологиями. Обычные токены имеют более свободную форму, что позволяет разработчику решать, как лучше всего реализовывать токены.

JWT содержат информацию о пользователе, что удобно для клиентской стороны.

Библиотеки здесь берут на себя основную тяжелую работу. Развертывание собственной системы аутентификации опасно, поэтому мы оставляем важные вещи проверенным «в боях» библиотекам, которым можем доверять.

Давайте начнем.

Создайте каталог, в котором вы будете содержать свой проект, а также виртуальную среду для установки зависимостей проекта.

****

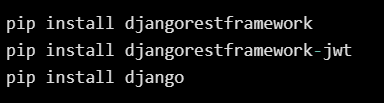
Активируйте виртуальную среду:



Создайте проект Django.



Установите DRF и django-rest-framework-jwt, используя pip.

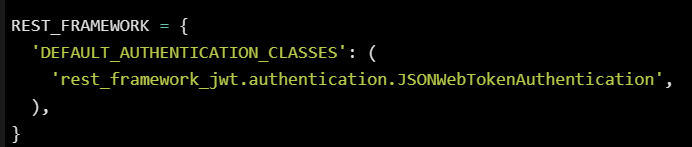
****

Давайте продолжим и добавим DRF в список установленных приложений в файле settings.py.

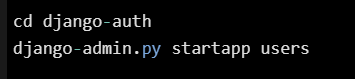
**Настройка параметров JWT**

Чтобы использовать JWT, нам нужно настроить разрешения django-rest-framework для принятия JSON Web Tokens.

В файле settings.py добавьте следующие конфигурации:

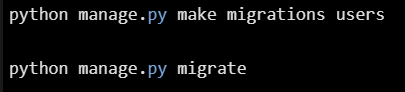
****

Создайте новое приложение users, которое будет обрабатывать аутентификацию и управление пользователями.

****

Добавьте приложение users в список установленных приложений в файле settings.py.

Создайте первоначальную миграцию для модели наших пользователей и выполните синхронизацию базы данных.

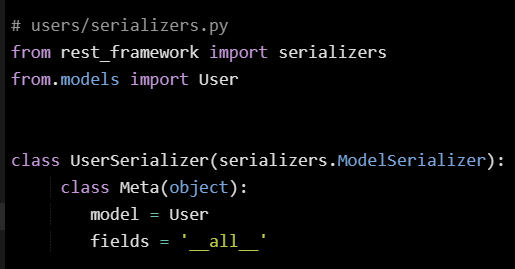
****

**Создание суперпользователя**

Создайте суперпользователя, выполнив следующую команду:



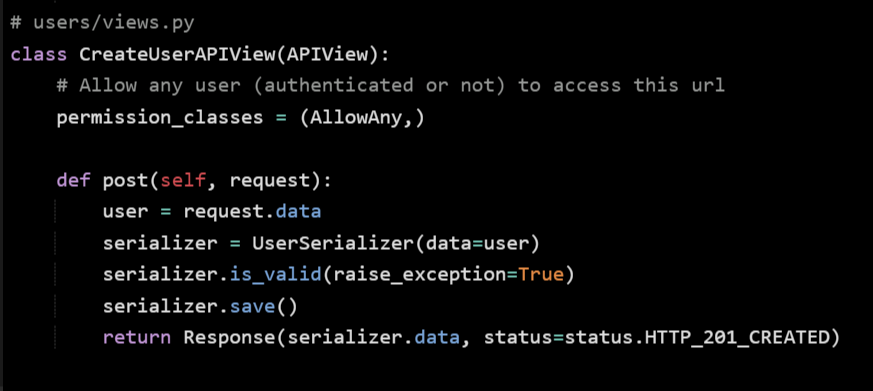
Создайте users/serializers.py и добавьте следующий код.

****

**CreateUserAPIView**

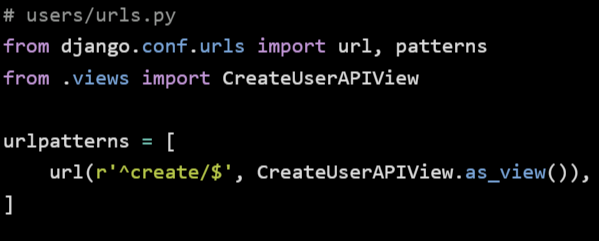
Затем мы хотим создать представление, чтобы у клиента был URL-адрес для создания новых пользователей.

