МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА» (БГТУ им. В.Г. Шухова)
Кафедра программного обеспечения
вычислительной техники и автоматизированных
систем

Курсовая работа

по дисциплине: БД
тема: «Разработка веб-приложения для управления и учета оборудования»

Выполнил: студент группы ВТ-231
Ковалев А. В.
Проверили:
Панченко М. В.

Белгород 2025

Цель курсовой работы

Проектирование и разработка информационной системы для автоматизации учета оборудования, включающая создание реляционной базы данных, реализацию интерфейса взаимодействия с БД и обеспечение целостности данных средствами ORM.

Основные задачи:

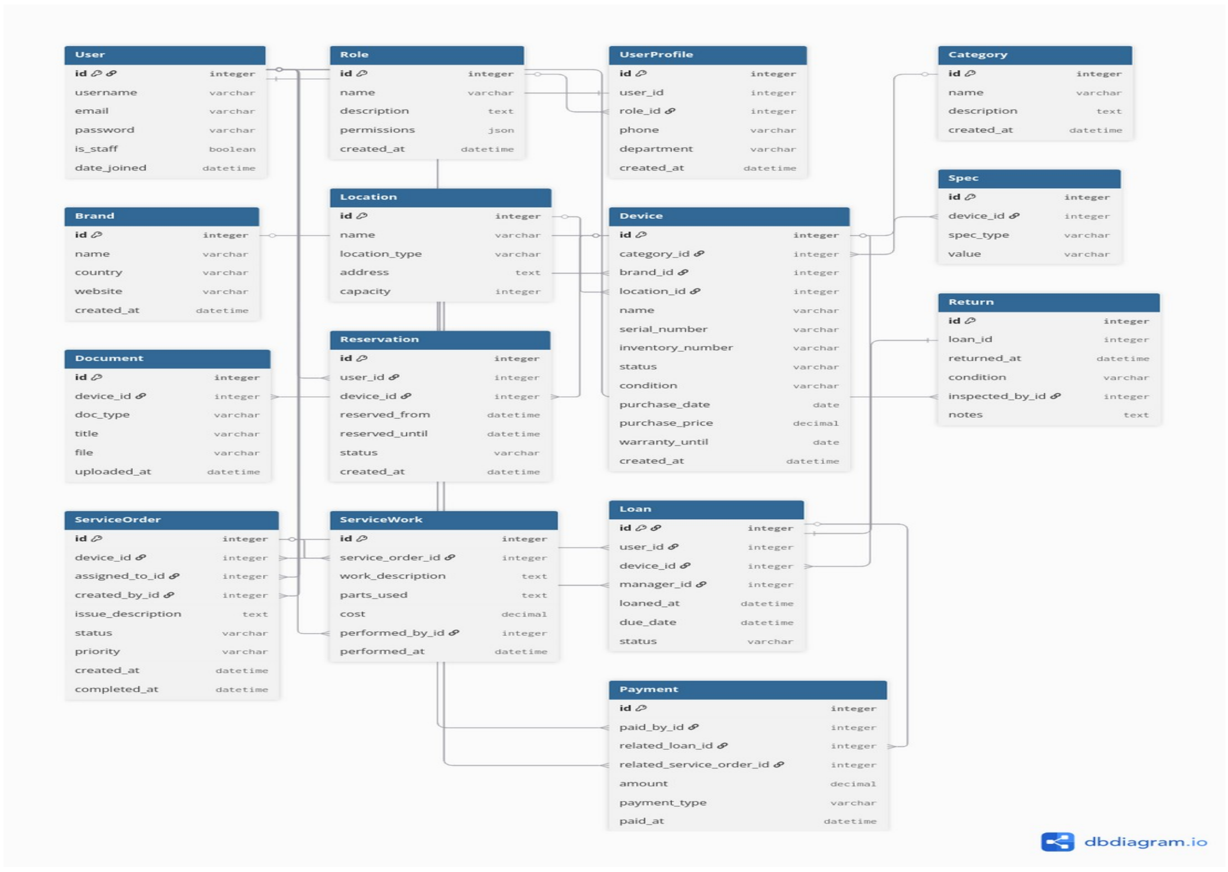
- Проектирование инфологической и даталогической моделей предметной области.
- Реализация операций манипулирования данными (CRUD) для всех сущностей.Обеспечение ссылочной целостности и нормализация базы данных (3НФ).
- Настройка автоматизированного резервного копирования данных.

Стек технологий и инструменты

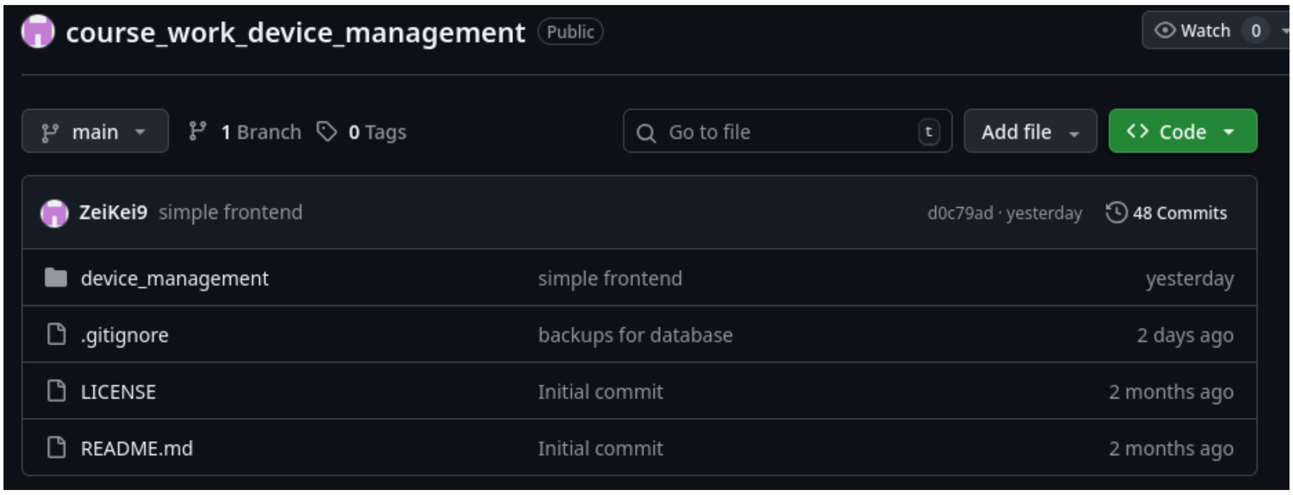
- **Язык программирования:** Python
- **СУБД:** PostgreSQL.
- **Фреймворк для работы с БД:** Django
- **Управление схемой БД:** Django Migrations.
- **ORM:** Django ORM.

Схема базы данных (ER-диаграмма)

Разработанная база данных находится в третьей нормальной форме (3НФ) и состоит из 15 сущностей. Логическая схема связей представлена на рисунке ниже:



Репозиторий с проектом: https://github.com/ZeiKei9/course_work_device_management



ZeiKei9 simple frontend d0c79ad · yesterday 48 Commits

device_management	simple frontend	yesterday
.gitignore	backups for database	2 days ago
LICENSE	Initial commit	2 months ago
README.md	Initial commit	2 months ago

Реализация экспорта данных

В соответствии с заданием реализован механизм выгрузки данных из базы в три формата: **CSV, XLSX (Excel), JSON**.

