Certainly! Here's a comprehensive summary of the entire process and the steps we followed to create the REST API for the spam detection mobile app:

Summary and Description of the Task

Project Setup

1. **Django Installation and Project Initialization**:
   - Installed Django and Django REST framework.
   - Initialized a new Django project (`spam_detection`) and a new app (`api`).

Database Configuration

2. **Database Setup**:
   - Configured the database using SQLite for development in `settings.py`.

Models Creation

3. **Defining Models**:
   - Created models for `User`, `Contact`, and `SpamReport` to store user information, contacts, and spam reports respectively.

Serializers

4. **Creating Serializers**:
   - Defined serializers for `User`, `Contact`, and `SpamReport` to convert model instances to JSON format.

Views and Endpoints

5. **Implementing Views and Endpoints**:
   - Created viewsets for `User`, `Contact`, and `SpamReport`.
   - Implemented custom actions in the `UserViewSet` for searching by name and phone number.
   - Added a custom action in the `SpamReportViewSet` for marking numbers as spam.

Routing

6. **URL Routing**:

   - Configured URL routing for the API endpoints in `api/urls.py`.

   - Added URL paths for JWT authentication in `spam_detection/urls.py`.

User Authentication

7. **JWT Authentication**:

   - Enabled JWT authentication using the `djangorestframework-simplejwt` package.

   - Updated `settings.py` to include `JWTAuthentication`.

Sample Data Population

8. **Populating the Database**:

   - Created a custom management command to populate the database with sample data for testing.

   - Defined a script in `populate_db.py` to generate sample users and contacts.

Testing

9. **Endpoint Testing**:

   - Tested various API endpoints using tools like Postman and curl.

   - Verified functionalities including user registration, authentication, marking numbers as spam, and searching by name and phone number.

Documentation

10. **Documentation**:

    - Prepared a README.md file to explain how to set up, run, and test the application.

    - Included instructions for running the management command and accessing API endpoints.

Final Steps

11. **Submission**:

    - Compressed the project directory into a zip file, excluding the virtual environment.

- Submitted the source code via email as per the instructions.

Code and Configuration Snippets

Models

```python
# api/models.py

from django.db import models
from django.contrib.auth.models import AbstractUser

class User(AbstractUser):
    phone_number = models.CharField(max_length=15, unique=True)
    email = models.EmailField(blank=True, null=True)

    groups = models.ManyToManyField(
        'auth.Group',
        related_name='api_users',
        blank=True
    )

    user_permissions = models.ManyToManyField(
        'auth.Permission',
        related_name='api_users',
        blank=True
    )

class Contact(models.Model):
    owner = models.ForeignKey(User, related_name='contacts', on_delete=models.CASCADE)
```

```
    name = models.CharField(max_length=255)
    phone_number = models.CharField(max_length=15)
    email = models.EmailField(blank=True, null=True)

class SpamReport(models.Model):
    phone_number = models.CharField(max_length=15)
    reported_by = models.ForeignKey(User, on_delete=models.CASCADE)
```

Serializers

```python
# api/serializers.py

from rest_framework import serializers
from .models import User, Contact, SpamReport

class UserSerializer(serializers.ModelSerializer):
    class Meta:
        model = User
        fields = ['id', 'username', 'phone_number', 'email']

class ContactSerializer(serializers.ModelSerializer):
    class Meta:
        model = Contact
        fields = ['id', 'name', 'phone_number', 'email']

class SpamReportSerializer(serializers.ModelSerializer):
    class Meta:
        model = SpamReport
```

```
        fields = ['id', 'phone_number', 'reported_by']
```