В user.views.py добавьте следующее:

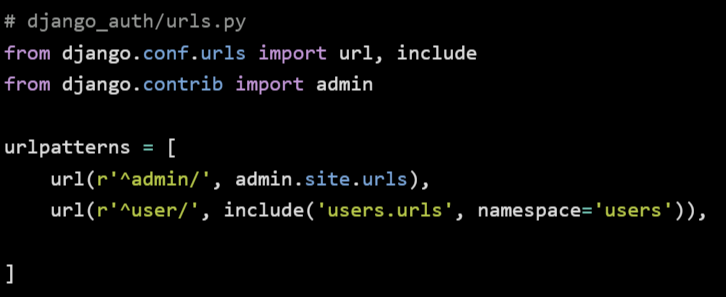
****

**Настройка URL-адресов**

Создайте файл users/urls.py и добавьте URL-адрес в соответствие с созданным нами представлением. Также добавьте следующий код.

****

Нам также нужно импортировать URL-адреса из приложения-пользователя в основной файл django\_auth/urls.py. Так что делаем это и двигаемся дальше. Мы используем здесь функцию include, поэтому не забудьте ее импортировать.

****

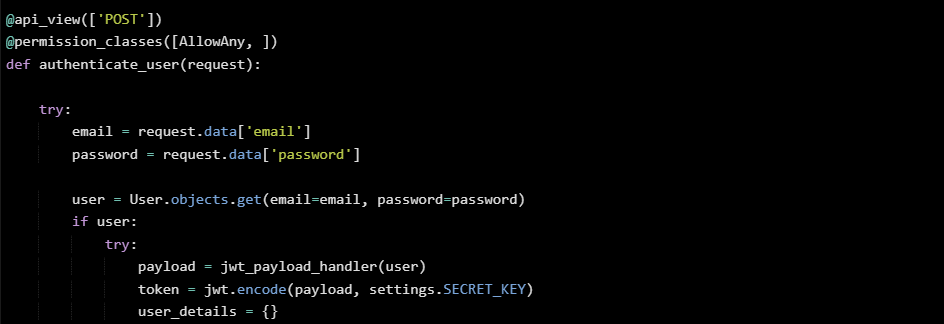
**Аутентификация пользователей**

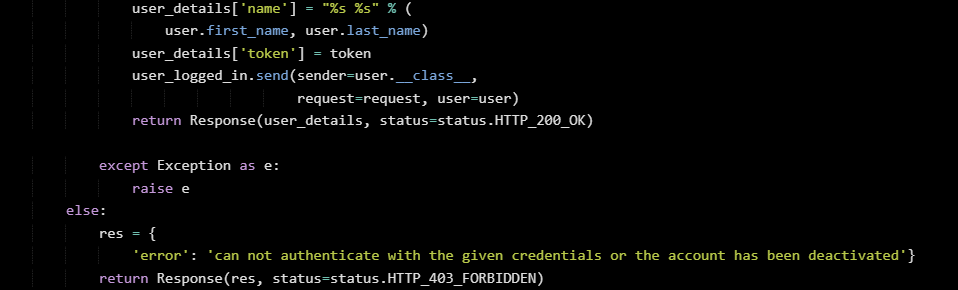
Мы будем использовать модуль Django-REST Framework JWT Python, который мы установили в начале этого руководства. Он добавляет поддержку JWT-аутентификации для приложений Django Rest Framework.

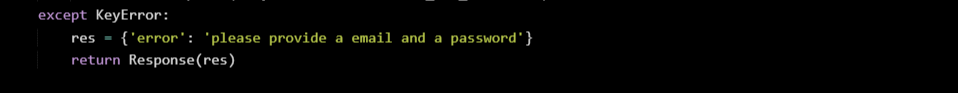
Но сначала давайте определим некоторые параметры конфигурации для наших токенов и как они сгенерированы в файле settings.py.

****

В users/views.py добавьте следующий код.