1) Для **формата CSV** используется ввстроенная библиотека csv

2) Для формата XLSX используется библиотека openpyxl

3) Для формата JSON используется одноименная библиотека и встроенные в джанго сериализаторы

Автоматизированное резервное копирование (Backup)

Для обеспечения сохранности данных реализована подсистема автоматического создания резервных копий базы данных PostgreSQL.

Механизм работы:

1. **Инструмент:** Используется утилита pg_dump, запускаемая через системный вызов (subprocess) внутри Django management command. Т. е. мы вызываем из терминала python manage.py backup_db и у нас автоматически делается бекап
2. **Автоматизация:** Скрипт auto_backup.py может быть добавлен в cron (планировщик задач) для ежедневного выполнения.

Хранение:
В соответствии с требованием размещения копии на удаленном компьютере, система настроена на сохранение файлов в директорию /backups.

Документирование

Так же в течении всего цикла разработки моей системы я активно использовал Swagger UI — интерактивную документацию для проверки работоспособности моих эндпоинтов

Листинг программного кода

1. Файлы конфигурации проекта

Файл: manage.py

```
#!/usr/bin/env python

"""Django's command-line utility for administrative tasks."""

import os

import sys


if __name__ == '__main__':

    os.environ.setdefault('DJANGO_SETTINGS_MODULE',
'device_management.settings')

    try:

        from django.core.management import execute_from_command_line

    except ImportError as exc:

        raise ImportError(

            "Couldn't import Django. Are you sure it's installed and "

            "available on your PYTHONPATH environment variable? Did you "

            "forget to activate a virtual environment?"

        ) from exc

    execute_from_command_line(sys.argv)
```

Файл: device_management/settings.py

```
import os
from datetime import timedelta

from pathlib import Path


from decouple import config


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

SECRET_KEY = config("SECRET_KEY")

DEBUG = config("DEBUG", default=False, cast=bool)


ALLOWED_HOSTS = []


INSTALLED_APPS = [
    "django.contrib.admin",
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.sessions",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "django.contrib.sites",
    "rest_framework",
    "rest_framework.authtoken",
    "corsheaders",
    "allauth",
    "allauth.account",
    "allauth.socialaccount",
    "django_filters",
    "drf_spectacular",
    "dbbackup",
    "devices",
    "users",
    "services",
]

SITE_ID = 1

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

```
ROOT_URLCONF = "device_management.urls"

TEMPLATES = [
    {
        "BACKEND": "django.template.backends.django.DjangoTemplates",
        "DIRS": [BASE_DIR / "templates"],
        "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",
            ],
        },
    },
]

STATIC_URL = "/static/"
STATICFILES_DIRS = [BASE_DIR / "static"]
STATIC_ROOT = BASE_DIR / "staticfiles"

WSGI_APPLICATION = "device_management.wsgi.application"



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"),
    }
}



AUTH_PASSWORD_VALIDATORS = [
    {
        "NAME":
"django.contrib.auth.password_validation.UserAttributeSimilarityValidator",
    },
    {
        "NAME":
"django.contrib.auth.password_validation.MinimumLengthValidator",
    },
    {
        "NAME":
"django.contrib.auth.password_validation.CommonPasswordValidator",
    },
    {
        "NAME":
"django.contrib.auth.password_validation.NumericPasswordValidator",
    },
]



LANGUAGE_CODE = "en-us"


TIME_ZONE = "UTC"


USE_I18N = True


USE_TZ = True


DEFAULT_AUTO_FIELD = "django.db.models.BigAutoField"


REST_FRAMEWORK = {
    "DEFAULT_AUTHENTICATION_CLASSES": [
        "rest_framework_simplejwt.authentication.JWTAuthentication",
        "rest_framework.authentication.SessionAuthentication",
    ],
    "DEFAULT_PERMISSION_CLASSES": [
        "rest_framework.permissions.IsAuthenticatedOrReadOnly",
    ],
    "DEFAULT_PAGINATION_CLASS":
"rest_framework.pagination.PageNumberPagination",
    "PAGE_SIZE": 20,
    "DEFAULT_RENDERER_CLASSES": [
        "rest_framework.renderers.JSONRenderer",
        "rest_framework.renderers.BrowsableAPIRenderer",
    ],
    "DEFAULT_SCHEMA_CLASS": "drf_spectacular.openapi.AutoSchema",
}

CORS_ALLOWED_ORIGINS = [
    "http://localhost:3000",
    "http://127.0.0.1:3000",
]

CORS_ALLOW_CREDENTIALS = True

SIMPLE_JWT = {
    "ACCESS_TOKEN_LIFETIME": timedelta(hours=5),
    "REFRESH_TOKEN_LIFETIME": timedelta(days=1),
    "ROTATE_REFRESH_TOKENS": False,
    "BLACKLIST_AFTER_ROTATION": True,
    "ALGORITHM": "HS256",
    "SIGNING_KEY": SECRET_KEY,
    "AUTH_HEADER_TYPES": ("Bearer",),
}

AUTHENTICATION_BACKENDS = [
    "django.contrib.auth.backends.ModelBackend",
    "allauth.account.auth_backends.AuthenticationBackend",
]

ACCOUNT_LOGIN_METHODS = {"username"}
ACCOUNT_SIGNUP_FIELDS = ["email", "username*", "password1*", "password2*"]

MEDIA_URL = "/media/"
MEDIA_ROOT = BASE_DIR / "media"

SPECTACULAR_SETTINGS = {
    "TITLE": "Device Management System API",
    "DESCRIPTION": "API для системы учёта техники",
    "VERSION": "1.0.0",
    "SERVE_INCLUDE_SCHEMA": False,
    "COMPONENT_SPLIT_REQUEST": True,
}



DBBACKUP_STORAGE = "django.core.files.storage.FileSystemStorage"
DBBACKUP_STORAGE_OPTIONS = {"location": BASE_DIR / "backups"}

DBBACKUP_DATABASES = ["default"]

DBBACKUP_CLEANUP_KEEP = 10
DBBACKUP_CLEANUP_KEEP_MEDIA = 10

DBBACKUP_FILENAME_TEMPLATE = "backup_{databasename}_{datetime}.{extension}"
DBBACKUP_MEDIA_FILENAME_TEMPLATE = "media_{datetime}.{extension}"
```

Файл: device_management/urls.py

```
from devices.template_views import device_catalog_view, home_view
from django.conf import settings

from django.conf.urls.static import static

from django.contrib import admin
```

```python
from django.urls import include, path
from drf_spectacular.views import (
    SpectacularAPIView,
    SpectacularRedocView,
    SpectacularSwaggerView,
)
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

urlpatterns = [
    path("", home_view, name="home"),
    path("catalog/", device_catalog_view, name="device_catalog"),
    path("admin/", admin.site.urls),
    path("api/token/", TokenObtainPairView.as_view(), name="token_obtain_pair"),
    path("api/token/refresh/", TokenRefreshView.as_view(), name="token_refresh"),
    path("api/auth/", include("allauth.urls")),
    path("api/", include("devices.urls")),
    path("api/", include("users.urls")),
    path("api/", include("services.urls")),
    path("api/schema/", SpectacularAPIView.as_view(), name="schema"),
    path(
        "api/docs/",
        SpectacularSwaggerView.as_view(url_name="schema"),
        name="swagger-ui",
    ),
    path("api/redoc/", SpectacularRedocView.as_view(url_name="schema"), name="redoc"),
]

