**RepeatPro documentation**

**Tutors service**

1. **Project Database:**

**БД проекта состоит их таких сущностей как :**

**1.Таблица TutorUser**

**-tutor\_id : Уникальный идентификатор репетитора (PK) \***

**-email :Электронная почта репетитора(EmailField) - Unique \***

**-password :Пароль репетитора(CharField) \***

**-first\_name: Имя репетитора(CharField) \***

**-last\_name : Фамилия репетитора(CharField) \***

**-phone\_number : Номер телефона репетитора (CharField) - Unique \***

**-bio : Био репетитора (TextField)**

**-date\_of\_birth :Дата рождения репетитора(DateTimeField)**

**-experience: Опыт работы (ChoiceField)**

**-education : Образование,ВУЗ(CharField)**

**-degree : Академическая степень (ChoiceField)**

**-year\_of\_ending: Год окончания ВУЗа (IntegerField)**

**-avatar : Аватар Репетитора (ImageField)**

**-salary : Ожидаемая ставка за час работы репетитора (СhoiceField)**

**-link: Ссылка на видео презентацию репетитора(URLField)**

**-activate\_post: Флаг для активации/деактивации профиля репетитора на сайте(BooleanField)**

**-courses: Связь с таблицей Courses через TutorCourse, репетитор может иметь несколько курсов**

**`EXPERIENCE\_CHOICES` (tuple): Кортеж с опциями опыта работы. `SALARY\_CHOICES` (tuple): Кортеж с опциями примерной ставки за час работы. `DEGREE\_CHOICES` (tuple): Кортеж с опциями академических степеней.**

**2.Таблица Courses**

**-course\_id:Уникальный идентификатор курса (PK)**

**-name:Название курса(СharField)**

**-description:Описание курса(CharField)**

**3.Таблица TutorCourse**

**-tutor\_course\_id : Уникальный идентификатор (PK)**

**-tutor\_id : Cвязь с таблицей TutorUser(FK)**

**-course\_id :Связь с таблицей Course(FK)**

**4.Таблица TutorRequest**

**-tutor\_request\_id : Уникальный идентификатор (PK)**

**-tutor\_id : Cвязь с таблицей TutorUser(FK)**

**-user\_id: Cвязь с таблицей User(FK)**

**-status : Статус запроса на обучение , по умолчанию Pending, есть опция Accept/Reject**

**-date\_time: Дата для обучения**

**-text : доп информация , текст для репетитора от клиента**

**5.Таблица Review**

**-tutor\_id : Cвязь с таблицей TutorUser(FK)**

**-user\_id: Cвязь с таблицей User(FK)**

**-text:Текст отзыва(СharField)**

**-rating : Оценка от клиента для репетитора, от 1 до 5 звезд(IntegerField,ChoiceField)**

**-created\_at: Дата создания отзыва(DateTimeField)**

**ERD >**[**тут**](https://lucid.app/lucidchart/fb2abe3d-ae77-4641-8d42-a004463b9f66/edit?viewport_loc=-1467%2C-1079%2C2556%2C1150%2C0_0&invitationId=inv_a51fc63d-13d0-4834-b553-69fb9abb4e49)**<**

**Project Serializers**

**1.Tutor List Serializer**

**======================================================================**

**Сериализатор для вывода всех активных репетиторов**

**=======================================================================**

**class TutorListSerializer(serializers.ModelSerializer):**

**"""Display All Active Tutors """**

**courses = serializers.SerializerMethodField()**

**def get\_courses(self, obj):**

**return [course.name for course in obj.courses.all()]**

**def get\_average\_rating(self, tutor):**

**reviews = tutor.reviews.all()**

**total\_rating = sum(review.rating for review in reviews)**

**return round(total\_rating / reviews.count(),2) if reviews.count() > 0 else 0.0**

**average\_rating = serializers.SerializerMethodField('get\_average\_rating')**

**class Meta:**

**model = TutorUser**

**fields = ('id','first\_name','last\_name','bio','salary','avatar','average\_rating','experience','courses')**

**=======================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Метод get\_courses отвечает за получения списка курсов , которые ведет репетитор**

**Метод get\_average\_rating отвечает за вывод среднего рейтинга репетитора**

**2.Tutor User Profile Serializer**

**=======================================================================**

**Сериализатор для вывода профиля репетитора для клиента**

**=======================================================================**

**class TutorUserProfile(serializers.ModelSerializer):**

**""" Display Tutor Profile for Client """**

**def get\_average\_rating(self, tutor):**

**reviews = tutor.reviews.all()**

**total\_rating = sum(review.rating for review in reviews)**

**return round(total\_rating / reviews.count(),2) if reviews.count() > 0 else 0.0**

**average\_rating = serializers.SerializerMethodField('get\_average\_rating')**

**reviews = ReviewViewSerializer(many=True,read\_only=True)**

**courses = serializers.SerializerMethodField()**

**def get\_courses(self, obj):**

**return [course.name for course in obj.courses.all()]**

**class Meta:**

**model = TutorUser**

**fields = ('first\_name','last\_name','email','phone\_number','bio','avatar','files','date\_of\_birth','experience','education','degree','courses','yof','salary','link','average\_rating','reviews')**

**=========================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Метод get\_courses отвечает за получения списка курсов , которые ведет репетитор**

**Метод get\_average\_rating отвечает за вывод среднего рейтинга репетитора**

**average\_rating ,reviews,courses отвечают за вывод среднего рейтинга, отзывов о репетиторе и за вывод курсов , которые преподает репетитор**