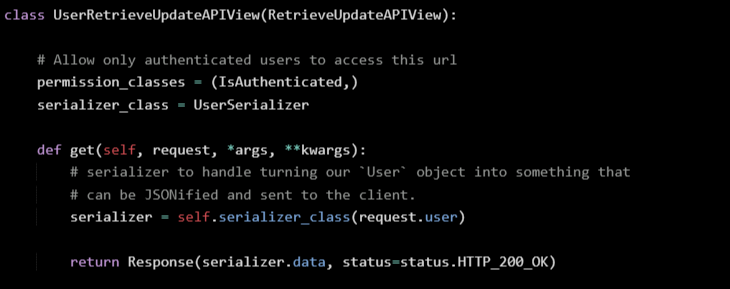


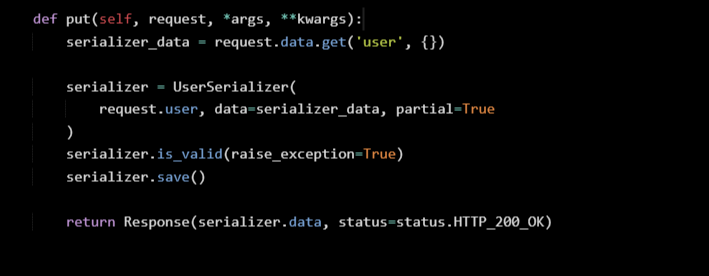


**Получение и обновление пользователей**

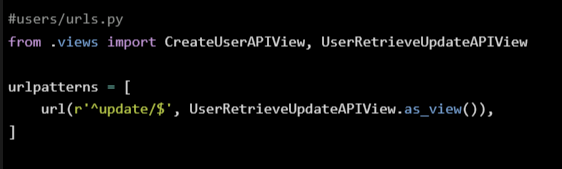
Пока пользователи могут зарегистрироваться и пройти проверку подлинности. Тем не менее, так же нужно предоставить способ получения и обновления этой информации. Давайте реализуем это.

В файле user.views.py добавьте следующий код.





Обновите URL-адреса, чтобы определить ендпоинт следующим образом.

****

Чтобы запрос был успешным, заголовки должны содержать JWT токен.

1. Библиотека JWT – json web token.

Django поставляется с системой аутентификации, основанной на сеансах, и это работает из коробки. Это включает в себя все модели (models), представления (views) и шаблоны (templates), которые могут быть нужны вам для создания и дальнейшего логина пользователей. Но вот в чем загвоздка: стандартная система аутентификации Django работает только с традиционным 'запрос-ответ' циклом HTML.

Что мы имеем ввиду под «традиционным 'запрос-ответ' циклом HTML»? Исторически, когда пользователь хотел выполнить какое-то действие (например, создать новый аккаунт), он заполнял определенную форму в браузере. Далее, когда он кликал на кнопку «Отправить», браузер формировал запрос - который включал в себя данные, введенные пользователем - и отправлял на сервер, сервер обрабатывал запрос, и отвечал либо HTML страницей, либо редиректом на новую страницу. Это то, что мы имеем ввиду, когда говорим о «полном обновлении страницы».

Почему важно знать, что встроенная система аутентификации Django работает только с традиционным 'запрос-ответ' циклом HTML? Потому что клиент, для которого мы создадим данный API, не придерживается этого цикла. Вместо этого, клиент будет ожидать, что сервер вернет JSON, вместе обычного HTML. Возвращая JSON, мы можем позволить решать клиенту, а не серверу, что делать дальше. В цикле 'запрос-ответ' JSON, сервер получает данные, обрабатывает их и возвращает ответ (пока что как и в цикле 'запрос-ответ' HTML), но ответ не управляет поведением браузера. Ответ просто сообщает браузеру результат запроса.

К счастью, команда разработки Django поняла, что тренды веб разработки движутся именно в этом направлении. Они также знали, что некоторые проекты могут не захотеть использовать встроенные модели, представления и шаблоны. Вместо этого, они могут использовать собственные. Чтобы убедиться, что все усилия, затраченные на создание встроенной системы аутентификации Django, не потрачены зря, они решили сделать возможным использование наиболее важных частей, сохраняя при этом возможность настройки конечного результата.