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

Файл: device_management/asgi.py

```
import os
from django.core.asgi import get_asgi_application


os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'device_management.settings')

application = get_asgi_application()
```

Файл: device_management/wsgi.py

```
import os
from django.core.wsgi import get_wsgi_application


os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'device_management.settings')

application = get_wsgi_application()
```

2. Приложение Devices (Ядро системы)

Файл: devices/admin.py

```
from django.contrib import admin
from .models import (

    Category,

    Brand,

    Location,

    Device,

    Spec,

    Document,

    Reservation,

    Loan,

    Return,

)


@admin.register(Category)

class CategoryAdmin(admin.ModelAdmin):

    list_display = ["name", "created_at", "updated_at"]

    search_fields = ["name", "description"]

    list_filter = ["created_at"]
```

```python
@admin.register(Brand)
class BrandAdmin(admin.ModelAdmin):
    list_display = ["name", "country", "website", "created_at"]
    search_fields = ["name", "country"]
    list_filter = ["country", "created_at"]


@admin.register(Location)
class LocationAdmin(admin.ModelAdmin):
    list_display = ["name", "location_type", "capacity", "created_at"]
    search_fields = ["name", "address"]
    list_filter = ["location_type", "created_at"]


@admin.register(Device)
class DeviceAdmin(admin.ModelAdmin):
    list_display = [
        "name",
        "serial_number",
        "category",
        "brand",
        "status",
        "condition",
        "location",
        "created_at",
    ]
    search_fields = ["name", "serial_number", "inventory_number"]
    list_filter = ["status", "condition", "category", "brand", "created_at"]
    readonly_fields = ["created_at", "updated_at"]
    fieldsets = [
        (
            "Basic Information",
            {
                "fields": [
                    "name",
                    "serial_number",
                    "inventory_number",
                    "category",
                    "brand",
                ]
            },
        ),
        ("Status", {"fields": ["status", "condition", "location"]}),
        (
            "Purchase Information",
            {"fields": ["purchase_date", "purchase_price", "warranty_until"]},
        ),
        ("Additional", {"fields": ["notes", "created_at", "updated_at"]}),
    ]


@admin.register(Spec)
class SpecAdmin(admin.ModelAdmin):
    list_display = ["device", "spec_type", "value", "created_at"]
    search_fields = ["device__name", "value"]
    list_filter = ["spec_type", "created_at"]


@admin.register(Document)
class DocumentAdmin(admin.ModelAdmin):
    list_display = ["device", "doc_type", "title", "uploaded_at"]
    search_fields = ["device__name", "title"]
    list_filter = ["doc_type", "uploaded_at"]


@admin.register(Reservation)
class ReservationAdmin(admin.ModelAdmin):
    list_display = [
        "user",
        "device",
        "reserved_from",
        "reserved_until",
        "status",
        "created_at",
    ]
    search_fields = ["user__username", "device__name"]
    list_filter = ["status", "created_at"]
    readonly_fields = ["created_at", "updated_at"]


@admin.register(Loan)
class LoanAdmin(admin.ModelAdmin):
    list_display = ["user", "device", "manager", "loaned_at", "due_date",
"status"]
    search_fields = ["user__username", "device__name", "manager__username"]
    list_filter = ["status", "loaned_at"]
    readonly_fields = ["loaned_at", "created_at", "updated_at"]


@admin.register(Return)
class ReturnAdmin(admin.ModelAdmin):
    list_display = ["loan", "returned_at", "condition", "inspected_by"]
    search_fields = ["loan__user__username", "loan__device__name"]
    list_filter = ["condition", "returned_at"]
    readonly_fields = ["returned_at", "created_at"]
```

Файл: devices/apps.py

```
from django.apps import AppConfig


class DevicesConfig(AppConfig):
    default_auto_field = "django.db.models.BigAutoField"
    name = "devices"
    def ready(self):
        import devices.signals
```

Файл: devices/models.py

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


class Category(models.Model):
    name = models.CharField(max_length=100, unique=True)
    description = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Category"
        verbose_name_plural = "Categories"
        ordering = ["name"]

    def __str__(self):
        return self.name


class Brand(models.Model):
    name = models.CharField(max_length=100, unique=True)
    country = models.CharField(max_length=100, blank=True, null=True)
    website = models.URLField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Brand"
        verbose_name_plural = "Brands"
        ordering = ["name"]

    def __str__(self):
        return self.name


class Location(models.Model):
    LOCATION_TYPES = [
        ("WAREHOUSE", "Warehouse"),
        ("OFFICE", "Office"),
        ("STORAGE", "Storage"),
    ]

    name = models.CharField(max_length=100)
```

```python
    location_type = models.CharField(
        max_length=20, choices=LOCATION_TYPES, default="WAREHOUSE"
    )
    address = models.TextField(blank=True, null=True)
    capacity = models.IntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Location"
        verbose_name_plural = "Locations"
        ordering = ["name"]

    def __str__(self):
        return f"{self.name} ({self.get_location_type_display()})"


class Device(models.Model):
    STATUS_CHOICES = [
        ("AVAILABLE", "Available"),
        ("RESERVED", "Reserved"),
        ("LOANED", "Loaned"),
        ("IN_SERVICE", "In Service"),
        ("RETIRED", "Retired"),
    ]

    CONDITION_CHOICES = [
        ("EXCELLENT", "Excellent"),
        ("GOOD", "Good"),
        ("FAIR", "Fair"),
        ("POOR", "Poor"),
    ]

    name = models.CharField(max_length=200)
    serial_number = models.CharField(max_length=100, unique=True)
    inventory_number = models.CharField(max_length=100, unique=True)
    category = models.ForeignKey(
        Category, on_delete=models.PROTECT, related_name="devices"
    )
    brand = models.ForeignKey(Brand, on_delete=models.PROTECT,
related_name="devices")
    location = models.ForeignKey(
        Location,
        on_delete=models.SET_NULL,
        null=True,
        blank=True,
        related_name="devices",
    )
    status = models.CharField(
        max_length=20, choices=STATUS_CHOICES, default="AVAILABLE"
    )
    purchase_date = models.DateField(null=True, blank=True)
    purchase_price = models.DecimalField(
        max_digits=10, decimal_places=2, null=True, blank=True
    )
    warranty_until = models.DateField(null=True, blank=True)
    condition = models.CharField(
        max_length=20, choices=CONDITION_CHOICES, default="GOOD"
    )
    notes = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Device"
        verbose_name_plural = "Devices"
        ordering = ["-created_at"]

    def __str__(self):
        return f"{self.name} ({self.serial_number})"

