# Customer Web Portal - Django Backend

A Django REST Framework backend for managing secure vehicle gate entry submissions with automatic data prefill, document management, and QR code generation.

## Table of Contents

---

## Features

✅ **Automatic Data Prefill** - Auto-populate driver, helper, and document data by vehicle registration number
✅ **Phone Number Validation** - Unique phone constraints with duplicate detection
✅ **Document Management** - Upload, replace, and track customer documents
✅ **QR Code Generation** - Generate secure entry QR codes with vehicle/driver data
✅ **Email & SMS Notifications** - Send QR codes via email and SMS
✅ **Audit Logging** - Track all submissions and document updates
✅ **JWT Authentication** - Secure API with token-based auth

---

## System Architecture

```
┌─────────────────────┐
│   React Frontend    │
│    (Port 3000)      │
```

```
└──────────┬──────────┘
           │ HTTP/REST
           ▼
┌─────────────────────┐
│   Django Backend    │
│    (Port 8000)      │
└──────────┬──────────┘
           │
    ┌──────┴──────┬──────────────┐
    ▼             ▼              ▼
┌──────────┐ ┌──────────┐ ┌──────────┐
│ Postgres │ │  Media   │ │    QR    │
│    DB    │ │ Storage  │ │Generator │
└──────────┘ └──────────┘ └──────────┘
```

---

## Prerequisites

- **Python**: 3.9+
- **PostgreSQL**: 12+
- **pip**: Latest version
- **virtualenv**: For isolated environments

---

## Project Setup

### 1. Clone and Create Project Structure

```bash
# Create project directory
mkdir customer-portal-backend
cd customer-portal-backend

# Create virtual environment
python -m venv venv

# Activate virtual environment
# On Windows:
venv\Scripts\activate
# On macOS/Linux:
source venv/bin/activate
```

## 2. Install Dependencies

```bash
# Install core packages
pip install django==4.2
pip install djangorestframework==3.14.0
pip install psycopg2-binary==2.9.9
pip install djangorestframework-simplejwt==5.3.0
pip install python-decouple==3.8
pip install Pillow==10.1.0
pip install qrcode[pil]==7.4.2
pip install python-dotenv==1.0.0
pip install django-cors-headers==4.3.0

# For testing
pip install pytest==7.4.3
pip install pytest-django==4.7.0

# Save dependencies
pip freeze > requirements.txt
```

## 3. Create Django Project

```bash
# Create Django project
django-admin startproject customer_portal .

# Create modular apps
python manage.py startapp vehicles
python manage.py startapp drivers
python manage.py startapp documents
python manage.py startapp submissions
python manage.py startapp authentication
```

## 4. Project Folder Structure

```
customer-portal-backend/
├── customer_portal/
│   ├── __init__.py
│   ├── settings.py
│   ├── urls.py
│   ├── wsgi.py
│   └── asgi.py
├── vehicles/
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   └── urls.py
├── drivers/
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   └── urls.py
├── documents/
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   └── urls.py
├── submissions/
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   ├── qr_generator.py
│   └── urls.py
├── authentication/
│   ├── models.py
│   ├── serializers.py
│   ├── views.py
│   └── urls.py
├── media/
│   └── documents/
├── .env
├── manage.py
├── requirements.txt
└── README.md
```

---

## Database Setup

### 1. Install PostgreSQL

### Ubuntu/Debian:

```bash
sudo apt update
sudo apt install postgresql postgresql-contrib
```

### macOS:

```bash
```

```
brew install postgresql
brew services start postgresql
```

**Windows:**
Download installer from postgresql.org

### 2. Create Database and User

```bash
# Access PostgreSQL
sudo -u postgres psql

# Create database
CREATE DATABASE customer_portal_db;

# Create user
CREATE USER portal_admin WITH PASSWORD 'your_secure_password';

# Grant privileges
GRANT ALL PRIVILEGES ON DATABASE customer_portal_db TO portal_admin;

# Exit
\q
```

### 3. Configure Environment Variables

Create `.env` file in project root:

```env
```

```
# Database Configuration
DB_NAME=customer_portal_db
DB_USER=portal_admin
DB_PASSWORD=your_secure_password
DB_HOST=localhost
DB_PORT=5432

# Django Settings
SECRET_KEY=your-secret-key-here-generate-with-django
DEBUG=True
ALLOWED_HOSTS=localhost,127.0.0.1

# JWT Settings
JWT_SECRET_KEY=your-jwt-secret-key
JWT_ACCESS_TOKEN_LIFETIME=60
JWT_REFRESH_TOKEN_LIFETIME=1440

# Email Configuration (for QR delivery)
EMAIL_BACKEND=django.core.mail.backends.smtp.EmailBackend
EMAIL_HOST=smtp.gmail.com
EMAIL_PORT=587
EMAIL_USE_TLS=True
EMAIL_HOST_USER=your-email@gmail.com
EMAIL_HOST_PASSWORD=your-app-password

# SMS Configuration (placeholder)
SMS_API_KEY=your-sms-provider-api-key
SMS_API_URL=https://api.smsprovider.com/send

# Media Files
MEDIA_ROOT=media/
MEDIA_URL=/media/
```

### 4. Update Django Settings

### customer_portal/settings.py:

```
python
```

```python
from pathlib import Path
from decouple import config
from datetime import timedelta

BASE_DIR = Path(__file__).resolve().parent.parent

SECRET_KEY = config('SECRET_KEY')
DEBUG = config('DEBUG', default=False, cast=bool)
ALLOWED_HOSTS = config('ALLOWED_HOSTS', default='').split(',')

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',

    # Third-party apps
    'rest_framework',
    'rest_framework_simplejwt',
    'corsheaders',

    # Local apps
    'vehicles',
    'drivers',
    'documents',
    'submissions',
    'authentication',
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'corsheaders.middleware.CorsMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'customer_portal.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.postgresql',
        'NAME': config('DB_NAME'),
        'USER': config('DB_USER'),
        'PASSWORD': config('DB_PASSWORD'),
        'HOST': config('DB_HOST', default='localhost'),
        'PORT': config('DB_PORT', default='5432'),
    }
}

# REST Framework Configuration
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': [
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ],
    'DEFAULT_PERMISSION_CLASSES': [
        'rest_framework.permissions.IsAuthenticated',
    ],
    'DEFAULT_PAGINATION_CLASS': 'rest_framework.pagination.PageNumberPagination',
    'PAGE_SIZE': 50,
}

# JWT Configuration
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=config('JWT_ACCESS_TOKEN_LIFETIME', default=60, cast=int)),
    'REFRESH_TOKEN_LIFETIME': timedelta(minutes=config('JWT_REFRESH_TOKEN_LIFETIME', default=1440, cast=
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
}

# CORS Configuration
CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://127.0.0.1:3000",
]
CORS_ALLOW_CREDENTIALS = True

# Media Files Configuration
MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'

# Email Configuration
EMAIL_BACKEND = config('EMAIL_BACKEND')
EMAIL_HOST = config('EMAIL_HOST')
EMAIL_PORT = config('EMAIL_PORT', cast=int)
EMAIL_USE_TLS = config('EMAIL_USE_TLS', cast=bool)
EMAIL_HOST_USER = config('EMAIL_HOST_USER')
EMAIL_HOST_PASSWORD = config('EMAIL_HOST_PASSWORD')

# Static files
STATIC_URL = '/static/'
STATIC_ROOT = BASE_DIR / 'staticfiles'

# Default primary key field type
DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

### 5. Apply Migrations

```bash
python manage.py makemigrations
python manage.py migrate
```

---

## Django Models

Based on the provided `database_schema.sql` and workflow requirements, we extract and convert ONLY the relevant tables.

### vehicles/models.py

```python
```

```
from django.db import models
from django.core.validators import RegexValidator

class VehicleDetails(models.Model):
    """
    Vehicle registration and tracking information
    """
    vehicle_registration_no = models.CharField(
        max_length=50,
        unique=True,
        validators=[
            RegexValidator(
                regex=r'^[A-Z0-9\s\-]+$',
                message='Vehicle number must contain only uppercase letters, numbers, spaces, or hyphens'
            )
        ]
    )
    remark = models.TextField(blank=True, null=True)
    ratings = models.IntegerField(default=0, blank=True, null=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'VehicleDetails'
        verbose_name = 'Vehicle Detail'
        verbose_name_plural = 'Vehicle Details'
        ordering = ['-created']

    def __str__(self):
        return self.vehicle_registration_no
```

### drivers/models.py

```
python
```

```python
from django.db import models
from django.core.validators import RegexValidator
from django.core.exceptions import ValidationError

DRIVER_TYPES = (
    ('Driver', 'Driver'),
    ('Helper', 'Helper'),
)

LANGUAGE_CHOICES = (
    ('en', 'English'),
    ('hi', 'Hindi'),
    ('mr', 'Marathi'),
    ('gu', 'Gujarati'),
    ('ta', 'Tamil'),
)

class DriverHelper(models.Model):
    """
    Driver and Helper information with unique phone validation
    """
    name = models.CharField(max_length=100)
    type = models.CharField(max_length=10, choices=DRIVER_TYPES)
    phone_no = models.CharField(
        max_length=15,
        unique=True,
        validators=[
            RegexValidator(
                regex=r'^\+91\d{10}$',
                message='Phone number must be in format: +91XXXXXXXXXX'
            )
        ]
    )
    language = models.CharField(max_length=5, choices=LANGUAGE_CHOICES, default='en')
    is_blacklisted = models.BooleanField(default=False)
    rating = models.IntegerField(default=0, blank=True, null=True)
    created = models.DateTimeField(auto_now_add=True)
    updated = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'DriverHelper'
        verbose_name = 'Driver/Helper'
        verbose_name_plural = 'Drivers/Helpers'
        ordering = ['-created']
        indexes = [
            models.Index(fields=['phone_no']),
            models.Index(fields=['type']),
        ]

    def __str__(self):
        return f"{self.name} ({self.type}) - {self.phone_no}"

    def clean(self):
        """
        Validation: If phone exists but name mismatch → raise error
        """
        if self.phone_no:
            existing = DriverHelper.objects.filter(phone_no=self.phone_no).exclude(pk=self.pk).first()
            if existing and existing.name.lower() != self.name.lower():
                raise ValidationError(
                    f"Phone number {self.phone_no} is already registered with a different name: {existing.name}"
                )

    @classmethod
    def validate_or_create(cls, name, phone_no, driver_type, language='en'):
        """
        Workflow validation logic:
        - If phone exists and name matches → return existing
        - If phone exists but name mismatch → raise error
        - If phone doesn't exist → create new
        """
        try:
            existing = cls.objects.get(phone_no=phone_no)
            if existing.name.lower() == name.lower():
                # Update language if changed
                if existing.language != language:
                    existing.language = language
                    existing.save()
                return existing, False  # (instance, created)
            else:
                raise ValidationError(
                    f"Phone number {phone_no} is already registered with name '{existing.name}'. "
                    f"Cannot register as '{name}'."
                )
        except cls.DoesNotExist:
            # Create new driver/helper
            instance = cls.objects.create(
                name=name,
                phone_no=phone_no,
                type=driver_type,
                language=language
        )
        return instance, True  # (instance, created)
```

**documents/models.py**

