Согласно задачи, реализован API, который позволяет осуществлять доступ ко всем записям поездок выбранного водителя, либо к их части, путем указания диапазона времени, в который должна попадать поездка.

Протестировать проект можно по адресу: https://demo-taxi.herokuapp.com

> *Примечание*
>
> *Бесплатный тарифный план Heroku не предполагает постоянное нахождение контейнера с проектом в активном состоянии, поэтому первое подключение может занимать 1-2 минуты времени, пока контейнер будет вновь активирован и развернут.*

Тестовые учетные записи:

root/9907test – супер-пользователь; водитель

Driver1/9907test – пользователь; водитель

User1/9907test – пользователь

С помощью супер-пользователя можно зайти в админ-панель и создать новых пользователей/поездки.

Маршруты API (завершающий слеш обязателен!):

Маршрут	Описание

https://demo-taxi.herokuapp.com/teslooptest/rest-auth/registration/	Регистрация пользователя.
	
	Метод запроса: POST
	
	Пример параметров запроса:
	
	`{`
	`"username": "USERNAME",`
	`"password1": "PASSWORD",`
	`"password2": "PASSWORD",`
	`"email": "OPTIONAL_EMAIL"`
	`}`

https://demo-taxi.herokuapp.com/teslooptest/rest-auth/login/	Вход пользователя на сервис.
	
	Метод запроса: POST
	
	Пример параметров запроса:
	
	`{`
	`"username": "USERNAME",`

	"password": "PASSWORD" }

https://demo-taxi.herokuapp.com/teslooptest/rest-auth/logout/	Выход пользователя из сервиса.
	
	Метод запроса: POST
	
	Пример параметров запроса:
	
	{
	"username": "USERNAME",
	"password": "PASSWORD"
	}

https://demo-taxi.herokuapp.com/teslooptest/rest-auth/refresh-token/	Обновление токена перед его инвалидацией во время текущей сессии.
	
	Метод запроса: POST
	
	Пример заголовка запроса:
	
	"authorization": "JWT YOUR_TOKEN"
	
	Пример параметров запроса:
	
	{
	"token": "YOUR_OLD_TOKEN"
	}

https://demo-taxi.herokuapp.com/teslooptest/user/	Получить список всех пользователей. Метод запроса: GET Пример заголовка запроса: "authorization": "JWT YOUR_TOKEN"
https://demo-taxi.herokuapp.com/teslooptest/user/<user_pk>/	Получить запись пользователя, для которого primary_key = user_pk. Метод запроса: GET Пример заголовка запроса: "authorization": "JWT YOUR_TOKEN"

https://demo-taxi.herokuapp.com/teslooptest/user/<user_pk>/trips/	Получить список всех поездок пользователя, для которого primary_key = user_pk. Метод запроса: GET Пример заголовка запроса: `"authorization": "JWT YOUR_TOKEN"`
https://demo-taxi.herokuapp.com/teslooptest/user/<user_pk>/trips/<trip_pk>/	Получить запись поездки, для которой primary_key = trip_pk. Для Для пользователя должно выполнятся условие primary_key = = user_pk. Метод запроса: GET Пример заголовка запроса: `"authorization": "JWT YOUR_TOKEN"`
https://demo-taxi.herokuapp.com /teslooptest/user/<user_pk>/trips? start=<start_of_range>&end=<end_of_range>	Получить список поездок, для которых start timestamp <= <start_of_range> и end timestamp >= <end_of_range> Метод запроса: GET Пример заголовка запроса: `"authorization": "JWT YOUR_TOKEN"`

https://demo-taxi.herokuapp.com/admin//	Выполнить вход в админ-панель. Метод запроса: GET

Базовый функционал

- регистрация пользователей через админ-панель либо посредством web-запроса.
- вход/выход, получение токена авторизации и его обновление.
- возможность просматривать как список пользователей, так и информацию о конкретном пользователе.
- возможность просматривать список поездок для конкретного пользователя, конкретную поездку, либо все поездки время старта/окончания которых находятся в заданном интервале.

Простой пример взаимодействия

Примечание.

Для тестирования проекта использовалась программа Advanced REST client.В общем случае, при возможности отправлять произвольные заголовки подойдет любое ПО, включая браузерные расширения.

