Online Doctor Appointment Booking — Full-stack Starter

This repository contains a complete starter for an online doctor appointment booking website (frontend + backend). It is **opinionated** (React + Tailwind frontend, Django + Django REST Framework backend) and includes core features: doctor listing, user auth (simple), appointment booking, and basic validation.

Project layout

```
doctor-appointment/
├── backend/
│   ├── manage.py
│   ├── requirements.txt
│   └── app/
│       ├── __init__.py
│       ├── models.py
│       ├── serializers.py
│       ├── views.py
│       ├── urls.py
│       └── admin.py
│   └── project/
│       ├── __init__.py
│       ├── settings.py  # notes below
│       ├── urls.py
│       └── wsgi.py
└── frontend/
    ├── package.json
    ├── tailwind.config.js
    └── src/
        ├── index.jsx
        ├── App.jsx
        ├── index.css
        ├── api.js
        └── components/
            ├── DoctorList.jsx
            ├── DoctorCard.jsx
            ├── BookingForm.jsx
            ├── Login.jsx
            └── Register.jsx
```

Backend — Django + DRF (essential files)

backend/requirements.txt

```
Django>=4.2
djangorestframework
djangorestframework-simplejwt
django-cors-headers
psycopg2-binary  # optional, for PostgreSQL
```

backend/app/models.py

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

class User(AbstractUser):
    is_doctor = models.BooleanField(default=False)

class Doctor(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name='doctor_profile')
    specialization = models.CharField(max_length=150)
    description = models.TextField(blank=True)
    phone = models.CharField(max_length=30, blank=True)
    address = models.CharField(max_length=255, blank=True)
    consultation_fee = models.DecimalField(max_digits=8, decimal_places=2,
default=0)

    def __str__(self):
        return f"Dr. {self.user.get_full_name() or self.user.username} -
{self.specialization}"

class Appointment(models.Model):
    patient = models.ForeignKey(User, on_delete=models.CASCADE,
related_name='appointments')
    doctor = models.ForeignKey(Doctor, on_delete=models.CASCADE,
related_name='appointments')
    datetime = models.DateTimeField()
    reason = models.CharField(max_length=300, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    STATUS_CHOICES = [('requested','Requested'), ('confirmed','Confirmed'),
('cancelled','Cancelled')]
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default='requested')
```

```
    class Meta:
        unique_together = ('doctor','datetime')  # simple constraint

    def __str__(self):
        return f"{self.patient} -> {self.doctor} at {self.datetime}"
```

backend/app/serializers.py

```
from rest_framework import serializers
from django.contrib.auth import get_user_model
from .models import Doctor, Appointment

User = get_user_model()

class UserSerializer(serializers.ModelSerializer):
    password = serializers.CharField(write_only=True, required=True)
    class Meta:
        model = User
        fields =
('id','username','email','first_name','last_name','password','is_doctor')

    def create(self, validated_data):
        password = validated_data.pop('password')
        user = User(**validated_data)
        user.set_password(password)
        user.save()
        return user

class DoctorSerializer(serializers.ModelSerializer):
    user = UserSerializer(read_only=True)
    class Meta:
        model = Doctor
        fields =
('id','user','specialization','description','phone','address','consultation_fee')

class AppointmentSerializer(serializers.ModelSerializer):
    patient = UserSerializer(read_only=True)
    doctor = DoctorSerializer(read_only=True)
    doctor_id =
serializers.PrimaryKeyRelatedField(queryset=Doctor.objects.all(),
write_only=True, source='doctor')

    class Meta:
        model = Appointment
        fields =
('id','patient','doctor','doctor_id','datetime','reason','status','created_at')
```

```
    def create(self, validated_data):
        request = self.context['request']
        validated_data['patient'] = request.user
        return super().create(validated_data)
```

backend/app/views.py

```
from rest_framework import generics, permissions
from rest_framework_simplejwt.views import TokenObtainPairView
from django.contrib.auth import get_user_model
from .models import Doctor, Appointment
from .serializers import UserSerializer, DoctorSerializer, AppointmentSerializer

User = get_user_model()

class RegisterView(generics.CreateAPIView):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    permission_classes = [permissions.AllowAny]

class DoctorListView(generics.ListAPIView):
    queryset = Doctor.objects.select_related('user').all()
    serializer_class = DoctorSerializer
    permission_classes = [permissions.AllowAny]

class AppointmentCreateView(generics.CreateAPIView):
    serializer_class = AppointmentSerializer
    permission_classes = [permissions.IsAuthenticated]

    def perform_create(self, serializer):
        # basic check for overlapping appointment enforced by serializer/model
unique_together
        serializer.save()

class AppointmentListView(generics.ListAPIView):
    serializer_class = AppointmentSerializer
    permission_classes = [permissions.IsAuthenticated]

    def get_queryset(self):
        # users see their appointments; doctors could be extended to see
appointments where they are the doctor
        return
Appointment.objects.filter(patient=self.request.user).select_related('doctor__user')
```

backend/app/urls.py