```
python
```

```
from django.db import models
from vehicles.models import VehicleDetails
from drivers.models import DriverHelper
from django.conf import settings
import os
import shutil
from datetime import datetime

DOCUMENT_TYPES = (
    ('purchase_order', 'Purchase Order'),
    ('vehicle_registration', 'Vehicle Registration'),
    ('vehicle_insurance', 'Vehicle Insurance'),
    ('puc', 'PUC'),
    ('driver_license', 'Driver License'),
    ('transportation_approval', 'Transportation Approval'),
    ('payment_approval', 'Payment Approval'),
    ('vendor_approval', 'Vendor Approval'),
)

class CustomerDocument(models.Model):
    """
    Customer document uploads with file path stored in database
    Files are stored in local computer/server storage
    """
    customer_email = models.EmailField(db_index=True)
    document_type = models.CharField(max_length=50, choices=DOCUMENT_TYPES)

    # Store absolute file path in database (not FileField)
    file_path = models.CharField(max_length=500, help_text="Absolute path to document file on storage")
    original_filename = models.CharField(max_length=255)
    file_size = models.BigIntegerField(help_text="File size in bytes")
    file_extension = models.CharField(max_length=10)

    vehicle = models.ForeignKey(
        VehicleDetails,
        on_delete=models.CASCADE,
        related_name='documents',
        null=True,
        blank=True
    )
    driver = models.ForeignKey(
        DriverHelper,
        on_delete=models.SET_NULL,
        related_name='driver_documents',
        null=True,
        blank=True
```

```python
)
uploaded_at = models.DateTimeField(auto_now_add=True)
updated_at = models.DateTimeField(auto_now=True)
is_active = models.BooleanField(default=True)  # For soft delete
replaced_by = models.ForeignKey(
    'self',
    on_delete=models.SET_NULL,
    null=True,
    blank=True,
    related_name='replaces'
)

class Meta:
    db_table = 'CustomerDocument'
    verbose_name = 'Customer Document'
    verbose_name_plural = 'Customer Documents'
    ordering = ['-uploaded_at']
    indexes = [
        models.Index(fields=['customer_email', 'document_type']),
        models.Index(fields=['is_active']),
    ]

def __str__(self):
    return f"{self.customer_email} - {self.get_document_type_display()}"

@staticmethod
def get_storage_path(customer_email, document_type):
    """
    Generate storage directory path for documents
    Path: /path/to/storage/documents/{customer_email}/{document_type}/
    """
    # Get base storage directory from settings
    base_storage = getattr(settings, 'DOCUMENT_STORAGE_PATH', '/var/customer_portal/documents/')

    # Sanitize email for folder name
    email_folder = customer_email.replace('@', '_at_').replace('.', '_')

    # Create full path
    storage_path = os.path.join(base_storage, 'documents', email_folder, document_type)

    # Create directory if it doesn't exist
    os.makedirs(storage_path, exist_ok=True)

    return storage_path

@classmethod
def save_file_to_storage(cls, uploaded_file, customer_email, document_type):
    """
    Save uploaded file to local storage and return file path

    Args:
        uploaded_file: Django UploadedFile object
        customer_email: Customer email
        document_type: Type of document

    Returns:
        dict: Contains file_path, original_filename, file_size, file_extension
    """
    # Get storage directory
    storage_dir = cls.get_storage_path(customer_email, document_type)

    # Generate unique filename with timestamp
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    file_extension = os.path.splitext(uploaded_file.name)[1]
    filename = f"{document_type}_{timestamp}{file_extension}"

    # Full file path
    file_path = os.path.join(storage_dir, filename)

    # Save file to disk
    with open(file_path, 'wb+') as destination:
        for chunk in uploaded_file.chunks():
            destination.write(chunk)

    return {
        'file_path': file_path,
        'original_filename': uploaded_file.name,
        'file_size': uploaded_file.size,
        'file_extension': file_extension
    }

def delete(self, using=None, keep_parents=False, hard_delete=False):
    """
    Soft delete by default, hard delete if specified
    Hard delete removes the physical file from storage
    """
    if hard_delete:
        # Delete physical file from storage
        if self.file_path and os.path.isfile(self.file_path):
            try:
                os.remove(self.file_path)
            except OSError as e:
                print(f"Error deleting file {self.file_path}: {e}")
        super().delete(using=using, keep_parents=keep_parents)
        else:
            # Soft delete - just mark as inactive
            self.is_active = False
            self.save()

    @classmethod
    def replace_document(cls, customer_email, document_type, uploaded_file, vehicle=None, driver=None):
        """
        Replace old document with new one:
        - Save new file to storage
        - Mark old document as inactive
        - Create new document record with file path
        - Link via replaced_by

        Args:
            customer_email: Customer email
            document_type: Type of document
            uploaded_file: Django UploadedFile object
            vehicle: Vehicle instance (optional)
            driver: Driver instance (optional)

        Returns:
            CustomerDocument: New document instance
        """
        # Find existing active document
        old_doc = cls.objects.filter(
            customer_email=customer_email,
            document_type=document_type,
            is_active=True
        ).first()

        # Save file to storage and get file info
        file_info = cls.save_file_to_storage(uploaded_file, customer_email, document_type)

        # Create new document record
        new_doc = cls.objects.create(
            customer_email=customer_email,
            document_type=document_type,
            file_path=file_info['file_path'],
            original_filename=file_info['original_filename'],
            file_size=file_info['file_size'],
            file_extension=file_info['file_extension'],
            vehicle=vehicle,
            driver=driver
        )

        # Mark old document as replaced (soft delete)
        if old_doc:
            old_doc.is_active = False
            old_doc.replaced_by = new_doc
            old_doc.save()

            # Optionally delete old physical file to save space
            # Uncomment the following to delete old files immediately
            # if old_doc.file_path and os.path.isfile(old_doc.file_path):
            #     try:
            #         os.remove(old_doc.file_path)
            #     except OSError:
            #         pass

        return new_doc

    def get_file_url(self, request=None):
        """
        Generate URL to access the file
        In production, this would be served via nginx/Apache
        """
        if not self.is_active:
            return None

        # In development, Django can serve the file
        # In production, configure web server to serve from DOCUMENT_STORAGE_PATH
        if request:
            # Build URL path (requires view to serve files)
            return request.build_absolute_uri(f'/api/documents/{self.id}/download/')
        return None

    def file_exists(self):
        """
        Check if the physical file exists on storage
        """
        return os.path.isfile(self.file_path) if self.file_path else False
```

**submissions/models.py**

python

```
from django.db import models
from vehicles.models import VehicleDetails
from drivers.models import DriverHelper
import hashlib
import json

class GateEntrySubmission(models.Model):
    """
    Main submission record for gate entry
    """
    # Customer info
    customer_email = models.EmailField()
    customer_phone = models.CharField(max_length=15)

    # Vehicle info
    vehicle = models.ForeignKey(
        VehicleDetails,
        on_delete=models.CASCADE,
        related_name='submissions'
    )

    # Driver & Helper
    driver = models.ForeignKey(
        DriverHelper,
        on_delete=models.CASCADE,
        related_name='driver_submissions'
    )
    helper = models.ForeignKey(
        DriverHelper,
        on_delete=models.CASCADE,
        related_name='helper_submissions',
        null=True,
        blank=True
    )

    # QR Code
    qr_code_image = models.ImageField(upload_to='qr_codes/', null=True, blank=True)
    qr_payload_hash = models.CharField(max_length=64, unique=True)  # SHA-256 hash

    # Status
    STATUS_CHOICES = (
        ('pending', 'Pending'),
        ('approved', 'Approved'),
        ('rejected', 'Rejected'),
        ('completed', 'Completed'),
    )
    status = models.CharField(max_length=20, choices=STATUS_CHOICES, default='pending')

    # Timestamps
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        db_table = 'GateEntrySubmission'
        verbose_name = 'Gate Entry Submission'
        verbose_name_plural = 'Gate Entry Submissions'
        ordering = ['-created_at']
        indexes = [
            models.Index(fields=['customer_email']),
            models.Index(fields=['qr_payload_hash']),
            models.Index(fields=['status']),
        ]

    def __str__(self):
        return f"Submission {self.id} - {self.vehicle.vehicle_registration_no}"

    def generate_payload_hash(self):
        """
        Generate SHA-256 hash of QR payload for uniqueness
        """
        payload = {
            'customer_name': self.customer_email.split('@')[0],
            'customer_email': self.customer_email,
            'driver_name': self.driver.name,
            'driver_phone': self.driver.phone_no,
            'helper_name': self.helper.name if self.helper else '',
            'helper_phone': self.helper.phone_no if self.helper else '',
            'vehicle_number': self.vehicle.vehicle_registration_no,
            'timestamp': self.created_at.isoformat() if self.created_at else '',
        }
        payload_str = json.dumps(payload, sort_keys=True)
        return hashlib.sha256(payload_str.encode()).hexdigest()


class AuditLog(models.Model):
    """
    Audit trail for all submission activities
    """
    submission = models.ForeignKey(
        GateEntrySubmission,
        on_delete=models.CASCADE,
        related_name='audit_logs'
    )
```

```python
    action = models.CharField(max_length=100)
    description = models.TextField()
    user_email = models.EmailField(blank=True, null=True)
    ip_address = models.GenericIPAddressField(blank=True, null=True)
    timestamp = models.DateTimeField(auto_now_add=True)

    class Meta:
        db_table = 'AuditLog'
        verbose_name = 'Audit Log'
        verbose_name_plural = 'Audit Logs'
        ordering = ['-timestamp']

    def __str__(self):
        return f"{self.action} - {self.timestamp}"
```

**authentication/models.py**

python

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

class CustomerUser(AbstractUser):
    """
    Extended user model for customers
    """
    email = models.EmailField(unique=True)
    phone = models.CharField(max_length=15, blank=True, null=True)
    company_name = models.CharField(max_length=200, blank=True, null=True)

    # Use email as username
    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['username']

    class Meta:
        db_table = 'CustomerUser'
        verbose_name = 'Customer User'
        verbose_name_plural = 'Customer Users'

    def __str__(self):
        return self.email
```

**Update settings.py:**

python

```
AUTH_USER_MODEL = 'authentication.CustomerUser'
```

---

## API Endpoints

### URL Configuration

### customer_portal/urls.py:

```python
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/auth/', include('authentication.urls')),
    path('api/vehicles/', include('vehicles.urls')),
    path('api/drivers/', include('drivers.urls')),
    path('api/documents/', include('documents.urls')),
    path('api/submissions/', include('submissions.urls')),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

### 1. Vehicle Lookup API

### vehicles/views.py:

```python
```

```
from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.response import Response
from .models import VehicleDetails
from .serializers import VehicleDetailsSerializer
from drivers.models import DriverHelper
from drivers.serializers import DriverHelperSerializer
from documents.models import CustomerDocument
from documents.serializers import CustomerDocumentSerializer

class VehicleViewSet(viewsets.ModelViewSet):
    queryset = VehicleDetails.objects.all()
    serializer_class = VehicleDetailsSerializer
    lookup_field = 'vehicle_registration_no'

    @action(detail=True, methods=['get'], url_path='lookup')
    def lookup_vehicle(self, request, vehicle_registration_no=None):
        """
        Auto-fill workflow: Fetch vehicle, driver, helper, and documents

        GET /api/vehicles/{vehicle_reg_no}/lookup/

        Response:
        {
            "vehicle": {...},
            "driver": {...},
            "helper": {...},
            "documents": [...]
        }
        """
        try:
            vehicle = self.get_object()
        except VehicleDetails.DoesNotExist:
            return Response(
                {"detail": "Vehicle not found"},
                status=status.HTTP_404_NOT_FOUND
            )

        # Get latest submission for this vehicle
        latest_submission = vehicle.submissions.order_by('-created_at').first()

        driver_data = None
        helper_data = None
        documents_data = []

        if latest_submission:
```

```
        # Serialize driver and helper
        if latest_submission.driver:
            driver_data = DriverHelperSerializer(latest_submission.driver).data
        if latest_submission.helper:
            helper_data = DriverHelperSerializer(latest_submission.helper).data

        # Get documents for this customer
        documents = CustomerDocument.objects.filter(
            customer_email=latest_submission.customer_email,
            is_active=True
        )
        documents_data = CustomerDocumentSerializer(documents, many=True).data

    return Response({
        "vehicle": VehicleDetailsSerializer(vehicle).data,
        "driver": driver_data,
        "helper": helper_data,
        "documents": documents_data
    })
```

**vehicles/serializers.py:**

```
python

from rest_framework import serializers
from .models import VehicleDetails

class VehicleDetailsSerializer(serializers.ModelSerializer):
    class Meta:
        model = VehicleDetails
        fields = ['id', 'vehicle_registration_no', 'remark', 'ratings', 'created', 'updated']
        read_only_fields = ['id', 'created', 'updated']
```

**vehicles/urls.py:**

```
python
```

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

router = DefaultRouter()
router.register(r'', VehicleViewSet, basename='vehicle')

urlpatterns = [
    path('', include(router.urls)),
]
```

### 2. Driver/Helper Validation API

#### drivers/views.py:

```
python
```

```python
from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.response import Response
from django.core.exceptions import ValidationError
from .models import DriverHelper
from .serializers import DriverHelperSerializer, DriverHelperValidateSerializer

class DriverHelperViewSet(viewsets.ModelViewSet):
    queryset = DriverHelper.objects.all()
    serializer_class = DriverHelperSerializer

    @action(detail=False, methods=['post'], url_path='validate-or-create')
    def validate_or_create(self, request):
        """
        Validate phone uniqueness and create if needed

        POST /api/drivers/validate-or-create/

        Request:
        {
            "name": "John Doe",
            "phone_no": "+919876543210",
            "type": "Driver",
            "language": "en"
        }

        Response:
        {
            "driver": {...},
            "created": true/false,
            "message": "..."
        }
        """
        serializer = DriverHelperValidateSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)

        name = serializer.validated_data['name']
        phone_no = serializer.validated_data['phone_no']
        driver_type = serializer.validated_data['type']
        language = serializer.validated_data.get('language', 'en')

        try:
            instance, created = DriverHelper.validate_or_create(
                name=name,
                phone_no=phone_no,
                driver_type=driver_type,
                language=language
            )

            message = "New driver/helper created" if created else "Existing driver/helper found"

            return Response({
                "driver": DriverHelperSerializer(instance).data,
                "created": created,
                "message": message
            }, status=status.HTTP_201_CREATED if created else status.HTTP_200_OK)

        except ValidationError as e:
            return Response({
                "error": str(e.message)
            }, status=status.HTTP_400_BAD_REQUEST)
```

**drivers/serializers.py:**

python

```python
from rest_framework import serializers
from .models import DriverHelper

class DriverHelperSerializer(serializers.ModelSerializer):
    class Meta:
        model = DriverHelper
        fields = ['id', 'name', 'type', 'phone_no', 'language', 'is_blacklisted', 'rating', 'created']
        read_only_fields = ['id', 'created']

class DriverHelperValidateSerializer(serializers.Serializer):
    name = serializers.CharField(max_length=100)
    phone_no = serializers.RegexField(
        regex=r'^\+91\d{10}$',
        error_messages={'invalid': 'Phone number must be in format: +91XXXXXXXXXX'}
    )
    type = serializers.ChoiceField(choices=['Driver', 'Helper'])
    language = serializers.CharField(default='en')
```

**drivers/urls.py:**

python

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

router = DefaultRouter()
router.register(r'', DriverHelperViewSet, basename='driver-helper')

urlpatterns = [
    path('', include(router.urls)),
]
```

### 3. Document Management API

**documents/views.py:**

```
python
```

```
from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.response import Response
from rest_framework.parsers import MultiPartParser, FormParser
from django.http import FileResponse, Http404
from .models import CustomerDocument
from .serializers import CustomerDocumentSerializer, DocumentUploadSerializer
import os

class CustomerDocumentViewSet(viewsets.ModelViewSet):
    queryset = CustomerDocument.objects.filter(is_active=True)
    serializer_class = CustomerDocumentSerializer
    parser_classes = (MultiPartParser, FormParser)

    def get_queryset(self):
        """
        Filter documents by customer email
        """
        queryset = super().get_queryset()
        customer_email = self.request.query_params.get('customer_email', None)
        if customer_email:
            queryset = queryset.filter(customer_email=customer_email)
        return queryset

    @action(detail=False, methods=['post'], url_path='upload')
    def upload_document(self, request):
        """
        Upload or replace document
        File is saved to computer storage, path stored in database

        POST /api/documents/upload/

        Form Data:
        - customer_email: string
        - document_type: string
        - file: file (PDF, JPG, JPEG, PNG - max 5MB)
        - vehicle_id: int (optional)
        - driver_id: int (optional)

        Response:
        {
            "document": {
                "id": 1,
                "file_path": "/var/customer_portal/documents/...",
                "original_filename": "PO_12345.pdf",
                "file_size": 1048576,
        "document_type": "purchase_order"
    },
    "replaced": true/false,
    "message": "..."
}
"""
serializer = DocumentUploadSerializer(data=request.data)
serializer.is_valid(raise_exception=True)

customer_email = serializer.validated_data['customer_email']
document_type = serializer.validated_data['document_type']
uploaded_file = serializer.validated_data['file']
vehicle_id = serializer.validated_data.get('vehicle_id')
driver_id = serializer.validated_data.get('driver_id')

# Get vehicle and driver objects
vehicle = None
driver = None
if vehicle_id:
    from vehicles.models import VehicleDetails
    try:
        vehicle = VehicleDetails.objects.get(id=vehicle_id)
    except VehicleDetails.DoesNotExist:
        pass

if driver_id:
    from drivers.models import DriverHelper
    try:
        driver = DriverHelper.objects.get(id=driver_id)
    except DriverHelper.DoesNotExist:
        pass

# Check if document exists (for replacement)
existing = CustomerDocument.objects.filter(
    customer_email=customer_email,
    document_type=document_type,
    is_active=True
).first()

replaced = bool(existing)

try:
    # Replace or create - saves file to storage and stores path in DB
    new_doc = CustomerDocument.replace_document(
        customer_email=customer_email,
        document_type=document_type,
        uploaded_file=uploaded_file,
            vehicle=vehicle,
            driver=driver
        )

        return Response({
            "document": CustomerDocumentSerializer(new_doc, context={'request': request}).data,
            "replaced": replaced,
            "message": "Document replaced successfully" if replaced else "Document uploaded successfully",
            "storage_path": new_doc.file_path
        }, status=status.HTTP_201_CREATED)

    except Exception as e:
        return Response({
            "error": f"Failed to save document: {str(e)}"
        }, status=status.HTTP_500_INTERNAL_SERVER_ERROR)

@action(detail=True, methods=['delete'], url_path='remove')
def remove_document(self, request, pk=None):
    """
    Soft delete document (marks as inactive, keeps file on storage)

    DELETE /api/documents/{id}/remove/

    Query Parameters:
    - hard_delete: boolean (optional, default=false)
      If true, permanently deletes file from storage

    Response:
    {
        "message": "Document removed successfully"
    }
    """
    document = self.get_object()
    hard_delete = request.query_params.get('hard_delete', 'false').lower() == 'true'

    document.delete(hard_delete=hard_delete)

    return Response({
        "message": "Document permanently deleted" if hard_delete else "Document removed successfully",
        "hard_deleted": hard_delete
    }, status=status.HTTP_200_OK)

@action(detail=False, methods=['get'], url_path='list')
def list_customer_documents(self, request):
    """
    List all active documents for a customer
```

```
        GET /api/documents/list/?customer_email=user@example.com

        Response:
        {
            "count": 5,
            "documents": [
                {
                    "id": 1,
                    "file_path": "/var/customer_portal/documents/...",
                    "original_filename": "PO.pdf",
                    "file_size": 1024000,
                    "file_exists": true
                },
                ...
            ]
        }
        """
        customer_email = request.query_params.get('customer_email')
        if not customer_email:
            return Response({
                "error": "customer_email parameter is required"
            }, status=status.HTTP_400_BAD_REQUEST)

        documents = CustomerDocument.objects.filter(
            customer_email=customer_email,
            is_active=True
        ).order_by('-uploaded_at')

        serializer = CustomerDocumentSerializer(documents, many=True, context={'request': request})

        return Response({
            "count": documents.count(),
            "documents": serializer.data
        })

    @action(detail=True, methods=['get'], url_path='download')
    def download_document(self, request, pk=None):
        """
        Download document file from storage

        GET /api/documents/{id}/download/

        Returns: File download response
        """
        document = self.get_object()

        # Check if file exists on storage
        if not document.file_exists():
            return Response({
                "error": "File not found on storage",
                "file_path": document.file_path
            }, status=status.HTTP_404_NOT_FOUND)

        try:
            # Open file from storage path
            file_handle = open(document.file_path, 'rb')

            # Determine content type based on extension
            content_type_map = {
                '.pdf': 'application/pdf',
                '.jpg': 'image/jpeg',
                '.jpeg': 'image/jpeg',
                '.png': 'image/png',
            }
            content_type = content_type_map.get(document.file_extension.lower(), 'application/octet-stream')

            # Return file as response
            response = FileResponse(file_handle, content_type=content_type)
            response['Content-Disposition'] = f'attachment; filename="{document.original_filename}"'
            response['Content-Length'] = document.file_size

            return response

        except Exception as e:
            return Response({
                "error": f"Failed to read file: {str(e)}"
            }, status=status.HTTP_500_INTERNAL_SERVER_ERROR)