1. Новый пользователь регистрируется через админ-панель (либо сервис), для него автоматически регистрируется запись профиля (таблица Profile), в которой устанавливается поле driver = False.
2. Через админ-панель для поля driver устанавливается значение True. Таким образом, пользователь теперь является водителем.
3. Для пользователя регистрируется запись (Trip) с указанием названия (name), времени старта (start) и времени окончания (end).
4. Для доступа к сервису пользователь отправляет POST запрос по адресу https://demo-taxi.herokuapp.com/teslooptest/rest-auth/login/ , с указанием его регистрационных данных (имя пользователя и пароль). В ответе от сервера содержится токен авторизации. Для любого дальнейшего взаимодействия с сервисом необходимо отправлять этот токен в Authorization заголовке запроса. Срок годности токена – 1 час.
5. Пользователь переходит по адресу https://demo-taxi.herokuapp.com/teslooptest/user/<user_pk>/trips?start=<start>&end=<end>, где:
   <user_pk> - первичный ключ пользователя
   <start> - время старта

<end> - время окончания
В ответ сервис отправляет список поездок (записей `Trip`) в JSON формате. В случае отсутствия записей входящих в заданный временной интервал, список будет пустым.

Аутентификация и авторизация

Доступ к записям пользователей и поездок осуществляется только после входа через https://demo-taxi.herokuapp.com/teslooptest/rest-auth/login/ и только с использованием полученного токена.

Для процесса аутентификации необходимы логин и пароль зарегистрированного пользователя.

Просматривать свою запись может любой тип пользователя. Доступ к спискам поездок открыт только для пользователей со значением True для поля driver в таблице `Profile`. Все пользователи, кроме супер-пользователя, могут просматривать только свои записи (записи, для которых пользователь является владельцем). Супер-пользователь может просматривать любые записи вне зависимости от прав собственности.

Авторизация на сервисе выполняется с помощью токена - JWT (JSON Web Token), который пользователь получает после входа. В ответ на любые действия без токена, даже для аутентифицированного пользователя (выполнившего вход), сервис будет отвечать ошибкой 401:

```
{
"detail": "Error decoding signature."
}
```

При попытке получить доступ к ресурсу, который не является собственностью пользователя (например, получить данные поездок другого пользователя), кроме случая, когда пользователь – это супер-пользователь, сервис будет отвечать ошибкой 403:

```
{
"detail": "You do not have permission to perform this action."
}
```

Как упоминалось выше, срок действия токена – 1 час. До истечения этого времени пользователь может обновить токен обратившись POST запросом по адресу https://demo-taxi.herokuapp.com/teslooptest/rest-auth/refresh-token/ , с указанием в запросе своего текущего токена.

JWT токен не хранится на сервере (stateless), поэтому должен содержаться в заголовке каждого запроса клиента к сервису.

Стек технологий

Основой этого сервиса является связка Django + Django Rest Framework (DRF). Фреймворк Django предоставляет функционал для построения web-приложения, который включает ORM, необходимый набор базовых представлений, механизм маршрутизации, шаблонизатор, админ-панель и многое другое. DRF, в свою очередь, позволяет с минимальными усилиями развернуть

REST архитектуру для web-приложения. В данном проекте DRF обеспечивает методы доступа к ресурсам с помощью простого и понятного API. Он же реализует контроль разрешений и ограничений доступа к маршрутам/ресурсам.

Кроме этого, в проекте используются вспомогательные модули:

- drf nested routers, для реализации вложенности маршрутов для связанных ресурсов
- drf jwt, для замены стандартной системы авторизации на JSON Web Token
- django_unixdatetimefield, для поля с типом UnixDateTimeField в моделях Django ORM
- django debug toolbar, для анализа и оптимизации запросов к БД

В качестве системы управления базой данных выбрана PostgreSQL. Никаких специфичных расширений БД не использовалось.

Механика работы

Модель данных в проекте состоит из трех таблиц – стандартной таблицы с записями пользователей (`User`), таблицы с записями профилей пользователя (`Profile`) и таблицы с записями поездок (`Trip`).

Таблица `Profile` используется для различия простых пользователей от водителей. Различие происходит на основе значения поля `driver`. Значение `True` определяет в пользователе водителя, что дает возможность просматривать записи поездок. Запись в таблице `Profile` создается автоматически вместе с каждой новой записью в таблице `User`. Этот функционал реализован с помощью `post_save` сигналов. По умолчанию для каждого вновь созданного профиля поле driver имеет значение `False`.

Создание записей в таблице `Trip` возможно только для водителей. Попытка создать запись поездки для простого пользователя вызовет ошибку валидации поля `user`:

Only drivers are allowed to register trips.

После регистрации водители могут просматривать эти записи через API.