**3.UpdateUserSerializer**

**=========================================================================**

**Сериализатор для изменения профиля репетитора**

**=========================================================================**

**class UpdateUserSerializer(serializers.ModelSerializer):**

**""" Update Tutor Profile """**

**tutor\_user\_model=TutorUser**

**def getEmail(self, tutor\_user\_model):**

**return tutor\_user\_model.email**

**def getFirstName(self, tutor\_user\_model):**

**return tutor\_user\_model.first\_name**

**def getLastName(self, tutor\_user\_model):**

**return tutor\_user\_model.last\_name**

**def getPhone(self, tutor\_user\_model):**

**return tutor\_user\_model.phone\_number**

**def getCourses(self, tutor\_user\_model):**

**return tutor\_user\_model.courses**

**email = serializers.SerializerMethodField("getEmail")**

**first\_name = serializers.SerializerMethodField("getFirstName")**

**last\_name = serializers.SerializerMethodField("getLastName")**

**phone\_num = serializers.SerializerMethodField("getPhone")**

**courses = serializers.SlugRelatedField(**

**queryset=Courses.objects.all(),**

**many=True,**

**slug\_field='name',**

**required=False**

**)**

**class Meta:**

**model = TutorUser**

**fields = ('email','first\_name','last\_name','phone\_num',"avatar",'bio','date\_of\_birth','experience','education','degree','yof','courses','salary','files','link',"slots",'activate\_post')**

**def update(self, instance, validated\_data):**

**courses\_data = validated\_data.pop('courses', None)**

**for attr, value in validated\_data.items():**

**if value is not None and getattr(instance, attr) != value:**

**setattr(instance, attr, value)**

**if courses\_data is not None:**

**instance.courses.clear()**

**for course\_data in courses\_data:**

**instance.courses.add(course\_data)**

**instance.save()**

**return instance**

**def to\_representation(self, instance):**

**representation = super().to\_representation(instance)**

**courses = instance.courses.all()**

**course\_names = [course.name for course in courses]**

**representation['courses'] = course\_names**

**return representation**

**=========================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Метод getEmail отвечает за получение email репетитора с таблицы TutorUser**

**Метод getFirstName отвечает за получение FirstName репетитора с таблицы TutorUser**

**Метод getLastName отвечает за получение LastName репетитора с таблицы TutorUser**

**Метод getPhone отвечает за получение PhoneNumber репетитора с таблицы TutorUser**

**email = serializers.SerializerMethodField("getEmail")**

**first\_name = serializers.SerializerMethodField("getFirstName")**

**last\_name = serializers.SerializerMethodField("getLastName")**

**phone\_num = serializers.SerializerMethodField("getPhone")**

**Здесь мы используем SerializerMethodField для создания полей, которые нужны нам только для чтения, в атрибуте мы указываем имя метода, которая будет вызываться для получения значении из выше упомянутых методов(email,first\_name,last\_name,phone\_number).**

**courses = serializers.SlugRelatedField(**

**queryset=Courses.objects.all(),**

**many=True,**

**slug\_field='name',**

**required=False**

**)**

**Поле courses представляет нам связанные курсы репетитора и выводит списком строк**

**queryset - запрос для выбора всех курсов из таблицы Courses**

**many - Определяет , что репетитор может иметь множество курсов**

**slug\_field - Определяет, что в поле будет храниться строковое представление связанных объектов курсов, используя значение поля 'name'.**

**required=False - озночает , что поле является не обязательным для заполнения.**

**Метод update используется для обновления полей-данных Репетитора**

**Сначала метод извлекает данные о курсах (courses\_data) из словаря validated\_data с помощью метода pop. Затем происходит перебор всех ключей и значений в словаре validated\_data. Если значение атрибута в словаре validated\_data не является None и отличается от значения соответствующего атрибута в экземпляре модели (instance), то значение атрибута в экземпляре модели обновляется с помощью функции setattr. Если данные о курсах (courses\_data) не являются None, то все текущие курсы, связанные с репетитором (содержащиеся в поле courses экземпляра модели), удаляются с помощью метода clear. Затем происходит перебор данных о курсах (courses\_data) и каждый курс добавляется к полю courses репетитора с помощью метода add. После обновления всех полей экземпляра модели, сохраняем изменения в базе данных с помощью метода save. В конце метод возвращает обновленный экземпляр модели TutorUser.**

**Метод to\_representation переопределяет метод сериализации для TutorUser, добавляет дополнительные данные о курсах репетитора.Сначала метод вызывает стандартный метод сериализации через super().to\_representation(instance**)**, чтобы получить базовое представление экземпляра модели TutorUser**

**4.ReviewViewSerializer**

**=========================================================================**

**Сериализатор для вывода отзывов репетитора**

**=========================================================================**

**class ReviewViewSerializer(serializers.ModelSerializer):**

**""" Display All Tutor Reviews """**

**client\_name = serializers.CharField(source='client.get\_full\_name', read\_only=True)**

**class Meta:**

**model = Review**

**fields = ('client\_name','description','rating','created\_at')**

**===================================================================**

**Serializer работает с таблицей Review и ее полями , которые указаны в fields**

**client\_name - атрибут для для получения имени автора отзыва , получаемая через связанную таблицу User-а**

**5.ReviewCreateSerializer**

**========================================================================**

**Сериализатор для создания отзыва клиентом для репетитора**

**========================================================================**

**class ReviewCreateSerializer(serializers.ModelSerializer):**

**""" Create Review for a Tutor """**

**tutor = serializers.HiddenField(default = serializers.CurrentUserDefault())**

**class Meta:**

**model = Review**

**fields = '\_\_all\_\_'**

**==================================================================**

**tutor - Репетитор, связанный с отзывом. Это поле скрыто и по умолчанию устанавливается на текущего репетитора**

**6.Tutor Register Serializer**

**========================================================================**

**Сериализатор для регистрации репетитора**

**========================================================================**

**class TutorUserCreateSerializer(UserCreateSerializer):**

**""" Tutor Sign Up using Djoser JWT """**

**class Meta(UserCreateSerializer.Meta):**

**model = TutorUser**

**fields = ('id','email','first\_name','last\_name','phone\_number','password')**

**==================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**TutorUserCreateSerializer - настраиваемый сериализатор, используемая для регистрации репетиторов на сервиса с помощью Djoser + JWT  
Он наследуется он UserCreateSerializer сериализатора представленного Djoser для обработки регистрации репетиторов**