    def is_available(self):
        return self.status == "AVAILABLE"

    def get_specifications(self):
        return self.specifications.all()

    def get_full_name(self):
        return f"{self.brand.name} {self.name}"


class Spec(models.Model):
    SPEC_TYPE_CHOICES = [
        ("CPU", "Processor"),
        ("RAM", "RAM"),
        ("STORAGE", "Storage"),
        ("SCREEN", "Screen Size"),
        ("COLOR", "Color"),
        ("OS", "Operating System"),
        ("OTHER", "Other"),
    ]

    device = models.ForeignKey(
        Device, on_delete=models.CASCADE, related_name="specifications"
    )
    spec_type = models.CharField(max_length=20, choices=SPEC_TYPE_CHOICES)
    value = models.CharField(max_length=200)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = "Specification"
        verbose_name_plural = "Specifications"
        ordering = ["spec_type"]

    def __str__(self):
        return f"{self.device.name} - {self.get_spec_type_display()}: 
{self.value}"


class Document(models.Model):
    DOC_TYPE_CHOICES = [
        ("MANUAL", "Manual"),
        ("WARRANTY", "Warranty"),
        ("RECEIPT", "Receipt"),
        ("CONTRACT", "Contract"),
        ("OTHER", "Other"),
    ]

    device = models.ForeignKey(
        Device, on_delete=models.CASCADE, related_name="documents"
    )
    doc_type = models.CharField(max_length=20, choices=DOC_TYPE_CHOICES)
    title = models.CharField(max_length=200)
    file = models.FileField(upload_to="documents/%Y/%m/%d/")
    description = models.TextField(blank=True, null=True)
    uploaded_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = "Document"
        verbose_name_plural = "Documents"
        ordering = ["-uploaded_at"]

    def __str__(self):
        return f"{self.device.name} - {self.get_doc_type_display()}: 
{self.title}"


class Reservation(models.Model):
    STATUS_CHOICES = [
        ("ACTIVE", "Active"),
        ("CANCELLED", "Cancelled"),
        ("COMPLETED", "Completed"),
    ]

    user = models.ForeignKey(
        User, on_delete=models.CASCADE, related_name="reservations"
    )
    device = models.ForeignKey(
        Device, on_delete=models.CASCADE, related_name="reservations"
    )
    reserved_from = models.DateTimeField()
    reserved_until = models.DateTimeField()
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default="ACTIVE")
    notes = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Reservation"
        verbose_name_plural = "Reservations"
        ordering = ["-created_at"]

    def __str__(self):
        return f"{self.user.username} - {self.device.name} ({self.status})"


class Loan(models.Model):
    STATUS_CHOICES = [
        ("ACTIVE", "Active"),
        ("OVERDUE", "Overdue"),
        ("RETURNED", "Returned"),
    ]

    user = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="loans")
    device = models.ForeignKey(Device, on_delete=models.CASCADE,
related_name="loans")
    manager = models.ForeignKey(
        User, on_delete=models.SET_NULL, null=True, related_name="managed_loans"
    )
    loaned_at = models.DateTimeField(auto_now_add=True)
    due_date = models.DateTimeField()
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default="ACTIVE")
    notes = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

    class Meta:
        verbose_name = "Loan"
        verbose_name_plural = "Loans"
        ordering = ["-loaned_at"]

    def __str__(self):
        return f"{self.user.username} - {self.device.name} ({self.status})"


class Return(models.Model):
    loan = models.OneToOneField(
        Loan, on_delete=models.CASCADE, related_name="return_record"
    )
    returned_at = models.DateTimeField(auto_now_add=True)
    condition = models.CharField(max_length=20,
choices=Device.CONDITION_CHOICES)
    inspected_by = models.ForeignKey(
        User, on_delete=models.SET_NULL, null=True,
related_name="inspected_returns"
    )
    notes = models.TextField(blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name = "Return"
        verbose_name_plural = "Returns"
        ordering = ["-returned_at"]

    def __str__(self):
        return f"Return: {self.loan.device.name} by {self.loan.user.username}"
```

Файл: devices/permissions.py

```
from rest_framework import permissions


class IsAdminOrReadOnly(permissions.BasePermission):
    def has_permission(self, request, view):
        if request.method in permissions.SAFE_METHODS:
            return True
        return request.user and request.user.is_staff


class IsManagerOrAdmin(permissions.BasePermission):
    def has_permission(self, request, view):
        if not request.user or not request.user.is_authenticated:
            return False

        if request.user.is_staff:
            return True

        try:
            profile = request.user.profile
            return profile.role and profile.role.name in ["MANAGER", "ADMIN"]
        except:
            return False


class IsOwnerOrManager(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.user.is_staff:
            return True

        try:
            profile = request.user.profile
            if profile.role and profile.role.name in ["MANAGER", "ADMIN"]:
                return True
        except:
            pass

        if hasattr(obj, "user"):
            return obj.user == request.user

        return False
```

Файл: devices/signals.py

```
from django.db.models.signals import post_save, pre_save
from django.dispatch import receiver
from django.utils import timezone
from .models import Loan, Return, Device


@receiver(post_save, sender=Loan)
def update_device_status_on_loan(sender, instance, created, **kwargs):
    if created:
        device = instance.device
        device.status = "LOANED"
        device.save()


@receiver(post_save, sender=Return)
def update_device_status_on_return(sender, instance, created, **kwargs):
    if created:
        loan = instance.loan
        loan.status = "RETURNED"
        loan.save()

        device = loan.device
        device.status = "AVAILABLE"
        device.condition = instance.condition
        device.save()


@receiver(pre_save, sender=Loan)
def check_overdue_loan(sender, instance, **kwargs):
    if instance.pk:
        if instance.status == "ACTIVE" and instance.due_date < timezone.now():
            instance.status = "OVERDUE"
```

Файл: devices/template_views.py

```
from django.shortcuts import render
```

```
def home_view(request):
    return render(request, "home.html")


def device_catalog_view(request):
    return render(request, "device_catalog.html")
```

Файл: devices/urls.py

```
from django.urls import include, path
from rest_framework.routers import DefaultRouter
from .views import BrandViewSet, CategoryViewSet, DeviceViewSet, LoanViewSet,
LocationViewSet, ReservationViewSet, ReturnViewSet

router = DefaultRouter()
router.register("categories", CategoryViewSet)
router.register("brands", BrandViewSet)
router.register("locations", LocationViewSet)
router.register("devices", DeviceViewSet)
router.register("reservations", ReservationViewSet)
router.register("loans", LoanViewSet)
router.register("returns", ReturnViewSet)