Запрет на просмотр чужих записей (созданных не пользователем, от имени которого осуществляется запрос) реализован с помощью класса `IsOwner` (файл `permissions.py` в приложении `rest_core`). Общее ограничение на просмотр поездок “не водителями” осуществляется с помощью класса `IsDriver`.

Фильтрация записей поездок осуществляется классом `TripViewSet` (файл `views.py` в приложении `rest_core`). Метод `list` может принимать необязательные аргументы start и end, которые должны являться метками времени Unix-эпохи (Unix Epoch Timestamp, сек.). В случае отправки клиентом некорректной метки, сервер ответит следующим сообщением со статусом 400:

```
{
  ' detail': 'START|END argument: ERROR_MESSAGE'
}
```

В случае корректности метки (меток), записи таблицы `Trip` для текущего пользователя будут отфильтрованы и отправленные в JSON формате в ответ. Принцип фильтрации:

- поле `start` меньше или равно аргументу start запроса
- поле `end` больше или равно аргументу end запроса

Таким образом, после фильтрации останутся только те записи, которые укладываются в диапазон заданных временных рамок.

Примечания

Данное условие –

“Being a driver does not involve any additional information above that categorization”

формально не описывает требования к реализации различий водителей от простых пользователей, поэтому я посчитал разумным использовать компромиссный вариант. Вместо переопределения стандартной модели пользователя или иного, потенциально-проблемного в перспективе действия, использовать стандартный прием с таблицей “профилем”, запись, в которой однозначно связана с конкретным пользователем, и создается автоматически без его непосредственного участия.

Еще один выбор продиктован этим решением – поскольку в задании не указана необходимость настраивать отображение полей в админке, админ панель осталась стандартной с раздельным управлением моделями. Из-за этого для назначения пользователя водителем, необходимо отдельно создать пользователя и затем отдельно же внести изменения в его профиль.

Требование касательно маршрута API повлияло на выбор DRF и необходимость использовать дополнительный модуль. Собственноручная реализация API без сторонних модулей показалась мне весьма ограничивающим фактором по мере роста проекта (если уж рассматривать это не как тестовое задание, а часть реального проекта), как в плане масштабирования, так и в плане отладки, да и понимания архитектуры в целом. Поэтому выбор сразу пал на DRF.
Что касается модуля `drf nested routers` – без него нет стандартной возможности организовать вложенность маршрутов одних представлений в другие, в том виде, которого требовал предоставленный пример:

/teslooptest/user/USERID:/trips?start=START:&end=END:

Далее, не хватает одной детали для этого условия:

“If the user is not a driver, hitting the API endpoint with auth tokens in header or cookie will result in authorization error”

Не указано, какой именно код статуса сервер должен отдавать в таком случае – 401 (Unauthorized) или 403 (Forbidden). В первом случае обычно предполагается нехватка полномочий, во втором - отказ сервера в доступе. По факту обе ошибки говорят о проблеме с авторизацией с разного ракурса, но в реализации системы разрешений DRF выбор пал на 403. Я не стал это переопределять, но оставил такую возможность – для этого необходимо в представлении `TripViewSet` закомментировать метод `get_permissions` и раскомментировать помеченный комментарием блок там же.

Также, ввиду наличия выбора, где хранить токен авторизации (auth tokens in header or cookie), выбор был сделан в пользу заголовка – хотя бы потому, что “из коробки” DRF JWT не поддерживает кукис (хотя запрос на добавление этого функционала висит с 15-го года). Вариант исправить это конечно есть, но, не зная как планируется использовать сервис, нет уверенности, что это действительно необходимо.

Примечание.

Тут же следует заметить, что в стандартной реализации JWT с токеном в заголовке существует уязвимость – если удалось перехватить токен, даже в случае если пользователь уже выполнил выход из сервиса, токен остается полностью валидным до исчерпания срока его действия. Это означает, что токеном можно успешно пользоваться даже не зная учетные данные пользователя, которому он принадлежит. Поскольку сам токен не храниться на сервере, есть еще более интересная возможность – просто обновлять токен до истечения срока действия и таким образом продлевать возможность “хозяйничать” на сервере сколь угодно долго (пока сервер не упадет/не изменится SECRET_KEY или не произойдет принудительная инвалидация всех токенов).

И заключительное. В задании не было указано может ли любой водитель просматривать поездки других водителей. Я пришел к выводу, что такое поведение является недопустимым и ограничил такую возможность. Если все же любые поездки должны быть доступны любому водителю, это легко исправить, удалив из списка разрешений класс `IsOwner`.