**(Сериализатор прописан в settings->Djoser)**

**7.Tutor Request Serializer**

**=========================================================================**

**Сериализатор отвечает за вывод всех запросов репетитора от клиентов.Где он также может принимать/отклонять запросы**

**=========================================================================**

**class TutorRequestSerializer(serializers.ModelSerializer):**

**client = serializers.CharField(source = 'client.get\_full\_name')**

**class Meta:**

**model=TutorRequest**

**fields = ("id","client","status","text","date\_time",)**

**========================================================================**

**Serializer работает с таблицей TutorRequest и ее полями , которые указаны в fields**

**8.Tutor Requisites Serializer**

**========================================================================**

**Сериализатор отвечает за реквезиты репетитора**

**========================================================================**

**class TutorRequisitesSerializer(serializers.ModelSerializer):**

**class Meta:**

**model=TutorUser**

**fields = ('id','email','first\_name','last\_name','phone\_number')**

**===================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Project Views**

**1.Tutor List View**

**=============================================================================**

**class TutorsListView(generics.ListAPIView):**

**"""Display All Active Tutors with filter by salary,experience,degree,average rating and courses """**

**queryset = TutorUser.objects.all()**

**serializer\_class = TutorListSerializer**

**permission\_classes = [IsAuthenticatedOrReadOnly,]**

**filter\_backends = (DjangoFilterBackend,SearchFilter,OrderingFilter)**

**search\_fields = ('id','first\_name','tutorcourse\_\_course\_\_name','last\_name','bio')**

**ordering\_fields = ['salary','experience']**

**url path = [**

**path('tutors/', TutorsListView.as\_view(), name='tutors\_list'),**

**]**

**=============================================================================**

**Данный API позволяет получить список всех активных репетиторов c помощью GET запроса с параметрами salary,first\_name,last\_name,bio,avatar,salary,experience,rating,course**

**Фильтрация,Поиск,Сортировка работает через django-filerter**

**Для поиска используется используются поля в search\_fields**

**Для сортировки используется используются поля в ordering\_fields**

**Так же прописан Permission , он дает доступ к взаимодействию с endpoint авторизованным пользователям , иначе только для чтения**

**Для фильтрации используются поля salary,experience,degree,rating,course**

**Для реализации фильтрации используется Filter Class**

**class TutorFilter(filters.FilterSet):**

**salary =filters.RangeFilter( label='Price per lesson')**

**experience = filters.RangeFilter(label='Enter experience range (0-10)')**

**degree = filters.ChoiceFilter(choices=TutorUser.DEGREE\_CHOICES,label = f'Сhoose a tutor by degree',empty\_label="Degrees")**

**# rating = filters.NumberFilter( method='filter\_by\_rating')**

**rating = filters.ChoiceFilter(choices=Review.RATING\_CHOICES,method='filter\_by\_rating',label = 'Choose a rating',empty\_label="Ratings")**

**course = filters.ModelMultipleChoiceFilter(**

**queryset=Courses.objects.all(),**

**field\_name='tutorcourse\_\_course\_\_id',**

**to\_field\_name='id',label = 'Choose courses'**

**)**

**class Meta:**

**model = TutorUser**

**fields = ('salary', 'experience', 'degree','rating','course')**

**def filter\_by\_rating(self, queryset, name, value):**

**queryset = queryset.annotate(average\_rating=models.Avg('reviews\_\_rating'))**

**queryset = queryset.filter(average\_rating\_\_gte=(value))**

**return queryset**

**Доступные фильтры:**

**- Зарплата: Фильтруйте репетиторов по диапазону их зарплаты за урок.**

**- Опыт: Фильтр репетиторов по их многолетнему опыту в заданном диапазоне (0-10).**

**- Степень: Отфильтруйте преподавателей по их ученой степени.**

**- Рейтинг: Фильтр репетиторов по их среднему рейтингу, предоставленному студентами.**

**- Курс: фильтрация репетиторов по курсам, которые они предлагают**

**Пример :GET /api/?rating=4**

**2.Tutor User Profile for Сlient**

**=============================================================================**

**class TutorProfileClientView(generics.RetrieveAPIView):**

**""" Display Tutor Profile for Client """**

**queryset = TutorUser.objects.all()**

**serializer\_class = TutorUserProfile**

**permission\_classes = [IsAuthenticatedOrReadOnly,]**

**lookup\_field = 'pk'**

**url path = [  
 path('tutors/<int:pk>/',TutorProfileClientView.as\_view(), name='tutor\_detail\_page'),**

**]**

**============================================================================**

**Данный API позволяет получить профиль репетитора и посмотреть более подробную информацию о репетиторе**

**Работает с моделью TutorUser и с сериализатором TutorUserProfile**

**Прописан Permission Class - только авторизованный пользователь может взаимодействовать со странице , а иначе только просматривать**

**lookup\_field используется для вывода определенного обьекта(репетитора) в модели по его id(pk)**

**3.Update Tutor Profile**

**============================================================================**

**class UpdateProfileView(generics.RetrieveUpdateDestroyAPIView):**

**permission\_classes = [IsAuthenticated]**

**"""Update Profile API"""**

**serializer\_class = UpdateUserSerializer**

**def get\_object(self):**

**return self.request.user**

**def put(self, request):**

**user = self.get\_object()**

**serializer = UpdateUserSerializer(user, data=request.data)**

**if serializer.is\_valid():**

**serializer.save()**

**return Response(serializer.data)**

**return Response(serializer.errors, status=400)**

**url path = [**

**path('tutor/<int:pk>/', UpdateProfileView.as\_view(), name='update\_profile'),**

**]**

**=============================================================================**

**Данный API позволяет изменить профиль репетитора**

**Работает с моделью TutorUser и с сериализатором UpdateUserSerializer  
метод get\_object извлекает аутентифицированный юзер объек по Bearer токену , затем с   
  
