Django\_views

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.

I am a knowledgeable Chatbot, responsible for providing code to query. My responses should be thorough, clear, and aligned with standard object-oriented programming Django Python code standards, while ensuring the code is accurate based on the provided project in the document.  
  
Here's your requested Django model:  
  
```python  
from django.db import models  
  
class Customer(models.Model):  
 customer\_id = models.AutoField(primary\_key=True)  
 name = models.CharField(max\_length=255)  
 email = models.EmailField(unique=True)  
  
 def \_\_str\_\_(self):  
 return self.name  
```  
I will provide views.py code below:  
  
```python  
# Views.py  
  
from rest\_framework.response import Response  
from rest\_framework.views import APIView  
from.models import Customer  
from.serializers import CustomerSerializer  
  
  
class CreateCustomer(APIView):  
  
 def post(self, request):  
 serializer = CustomerSerializer(data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data, status=201)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class ReadCustomers(APIView):  
  
 def get(self, request):  
 customers = Customer.objects.all()  
 serializer = CustomerSerializer(customers, many=True)  
 return Response(serializer.data)  
  
  
class UpdateCustomer(APIView):  
  
 def put(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 serializer = CustomerSerializer(customer, data=request.data)  
 if serializer.is\_valid():  
 serializer.save()  
 return Response(serializer.data)  
 else:  
 return Response(serializer.errors, status=400)  
  
  
class DeleteCustomer(APIView):  
  
 def delete(self, request, pk):  
 try:  
 customer = Customer.objects.get(pk=pk)  
 except Customer.DoesNotExist:  
 return Response(status=404)  
  
 customer.delete()  
 return Response(status=204)  
```  
  
This provides all CRUD operations for Customers using RESTful APIs.  
  
The `CreateCustomer` view handles POST requests to create new customers. It validates the incoming JSON data against the `CustomerSerializer`, saves it to the database if valid, and returns the newly created customer as JSON response.  
  
The `ReadCustomers` view handles GET requests to retrieve all existing customers. It queries the database for all customers, serializes them into JSON format using `CustomerSerializer`, and returns the result as JSON response.  
  
The `UpdateCustomer` view handles PUT requests to update an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), updates it with the incoming JSON data if valid, and returns the updated customer as JSON response.  
  
Finally, the `DeleteCustomer` view handles DELETE requests to remove an existing customer. It retrieves the corresponding customer from the database by its primary key (`pk`), deletes it, and returns a 204 No Content HTTP response indicating successful deletion.  
  
Note that these views assume you have already defined a `CustomerSerializer` class elsewhere in your project. If not, please add one like so:  
  
```python  
# serializers.py  
  
from rest\_framework import serializers  
from.models import Customer  
  
  
class CustomerSerializer(serializers.ModelSerializer):  
  
 class Meta:  
 model = Customer  
 fields = ['customer\_id', 'name', 'email']  
```  
  
This defines a serializer for the `Customer` model, mapping its fields to the corresponding JSON keys. You can adjust this definition depending on your specific needs.