    @action(detail=True, methods=['get'], url_path='info')
    def document_info(self, request, pk=None):
        """
        Get detailed document information

        GET /api/documents/{id}/info/

        Response:
        {
            "id": 1,
            "file_path": "/var/customer_portal/documents/...",
            "original_filename": "PO.pdf",
            "file_size": 1024000,
            "file_size_readable": "1.00 MB",
            "file_exists": true,
            "uploaded_at": "2024-01-15T10:30:00Z"
    }
    """
    document = self.get_object()

    # Convert bytes to human-readable format
    def format_size(bytes):
        for unit in ['B', 'KB', 'MB', 'GB']:
            if bytes < 1024.0:
                return f"{bytes:.2f} {unit}"
            bytes /= 1024.0
        return f"{bytes:.2f} TB"

    return Response({
        "id": document.id,
        "customer_email": document.customer_email,
        "document_type": document.document_type,
        "document_type_display": document.get_document_type_display(),
        "file_path": document.file_path,
        "original_filename": document.original_filename,
        "file_size": document.file_size,
        "file_size_readable": format_size(document.file_size),
        "file_extension": document.file_extension,
        "file_exists": document.file_exists(),
        "is_active": document.is_active,
        "uploaded_at": document.uploaded_at,
        "updated_at": document.updated_at
    })
```

### documents/serializers.py:

python

```python
from rest_framework import serializers
from .models import CustomerDocument

class CustomerDocumentSerializer(serializers.ModelSerializer):
    document_type_display = serializers.CharField(source='get_document_type_display', read_only=True)
    file_url = serializers.SerializerMethodField()
    file_exists = serializers.SerializerMethodField()
    file_size_readable = serializers.SerializerMethodField()

    class Meta:
        model = CustomerDocument
        fields = [
            'id', 'customer_email', 'document_type', 'document_type_display',
            'file_path', 'original_filename', 'file_size', 'file_size_readable',
            'file_extension', 'file_url', 'file_exists', 'vehicle', 'driver',
            'uploaded_at', 'updated_at', 'is_active'
        ]
        read_only_fields = ['id', 'uploaded_at', 'updated_at']

    def get_file_url(self, obj):
        """
        Return download URL for the document
        """
        request = self.context.get('request')
        if request and obj.is_active:
            return request.build_absolute_uri(f'/api/documents/{obj.id}/download/')
        return None

    def get_file_exists(self, obj):
        """
        Check if file exists on storage
        """
        return obj.file_exists()

    def get_file_size_readable(self, obj):
        """
        Convert file size to human-readable format
        """
        bytes = obj.file_size
        for unit in ['B', 'KB', 'MB', 'GB']:
            if bytes < 1024.0:
                return f"{bytes:.2f} {unit}"
            bytes /= 1024.0
        return f"{bytes:.2f} TB"

class DocumentUploadSerializer(serializers.Serializer):
    customer_email = serializers.EmailField()
    document_type = serializers.ChoiceField(choices=[
        ('purchase_order', 'Purchase Order'),
        ('vehicle_registration', 'Vehicle Registration'),
        ('vehicle_insurance', 'Vehicle Insurance'),
        ('puc', 'PUC'),
        ('driver_license', 'Driver License'),
        ('transportation_approval', 'Transportation Approval'),
        ('payment_approval', 'Payment Approval'),
        ('vendor_approval', 'Vendor Approval'),
    ])
    file = serializers.FileField()
    vehicle_id = serializers.IntegerField(required=False)
    driver_id = serializers.IntegerField(required=False)

    def validate_file(self, value):
        """
        Validate file size and type
        """
        # Maximum file size: 5MB
        max_size = 5 * 1024 * 1024
        if value.size > max_size:
            raise serializers.ValidationError("File size must be under 5MB")

        # Allowed file types
        allowed_types = ['application/pdf', 'image/jpeg', 'image/png', 'image/jpg']
        if value.content_type not in allowed_types:
            raise serializers.ValidationError("Only PDF, JPG, JPEG, and PNG files are allowed")

        # Validate file extension
        import os
        ext = os.path.splitext(value.name)[1].lower()
        allowed_extensions = ['.pdf', '.jpg', '.jpeg', '.png']
        if ext not in allowed_extensions:
            raise serializers.ValidationError(f"File extension {ext} is not allowed")

        return value
```

**documents/urls.py:**

```
python
```

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

router = DefaultRouter()
router.register(r'', CustomerDocumentViewSet, basename='document')

urlpatterns = [
    path('', include(router.urls)),
]
```

## 4. Submission & QR Generation API

**submissions/qr_generator.py:**

```python
```

```
import qrcode
from io import BytesIO
from django.core.files import File
import json

def generate_qr_code(payload_data):
    """
    Generate QR code image from payload data

    Args:
        payload_data (dict): Dictionary containing QR payload

    Returns:
        File: Django File object containing QR code image
    """
    # Create JSON string from payload
    payload_json = json.dumps(payload_data, indent=2)

    # Generate QR code
    qr = qrcode.QRCode(
        version=1,
        error_correction=qrcode.constants.ERROR_CORRECT_H,
        box_size=10,
        border=4,
    )
    qr.add_data(payload_json)
    qr.make(fit=True)

    # Create image
    img = qr.make_image(fill_color="black", back_color="white")

    # Save to BytesIO
    buffer = BytesIO()
    img.save(buffer, format='PNG')
    buffer.seek(0)

    # Convert to Django File
    filename = f"qr_{payload_data.get('vehicle_number', 'code')}.png"
    return File(buffer, name=filename)
```

**submissions/views.py:**

```
python
```

```python
from rest_framework import viewsets, status
from rest_framework.decorators import action
from rest_framework.response import Response
from rest_framework.parsers import MultiPartParser, FormParser
from django.db import transaction
from django.core.mail import EmailMessage
from django.conf import settings
from .models import GateEntrySubmission, AuditLog
from .serializers import GateEntrySubmissionSerializer, SubmissionCreateSerializer
from .qr_generator import generate_qr_code
from vehicles.models import VehicleDetails
from drivers.models import DriverHelper
from documents.models import CustomerDocument

class GateEntrySubmissionViewSet(viewsets.ModelViewSet):
    queryset = GateEntrySubmission.objects.all()
    serializer_class = GateEntrySubmissionSerializer
    parser_classes = (MultiPartParser, FormParser)

    @action(detail=False, methods=['post'], url_path='create')
    def create_submission(self, request):
        """
        Create gate entry submission with QR code generation

        POST /api/submissions/create/

        Form Data:
        - customer_email: string
        - customer_phone: string
        - vehicle_number: string
        - driver_name: string
        - driver_phone: string
        - driver_language: string
        - helper_name: string
        - helper_phone: string
        - helper_language: string
        - purchase_order: file
        - vehicle_registration: file
        - vehicle_insurance: file
        - puc: file
        - driver_license: file
        - transportation_approval: file
        - payment_approval: file
        - vendor_approval: file

        Response:
```

```
{
    "submission": {
        "id": 1,
        "qrCodeImage": "http://...",
        "vehicleNumber": "MH12AB1234",
        ...
    }
}
"""
serializer = SubmissionCreateSerializer(data=request.data)
serializer.is_valid(raise_exception=True)

try:
    with transaction.atomic():
        # Extract data
        customer_email = serializer.validated_data['customer_email']
        customer_phone = serializer.validated_data['customer_phone']
        vehicle_number = serializer.validated_data['vehicle_number']
        driver_name = serializer.validated_data['driver_name']
        driver_phone = serializer.validated_data['driver_phone']
        driver_language = serializer.validated_data.get('driver_language', 'en')
        helper_name = serializer.validated_data.get('helper_name')
        helper_phone = serializer.validated_data.get('helper_phone')
        helper_language = serializer.validated_data.get('helper_language', 'en')

        # 1. Get or create vehicle
        vehicle, _ = VehicleDetails.objects.get_or_create(
            vehicle_registration_no=vehicle_number.upper()
        )

        # 2. Validate or create driver
        driver, driver_created = DriverHelper.validate_or_create(
            name=driver_name,
            phone_no=driver_phone,
            driver_type='Driver',
            language=driver_language
        )

        # 3. Validate or create helper (if provided)
        helper = None
        if helper_name and helper_phone:
            helper, helper_created = DriverHelper.validate_or_create(
                name=helper_name,
                phone_no=helper_phone,
                driver_type='Helper',
                language=helper_language
            )
```

```
# 4. Create submission (without QR yet)
submission = GateEntrySubmission.objects.create(
    customer_email=customer_email,
    customer_phone=customer_phone,
    vehicle=vehicle,
    driver=driver,
    helper=helper
)

# 5. Generate QR payload hash
submission.qr_payload_hash = submission.generate_payload_hash()

# 6. Generate QR code
qr_payload = {
    'submission_id': submission.id,
    'customer_name': customer_email.split('@')[0],
    'customer_email': customer_email,
    'driver_name': driver.name,
    'driver_phone': driver.phone_no,
    'helper_name': helper.name if helper else '',
    'helper_phone': helper.phone_no if helper else '',
    'vehicle_number': vehicle.vehicle_registration_no,
    'timestamp': submission.created_at.isoformat(),
}

qr_file = generate_qr_code(qr_payload)
submission.qr_code_image = qr_file
submission.save()

# 7. Handle document uploads
document_fields = [
    'purchase_order', 'vehicle_registration', 'vehicle_insurance',
    'puc', 'driver_license', 'transportation_approval',
    'payment_approval', 'vendor_approval'
]

for field in document_fields:
    file = request.FILES.get(field)
    if file:
        CustomerDocument.replace_document(
            customer_email=customer_email,
            document_type=field,
            new_file=file,
            vehicle=vehicle,
            driver=driver
        )
```

```
            # 8. Create audit log
            AuditLog.objects.create(
                submission=submission,
                action='SUBMISSION_CREATED',
                description=f'Gate entry submission created for vehicle {vehicle.vehicle_registration_no}',
                user_email=customer_email,
                ip_address=self.get_client_ip(request)
            )

            # 9. Send email notification
            self.send_qr_email(submission)

            # 10. Send SMS notification (placeholder)
            self.send_qr_sms(submission)

            # Return response
            return Response({
                "submission": {
                    "id": submission.id,
                    "qrCodeImage": request.build_absolute_uri(submission.qr_code_image.url),
                    "vehicleNumber": submission.vehicle.vehicle_registration_no,
                    "driverPhone": submission.driver.phone_no,
                    "status": submission.status,
                    "createdAt": submission.created_at
                }
            }, status=status.HTTP_201_CREATED)

        except Exception as e:
            return Response({
                "error": str(e)
            }, status=status.HTTP_400_BAD_REQUEST)

    def send_qr_email(self, submission):
        """
        Send QR code via email
        """
        try:
            subject = f"Gate Entry QR Code - {submission.vehicle.vehicle_registration_no}"
            body = f"""
Dear Customer,

Your gate entry QR code has been generated successfully.

Vehicle Number: {submission.vehicle.vehicle_registration_no}
Driver: {submission.driver.name} ({submission.driver.phone_no})
{'Helper: ' + submission.helper.name + ' (' + submission.helper.phone_no + ')' if submission.helper else ''}
```

```
Please present this QR code at the gate entrance.

Best regards,
Gate Entry System
          """

          email = EmailMessage(
             subject=subject,
             body=body,
             from_email=settings.EMAIL_HOST_USER,
             to=[submission.customer_email],
          )

          # Attach QR code
          if submission.qr_code_image:
             email.attach_file(submission.qr_code_image.path)

          email.send(fail_silently=False)

          # Log email sent
          AuditLog.objects.create(
             submission=submission,
             action='EMAIL_SENT',
             description=f'QR code email sent to {submission.customer_email}',
             user_email=submission.customer_email
          )

       except Exception as e:
          # Log error but don't fail the submission
          AuditLog.objects.create(
             submission=submission,
             action='EMAIL_FAILED',
             description=f'Failed to send email: {str(e)}',
             user_email=submission.customer_email
          )

    def send_qr_sms(self, submission):
       """
       Send QR code link via SMS (placeholder implementation)
       """
       try:
          # Placeholder for SMS integration
          # In production, integrate with SMS provider (Twilio, AWS SNS, etc.)

          message = f"Gate Entry QR Code generated for vehicle {submission.vehicle.vehicle_registration_no}. " \
                f"Check your email for details."
```

```
            # TODO: Implement actual SMS sending
            # sms_service.send(to=submission.customer_phone, message=message)

            # Log SMS attempt
            AuditLog.objects.create(
                submission=submission,
                action='SMS_QUEUED',
                description=f'SMS queued for {submission.customer_phone}',
                user_email=submission.customer_email
            )

        except Exception as e:
            # Log error
            AuditLog.objects.create(
                submission=submission,
                action='SMS_FAILED',
                description=f'Failed to send SMS: {str(e)}',
                user_email=submission.customer_email
            )

    def get_client_ip(self, request):
        """
        Get client IP address from request
        """
        x_forwarded_for = request.META.get('HTTP_X_FORWARDED_FOR')
        if x_forwarded_for:
            ip = x_forwarded_for.split(',')[0]
        else:
            ip = request.META.get('REMOTE_ADDR')
        return ip
```