Метод put обрабатывает запрос PUT на обновление профиля пользователя. Сначала он проверяет данные, отправленные в запросе, по определенным полям сериализатора. Если данные действительны, он сохраняет обновленные данные в пользовательский объект и базу данных и возвращает обновленные данные профиля в ответе.А иначе ошибку с кодом 400**

**4.Review Create View**

**============================================================================**

**class ReviewCreateView(generics.CreateAPIView):**

**""" Create Review for a Tutor """**

**queryset = Review.objects.all()**

**serializer\_class = ReviewCreateSerializer**

**permission\_classes = [IsAuthenticated,]**

**def post(self, request, \*args, \*\*kwargs):**

**response = super().post(request, \*args, \*\*kwargs)**

**return Response(response.data, status=201)**

**url path = [**

**path('tutor/<int:pk>/review/', ReviewCreateView.as\_view(), name='review\_page'),**

**]**

**==========================================================================**

**Данный API позволяет писать отзыв репетитору**

**Работает с моделью Review и с сериализатором ReviewCreateSerializer  
Permission Class - только владелец авторизованный пользователь может написать отзыв**

**Метод post - обрабатывает POST запрос в виде отзыва и сохраняет отзыв в таблицу Review таблицу и возвращает статус 201 - Created**

**5.Tutor Request View**

**==========================================================================**

**class TutorRequestView(APIView):**

**def get(self, request, tutor\_id):**

**tutor\_requests = TutorRequest.objects.filter(tutor\_id=tutor\_id)**

**serializer = TutorRequestSerializer(tutor\_requests, many=True)**

**return Response(serializer.data)**

**def patch(self, request, tutor\_id):**

**tutor\_request\_id = request.data.get('tutor\_request\_id')**

**status\_value = request.data.get('status')**

**allowed\_statuses = [choice[0] for choice in TutorRequest.STATUS\_CHOICES]**

**if status\_value not in allowed\_statuses:**

**return Response({'error': 'Invalid status value'}, status=status.HTTP\_400\_BAD\_REQUEST)**

**try:**

**tutor\_request = TutorRequest.objects.get(id=tutor\_request\_id, tutor\_id=tutor\_id)**

**except TutorRequest.DoesNotExist:**

**return Response({'error': 'Tutor request not found'}, status=status.HTTP\_404\_NOT\_FOUND)**

**tutor\_request.status = status\_value**

**tutor\_request.save()**

**clients\_api\_url = 'http://clients-service:8000/api/'**

**response = requests.post(clients\_api\_url, data=serializers.data)**

**if response.status\_code == 200:**

**return Response({'message': 'Successfully'}, status=status.HTTP\_200\_OK)**

**else:**

**return Response({'error': 'Failed'}, status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)**

**url path[  
 path('tutor\_requests/<int:tutor\_id>/', TutorRequestView.as\_view(), name='tutor\_requests'),  
]**

**=============================================================================**

**Данный API позволяет просматривать запросы репетитора и принимать/отклонять запросы**

**Работает с моделью TutorRequest и с сериализатором TutorRequestSerializer  
Permission Class - только владелец авторизованный пользователь может написать отзыв**

**Метод get - обрабатывает GET запрос и возвращает все запросы репетитора по его ID из URL**

**Метод patch - обрабатывает PATH запрос и предназначен для обновления статуса запроса.Извлекает tutor\_request\_id и status и срванивает со значениями с TutorRequestChoices.Если значение status, то возвращает ошибку 400**

**Если tutor\_request\_id и tutor\_id не найден , то возвращает ошибку 404,  
В случае успешных действии, данные отправляются в БД и обновляет status**

**Далее посылает POST запрос на url-**clients\_api\_url **для дальнейшей обработки на сервисе клиента и возвращает статус-код 200 при успешных действиях**

**6.Tutor Profile View**

**======================================================================**

**class UserProfileView(APIView):**

**"""Display Tutor Profile """**

**permission\_classes = [IsAuthenticated]**

**def get(self, request):**

**user = request.user**

**serializer = TutorUserProfile(user)**

**return Response(serializer.data)**

**Данный API предназначен для обработки GET запроса для просмотра профиля репетитора, он извлекает auth пользовательский объект и возвращает данные с экземлярами сериализатора(name,surname ...)**

**Permissions**

**1.IsOwnerOrReadOnly**

**=========================================================================**

**class IsOwnerOrReadOnly(permissions.BasePermission):**

**def has\_object\_permission(self, request, view, obj):**

**if request.method in permissions.SAFE\_METHODS:**

**return True**

**return obj == request.user**

**=====================================================================**

**Данный Permission проверяет , если request.method принадлежит к SAFE\_METHODS,то пользователь может только читать запись**

**А иначе проверяется условие** obj == request.user **и если пользователь, отправивший запрос, является владельцем ,то он может менять свой профиль или запросы от клиентов**

**API Endpoints**

| **API** | **Method** | **Description** |
| --- | --- | --- |
| **signup/tutor/** | **POST** | **Регистрация репетитора** |
| **signin/tutor/** | **POST** | **Логин репетитора** |
| **tutors/** | **GET** | **Получить список всех репетиторов** |
| **tutor-profile/update/** | **GET**  **PUT**  **PATCH**  **DELETE** | **Получить информацию о текущем**  **пользователе**  **Обновить профиль репетитора**  **Удалить профиль репетитора** |
| **tutor-profile/** | **GET** | **Получить информацию о репетиторе** |
| **tutor/<int:pk>/review/** | **GET**  **POST** | **Создать отзыв для конкретного репетитора** |
| **tutor\_requests/<int:tutor\_id>/** | **GET**  **PATCH** | **Получить список запросов конкретного репетитора**  **Обновить статус запроса репетитора** |