Аутентификация, основанная на токенах наиболее распространенной альтернативой аутентификации на основе сессий/сеансов, является т.н. аутентификация на основе токенов. Мы будем использовать особую форму такой аутентификации для защиты нашего приложения. При аутентификации на основе токенов сервер предоставляет клиенту токен после успешного запроса на вход. Этот токен уникален для пользователя и хранится в базе данных вместе идентификатором пользователя (если точнее, возможны раные вариации генерации токена, основная же идея в том, чтобы он аутентифицировал пользователя, позволяя знать кто это и давая доступ к апи, и имел время жизни, по истечении которого "протухал"). Ожидается, что клиент отправит токен вместе с будущими запросами, чтобы сервер мог идентифицировать пользователя. Сервер делает это путем поиска в таблице базы данных, содержащей все созданные токены. Если соответствующий токен найден, то сервер продолжает проверять, действителен ли токен. Если не найден, мы говорим, что пользователь не аутентифицирован. Поскольку токены хранятся в базе данных, а не в файлах куки, аутентификация на основе токенов соответствует нашим потребностям.

**Верификация токенов**

Мы всегда имеем возможность сохранить не только идентификатор пользователя (ID) с его токеном. Мы также можем хранить такие вещи, как дата истечения срока действия токена. В данном примере нам необходимо убедиться, что срок действия токена не прошел. Если прошел - считать, что токен недействителен. В таком случае, мы удаляем его из базы данных и просим пользователя снова войти в систему.

**JSON Web Tokens**

JSON Web Token (сокр. JWT) - это открытый стандарт (RFC 7519), который определяет компактный и автономный способ безопасной передачи информации между двумя сторонами. Можно думать о JWT как о токенах аутентификации на стероидах.

Помните, что мы сказали, что будем использовать особую форму аутентификации на основе токенов? JWT это как раз то, что имелось ввиду.

**Почему JSON Web Tokes лучше обычных токенов?**

При переходе с обычных токенов на JWT мы получаем несколько преимуществ:

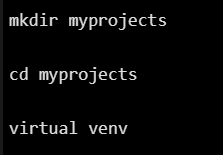
JWT - открытый стандарт. Это означает, что что все реализации JWT должны быть довольно похожими, что является преимуществом при работе с разными языками и технологиями. Обычные токены имеют более свободную форму, что позволяет разработчику решать, как лучше всего реализовывать токены.

JWT содержат информацию о пользователе, что удобно для клиентской стороны.

Библиотеки здесь берут на себя основную тяжелую работу. Развертывание собственной системы аутентификации опасно, поэтому мы оставляем важные вещи проверенным «в боях» библиотекам, которым можем доверять.

Давайте начнем.

Создайте каталог, в котором вы будете содержать свой проект, а также виртуальную среду для установки зависимостей проекта.

****

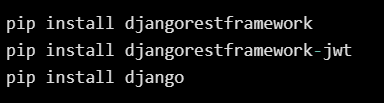
Активируйте виртуальную среду:



Создайте проект Django.



Установите DRF и django-rest-framework-jwt, используя pip.

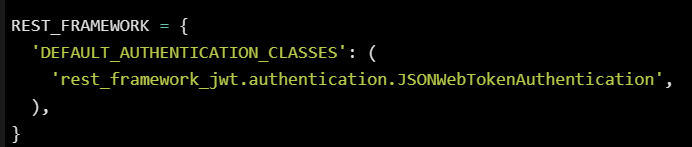
****

Давайте продолжим и добавим DRF в список установленных приложений в файле settings.py.

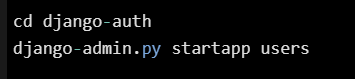
**Настройка параметров JWT**

Чтобы использовать JWT, нам нужно настроить разрешения django-rest-framework для принятия JSON Web Tokens.

В файле settings.py добавьте следующие конфигурации:

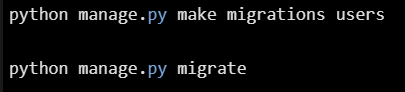
****

Создайте новое приложение users, которое будет обрабатывать аутентификацию и управление пользователями.

****

Добавьте приложение users в список установленных приложений в файле settings.py.

Создайте первоначальную миграцию для модели наших пользователей и выполните синхронизацию базы данных.