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

Файл: devices/utils.py

```
import csv
import json

from django.http import HttpResponse
from openpyxl import Workbook
from openpyxl.styles import Alignment, Font, PatternFill


def export_devices_to_csv(queryset):
    response = HttpResponse(content_type="text/csv")
    response["Content-Disposition"] = 'attachment; filename="devices.csv"'
    response.write("\ufeff".encode("utf8"))

    writer = csv.writer(response)
    writer.writerow(
        [
            "ID",
            "Name",
            "Serial Number",
            "Inventory Number",
            "Category",
            "Brand",
            "Status",
            "Condition",
            "Location",
            "Purchase Date",
            "Purchase Price",
            "Warranty Until",
            "Created At",
        ]
    )

    for device in queryset:
        writer.writerow(
            [
                device.id,
                device.name,
                device.serial_number,
                device.inventory_number,
                device.category.name if device.category else "",
                device.brand.name if device.brand else "",
                device.get_status_display(),
                device.get_condition_display(),
                device.location.name if device.location else "",
                device.purchase_date,
                device.purchase_price,
                device.warranty_until,
                device.created_at.strftime("%Y-%m-%d %H:%M:%S"),
            ]
        )

    return response


def export_loans_to_csv(queryset):
    response = HttpResponse(content_type="text/csv")
    response["Content-Disposition"] = 'attachment; filename="loans.csv"'
    response.write("\ufeff".encode("utf8"))

    writer = csv.writer(response)
    writer.writerow(
        [
            "ID",
            "User",
            "Device",
            "Serial Number",
            "Manager",
            "Loaned At",
            "Due Date",
            "Status",
        ]
    )

    for loan in queryset:
```

```
        writer.writerow(
            [
                loan.id,
                loan.user.username,
                loan.device.name,
                loan.device.serial_number,
                loan.manager.username if loan.manager else "",
                loan.loaned_at.strftime("%Y-%m-%d %H:%M:%S"),
                loan.due_date.strftime("%Y-%m-%d %H:%M:%S"),
                loan.get_status_display(),
            ]
        )

    return response


def export_devices_to_excel(queryset):
    response = HttpResponse(
        content_type="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"
    )
    response["Content-Disposition"] = 'attachment; filename="devices.xlsx"'

    wb = Workbook()
    ws = wb.active
    ws.title = "Devices"

    headers = [
        "ID",
        "Name",
        "Serial Number",
        "Inventory Number",
        "Category",
        "Brand",
        "Status",
        "Condition",
        "Location",
        "Purchase Date",
        "Purchase Price",
        "Warranty Until",
        "Created At",
    ]

    header_fill = PatternFill(
        start_color="366092", end_color="366092", fill_type="solid"
    )
    header_font = Font(bold=True, color="FFFFFF")

    for col_num, header in enumerate(headers, 1):
        cell = ws.cell(row=1, column=col_num)
        cell.value = header
```

```
            cell.fill = header_fill
            cell.font = header_font
            cell.alignment = Alignment(horizontal="center", vertical="center")

    for row_num, device in enumerate(queryset, 2):
        ws.cell(row=row_num, column=1).value = device.id
        ws.cell(row=row_num, column=2).value = device.name
        ws.cell(row=row_num, column=3).value = device.serial_number
        ws.cell(row=row_num, column=4).value = device.inventory_number
        ws.cell(row=row_num, column=5).value = (
            device.category.name if device.category else ""
        )
        ws.cell(row=row_num, column=6).value = device.brand.name if device.brand
else ""
        ws.cell(row=row_num, column=7).value = device.get_status_display()
        ws.cell(row=row_num, column=8).value = device.get_condition_display()
        ws.cell(row=row_num, column=9).value = (
            device.location.name if device.location else ""
        )
        ws.cell(row=row_num, column=10).value = device.purchase_date
        ws.cell(row=row_num, column=11).value = device.purchase_price
        ws.cell(row=row_num, column=12).value = device.warranty_until
        ws.cell(row=row_num, column=13).value = device.created_at.strftime(
            "%Y-%m-%d %H:%M:%S"
        )

    for col in ws.columns:
        max_length = 0
        column = col[0].column_letter
        for cell in col:
            try:
                if len(str(cell.value)) > max_length:
                    max_length = len(cell.value)
            except:
                pass
        adjusted_width = max_length + 2
        ws.column_dimensions[column].width = adjusted_width

    wb.save(response)
    return response


def export_loans_to_excel(queryset):
    response = HttpResponse(
        content_type="application/vnd.openxmlformats-
officedocument.spreadsheetml.sheet"
    )
    response["Content-Disposition"] = 'attachment; filename="loans.xlsx"'

    wb = Workbook()
    ws = wb.active
```

```python
ws.title = "Loans"

headers = [
    "ID",
    "User",
    "Device",
    "Serial Number",
    "Manager",
    "Loaned At",
    "Due Date",
    "Status",
]

header_fill = PatternFill(
    start_color="366092", end_color="366092", fill_type="solid"
)
header_font = Font(bold=True, color="FFFFFF")

for col_num, header in enumerate(headers, 1):
    cell = ws.cell(row=1, column=col_num)
    cell.value = header
    cell.fill = header_fill
    cell.font = header_font
    cell.alignment = Alignment(horizontal="center", vertical="center")

for row_num, loan in enumerate(queryset, 2):
    ws.cell(row=row_num, column=1).value = loan.id
    ws.cell(row=row_num, column=2).value = loan.user.username
    ws.cell(row=row_num, column=3).value = loan.device.name
    ws.cell(row=row_num, column=4).value = loan.device.serial_number
    ws.cell(row=row_num, column=5).value = (
        loan.manager.username if loan.manager else ""
    )
    ws.cell(row=row_num, column=6).value = loan.loaned_at.strftime(
        "%Y-%m-%d %H:%M:%S"
    )
    ws.cell(row=row_num, column=7).value = loan.due_date.strftime(
        "%Y-%m-%d %H:%M:%S"
    )
    ws.cell(row=row_num, column=8).value = loan.get_status_display()

for col in ws.columns:
    max_length = 0
    column = col[0].column_letter
    for cell in col:
        try:
            if len(str(cell.value)) > max_length:
                max_length = len(cell.value)
        except:
            pass
    adjusted_width = max_length + 2
```

```
            ws.column_dimensions[column].width = adjusted_width

        wb.save(response)
        return response


    def export_devices_to_json(queryset):
        response = HttpResponse(content_type="application/json")
        response["Content-Disposition"] = 'attachment; filename="devices.json"'

        devices_data = []
        for device in queryset:
            device_dict = {
                "id": device.id,
                "name": device.name,
                "serial_number": device.serial_number,
                "inventory_number": device.inventory_number,
                "category": {"id": device.category.id, "name": device.category.name}
                if device.category
                else None,
                "brand": {"id": device.brand.id, "name": device.brand.name}
                if device.brand
                else None,
                "status": device.status,
                "status_display": device.get_status_display(),
                "condition": device.condition,
                "condition_display": device.get_condition_display(),
                "location": {"id": device.location.id, "name": device.location.name}
                if device.location
                else None,
                "purchase_date": str(device.purchase_date)
                if device.purchase_date
                else None,
                "purchase_price": str(device.purchase_price)
                if device.purchase_price
                else None,
                "warranty_until": str(device.warranty_until)
                if device.warranty_until
                else None,
                "created_at": device.created_at.isoformat(),
            }
            devices_data.append(device_dict)

        response.write(json.dumps(devices_data, indent=2, ensure_ascii=False))
        return response


    def export_loans_to_json(queryset):
        response = HttpResponse(content_type="application/json")
        response["Content-Disposition"] = 'attachment; filename="loans.json"'
        loans_data = []
        for loan in queryset:
            loan_dict = {
                "id": loan.id,
                "user": {
                    "id": loan.user.id,
                    "username": loan.user.username,
                    "email": loan.user.email,
                },
                "device": {
                    "id": loan.device.id,
                    "name": loan.device.name,
                    "serial_number": loan.device.serial_number,
                },
                "manager": {"id": loan.manager.id, "username":
    loan.manager.username}
                if loan.manager
                else None,
                "loaned_at": loan.loaned_at.isoformat(),
                "due_date": loan.due_date.isoformat(),
                "status": loan.status,
                "status_display": loan.get_status_display(),
                "notes": loan.notes,
            }
            loans_data.append(loan_dict)

        response.write(json.dumps(loans_data, indent=2, ensure_ascii=False))
        return response
```