**ocumentation**

**Tutors service**

1. **Project Database:**

**БД проекта состоит их таких сущностей как :**

**1.Таблица TutorUser**

**-tutor\_id : Уникальный идентификатор репетитора (PK) \***

**-email :Электронная почта репетитора(EmailField) - Unique \***

**-password :Пароль репетитора(CharField) \***

**-first\_name: Имя репетитора(CharField) \***

**-last\_name : Фамилия репетитора(CharField) \***

**-phone\_number : Номер телефона репетитора (CharField) - Unique \***

**-bio : Био репетитора (TextField)**

**-date\_of\_birth :Дата рождения репетитора(DateTimeField)**

**-experience: Опыт работы (ChoiceField)**

**-education : Образование,ВУЗ(CharField)**

**-degree : Академическая степень (ChoiceField)**

**-year\_of\_ending: Год окончания ВУЗа (IntegerField)**

**-avatar : Аватар Репетитора (ImageField)**

**-salary : Ожидаемая ставка за час работы репетитора (СhoiceField)**

**-link: Ссылка на видео презентацию репетитора(URLField)**

**-activate\_post: Флаг для активации/деактивации профиля репетитора на сайте(BooleanField)**

**-courses: Связь с таблицей Courses через TutorCourse, репетитор может иметь несколько курсов**

**`EXPERIENCE\_CHOICES` (tuple): Кортеж с опциями опыта работы. `SALARY\_CHOICES` (tuple): Кортеж с опциями примерной ставки за час работы. `DEGREE\_CHOICES` (tuple): Кортеж с опциями академических степеней.**

**2.Таблица Courses**

**-course\_id:Уникальный идентификатор курса (PK)**

**-name:Название курса(СharField)**

**-description:Описание курса(CharField)**

**3.Таблица TutorCourse**

**-tutor\_course\_id : Уникальный идентификатор (PK)**

**-tutor\_id : Cвязь с таблицей TutorUser(FK)**

**-course\_id :Связь с таблицей Course(FK)**

**4.Таблица TutorRequest**

**-tutor\_request\_id : Уникальный идентификатор (PK)**

**-tutor\_id : Cвязь с таблицей TutorUser(FK)**

**-user\_id: Cвязь с таблицей User(FK)**

**-status : Статус запроса на обучение , по умолчанию Pending, есть опция Accept/Reject**

**-date\_time: Дата для обучения**

**-text : доп информация , текст для репетитора от клиента**

**5.Таблица Review**

**-tutor\_id : Cвязь с таблицей TutorUser(FK)**

**-user\_id: Cвязь с таблицей User(FK)**

**-text:Текст отзыва(СharField)**

**-rating : Оценка от клиента для репетитора, от 1 до 5 звезд(IntegerField,ChoiceField)**

**-created\_at: Дата создания отзыва(DateTimeField)**

**ERD >**[**тут**](https://lucid.app/lucidchart/fb2abe3d-ae77-4641-8d42-a004463b9f66/edit?viewport_loc=-1467%2C-1079%2C2556%2C1150%2C0_0&invitationId=inv_a51fc63d-13d0-4834-b553-69fb9abb4e49)**<**

**Project Serializers**

**1.Tutor List Serializer**

**======================================================================**

**Сериализатор для вывода всех активных репетиторов**

**=======================================================================**

**class TutorListSerializer(serializers.ModelSerializer):**

**"""Display All Active Tutors """**

**courses = serializers.SerializerMethodField()**

**def get\_courses(self, obj):**

**return [course.name for course in obj.courses.all()]**

**def get\_average\_rating(self, tutor):**

**reviews = tutor.reviews.all()**

**total\_rating = sum(review.rating for review in reviews)**

**return round(total\_rating / reviews.count(),2) if reviews.count() > 0 else 0.0**

**average\_rating = serializers.SerializerMethodField('get\_average\_rating')**

**class Meta:**

**model = TutorUser**

**fields = ('id','first\_name','last\_name','bio','salary','avatar','average\_rating','experience','courses')**

**=======================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Метод get\_courses отвечает за получения списка курсов , которые ведет репетитор**

**Метод get\_average\_rating отвечает за вывод среднего рейтинга репетитора**

**2.Tutor User Profile Serializer**

**=======================================================================**

**Сериализатор для вывода профиля репетитора для клиента**

**=======================================================================**

**class TutorUserProfile(serializers.ModelSerializer):**

**""" Display Tutor Profile for Client """**

**def get\_average\_rating(self, tutor):**

**reviews = tutor.reviews.all()**

**total\_rating = sum(review.rating for review in reviews)**

**return round(total\_rating / reviews.count(),2) if reviews.count() > 0 else 0.0**

**average\_rating = serializers.SerializerMethodField('get\_average\_rating')**

**reviews = ReviewViewSerializer(many=True,read\_only=True)**

**courses = serializers.SerializerMethodField()**

**def get\_courses(self, obj):**

**return [course.name for course in obj.courses.all()]**

**class Meta:**

**model = TutorUser**

**fields = ('first\_name','last\_name','email','phone\_number','bio','avatar','files','date\_of\_birth','experience','education','degree','courses','yof','salary','link','average\_rating','reviews')**

**=========================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Метод get\_courses отвечает за получения списка курсов , которые ведет репетитор**

**Метод get\_average\_rating отвечает за вывод среднего рейтинга репетитора**

**average\_rating ,reviews,courses отвечают за вывод среднего рейтинга, отзывов о репетиторе и за вывод курсов , которые преподает репетитор**