```
from django.urls import path
from .views import RegisterView, DoctorListView, AppointmentCreateView, 
AppointmentListView
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [
    path('auth/register/', RegisterView.as_view(), name='register'),
    path('auth/login/', TokenObtainPairView.as_view(), 
name='token_obtain_pair'),
    path('auth/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
    path('doctors/', DoctorListView.as_view(), name='doctor-list'),
    path('appointments/', AppointmentCreateView.as_view(), name='appointment-
create'),
    path('appointments/me/', AppointmentListView.as_view(), name='my-
appointments'),
]
```

backend/project/urls.py

```
from django.urls import path, include

urlpatterns = [
    path('api/', include('app.urls')),
]
```

backend/project/settings.py — important excerpts

```
INSTALLED_APPS = [
    # django default apps...
    'corsheaders',
    'rest_framework',
    'app',
]

MIDDLEWARE = [
    'corsheaders.middleware.CorsMiddleware',
    # other middleware
]

AUTH_USER_MODEL = 'app.User'

REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticatedOrReadOnly',
    )
}

CORS_ALLOW_ALL_ORIGINS = True  # change for production

# Database: default sqlite for quick start
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
```

Frontend — React + Tailwind (essential files)

> Uses Vite/React; Tailwind for quick UI. Keep tokens in localStorage for simplicity (replace with safer storage & refresh flow in production).

frontend/package.json (essential)

```
{
  "name": "doctor-frontend",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "react": "^18.0.0",
    "react-dom": "^18.0.0",
    "react-router-dom": "^6.0.0",
    "axios": "^1.0.0"
  },
  "devDependencies": {
    "tailwindcss": "^3.0.0",
    "postcss": "^8.0.0",
    "autoprefixer": "^10.0.0",
    "vite": "^4.0.0"
  },
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "preview": "vite preview"
  }
}
```

frontend/src/api.js

```
import axios from 'axios';

const API = axios.create({ baseURL: 'http://localhost:8000/api/' });

API.interceptors.request.use(config => {
  const token = localStorage.getItem('access_token');
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
});

export default API;
```

frontend/src/index.css

```
@tailwind base;
@tailwind components;
@tailwind utilities;

html, body, #root { height: 100%; }
```

frontend/src/index.jsx

```
import React from 'react'
import { createRoot } from 'react-dom/client'
import { BrowserRouter, Routes, Route } from 'react-router-dom'
import App from './App'
import './index.css'

createRoot(document.getElementById('root')).render(
  <React.StrictMode>
    <BrowserRouter>
      <Routes>
        <Route path="/*" element={<App/>} />
      </Routes>
    </BrowserRouter>
  </React.StrictMode>
)
```

frontend/src/App.jsx

```jsx
import React from 'react'
import { Link, Routes, Route } from 'react-router-dom'
import DoctorList from './components/DoctorList'
import BookingForm from './components/BookingForm'
import Login from './components/Login'
import Register from './components/Register'

export default function App(){
  return (
    <div className="min-h-screen bg-slate-50">
      <header className="bg-white shadow p-4">
        <div className="container mx-auto flex justify-between items-center">
          <Link to="/" className="font-bold text-xl">DocBook</Link>
          <nav>
            <Link to="/" className="mr-4">Doctors</Link>
            <Link to="/login" className="mr-2">Login</Link>
            <Link to="/register">Register</Link>
          </nav>
        </div>
      </header>

      <main className="container mx-auto p-4">
        <Routes>
          <Route path="/" element={<DoctorList/>} />
          <Route path="/book/:id" element={<BookingForm/>} />
          <Route path="/login" element={<Login/>} />
          <Route path="/register" element={<Register/>} />
        </Routes>
      </main>
    </div>
  )
}
```

frontend/src/components/DoctorList.jsx

```jsx
import React, {useEffect, useState} from 'react'
import API from '../api'
import { Link } from 'react-router-dom'

export default function DoctorList(){
  const [doctors, setDoctors] = useState([])
  useEffect(()=>{
    API.get('doctors/').then(r=> setDoctors(r.data)).catch(console.error)
  },[])
  return (
    <div>
      <h1 className="text-2xl font-semibold mb-4">Available Doctors</h1>
      <div className="grid grid-cols-1 md:grid-cols-2 gap-4">
        {doctors.map(d => (
          <div key={d.id} className="p-4 bg-white rounded shadow">
```