Файл: devices/views.py

```
from django_filters.rest_framework import DjangoFilterBackend
from rest_framework import filters, status, viewsets
from rest_framework.decorators import action
from rest_framework.response import Response

from .models import Brand, Category, Device, Loan, Location, Reservation, Return
from .permissions import IsAdminOrReadOnly, IsManagerOrAdmin, IsOwnerOrManager
from .serializers import (
    BrandSerializer,
    CategorySerializer,
    DeviceListSerializer,
    DeviceSerializer,
    LoanSerializer,
    LocationSerializer,
    ReservationSerializer,
    ReturnSerializer,
)
from .utils import (
    export_devices_to_csv,
```

```python
    export_devices_to_excel,
    export_devices_to_json,
    export_loans_to_csv,
    export_loans_to_excel,
    export_loans_to_json,
)


class CategoryViewSet(viewsets.ModelViewSet):
    queryset = Category.objects.all()
    serializer_class = CategorySerializer
    permission_classes = [IsAdminOrReadOnly]
    filter_backends = [filters.SearchFilter, filters.OrderingFilter]
    search_fields = ["name", "description"]
    ordering_fields = ["name", "created_at"]
    ordering = ["name"]


class BrandViewSet(viewsets.ModelViewSet):
    queryset = Brand.objects.all()
    serializer_class = BrandSerializer
    permission_classes = [IsAdminOrReadOnly]
    filter_backends = [
        filters.SearchFilter,
        filters.OrderingFilter,
        DjangoFilterBackend,
    ]
    search_fields = ["name", "country"]
    ordering_fields = ["name", "created_at"]
    ordering = ["name"]
    filterset_fields = ["country"]


class LocationViewSet(viewsets.ModelViewSet):
    queryset = Location.objects.all()
    serializer_class = LocationSerializer
    permission_classes = [IsAdminOrReadOnly]
    filter_backends = [
        filters.SearchFilter,
        filters.OrderingFilter,
        DjangoFilterBackend,
    ]
    search_fields = ["name", "address"]
    ordering_fields = ["name", "created_at"]
    ordering = ["name"]
    filterset_fields = ["location_type"]


class DeviceViewSet(viewsets.ModelViewSet):
    queryset = Device.objects.select_related(
        "category", "brand", "location"
```

```
).prefetch_related("specifications", "documents")
serializer_class = DeviceSerializer
permission_classes = [IsManagerOrAdmin]
filter_backends = [
    filters.SearchFilter,
    filters.OrderingFilter,
    DjangoFilterBackend,
]
search_fields = ["name", "serial_number", "inventory_number"]
ordering_fields = ["name", "created_at", "purchase_date"]
ordering = ["-created_at"]
filterset_fields = ["category", "brand", "status", "condition", "location"]

def get_serializer_class(self):
    if self.action == "list":
        return DeviceListSerializer
    return DeviceSerializer

def get_permissions(self):
    if self.action in ["list", "retrieve", "available", "by_serial"]:
        return [IsAdminOrReadOnly()]
    return [IsManagerOrAdmin()]

@action(detail=False, methods=["get"])
def available(self, request):
    available_devices = self.queryset.filter(status="AVAILABLE")
    serializer = DeviceListSerializer(available_devices, many=True)
    return Response(serializer.data)

@action(detail=False, methods=["get"])
def by_serial(self, request):
    serial = request.query_params.get("serial", None)
    if not serial:
        return Response(
            {"error": "Serial number is required"},
            status=status.HTTP_400_BAD_REQUEST,
        )

    try:
        device = Device.objects.get(serial_number=serial)
        serializer = DeviceSerializer(device)
        return Response(serializer.data)
    except Device.DoesNotExist:
        return Response(
            {"error": "Device not found"}, status=status.HTTP_404_NOT_FOUND
        )

@action(detail=True, methods=["get"])
def history(self, request, pk=None):
    device = self.get_object()
    loans = device.loans.all().order_by("-loaned_at")
```

```
        reservations = device.reservations.all().order_by("-created_at")
        service_orders = device.service_orders.all().order_by("-created_at")

        return Response(
            {
                "device": DeviceSerializer(device).data,
                "loans_count": loans.count(),
                "reservations_count": reservations.count(),
                "service_orders_count": service_orders.count(),
            }
        )

    @action(detail=False, methods=["get"])
    def export_csv(self, request):
        queryset = self.filter_queryset(self.get_queryset())
        return export_devices_to_csv(queryset)

    @action(detail=False, methods=["get"])
    def export_excel(self, request):
        queryset = self.filter_queryset(self.get_queryset())
        return export_devices_to_excel(queryset)

    @action(detail=False, methods=["get"])
    def export_json(self, request):
        queryset = self.filter_queryset(self.get_queryset())
        return export_devices_to_json(queryset)

    @action(detail=False, methods=["get"])
    def export(self, request):
        format_type = request.query_params.get("format", "csv").lower()
        queryset = self.filter_queryset(self.get_queryset())

        if format_type == "csv":
            return export_devices_to_csv(queryset)
        elif format_type == "xlsx":
            return export_devices_to_excel(queryset)
        elif format_type == "json":
            return export_devices_to_json(queryset)
        else:
            return Response(
                {"error": "Invalid format. Use csv, xlsx, or json"},
                status=status.HTTP_400_BAD_REQUEST,
            )


class ReservationViewSet(viewsets.ModelViewSet):
    queryset = Reservation.objects.select_related("user", "device").all()
    serializer_class = ReservationSerializer
    permission_classes = [IsOwnerOrManager]
    filter_backends = [
        filters.SearchFilter,
        filters.OrderingFilter,
        DjangoFilterBackend,
    ]
    search_fields = ["user__username", "device__name", "device__serial_number"]
    ordering_fields = ["created_at", "reserved_from", "reserved_until"]
    ordering = ["-created_at"]
    filterset_fields = ["status", "user", "device"]

    def perform_create(self, serializer):
        serializer.save(user=self.request.user)

    def get_queryset(self):
        if self.request.user.is_staff:
            return self.queryset
        try:
            profile = self.request.user.profile
            if profile.role and profile.role.name in ["MANAGER", "ADMIN"]:
                return self.queryset
        except:
            pass
        return self.queryset.filter(user=self.request.user)

    @action(detail=False, methods=["get"])
    def my_reservations(self, request):
        reservations = self.queryset.filter(user=request.user)
        serializer = self.get_serializer(reservations, many=True)
        return Response(serializer.data)

    @action(detail=True, methods=["post"])
    def cancel(self, request, pk=None):
        reservation = self.get_object()
        if reservation.status == "ACTIVE":
            reservation.status = "CANCELLED"
            reservation.save()
            return Response({"status": "Reservation cancelled"})
        return Response(
            {"error": "Can only cancel active reservations"},
            status=status.HTTP_400_BAD_REQUEST,
        )


class LoanViewSet(viewsets.ModelViewSet):
    queryset = Loan.objects.select_related("user", "device", "manager").all()
    serializer_class = LoanSerializer
    permission_classes = [IsManagerOrAdmin]
    filter_backends = [
        filters.SearchFilter,
        filters.OrderingFilter,
        DjangoFilterBackend,
    ]
    search_fields = ["user__username", "device__name", "device__serial_number"]
```

```python
ordering_fields = ["loaned_at", "due_date"]
ordering = ["-loaned_at"]
filterset_fields = ["status", "user", "device"]

def get_permissions(self):
    if self.action in ["my_loans"]:
        from rest_framework.permissions import IsAuthenticated

        return [IsAuthenticated()]
    return [IsManagerOrAdmin()]

def get_queryset(self):
    if self.action == "my_loans":
        return self.queryset.filter(user=self.request.user)

    if self.request.user.is_staff:
        return self.queryset

    try:
        profile = self.request.user.profile
        if profile.role and profile.role.name in ["MANAGER", "ADMIN"]:
            return self.queryset
    except:
        pass

    return self.queryset.filter(user=self.request.user)

def perform_create(self, serializer):
    serializer.save(manager=self.request.user)

@action(detail=False, methods=["get"])
def active(self, request):
    active_loans = self.get_queryset().filter(status="ACTIVE")
    serializer = self.get_serializer(active_loans, many=True)
    return Response(serializer.data)

@action(detail=False, methods=["get"])
def overdue(self, request):
    overdue_loans = self.get_queryset().filter(status="OVERDUE")
    serializer = self.get_serializer(overdue_loans, many=True)
    return Response(serializer.data)

@action(detail=False, methods=["get"])
def my_loans(self, request):
    my_loans = self.queryset.filter(user=request.user)
    serializer = self.get_serializer(my_loans, many=True)
    return Response(serializer.data)

@action(detail=False, methods=["get"])
def export_csv(self, request):
    queryset = self.filter_queryset(self.get_queryset())
```

```python
        return export_loans_to_csv(queryset)

    @action(detail=False, methods=["get"])
    def export_excel(self, request):
        queryset = self.filter_queryset(self.get_queryset())
        return export_loans_to_excel(queryset)

    @action(detail=False, methods=["get"])
    def export_json(self, request):
        queryset = self.filter_queryset(self.get_queryset())
        return export_loans_to_json(queryset)

    @action(detail=False, methods=["get"])
    def export(self, request):
        format_type = request.query_params.get("format", "csv").lower()
        queryset = self.filter_queryset(self.get_queryset())

        if format_type == "csv":
            return export_loans_to_csv(queryset)
        elif format_type == "xlsx":
            return export_loans_to_excel(queryset)
        elif format_type == "json":
            return export_loans_to_json(queryset)
        else:
            return Response(
                {"error": "Invalid format. Use csv, xlsx, or json"},
                status=status.HTTP_400_BAD_REQUEST,
            )


class ReturnViewSet(viewsets.ModelViewSet):
    queryset = Return.objects.select_related(
        "loan", "loan__device", "loan__user", "inspected_by"
    ).all()
    serializer_class = ReturnSerializer
    permission_classes = [IsManagerOrAdmin]
    filter_backends = [
        filters.SearchFilter,
        filters.OrderingFilter,
        DjangoFilterBackend,
    ]
    search_fields = ["loan__user__username", "loan__device__name"]
    ordering_fields = ["returned_at"]
    ordering = ["-returned_at"]
    filterset_fields = ["condition", "inspected_by"]

    def perform_create(self, serializer):
        serializer.save(inspected_by=self.request.user)
```