**3.UpdateUserSerializer**

**=========================================================================**

**Сериализатор для изменения профиля репетитора**

**=========================================================================**

**class UpdateUserSerializer(serializers.ModelSerializer):**

**""" Update Tutor Profile """**

**tutor\_user\_model=TutorUser**

**def getEmail(self, tutor\_user\_model):**

**return tutor\_user\_model.email**

**def getFirstName(self, tutor\_user\_model):**

**return tutor\_user\_model.first\_name**

**def getLastName(self, tutor\_user\_model):**

**return tutor\_user\_model.last\_name**

**def getPhone(self, tutor\_user\_model):**

**return tutor\_user\_model.phone\_number**

**def getCourses(self, tutor\_user\_model):**

**return tutor\_user\_model.courses**

**email = serializers.SerializerMethodField("getEmail")**

**first\_name = serializers.SerializerMethodField("getFirstName")**

**last\_name = serializers.SerializerMethodField("getLastName")**

**phone\_num = serializers.SerializerMethodField("getPhone")**

**courses = serializers.SlugRelatedField(**

**queryset=Courses.objects.all(),**

**many=True,**

**slug\_field='name',**

**required=False**

**)**

**class Meta:**

**model = TutorUser**

**fields = ('email','first\_name','last\_name','phone\_num',"avatar",'bio','date\_of\_birth','experience','education','degree','yof','courses','salary','files','link',"slots",'activate\_post')**

**def update(self, instance, validated\_data):**

**courses\_data = validated\_data.pop('courses', None)**

**for attr, value in validated\_data.items():**

**if value is not None and getattr(instance, attr) != value:**

**setattr(instance, attr, value)**

**if courses\_data is not None:**

**instance.courses.clear()**

**for course\_data in courses\_data:**

**instance.courses.add(course\_data)**

**instance.save()**

**return instance**

**def to\_representation(self, instance):**

**representation = super().to\_representation(instance)**

**courses = instance.courses.all()**

**course\_names = [course.name for course in courses]**

**representation['courses'] = course\_names**

**return representation**

**=========================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Метод getEmail отвечает за получение email репетитора с таблицы TutorUser**

**Метод getFirstName отвечает за получение FirstName репетитора с таблицы TutorUser**

**Метод getLastName отвечает за получение LastName репетитора с таблицы TutorUser**

**Метод getPhone отвечает за получение PhoneNumber репетитора с таблицы TutorUser**

**email = serializers.SerializerMethodField("getEmail")**

**first\_name = serializers.SerializerMethodField("getFirstName")**

**last\_name = serializers.SerializerMethodField("getLastName")**

**phone\_num = serializers.SerializerMethodField("getPhone")**

**Здесь мы используем SerializerMethodField для создания полей, которые нужны нам только для чтения, в атрибуте мы указываем имя метода, которая будет вызываться для получения значении из выше упомянутых методов(email,first\_name,last\_name,phone\_number).**

**courses = serializers.SlugRelatedField(**

**queryset=Courses.objects.all(),**

**many=True,**

**slug\_field='name',**

**required=False**

**)**

**Поле courses представляет нам связанные курсы репетитора и выводит списком строк**

**queryset - запрос для выбора всех курсов из таблицы Courses**

**many - Определяет , что репетитор может иметь множество курсов**

**slug\_field - Определяет, что в поле будет храниться строковое представление связанных объектов курсов, используя значение поля 'name'.**

**required=False - озночает , что поле является не обязательным для заполнения.**

**Метод update используется для обновления полей-данных Репетитора**

**Сначала метод извлекает данные о курсах (courses\_data) из словаря validated\_data с помощью метода pop. Затем происходит перебор всех ключей и значений в словаре validated\_data. Если значение атрибута в словаре validated\_data не является None и отличается от значения соответствующего атрибута в экземпляре модели (instance), то значение атрибута в экземпляре модели обновляется с помощью функции setattr. Если данные о курсах (courses\_data) не являются None, то все текущие курсы, связанные с репетитором (содержащиеся в поле courses экземпляра модели), удаляются с помощью метода clear. Затем происходит перебор данных о курсах (courses\_data) и каждый курс добавляется к полю courses репетитора с помощью метода add. После обновления всех полей экземпляра модели, сохраняем изменения в базе данных с помощью метода save. В конце метод возвращает обновленный экземпляр модели TutorUser.**

**Метод to\_representation переопределяет метод сериализации для TutorUser, добавляет дополнительные данные о курсах репетитора.Сначала метод вызывает стандартный метод сериализации через super().to\_representation(instance**)**, чтобы получить базовое представление экземпляра модели TutorUser**

**4.ReviewViewSerializer**

**=========================================================================**

**Сериализатор для вывода отзывов репетитора**

**=========================================================================**

**class ReviewViewSerializer(serializers.ModelSerializer):**

**""" Display All Tutor Reviews """**

**client\_name = serializers.CharField(source='client.get\_full\_name', read\_only=True)**

**class Meta:**

**model = Review**

**fields = ('client\_name','description','rating','created\_at')**

**===================================================================**

**Serializer работает с таблицей Review и ее полями , которые указаны в fields**

**client\_name - атрибут для для получения имени автора отзыва , получаемая через связанную таблицу User-а**

**5.ReviewCreateSerializer**

**========================================================================**

**Сериализатор для создания отзыва клиентом для репетитора**

**========================================================================**

**class ReviewCreateSerializer(serializers.ModelSerializer):**

**""" Create Review for a Tutor """**

**tutor = serializers.HiddenField(default = serializers.CurrentUserDefault())**

**class Meta:**

**model = Review**

**fields = '\_\_all\_\_'**

**==================================================================**

**tutor - Репетитор, связанный с отзывом. Это поле скрыто и по умолчанию устанавливается на текущего репетитора**

**6.Tutor Register Serializer**

**========================================================================**

**Сериализатор для регистрации репетитора**

**========================================================================**

**class TutorUserCreateSerializer(UserCreateSerializer):**

**""" Tutor Sign Up using Djoser JWT """**

**class Meta(UserCreateSerializer.Meta):**

**model = TutorUser**

**fields = ('id','email','first\_name','last\_name','phone\_number','password')**

**==================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**TutorUserCreateSerializer - настраиваемый сериализатор, используемая для регистрации репетиторов на сервиса с помощью Djoser + JWT  
Он наследуется он UserCreateSerializer сериализатора представленного Djoser для обработки регистрации репетиторов**