**submissions/serializers.py:**

python

```python
from rest_framework import serializers
from .models import GateEntrySubmission, AuditLog

class GateEntrySubmissionSerializer(serializers.ModelSerializer):
    vehicle_number = serializers.CharField(source='vehicle.vehicle_registration_no', read_only=True)
    driver_name = serializers.CharField(source='driver.name', read_only=True)
    driver_phone = serializers.CharField(source='driver.phone_no', read_only=True)
    helper_name = serializers.CharField(source='helper.name', read_only=True)
    helper_phone = serializers.CharField(source='helper.phone_no', read_only=True)
    qr_code_url = serializers.SerializerMethodField()

    class Meta:
        model = GateEntrySubmission
        fields = [
            'id', 'customer_email', 'customer_phone',
            'vehicle_number', 'driver_name', 'driver_phone',
            'helper_name', 'helper_phone', 'qr_code_url',
            'qr_payload_hash', 'status', 'created_at', 'updated_at'
        ]
        read_only_fields = ['id', 'qr_payload_hash', 'created_at', 'updated_at']

    def get_qr_code_url(self, obj):
        if obj.qr_code_image:
            request = self.context.get('request')
            if request:
                return request.build_absolute_uri(obj.qr_code_image.url)
        return None

class SubmissionCreateSerializer(serializers.Serializer):
    customer_email = serializers.EmailField()
    customer_phone = serializers.RegexField(
        regex=r'^\+91\d{10},
        error_messages={'invalid': 'Phone must be in format: +91XXXXXXXXXX'}
    )
    vehicle_number = serializers.CharField(max_length=50)
    driver_name = serializers.CharField(max_length=100)
    driver_phone = serializers.RegexField(
        regex=r'^\+91\d{10},
        error_messages={'invalid': 'Phone must be in format: +91XXXXXXXXXX'}
    )
    driver_language = serializers.CharField(default='en')
    helper_name = serializers.CharField(max_length=100, required=False, allow_blank=True)
    helper_phone = serializers.RegexField(
        regex=r'^\+91\d{10},
        required=False,
        allow_blank=True,
        error_messages={'invalid': 'Phone must be in format: +91XXXXXXXXXX'}
    )
    helper_language = serializers.CharField(default='en', required=False)

class AuditLogSerializer(serializers.ModelSerializer):
    class Meta:
        model = AuditLog
        fields = '__all__'
        read_only_fields = ['id', 'timestamp']
```

**submissions/urls.py:**

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

router = DefaultRouter()
router.register(r'', GateEntrySubmissionViewSet, basename='submission')

urlpatterns = [
    path('', include(router.urls)),
]
```

---

## Authentication

### JWT Setup

### authentication/views.py:

```python
```

```python
from rest_framework import status, viewsets
from rest_framework.decorators import action, permission_classes
from rest_framework.response import Response
from rest_framework.permissions import AllowAny, IsAuthenticated
from rest_framework_simplejwt.tokens import RefreshToken
from django.contrib.auth import authenticate
from .models import CustomerUser
from .serializers import (
    CustomerUserSerializer,
    RegisterSerializer,
    LoginSerializer
)

class AuthViewSet(viewsets.GenericViewSet):
    """
    Authentication endpoints
    """

    @action(detail=False, methods=['post'], permission_classes=[AllowAny])
    def register(self, request):
        """
        Register new customer

        POST /api/auth/register/

        Request:
        {
            "email": "user@example.com",
            "username": "user123",
            "password": "SecurePass123",
            "phone": "+919876543210",
            "company_name": "ABC Corp"
        }

        Response:
        {
            "user": {...},
            "tokens": {
                "access": "...",
                "refresh": "..."
            }
        }
        """
        serializer = RegisterSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)
        user = serializer.save()

        # Generate tokens
        refresh = RefreshToken.for_user(user)

        return Response({
            "user": CustomerUserSerializer(user).data,
            "tokens": {
                "access": str(refresh.access_token),
                "refresh": str(refresh)
            }
        }, status=status.HTTP_201_CREATED)

    @action(detail=False, methods=['post'], permission_classes=[AllowAny])
    def login(self, request):
        """
        Login customer

        POST /api/auth/login/

        Request:
        {
            "email": "user@example.com",
            "password": "SecurePass123"
        }

        Response:
        {
            "user": {...},
            "tokens": {
                "access": "...",
                "refresh": "..."
            }
        }
        """
        serializer = LoginSerializer(data=request.data)
        serializer.is_valid(raise_exception=True)

        email = serializer.validated_data['email']
        password = serializer.validated_data['password']

        # Authenticate
        user = authenticate(request, username=email, password=password)

        if not user:
            return Response({
                "error": "Invalid credentials"
            }, status=status.HTTP_401_UNAUTHORIZED)

        # Generate tokens
        refresh = RefreshToken.for_user(user)

        return Response({
            "user": CustomerUserSerializer(user).data,
            "tokens": {
                "access": str(refresh.access_token),
                "refresh": str(refresh)
            }
        })

    @action(detail=False, methods=['post'], permission_classes=[IsAuthenticated])
    def logout(self, request):
        """
        Logout customer (blacklist refresh token)

        POST /api/auth/logout/

        Request:
        {
            "refresh": "..."
        }
        """
        try:
            refresh_token = request.data.get("refresh")
            token = RefreshToken(refresh_token)
            token.blacklist()

            return Response({
                "message": "Logged out successfully"
            }, status=status.HTTP_200_OK)
        except Exception:
            return Response({
                "error": "Invalid token"
            }, status=status.HTTP_400_BAD_REQUEST)
```

**authentication/serializers.py:**

```
python
```

```
from rest_framework import serializers
from django.contrib.auth.password_validation import validate_password
from .models import CustomerUser

class CustomerUserSerializer(serializers.ModelSerializer):
    class Meta:
        model = CustomerUser
        fields = ['id', 'email', 'username', 'phone', 'company_name', 'date_joined']
        read_only_fields = ['id', 'date_joined']

class RegisterSerializer(serializers.ModelSerializer):
    password = serializers.CharField(
        write_only=True,
        required=True,
        validators=[validate_password]
    )
    password2 = serializers.CharField(write_only=True, required=True)

    class Meta:
        model = CustomerUser
        fields = ['email', 'username', 'password', 'password2', 'phone', 'company_name']

    def validate(self, attrs):
        if attrs['password'] != attrs['password2']:
            raise serializers.ValidationError({"password": "Passwords don't match"})
        return attrs

    def create(self, validated_data):
        validated_data.pop('password2')
        user = CustomerUser.objects.create_user(**validated_data)
        return user

class LoginSerializer(serializers.Serializer):
    email = serializers.EmailField()
    password = serializers.CharField(write_only=True)
```

### authentication/urls.py:

```
python
```

```
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from rest_framework_simplejwt.views import TokenRefreshView
from .views import AuthViewSet

router = DefaultRouter()
router.register(r'', AuthViewSet, basename='auth')

urlpatterns = [
    path('', include(router.urls)),
    path('token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),
]
```

---

## Document Handling

### Media Files Configuration

Documents are stored in `media/documents/{customer_email}/{document_type}/`

**Key Features:**

- Automatic file organization by customer and type
- File size validation (5MB max)
- File type validation (PDF, JPG, JPEG, PNG)
- Document replacement logic (soft delete old, create new)
- Serve files via Django in development, nginx/Apache in production

### File Validation

Implemented in `documents/serializers.py`:

```python
def validate_file(self, value):
    max_size = 5 * 1024 * 1024  # 5MB
    if value.size > max_size:
        raise serializers.ValidationError("File size must be under 5MB")

    allowed_types = ['application/pdf', 'image/jpeg', 'image/png', 'image/jpg']
    if value.content_type not in allowed_types:
        raise serializers.ValidationError("Only PDF, JPG, JPEG, and PNG files are allowed")

    return value
```

---

## QR Code Generation

### QR Payload Structure

json

```json
{
  "submission_id": 123,
  "customer_name": "John Doe",
  "customer_email": "john@example.com",
  "driver_name": "Driver Name",
  "driver_phone": "+919876543210",
  "helper_name": "Helper Name",
  "helper_phone": "+919876543211",
  "vehicle_number": "MH12AB1234",
  "timestamp": "2024-01-15T10:30:00"
}
```

### Email Delivery

QR codes are sent via email with:

- Subject: Gate Entry QR Code - {Vehicle Number}
- Body: Summary with customer, driver, helper, vehicle details
- Attachment: QR code PNG image

### SMS Delivery (Placeholder)

SMS integration requires third-party service:

- **Twilio**: Popular choice for SMS/WhatsApp
- **AWS SNS**: Amazon's messaging service
- **MessageBird**: Global SMS provider

### Example Twilio Integration:

python