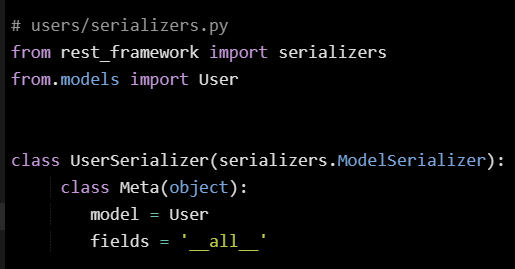
****

**Создание суперпользователя**

Создайте суперпользователя, выполнив следующую команду:



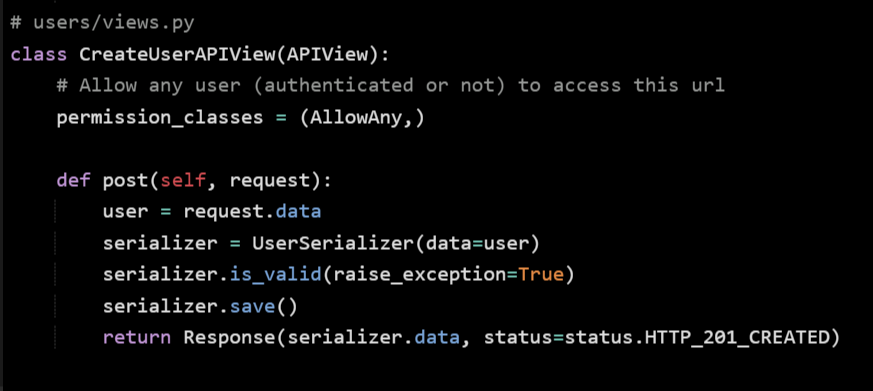
Создайте users/serializers.py и добавьте следующий код.

****

**CreateUserAPIView**

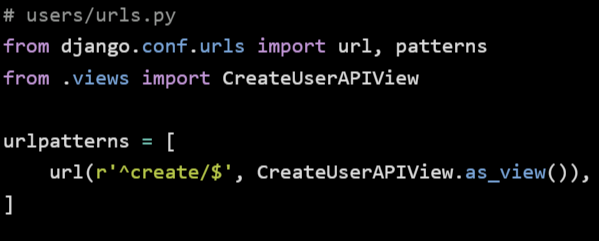
Затем мы хотим создать представление, чтобы у клиента был URL-адрес для создания новых пользователей.

В user.views.py добавьте следующее:

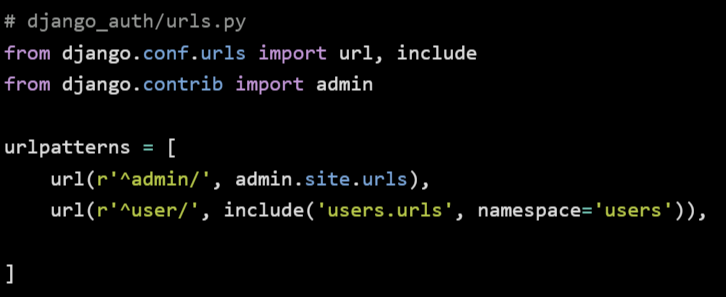
****

**Настройка URL-адресов**

Создайте файл users/urls.py и добавьте URL-адрес в соответствие с созданным нами представлением. Также добавьте следующий код.

****

Нам также нужно импортировать URL-адреса из приложения-пользователя в основной файл django\_auth/urls.py. Так что делаем это и двигаемся дальше. Мы используем здесь функцию include, поэтому не забудьте ее импортировать.