**(Сериализатор прописан в settings->Djoser)**

**7.Tutor Request Serializer**

**=========================================================================**

**Сериализатор отвечает за вывод всех запросов репетитора от клиентов.Где он также может принимать/отклонять запросы**

**=========================================================================**

**class TutorRequestSerializer(serializers.ModelSerializer):**

**client = serializers.CharField(source = 'client.get\_full\_name')**

**class Meta:**

**model=TutorRequest**

**fields = ("id","client","status","text","date\_time",)**

**========================================================================**

**Serializer работает с таблицей TutorRequest и ее полями , которые указаны в fields**

**8.Tutor Requisites Serializer**

**========================================================================**

**Сериализатор отвечает за реквезиты репетитора**

**========================================================================**

**class TutorRequisitesSerializer(serializers.ModelSerializer):**

**class Meta:**

**model=TutorUser**

**fields = ('id','email','first\_name','last\_name','phone\_number')**

**===================================================================**

**Serializer работает с таблицей TutorUser и ее полями , которые указаны в fields**

**Project Views**

**1.Tutor List View**

**=============================================================================**

**class TutorsListView(generics.ListAPIView):**

**"""Display All Active Tutors with filter by salary,experience,degree,average rating and courses """**

**queryset = TutorUser.objects.all()**

**serializer\_class = TutorListSerializer**

**permission\_classes = [IsAuthenticatedOrReadOnly,]**

**filter\_backends = (DjangoFilterBackend,SearchFilter,OrderingFilter)**

**search\_fields = ('id','first\_name','tutorcourse\_\_course\_\_name','last\_name','bio')**

**ordering\_fields = ['salary','experience']**

**url path = [**

**path('tutors/', TutorsListView.as\_view(), name='tutors\_list'),**

**]**

**=============================================================================**

**Данный API позволяет получить список всех активных репетиторов c помощью GET запроса с параметрами salary,first\_name,last\_name,bio,avatar,salary,experience,rating,course**

**Фильтрация,Поиск,Сортировка работает через django-filerter**

**Для поиска используется используются поля в search\_fields**

**Для сортировки используется используются поля в ordering\_fields**

**Так же прописан Permission , он дает доступ к взаимодействию с endpoint авторизованным пользователям , иначе только для чтения**

**Для фильтрации используются поля salary,experience,degree,rating,course**

**Для реализации фильтрации используется Filter Class**

**class TutorFilter(filters.FilterSet):**

**salary =filters.RangeFilter( label='Price per lesson')**

**experience = filters.RangeFilter(label='Enter experience range (0-10)')**

**degree = filters.ChoiceFilter(choices=TutorUser.DEGREE\_CHOICES,label = f'Сhoose a tutor by degree',empty\_label="Degrees")**

**# rating = filters.NumberFilter( method='filter\_by\_rating')**

**rating = filters.ChoiceFilter(choices=Review.RATING\_CHOICES,method='filter\_by\_rating',label = 'Choose a rating',empty\_label="Ratings")**

**course = filters.ModelMultipleChoiceFilter(**

**queryset=Courses.objects.all(),**

**field\_name='tutorcourse\_\_course\_\_id',**

**to\_field\_name='id',label = 'Choose courses'**

**)**

**class Meta:**

**model = TutorUser**

**fields = ('salary', 'experience', 'degree','rating','course')**

**def filter\_by\_rating(self, queryset, name, value):**

**queryset = queryset.annotate(average\_rating=models.Avg('reviews\_\_rating'))**

**queryset = queryset.filter(average\_rating\_\_gte=(value))**

**return queryset**

**Доступные фильтры:**

**- Зарплата: Фильтруйте репетиторов по диапазону их зарплаты за урок.**

**- Опыт: Фильтр репетиторов по их многолетнему опыту в заданном диапазоне (0-10).**

**- Степень: Отфильтруйте преподавателей по их ученой степени.**

**- Рейтинг: Фильтр репетиторов по их среднему рейтингу, предоставленному студентами.**

**- Курс: фильтрация репетиторов по курсам, которые они предлагают**

**Пример :GET /api/?rating=4**

**2.Tutor User Profile for Сlient**

**=============================================================================**

**class TutorProfileClientView(generics.RetrieveAPIView):**

**""" Display Tutor Profile for Client """**

**queryset = TutorUser.objects.all()**

**serializer\_class = TutorUserProfile**

**permission\_classes = [IsAuthenticatedOrReadOnly,]**

**lookup\_field = 'pk'**

**url path = [  
 path('tutors/<int:pk>/',TutorProfileClientView.as\_view(), name='tutor\_detail\_page'),**

**]**

**============================================================================**

**Данный API позволяет получить профиль репетитора и посмотреть более подробную информацию о репетиторе**

**Работает с моделью TutorUser и с сериализатором TutorUserProfile**

**Прописан Permission Class - только авторизованный пользователь может взаимодействовать со странице , а иначе только просматривать**

**lookup\_field используется для вывода определенного обьекта(репетитора) в модели по его id(pk)**

**3.Update Tutor Profile**

**============================================================================**

**class UpdateProfileView(generics.RetrieveUpdateDestroyAPIView):**

**permission\_classes = [IsAuthenticated]**

**"""Update Profile API"""**

**serializer\_class = UpdateUserSerializer**

**def get\_object(self):**

**return self.request.user**

**def put(self, request):**

**user = self.get\_object()**

**serializer = UpdateUserSerializer(user, data=request.data)**

**if serializer.is\_valid():**

**serializer.save()**

**return Response(serializer.data)**

**return Response(serializer.errors, status=400)**

**url path = [**

**path('tutor/<int:pk>/', UpdateProfileView.as\_view(), name='update\_profile'),**

**]**

**=============================================================================**

**Данный API позволяет изменить профиль репетитора**

**Работает с моделью TutorUser и с сериализатором UpdateUserSerializer  
метод get\_object извлекает аутентифицированный юзер объек по Bearer токену , затем с   
  