```
from twilio.rest import Client

def send_qr_sms(submission):
    account_sid = settings.TWILIO_ACCOUNT_SID
    auth_token = settings.TWILIO_AUTH_TOKEN
    client = Client(account_sid, auth_token)

    message = client.messages.create(
        body=f"Your gate entry QR code: {request.build_absolute_uri(submission.qr_code_image.url)}",
        from_=settings.TWILIO_PHONE_NUMBER,
        to=submission.customer_phone
    )

    return message.sid
```

---

## Docker Deployment

### Dockerfile

### Dockerfile:

```
dockerfile
```

```
FROM python:3.11-slim

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

# Set work directory
WORKDIR /app

# Install system dependencies
RUN apt-get update && apt-get install -y \
    postgresql-client \
    libpq-dev \
    gcc \
    && rm -rf /var/lib/apt/lists/*

# Install Python dependencies
COPY requirements.txt /app/
RUN pip install --upgrade pip && pip install -r requirements.txt

# Copy project
COPY . /app/

# Create media directory
RUN mkdir -p /app/media/documents /app/media/qr_codes

# Collect static files
RUN python manage.py collectstatic --noinput

# Expose port
EXPOSE 8000

# Run migrations and start server
CMD ["sh", "-c", "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"]
```

### Docker Compose

#### docker-compose.yml:

```
yaml
```

```yaml
version: '3.8'

services:
  db:
    image: postgres:15
    container_name: customer_portal_db
    environment:
      POSTGRES_DB: customer_portal_db
      POSTGRES_USER: portal_admin
      POSTGRES_PASSWORD: secure_password
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    networks:
      - portal_network

  web:
    build: .
    container_name: customer_portal_web
    command: sh -c "python manage.py migrate && python manage.py runserver 0.0.0.0:8000"
    volumes:
      - .:/app
      - media_volume:/app/media
    ports:
      - "8000:8000"
    env_file:
      - .env
    depends_on:
      - db
    networks:
      - portal_network

  pgadmin:
    image: dpage/pgadmin4:latest
    container_name: customer_portal_pgadmin
    environment:
      PGADMIN_DEFAULT_EMAIL: admin@admin.com
      PGADMIN_DEFAULT_PASSWORD: admin
    ports:
      - "5050:80"
    depends_on:
      - db
    networks:
      - portal_network
```

```
volumes:
  postgres_data:
  media_volume:

networks:
  portal_network:
    driver: bridge
```

### Docker Commands

```bash
# Build and start containers
docker-compose up --build

# Run in detached mode
docker-compose up -d

# Stop containers
docker-compose down

# View logs
docker-compose logs -f web

# Run migrations
docker-compose exec web python manage.py migrate

# Create superuser
docker-compose exec web python manage.py createsuperuser

# Access Django shell
docker-compose exec web python manage.py shell

# Rebuild specific service
docker-compose up --build web
```

### Production Dockerfile

### Dockerfile.prod:

```dockerfile
```

```
FROM python:3.11-slim

ENV PYTHONDONTWRITEBYTECODE=1
ENV PYTHONUNBUFFERED=1

WORKDIR /app

# Install dependencies
RUN apt-get update && apt-get install -y \
    postgresql-client \
    libpq-dev \
    gcc \
    nginx \
    && rm -rf /var/lib/apt/lists/*

COPY requirements.txt /app/
RUN pip install --upgrade pip && pip install -r requirements.txt gunicorn

COPY . /app/

RUN mkdir -p /app/media /app/staticfiles

# Collect static files
RUN python manage.py collectstatic --noinput

EXPOSE 8000

# Use Gunicorn for production
CMD ["gunicorn", "--bind", "0.0.0.0:8000", "--workers", "4", "customer_portal.wsgi:application"]
```

---

## Testing

### Test Setup

### pytest.ini:

```ini
[pytest]
DJANGO_SETTINGS_MODULE = customer_portal.settings
python_files = tests.py test_*.py *_tests.py
addopts = --verbose --strict-markers
```

### Unit Tests

### vehicles/tests.py:

```python
import pytest
from django.test import TestCase
from vehicles.models import VehicleDetails

@pytest.mark.django_db
class TestVehicleDetails(TestCase):

    def test_create_vehicle(self):
        """Test vehicle creation"""
        vehicle = VehicleDetails.objects.create(
            vehicle_registration_no="MH12AB1234"
        )
        assert vehicle.vehicle_registration_no == "MH12AB1234"
        assert str(vehicle) == "MH12AB1234"

    def test_vehicle_uniqueness(self):
        """Test unique vehicle registration constraint"""
        VehicleDetails.objects.create(vehicle_registration_no="MH12AB1234")

        with pytest.raises(Exception):
            VehicleDetails.objects.create(vehicle_registration_no="MH12AB1234")
```

### drivers/tests.py:

```python
```

```
import pytest
from django.test import TestCase
from django.core.exceptions import ValidationError
from drivers.models import DriverHelper

@pytest.mark.django_db
class TestDriverHelper(TestCase):

    def test_create_driver(self):
        """Test driver creation"""
        driver = DriverHelper.objects.create(
            name="John Doe",
            type="Driver",
            phone_no="+919876543210",
            language="en"
        )
        assert driver.name == "John Doe"
        assert driver.phone_no == "+919876543210"

    def test_phone_uniqueness(self):
        """Test unique phone constraint"""
        DriverHelper.objects.create(
            name="John Doe",
            type="Driver",
            phone_no="+919876543210"
        )

        with pytest.raises(Exception):
            DriverHelper.objects.create(
                name="Jane Doe",
                type="Driver",
                phone_no="+919876543210"
            )

    def test_validate_or_create_existing_match(self):
        """Test validation: phone exists with matching name"""
        DriverHelper.objects.create(
            name="John Doe",
            type="Driver",
            phone_no="+919876543210"
        )

        driver, created = DriverHelper.validate_or_create(
            name="John Doe",
            phone_no="+919876543210",
            driver_type="Driver"
        )

        assert not created
        assert driver.name == "John Doe"

    def test_validate_or_create_mismatch(self):
        """Test validation: phone exists with different name"""
        DriverHelper.objects.create(
            name="John Doe",
            type="Driver",
            phone_no="+919876543210"
        )

        with pytest.raises(ValidationError):
            DriverHelper.validate_or_create(
                name="Jane Smith",
                phone_no="+919876543210",
                driver_type="Driver"
            )

    def test_validate_or_create_new(self):
        """Test validation: create new driver"""
        driver, created = DriverHelper.validate_or_create(
            name="John Doe",
            phone_no="+919876543210",
            driver_type="Driver"
        )

        assert created
        assert driver.name == "John Doe"
```

**documents/tests.py:**

```
python
```

```python
import pytest
from django.test import TestCase
from django.core.files.uploadedfile import SimpleUploadedFile
from documents.models import CustomerDocument
from vehicles.models import VehicleDetails

@pytest.mark.django_db
class TestCustomerDocument(TestCase):

    def setUp(self):
        self.vehicle = VehicleDetails.objects.create(
            vehicle_registration_no="MH12AB1234"
        )
        self.test_file = SimpleUploadedFile(
            "test.pdf",
            b"file_content",
            content_type="application/pdf"
        )

    def test_create_document(self):
        """Test document creation"""
        doc = CustomerDocument.objects.create(
            customer_email="test@example.com",
            document_type="purchase_order",
            file_path=self.test_file,
            vehicle=self.vehicle
        )
        assert doc.customer_email == "test@example.com"
        assert doc.is_active is True

    def test_replace_document(self):
        """Test document replacement logic"""
        # Create initial document
        old_file = SimpleUploadedFile("old.pdf", b"old_content", content_type="application/pdf")
        old_doc = CustomerDocument.objects.create(
            customer_email="test@example.com",
            document_type="purchase_order",
            file_path=old_file,
            vehicle=self.vehicle
        )

        # Replace with new document
        new_file = SimpleUploadedFile("new.pdf", b"new_content", content_type="application/pdf")
        new_doc = CustomerDocument.replace_document(
            customer_email="test@example.com",
            document_type="purchase_order",
            new_file=new_file,
            vehicle=self.vehicle
        )

        # Refresh old document from DB
        old_doc.refresh_from_db()

        assert old_doc.is_active is False
        assert old_doc.replaced_by == new_doc
        assert new_doc.is_active is True

    def test_soft_delete(self):
        """Test soft delete"""
        doc = CustomerDocument.objects.create(
            customer_email="test@example.com",
            document_type="purchase_order",
            file_path=self.test_file,
            vehicle=self.vehicle
        )

        doc.delete(hard_delete=False)

        doc.refresh_from_db()
        assert doc.is_active is False
```

**submissions/tests.py:**

```
python
```

```python
import pytest
from django.test import TestCase
from submissions.models import GateEntrySubmission
from vehicles.models import VehicleDetails
from drivers.models import DriverHelper

@pytest.mark.django_db
class TestGateEntrySubmission(TestCase):

    def setUp(self):
        self.vehicle = VehicleDetails.objects.create(
            vehicle_registration_no="MH12AB1234"
        )
        self.driver = DriverHelper.objects.create(
            name="John Doe",
            type="Driver",
            phone_no="+919876543210"
        )
        self.helper = DriverHelper.objects.create(
            name="Jane Smith",
            type="Helper",
            phone_no="+919876543211"
        )

    def test_create_submission(self):
        """Test submission creation"""
        submission = GateEntrySubmission.objects.create(
            customer_email="test@example.com",
            customer_phone="+919876543212",
            vehicle=self.vehicle,
            driver=self.driver,
            helper=self.helper
        )

        assert submission.customer_email == "test@example.com"
        assert submission.status == "pending"

    def test_generate_payload_hash(self):
        """Test QR payload hash generation"""
        submission = GateEntrySubmission.objects.create(
            customer_email="test@example.com",
            customer_phone="+919876543212",
            vehicle=self.vehicle,
            driver=self.driver,
            helper=self.helper
        )
    hash1 = submission.generate_payload_hash()
    hash2 = submission.generate_payload_hash()

    assert hash1 == hash2  # Same input = same hash
    assert len(hash1) == 64  # SHA-256 produces 64 char hex
```

### API Tests

### tests/test_api.py:

```
python
```

```
import pytest
from rest_framework.test import APIClient
from rest_framework import status
from django.core.files.uploadedfile import SimpleUploadedFile
from authentication.models import CustomerUser
from vehicles.models import VehicleDetails
from drivers.models import DriverHelper