Обьедененные файлы сериализаторов: devices/serializers/.py*

```python
from rest_framework import serializers
from devices.models import Category, Brand, Location, Spec, Document, Device,
Reservation, Loan, Return

class CategorySerializer(serializers.ModelSerializer):
    class Meta:
        model = Category
        fields = "__all__"

class BrandSerializer(serializers.ModelSerializer):
    class Meta:
        model = Brand
        fields = "__all__"

class LocationSerializer(serializers.ModelSerializer):
    class Meta:
        model = Location
        fields = "__all__"

class SpecSerializer(serializers.ModelSerializer):
    class Meta:
        model = Spec
        fields = "__all__"

class DocumentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Document
        fields = "__all__"

class DeviceSerializer(serializers.ModelSerializer):
    specifications = SpecSerializer(many=True, read_only=True)
    documents = DocumentSerializer(many=True, read_only=True)
    class Meta:
        model = Device
        fields = "__all__"

class ReservationSerializer(serializers.ModelSerializer):
    class Meta:
        model = Reservation
        fields = "__all__"

class LoanSerializer(serializers.ModelSerializer):
    class Meta:
        model = Loan
        fields = "__all__"

class ReturnSerializer(serializers.ModelSerializer):
    class Meta:
        model = Return
        fields = "__all__"
```

Файл: devices/management/commands/auto_backup.py

```python
from datetime import datetime
from django.core.management import call_command
from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = "Automatically backup database and media files"
    def handle(self, *args, **options):
        try:
            call_command("dbbackup", "--clean")
            call_command("mediabackup", "--clean")
            self.stdout.write(self.style.SUCCESS(f"Backup completed at 
{datetime.now()}"))
        except Exception as e:
            self.stdout.write(self.style.ERROR(f"Backup failed: {str(e)}"))
```

Файл: devices/management/commands/backup_db.py

```python
import os
import subprocess
from datetime import datetime
from django.conf import settings
from django.core.management.base import BaseCommand

class Command(BaseCommand):
    help = "Create a backup of the PostgreSQL database"
    def add_arguments(self, parser):
        parser.add_argument("--output-dir", type=str, default="backups")

    def handle(self, *args, **options):
        output_dir = options["output_dir"]
        os.makedirs(output_dir, exist_ok=True)
        db_config = settings.DATABASES["default"]
        timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
        backup_path = os.path.join(output_dir, f"backup_{db_config['NAME']}
_{timestamp}.sql")

        env = os.environ.copy()
        env["PGPASSWORD"] = db_config["PASSWORD"]

        command = [
            "pg_dump", "-h", db_config["HOST"], "-p", str(db_config["PORT"]),
            "-U", db_config["USER"], "-F", "c", "-f", backup_path,
db_config["NAME"]
        ]
        subprocess.run(command, env=env)
        self.stdout.write(self.style.SUCCESS(f"Backup created: {backup_path}"))
```

3. Приложение Services (Сервис и Платежи)