Метод put обрабатывает запрос PUT на обновление профиля пользователя. Сначала он проверяет данные, отправленные в запросе, по определенным полям сериализатора. Если данные действительны, он сохраняет обновленные данные в пользовательский объект и базу данных и возвращает обновленные данные профиля в ответе.А иначе ошибку с кодом 400**

**4.Review Create View**

**============================================================================**

**class ReviewCreateView(generics.CreateAPIView):**

**""" Create Review for a Tutor """**

**queryset = Review.objects.all()**

**serializer\_class = ReviewCreateSerializer**

**permission\_classes = [IsAuthenticated,]**

**def post(self, request, \*args, \*\*kwargs):**

**response = super().post(request, \*args, \*\*kwargs)**

**return Response(response.data, status=201)**

**url path = [**

**path('tutor/<int:pk>/review/', ReviewCreateView.as\_view(), name='review\_page'),**

**]**

**==========================================================================**

**Данный API позволяет писать отзыв репетитору**

**Работает с моделью Review и с сериализатором ReviewCreateSerializer  
Permission Class - только владелец авторизованный пользователь может написать отзыв**

**Метод post - обрабатывает POST запрос в виде отзыва и сохраняет отзыв в таблицу Review таблицу и возвращает статус 201 - Created**

**5.Tutor Request View**

**==========================================================================**

**class TutorRequestView(APIView):**

**def get(self, request, tutor\_id):**

**tutor\_requests = TutorRequest.objects.filter(tutor\_id=tutor\_id)**

**serializer = TutorRequestSerializer(tutor\_requests, many=True)**

**return Response(serializer.data)**

**def patch(self, request, tutor\_id):**

**tutor\_request\_id = request.data.get('tutor\_request\_id')**

**status\_value = request.data.get('status')**

**allowed\_statuses = [choice[0] for choice in TutorRequest.STATUS\_CHOICES]**

**if status\_value not in allowed\_statuses:**

**return Response({'error': 'Invalid status value'}, status=status.HTTP\_400\_BAD\_REQUEST)**

**try:**

**tutor\_request = TutorRequest.objects.get(id=tutor\_request\_id, tutor\_id=tutor\_id)**

**except TutorRequest.DoesNotExist:**

**return Response({'error': 'Tutor request not found'}, status=status.HTTP\_404\_NOT\_FOUND)**

**tutor\_request.status = status\_value**

**tutor\_request.save()**

**clients\_api\_url = 'http://clients-service:8000/api/'**

**response = requests.post(clients\_api\_url, data=serializers.data)**

**if response.status\_code == 200:**

**return Response({'message': 'Successfully'}, status=status.HTTP\_200\_OK)**

**else:**

**return Response({'error': 'Failed'}, status=status.HTTP\_500\_INTERNAL\_SERVER\_ERROR)**

**url path[  
 path('tutor\_requests/<int:tutor\_id>/', TutorRequestView.as\_view(), name='tutor\_requests'),  
]**

**=============================================================================**

**Данный API позволяет просматривать запросы репетитора и принимать/отклонять запросы**

**Работает с моделью TutorRequest и с сериализатором TutorRequestSerializer  
Permission Class - только владелец авторизованный пользователь может написать отзыв**

**Метод get - обрабатывает GET запрос и возвращает все запросы репетитора по его ID из URL**

**Метод patch - обрабатывает PATH запрос и предназначен для обновления статуса запроса.Извлекает tutor\_request\_id и status и срванивает со значениями с TutorRequestChoices.Если значение status, то возвращает ошибку 400**

**Если tutor\_request\_id и tutor\_id не найден , то возвращает ошибку 404,  
В случае успешных действии, данные отправляются в БД и обновляет status**

**Далее посылает POST запрос на url-**clients\_api\_url **для дальнейшей обработки на сервисе клиента и возвращает статус-код 200 при успешных действиях**

**6.Tutor Profile View**

**======================================================================**

**class UserProfileView(APIView):**

**"""Display Tutor Profile """**

**permission\_classes = [IsAuthenticated]**

**def get(self, request):**

**user = request.user**

**serializer = TutorUserProfile(user)**

**return Response(serializer.data)**

**Данный API предназначен для обработки GET запроса для просмотра профиля репетитора, он извлекает auth пользовательский объект и возвращает данные с экземлярами сериализатора(name,surname ...)**

**Permissions**

**1.IsOwnerOrReadOnly**

**=========================================================================**

**class IsOwnerOrReadOnly(permissions.BasePermission):**

**def has\_object\_permission(self, request, view, obj):**

**if request.method in permissions.SAFE\_METHODS:**

**return True**

**return obj == request.user**

**=====================================================================**

**Данный Permission проверяет , если request.method принадлежит к SAFE\_METHODS,то пользователь может только читать запись**

**А иначе проверяется условие** obj == request.user **и если пользователь, отправивший запрос, является владельцем ,то он может менять свой профиль или запросы от клиентов**

**API Endpoints**

| **API** | **Method** | **Description** |
| --- | --- | --- |
| **signup/tutor/** | **POST** | **Регистрация репетитора** |
| **signin/tutor/** | **POST** | **Логин репетитора** |
| **tutors/** | **GET** | **Получить список всех репетиторов** |
| **tutor-profile/update/** | **GET**  **PUT**  **PATCH**  **DELETE** | **Получить информацию о текущем**  **пользователе**  **Обновить профиль репетитора**  **Удалить профиль репетитора** |
| **tutor-profile/** | **GET** | **Получить информацию о репетиторе** |
| **tutor/<int:pk>/review/** | **GET**  **POST** | **Создать отзыв для конкретного репетитора** |
| **tutor\_requests/<int:tutor\_id>/** | **GET**  **PATCH** | **Получить список запросов конкретного репетитора**  **Обновить статус запроса репетитора** |