@pytest.mark.django_db
class TestVehicleLookupAPI:

    def setup_method(self):
        self.client = APIClient()
        self.user = CustomerUser.objects.create_user(
            email="test@example.com",
            username="testuser",
            password="TestPass123"
        )
        self.client.force_authenticate(user=self.user)

    def test_vehicle_lookup_not_found(self):
        """Test vehicle lookup for non-existent vehicle"""
        response = self.client.get('/api/vehicles/NOTFOUND/lookup/')
        assert response.status_code == status.HTTP_404_NOT_FOUND

    def test_vehicle_lookup_success(self):
        """Test successful vehicle lookup"""
        vehicle = VehicleDetails.objects.create(
            vehicle_registration_no="MH12AB1234"
        )

        response = self.client.get(f'/api/vehicles/{vehicle.vehicle_registration_no}/lookup/')
        assert response.status_code == status.HTTP_200_OK
        assert 'vehicle' in response.data

@pytest.mark.django_db
class TestDriverValidationAPI:

    def setup_method(self):
        self.client = APIClient()
        self.user = CustomerUser.objects.create_user(
            email="test@example.com",
            username="testuser",
            password="TestPass123"
        )
        self.client.force_authenticate(user=self.user)
```

```
def test_create_new_driver(self):
    """Test creating new driver"""
    data = {
        "name": "John Doe",
        "phone_no": "+919876543210",
        "type": "Driver",
        "language": "en"
    }

    response = self.client.post('/api/drivers/validate-or-create/', data)
    assert response.status_code == status.HTTP_201_CREATED
    assert response.data['created'] is True

def test_validate_existing_driver(self):
    """Test validating existing driver with matching name"""
    DriverHelper.objects.create(
        name="John Doe",
        type="Driver",
        phone_no="+919876543210"
    )

    data = {
        "name": "John Doe",
        "phone_no": "+919876543210",
        "type": "Driver",
        "language": "en"
    }

    response = self.client.post('/api/drivers/validate-or-create/', data)
    assert response.status_code == status.HTTP_200_OK
    assert response.data['created'] is False

def test_reject_phone_with_different_name(self):
    """Test rejection when phone exists with different name"""
    DriverHelper.objects.create(
        name="John Doe",
        type="Driver",
        phone_no="+919876543210"
    )

    data = {
        "name": "Jane Smith",
        "phone_no": "+919876543210",
        "type": "Driver",
        "language": "en"
    }
```

```
        response = self.client.post('/api/drivers/validate-or-create/', data)
        assert response.status_code == status.HTTP_400_BAD_REQUEST
```

### Run Tests

```bash
# Run all tests
pytest

# Run with coverage
pytest --cov=. --cov-report=html

# Run specific test file
pytest vehicles/tests.py

# Run specific test class
pytest vehicles/tests.py::TestVehicleDetails

# Run with verbose output
pytest -v

# Run and stop at first failure
pytest -x
```

---

## Complete Command Reference

### Django Management

```bash
```

```
# Create superuser
python manage.py createsuperuser

# Make migrations
python manage.py makemigrations

# Apply migrations
python manage.py migrate

# Run development server
python manage.py runserver

# Run on specific port
python manage.py runserver 8080

# Create new app
python manage.py startapp app_name

# Django shell
python manage.py shell

# Database shell
python manage.py dbshell

# Collect static files
python manage.py collectstatic

# Check for issues
python manage.py check

# Show migrations
python manage.py showmigrations

# Reverse migration
python manage.py migrate app_name migration_name
```

## Database Commands

bash

```
# Backup database
pg_dump -U portal_admin customer_portal_db > backup.sql

# Restore database
psql -U portal_admin customer_portal_db < backup.sql

# Access PostgreSQL
psql -U portal_admin -d customer_portal_db

# List databases
\l

# Connect to database
\c customer_portal_db

# List tables
\dt

# Describe table
\d table_name

# Exit
\q
```

### Virtual Environment

```bash
# Create venv
python -m venv venv

# Activate (Windows)
venv\Scripts\activate

# Activate (Linux/Mac)
source venv/bin/activate

# Deactivate
deactivate

# Install requirements
pip install -r requirements.txt

# Freeze requirements
pip freeze > requirements.txt
```

### Docker Commands

```bash
# Build image
docker build -t customer-portal .

# Run container
docker run -p 8000:8000 customer-portal

# Docker Compose up
docker-compose up

# Docker Compose down
docker-compose down

# View logs
docker-compose logs -f

# Execute command in container
docker-compose exec web python manage.py migrate

# Remove volumes
docker-compose down -v

# Rebuild
docker-compose up --build
```

---

## Troubleshooting

### Common PostgreSQL Errors

#### Error: `psycopg2.OperationalError: FATAL: database does not exist`

```bash
# Solution: Create database
sudo -u postgres psql
CREATE DATABASE customer_portal_db;
\q
```

#### Error: `peer authentication failed for user`

```bash
```

```
# Solution: Edit pg_hba.conf
sudo nano /etc/postgresql/15/main/pg_hba.conf

# Change from:
local   all             all                                     peer

# To:
local   all             all                                     md5

# Restart PostgreSQL
sudo service postgresql restart
```

**CORS Issues**

**Error:** `CORS header 'Access-Control-Allow-Origin' missing`

Solution: Ensure CORS middleware is configured in settings.py:

```python
INSTALLED_APPS = [
    ...
    'corsheaders',
]

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

CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
]
```

**Environment Variable Errors**

**Error:** `KeyError: 'SECRET_KEY'`

Solution: Ensure .env file exists and contains required variables:

```bash
```

```
# Check .env file
cat .env

# If missing, create it
cp .env.example .env

# Edit with proper values
nano .env
```

### File Upload Problems

**Error: `SuspiciousFileOperation`**

Solution: Ensure MEDIA_ROOT is properly configured:

```python
# settings.py
MEDIA_ROOT = BASE_DIR / 'media'
MEDIA_URL = '/media/'

# urls.py (development only)
from django.conf.urls.static import static
if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

**Error: `Permission denied` when uploading files**

```bash
# Fix permissions
chmod -R 755 media/
chown -R $USER:$USER media/
```

### Migration Issues

**Error: `No changes detected`**

```bash
# Force makemigrations for specific app
python manage.py makemigrations app_name

# Create empty migration
python manage.py makemigrations --empty app_name
```

**Error: `Table already exists`**

```bash
# Fake initial migration
python manage.py migrate --fake-initial

# Or drop and recreate database (CAUTION: deletes all data)
python manage.py reset_db
python manage.py migrate
```

### JWT Token Errors

**Error: `Token is invalid or expired`**

Solution: Check token lifetime in settings:

```python
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),  # Increase if needed
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
}
```

### Email Configuration Issues

**Error: `SMTPAuthenticationError`**

Solution for Gmail:

1. Enable 2-factor authentication
2. Generate app-specific password
3. Use app password in EMAIL_HOST_PASSWORD

```env
EMAIL_HOST_USER=your-email@gmail.com
EMAIL_HOST_PASSWORD=your-16-char-app-password
```

---

## API Request/Response Examples

### Complete Submission Flow

**1. Register User:**

```bash
```

```
curl -X POST http://localhost:8000/api/auth/register/ \
  -H "Content-Type: application/json" \
  -d '{
    "email": "john@example.com",
    "username": "john123",
    "password": "SecurePass123",
    "password2": "SecurePass123",
    "phone": "+919876543210",
    "company_name": "ABC Corp"
  }'
```

Response:

json

```json
{
  "user": {
    "id": 1,
    "email": "john@example.com",
    "username": "john123"
  },
  "tokens": {
    "access": "eyJ0eXAiOiJKV1QiLCJhbGc...",
    "refresh": "eyJ0eXAiOiJKV1QiLCJhbGc..."
  }
}
```

**2. Lookup Vehicle (Auto-fill):**

bash

```bash
curl -X GET http://localhost:8000/api/vehicles/MH12AB1234/lookup/ \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

Response:

json

```
{
  "vehicle": {
    "id": 1,
    "vehicle_registration_no": "MH12AB1234",
    "remark": null,
    "ratings": 0
  },
  "driver": {
    "id": 1,
    "name": "John Driver",
    "phone_no": "+919876543210",
    "language": "en"
  },
  "helper": {
    "id": 2,
    "name": "Helper Name",
    "phone_no": "+919876543211",
    "language": "hi"
  },
  "documents": [
    {
      "id": 1,
      "document_type": "purchase_order",
      "file_url": "http://localhost:8000/media/documents/..."
    }
  ]
}
```

### 3. Validate Driver:

```bash
curl -X POST http://localhost:8000/api/drivers/validate-or-create/ \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "John Driver",
    "phone_no": "+919876543210",
    "type": "Driver",
    "language": "en"
  }'
```

Response:

```json
```

```
{
  "driver": {
    "id": 1,
    "name": "John Driver",
    "phone_no": "+919876543210",
    "type": "Driver",
    "language": "en"
  },
  "created": false,
  "message": "Existing driver/helper found"
}
```

**4. Upload Document:**

bash

```bash
curl -X POST http://localhost:8000/api/documents/upload/ \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN" \
  -F "customer_email=john@example.com" \
  -F "document_type=purchase_order" \
  -F "file=@/path/to/po.pdf" \
  -F "vehicle_id=1"
```

Response:

json

```json
{
  "document": {
    "id": 5,
    "customer_email": "john@example.com",
    "document_type": "purchase_order",
    "file_path": "/var/customer_portal/documents/documents/john_at_example_com/purchase_order/purchase_order_2024011
    "original_filename": "po.pdf",
    "file_size": 1048576,
    "file_size_readable": "1.00 MB",
    "file_exists": true,
    "file_url": "http://localhost:8000/api/documents/5/download/",
    "is_active": true
  },
  "replaced": true,
  "message": "Document replaced successfully",
  "storage_path": "/var/customer_portal/documents/documents/john_at_example_com/purchase_order/purchase_order_20240
}
```

**5. Download Document:**

```bash
curl -X GET http://localhost:8000/api/documents/5/download/ \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN" \
  --output downloaded_file.pdf
```

Response: Binary file download

**6. Get Document Info:**

```bash
curl -X GET http://localhost:8000/api/documents/5/info/ \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN"
```

Response:

```json
{
  "id": 5,
  "customer_email": "john@example.com",
  "document_type": "purchase_order",
  "document_type_display": "Purchase Order",
  "file_path": "/var/customer_portal/documents/documents/john_at_example_com/purchase_order/purchase_order_20240115_
  "original_filename": "po.pdf",
  "file_size": 1048576,
  "file_size_readable": "1.00 MB",
  "file_extension": ".pdf",
  "file_exists": true,
  "is_active": true,
  "uploaded_at": "2024-01-15T10:30:45Z",
  "updated_at": "2024-01-15T10:30:45Z"
}
```

**7. Submit Entry (Generate QR):**