****

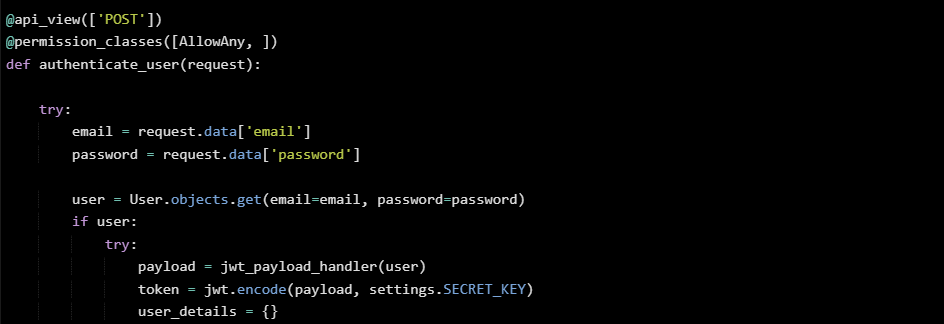
**Аутентификация пользователей**

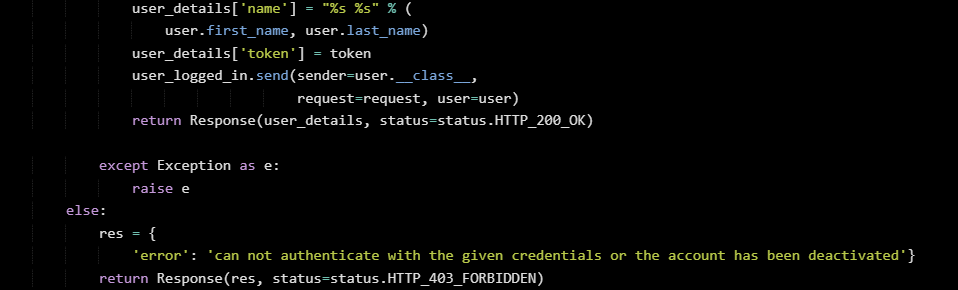
Мы будем использовать модуль Django-REST Framework JWT Python, который мы установили в начале этого руководства. Он добавляет поддержку JWT-аутентификации для приложений Django Rest Framework.

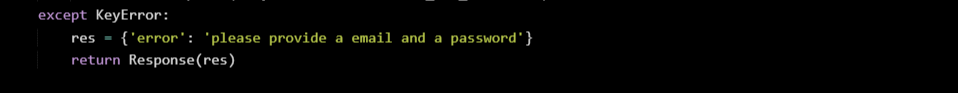
Но сначала давайте определим некоторые параметры конфигурации для наших токенов и как они сгенерированы в файле settings.py.

****

В users/views.py добавьте следующий код.



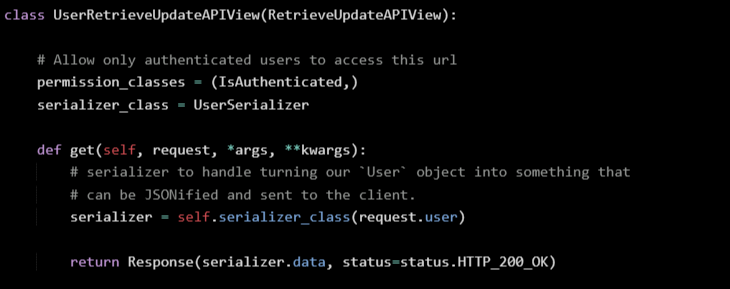


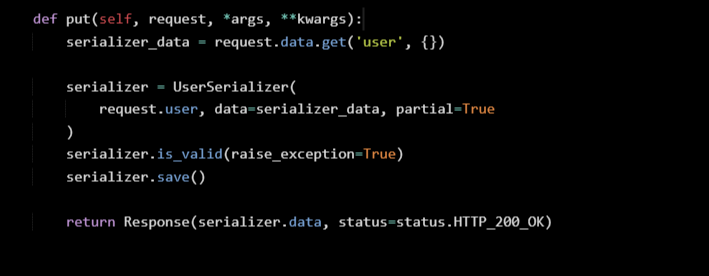


**Получение и обновление пользователей**

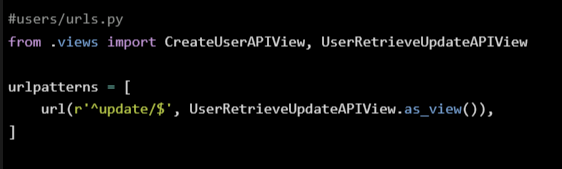
Пока пользователи могут зарегистрироваться и пройти проверку подлинности. Тем не менее, так же нужно предоставить способ получения и обновления этой информации. Давайте реализуем это.

В файле user.views.py добавьте следующий код.





Обновите URL-адреса, чтобы определить ендпоинт следующим образом.

****

Чтобы запрос был успешным, заголовки должны содержать JWT токен.