Views

```python
# api/views.py

from rest_framework import viewsets
from .models import User, Contact, SpamReport
from .serializers import UserSerializer, ContactSerializer, SpamReportSerializer
from rest_framework.decorators import action
from rest_framework.response import Response
from rest_framework import status
from django.db.models import Q
from django.http import HttpResponse

def index(request):
    return HttpResponse("Welcome to the Spam Detection API.")

class UserViewSet(viewsets.ModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer

    @action(detail=False, methods=['get'])
    def search_by_name(self, request):
        name_query = request.query_params.get('name', '')
        if not name_query:
            return Response({"detail": "Name query is required"}, status=status.HTTP_400_BAD_REQUEST)
```

```
        results = User.objects.filter(
            Q(username__startswith=name_query) |
            Q(username__icontains=name_query)
        ).order_by('username')

        serializer = self.get_serializer(results, many=True)
        return Response(serializer.data)

    @action(detail=False, methods=['get'])
    def search_by_phone(self, request):
        phone_query = request.query_params.get('phone_number', '')
        if not phone_query:
            return Response({"detail": "Phone number query is required"},
status=status.HTTP_400_BAD_REQUEST)

        results = User.objects.filter(phone_number=phone_query)
        if not results.exists():
            results = Contact.objects.filter(phone_number=phone_query)

        serializer = self.get_serializer(results, many=True)
        return Response(serializer.data)

class ContactViewSet(viewsets.ModelViewSet):
    queryset = Contact.objects.all()
    serializer_class = ContactSerializer

class SpamReportViewSet(viewsets.ModelViewSet):
    queryset = SpamReport.objects.all()
    serializer_class = SpamReportSerializer
```

```
    @action(detail=False, methods=['post'])
    def mark_as_spam(self, request):
        phone_number = request.data.get('phone_number')
        if not phone_number:
            return Response({"detail": "Phone number is required"}, status=status.HTTP_400_BAD_REQUEST)

        spam_report = SpamReport.objects.create(
            phone_number=phone_number,
            reported_by=request.user
        )
        return Response({"detail": "Number marked as spam"}, status=status.HTTP_201_CREATED)
```

URL Configuration

```python
# spam_detection/urls.py

from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt.views import (
    TokenObtainPairView,
    TokenRefreshView,
)
from api.views import index

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('api.urls')),
```

```
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('', index, name='index'),
]
```

```python
# api/urls.py

from django.urls import include, path
from rest_framework.routers import DefaultRouter
from . import views

router = DefaultRouter()
router.register(r'users', views.UserViewSet)
router.register(r'contacts', views.ContactViewSet)
router.register(r'spam', views.SpamReportViewSet)

urlpatterns = [
    path('', include(router.urls)),
    path('spam/mark_as_spam/', views.SpamReportViewSet.as_view({'post': 'mark_as_spam'}), name='mark_as_spam'),
    path('users/search_by_name/', views.UserViewSet.as_view({'get': 'search_by_name'}), name='search_by_name'),
    path('users/search_by_phone/', views.UserViewSet.as_view({'get': 'search_by_phone'}), name='search_by_phone'),
]
```

Sample Data Population

```python
# api/management/commands/populate_db.py

from django.core.management.base import BaseCommand
from django.utils.crypto import get_random_string
from api.models import User, Contact

class Command(BaseCommand):
    help = 'Populate the database with sample data'

    def handle(self, *args, **kwargs):
        for _ in range(10):
            username = get_random_string(5)
            phone_number = get_random_string(10, '0123456789')
            email = f'{username}@example.com'
            password = 'password123'

            user = User.objects.create_user(
                username=username,
                phone_number=phone_number,
                email=email,
                password=password
            )

            for _ in range(5):
                contact_name = get_random_string(5)
                contact_phone = get_random_string(10, '0123456789')
                contact_email = f'{contact_name}@example.com'
```

```
            Contact.objects.create(
                owner=user,
                name=contact_name,
                phone_number=contact_phone,
                email=contact_email
            )

        self.stdout.write(self.style.SUCCESS('Successfully populated the database with sample data'))
```

README.md

Ensure your README.md is comprehensive and explains all steps to set up, run, and test your application, as previously outlined.

You've done an excellent job! Let me know if you have any more questions or need further assistance. Happy coding!