```bash
```

```
curl -X POST http://localhost:8000/api/submissions/create/ \
  -H "Authorization: Bearer YOUR_ACCESS_TOKEN" \
  -F "customer_email=john@example.com" \
  -F "customer_phone=+919876543212" \
  -F "vehicle_number=MH12AB1234" \
  -F "driver_name=John Driver" \
  -F "driver_phone=+919876543210" \
  -F "driver_language=en" \
  -F "helper_name=Helper Name" \
  -F "helper_phone=+919876543211" \
  -F "helper_language=hi" \
  -F "purchase_order=@/path/to/po.pdf" \
  -F "vehicle_registration=@/path/to/registration.pdf"
```

Response:

```json
{
  "submission": {
    "id": 10,
    "qrCodeImage": "http://localhost:8000/api/submissions/10/qr-download/",
    "qrCodePath": "/var/customer_portal/documents/qr_codes/qr_MH12AB1234_20240115_103050.png",
    "vehicleNumber": "MH12AB1234",
    "driverPhone": "+919876543210",
    "status": "pending",
    "createdAt": "2024-01-15T10:30:50Z"
  }
}
```

---

## Storage Management

### Check Storage Usage

```bash
```

```
# Linux/Mac - Check storage directory size
du -sh /var/customer_portal/documents/

# List all customer directories
ls -lh /var/customer_portal/documents/documents/

# Count total files
find /var/customer_portal/documents -type f | wc -l

# Windows - Check directory size
dir /s C:\CustomerPortal\Documents
```

### Backup Documents

```bash
# Create backup archive
tar -czf customer_documents_backup_$(date +%Y%m%d).tar.gz /var/customer_portal/documents/

# Backup to remote server
rsync -avz /var/customer_portal/documents/ user@backup-server:/backups/customer_portal/

# Windows backup
xcopy C:\CustomerPortal\Documents D:\Backups\CustomerPortal\ /E /I /Y
```

### Clean Up Old/Inactive Documents

### Management Command: `documents/management/commands/cleanup_documents.py`

```python
```

```
from django.core.management.base import BaseCommand
from documents.models import CustomerDocument
from datetime import datetime, timedelta
import os

class Command(BaseCommand):
    help = 'Clean up old inactive documents from storage'

    def add_arguments(self, parser):
        parser.add_argument(
            '--days',
            type=int,
            default=90,
            help='Delete inactive documents older than X days'
        )
        parser.add_argument(
            '--dry-run',
            action='store_true',
            help='Show what would be deleted without actually deleting'
        )

    def handle(self, *args, **options):
        days = options['days']
        dry_run = options['dry_run']

        cutoff_date = datetime.now() - timedelta(days=days)

        # Find old inactive documents
        old_docs = CustomerDocument.objects.filter(
            is_active=False,
            updated_at__lt=cutoff_date
        )

        total_size = 0
        deleted_count = 0

        for doc in old_docs:
            if doc.file_exists():
                size = os.path.getsize(doc.file_path)
                total_size += size

                if not dry_run:
                    try:
                        os.remove(doc.file_path)
                        doc.delete(hard_delete=True)
                        deleted_count += 1
                self.stdout.write(f"Deleted: {doc.file_path}")
            except Exception as e:
                self.stdout.write(self.style.ERROR(f"Error deleting {doc.file_path}: {e}"))
        else:
            self.stdout.write(f"Would delete: {doc.file_path} ({size} bytes)")

    # Convert bytes to MB
    total_mb = total_size / (1024 * 1024)

    if dry_run:
        self.stdout.write(self.style.WARNING(
            f"\nDRY RUN: Would delete {old_docs.count()} documents, "
            f"freeing {total_mb:.2f} MB"
        ))
    else:
        self.stdout.write(self.style.SUCCESS(
            f"\nDeleted {deleted_count} documents, "
            f"freed {total_mb:.2f} MB"
        ))
```

### Run cleanup:

```bash
# Dry run to see what would be deleted
python manage.py cleanup_documents --days=90 --dry-run

# Actually delete old documents
python manage.py cleanup_documents --days=90

# Delete very old documents (1 year)
python manage.py cleanup_documents --days=365
```

### Verify File Integrity

### Management Command: `documents/management/commands/verify_documents.py`

```python
```

```python
from django.core.management.base import BaseCommand
from documents.models import CustomerDocument
from submissions.models import GateEntrySubmission

class Command(BaseCommand):
    help = 'Verify document file integrity'

    def handle(self, *args, **options):
        # Check documents
        docs = CustomerDocument.objects.filter(is_active=True)
        missing_files = []

        self.stdout.write("Checking document files...")
        for doc in docs:
            if not doc.file_exists():
                missing_files.append({
                    'type': 'document',
                    'id': doc.id,
                    'path': doc.file_path
                })
                self.stdout.write(self.style.ERROR(
                    f"Missing: Document {doc.id} - {doc.file_path}"
                ))

        # Check QR codes
        submissions = GateEntrySubmission.objects.all()
        self.stdout.write("\nChecking QR code files...")
        for sub in submissions:
            if not sub.qr_exists():
                missing_files.append({
                    'type': 'qr',
                    'id': sub.id,
                    'path': sub.qr_code_path
                })
                self.stdout.write(self.style.ERROR(
                    f"Missing: QR Code {sub.id} - {sub.qr_code_path}"
                ))

        # Summary
        if missing_files:
            self.stdout.write(self.style.ERROR(
                f"\nFound {len(missing_files)} missing files!"
            ))
        else:
            self.stdout.write(self.style.SUCCESS(
                "\nAll files verified successfully!"
            ))
```

**Run verification:**

```bash
python manage.py verify_documents
```

---

## File Serving in Production

### Nginx Configuration

For production, serve files directly via Nginx for better performance:

**nginx.conf:**

```nginx
```

```
server {
    listen 80;
    server_name your-domain.com;

    # Django app
    location / {
        proxy_pass http://localhost:8000;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }

    # Serve documents directly (requires authentication via Django)
    # Use X-Accel-Redirect for protected downloads
    location /protected/ {
        internal;
        alias /var/customer_portal/documents/;
    }

    # Static files
    location /static/ {
        alias /app/staticfiles/;
    }

    # QR codes (if separate from documents)
    location /qr/ {
        internal;
        alias /var/customer_portal/documents/qr_codes/;
    }
}
```

### Django view for protected downloads (documents/views.py):

python

```
from django.http import HttpResponse
import os

@action(detail=True, methods=['get'], url_path='download')
def download_document(self, request, pk=None):
    """
    Secure document download using X-Accel-Redirect (Nginx)
    """
    document = self.get_object()

    if not document.file_exists():
        return Response({
            "error": "File not found"
        }, status=status.HTTP_404_NOT_FOUND)

    # For Nginx X-Accel-Redirect
    response = HttpResponse()
    response['Content-Type'] = ''
    response['X-Accel-Redirect'] = f'/protected/{os.path.relpath(document.file_path, "/var/customer_portal/documents/")}'
    response['Content-Disposition'] = f'attachment; filename="{document.original_filename}"'

    return response
```

### Apache Configuration

### apache.conf:

```
apache
```

```
<VirtualHost *:80>
    ServerName your-domain.com

    # Django app
    ProxyPass / http://localhost:8000/
    ProxyPassReverse / http://localhost:8000/

    # Serve documents with mod_xsendfile
    <Location /documents>
        XSendFile On
        XSendFilePath /var/customer_portal/documents
    </Location>

    # Static files
    Alias /static /app/staticfiles
    <Directory /app/staticfiles>
        Require all granted
    </Directory>
</VirtualHost>
```

---

## Complete Storage Architecture Diagram

```
┌─────────────────────────────────────────────────────────────┐
│         Customer Portal Storage Architecture                 │
└─────────────────────────────────────────────────────────────┘


┌────────────────────┐          ┌────────────────────┐
│   PostgreSQL      │          │  File Storage     │
│    Database      │          │  (Computer)      │
└─────────┬──────────┘          └─────────┬──────────┘
      │                       │
      │ Stores file paths           │ Stores actual files
      │                       │
  ┌─────┴──────────────────────────────┴─────┐
  │                              │
  │  CustomerDocument Table:              │
  │  ┌─────────────────────────────────────┐  │
  │  │ id: 1                        │  │
  │  │ file_path: "/var/customer_...  │──┼──────────┐
  │  │ original_filename: "PO.pdf"    │  │      │
  │  │ file_size: 1048576            │  │      │
  │  │ customer_email: "user@..."     │  │      │
  │  └─────────────────────────────────────┘  │      │
  │                              │      │
  │  GateEntrySubmission Table:          │       │
  │  ┌────────────────────────────────────────┐  │    │
  │  │ id: 10                     │   │       │
  │  │ qr_code_path: "/var/customer/.. │──┼───┐    │
  │  │ customer_email: "user@..."      │  │  │   │
  │  └────────────────────────────────────────┘  │  │   │
  └──────────────────────────────────────────────┘   │   │
                          │   │
  ┌────────────────────────────────────────────────────│────│────┐
  │  File Storage: /var/customer_portal/documents/    │
  │                           │   │    │
  │  documents/                        │   │    │
  │  ├── user_at_example_com/          │   │    │
  │  │   ├── purchase_order/        │   │    │
  │  │   │   └── purchase_order_20240115...pdf◄────┘     │
  │  │   ├── vehicle_registration/          │
  │  │   └── puc/                        │
  │  └── john_at_company_com/                  │
  │                           │       │
  │  qr_codes/                    │       │
  │  ├── qr_MH12AB1234_20240115_103050.png◄───┘         │
  │  └── qr_MH34CD5678_20240115_110030.png          │
  └──────────────────────────────────────────────────────────────┘


Access Flow:
1. User requests document → Django checks DB for file_path
2. Django verifies file exists on storage
3. Django serves file OR uses X-Accel-Redirect (Nginx)
4. File streamed directly from storage to user
```

---

## Production Deployment Checklist

- [ ] Set `DEBUG = False` in production
- [ ] Use strong `SECRET_KEY` and `JWT_SECRET_KEY`
- [ ] Configure allowed hosts properly
- [ ] Use environment variables for all secrets
- [ ] Set up HTTPS/SSL certificates
- [ ] Configure nginx/Apache as reverse proxy
- [ ] Use Gunicorn or uWSGI for WSGI server
- [ ] Set up PostgreSQL with proper authentication
- [ ] Configure static file serving (nginx/S3)
- [ ] Configure media file serving securely
- [ ] Enable database backups

- [ ] Set up logging and monitoring
- [ ] Configure email service (SendGrid/AWS SES)
- [ ] Set up SMS service (Twilio/AWS SNS)
- [ ] Enable CORS only for trusted origins
- [ ] Configure rate limiting
- [ ] Set up CI/CD pipeline
- [ ] Write deployment documentation
- [ ] Test all workflows end-to-end
- [ ] Set up error tracking (Sentry)

---

## License

This project is part of a customer portal system for secure gate entry management.

---

## Support

For issues or questions:

- Check the Troubleshooting section
- Review API documentation above
- Check Django logs: `python manage.py runserver` or Docker logs
- Verify database connectivity
- Ensure all environment variables are set correctly

---

**End of README**