Файл: services/admin.py

```
from django.contrib import admin
from .models import ServiceOrder, ServiceWork, Payment

@admin.register(ServiceOrder)
class ServiceOrderAdmin(admin.ModelAdmin):
    list_display = ["device", "status", "priority", "created_at"]

@admin.register(ServiceWork)
class ServiceWorkAdmin(admin.ModelAdmin):
    list_display = ["service_order", "cost", "performed_by"]

@admin.register(Payment)
class PaymentAdmin(admin.ModelAdmin):
    list_display = ["paid_by", "amount", "payment_type", "paid_at"]

Файл: services/apps.py
from django.apps import AppConfig
class ServicesConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'services'
```

Файл: services/models.py

```
from django.db import models
from django.contrib.auth.models import User
from devices.models import Device, Loan

class ServiceOrder(models.Model):
    STATUS_CHOICES = [("PENDING", "Pending"), ("IN_PROGRESS", "In Progress"),
("COMPLETED", "Completed")]
    PRIORITY_CHOICES = [("LOW", "Low"), ("MEDIUM", "Medium"), ("HIGH", "High")]
    device = models.ForeignKey(Device, on_delete=models.CASCADE,
related_name="service_orders")
    issue_description = models.TextField()
    status = models.CharField(max_length=20, choices=STATUS_CHOICES,
default="PENDING")
    priority = models.CharField(max_length=20, choices=PRIORITY_CHOICES,
default="MEDIUM")
    assigned_to = models.ForeignKey(User, on_delete=models.SET_NULL, null=True,
related_name="assigned_orders")
    created_by = models.ForeignKey(User, on_delete=models.SET_NULL, null=True,
related_name="created_orders")
    created_at = models.DateTimeField(auto_now_add=True)

class ServiceWork(models.Model):
    service_order = models.ForeignKey(ServiceOrder, on_delete=models.CASCADE,
related_name="works")
    work_description = models.TextField()
    cost = models.DecimalField(max_digits=10, decimal_places=2, default=0)
    performed_by = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    performed_at = models.DateTimeField(auto_now_add=True)

class Payment(models.Model):
    TYPES = [("DEPOSIT", "Deposit"), ("FINE", "Fine"), ("RENTAL", "Rental Fee")]
    amount = models.DecimalField(max_digits=10, decimal_places=2)
    payment_type = models.CharField(max_length=20, choices=TYPES)
    paid_by = models.ForeignKey(User, on_delete=models.CASCADE,
related_name="payments")
    related_loan = models.ForeignKey(Loan, on_delete=models.SET_NULL, null=True)
    paid_at = models.DateTimeField(auto_now_add=True)
```

Файл: services/permissions.py

```
from rest_framework import permissions

class IsManagerOrAdmin(permissions.BasePermission):
    def has_permission(self, request, view):
        if not request.user.is_authenticated: return False
        if request.user.is_staff: return True
        try: return request.user.profile.role.name in ["MANAGER", "ADMIN"]
        except: return False

class IsOwnerOrReadOnly(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.method in permissions.SAFE_METHODS: return True
        return hasattr(obj, "paid_by") and obj.paid_by == request.user
```

Файл: services/serializers.py

```
from rest_framework import serializers
from .models import ServiceOrder, ServiceWork, Payment

class ServiceWorkSerializer(serializers.ModelSerializer):
    class Meta:
        model = ServiceWork
        fields = "__all__"

class ServiceOrderSerializer(serializers.ModelSerializer):
    works = ServiceWorkSerializer(many=True, read_only=True)
    class Meta:
        model = ServiceOrder
        fields = "__all__"

class PaymentSerializer(serializers.ModelSerializer):
    class Meta:
        model = Payment
        fields = "__all__"
```

Файл: services/urls.py

```
from django.urls import include, path
from rest_framework.routers import DefaultRouter
from .views import PaymentViewSet, ServiceOrderViewSet
router = DefaultRouter()

router.register("service-orders", ServiceOrderViewSet)
router.register("payments", PaymentViewSet)
urlpatterns = [path("", include(router.urls))]
```

Файл: services/views.py

```
from rest_framework import viewsets
from .models import Payment, ServiceOrder
from .permissions import IsManagerOrAdmin, IsOwnerOrReadOnly
from .serializers import PaymentSerializer, ServiceOrderSerializer

class ServiceOrderViewSet(viewsets.ModelViewSet):
    queryset = ServiceOrder.objects.all()
    serializer_class = ServiceOrderSerializer
    permission_classes = [IsManagerOrAdmin]

class PaymentViewSet(viewsets.ModelViewSet):
    queryset = Payment.objects.all()
    serializer_class = PaymentSerializer
    permission_classes = [IsOwnerOrReadOnly]
    def perform_create(self, serializer):
        serializer.save(paid_by=self.request.user)
```

4. Приложение Users (Пользователи)

Файл: users/admin.py

```
from django.contrib import admin
from .models import Role, UserProfile

@admin.register(Role)
class RoleAdmin(admin.ModelAdmin):
    list_display = ["name", "created_at"]

@admin.register(UserProfile)
class UserProfileAdmin(admin.ModelAdmin):
    list_display = ["user", "role", "department"]
```

Файл: users/apps.py

```
from django.apps import AppConfig
```

```
class UsersConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'users'
```

Файл: users/models.py

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

class Role(models.Model):
    ROLE_CHOICES = [("ADMIN", "Administrator"), ("MANAGER", "Manager"), ("USER",
"User")]
    name = models.CharField(max_length=20, choices=ROLE_CHOICES, unique=True)
    permissions = models.JSONField(default=dict, blank=True)
    created_at = models.DateTimeField(auto_now_add=True)
    def __str__(self): return self.get_name_display()

class UserProfile(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE,
related_name="profile")
    role = models.ForeignKey(Role, on_delete=models.SET_NULL, null=True,
blank=True)
    phone = models.CharField(max_length=20, blank=True, null=True)
    department = models.CharField(max_length=100, blank=True, null=True)
    created_at = models.DateTimeField(auto_now_add=True)
    def __str__(self): return f"{self.user.username} — {self.role}"
```

Файл: users/permissions.py

```
from rest_framework import permissions
class IsAdminOrSelf(permissions.BasePermission):
    def has_object_permission(self, request, view, obj):
        if request.user.is_staff: return True
        return hasattr(obj, "user") and obj.user == request.user
```

Файл: users/serializers.py

```
from rest_framework import serializers
from .models import Role, UserProfile
from django.contrib.auth.models import User

class RoleSerializer(serializers.ModelSerializer):
    class Meta:
        model = Role
        fields = "__all__"

class UserProfileSerializer(serializers.ModelSerializer):
    class Meta:
        model = UserProfile
        fields = "__all__"
```

```python
class UserSerializer(serializers.ModelSerializer):
    profile = UserProfileSerializer(read_only=True)
    class Meta:
        model = User
        fields = ["id", "username", "email", "profile"]
```

Файл: users/urls.py

```python
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import RoleViewSet, UserProfileViewSet, UserViewSet

router = DefaultRouter()
router.register("roles", RoleViewSet)
router.register("profiles", UserProfileViewSet)
router.register("users", UserViewSet)
urlpatterns = [path("", include(router.urls))]
```

Файл: users/views.py

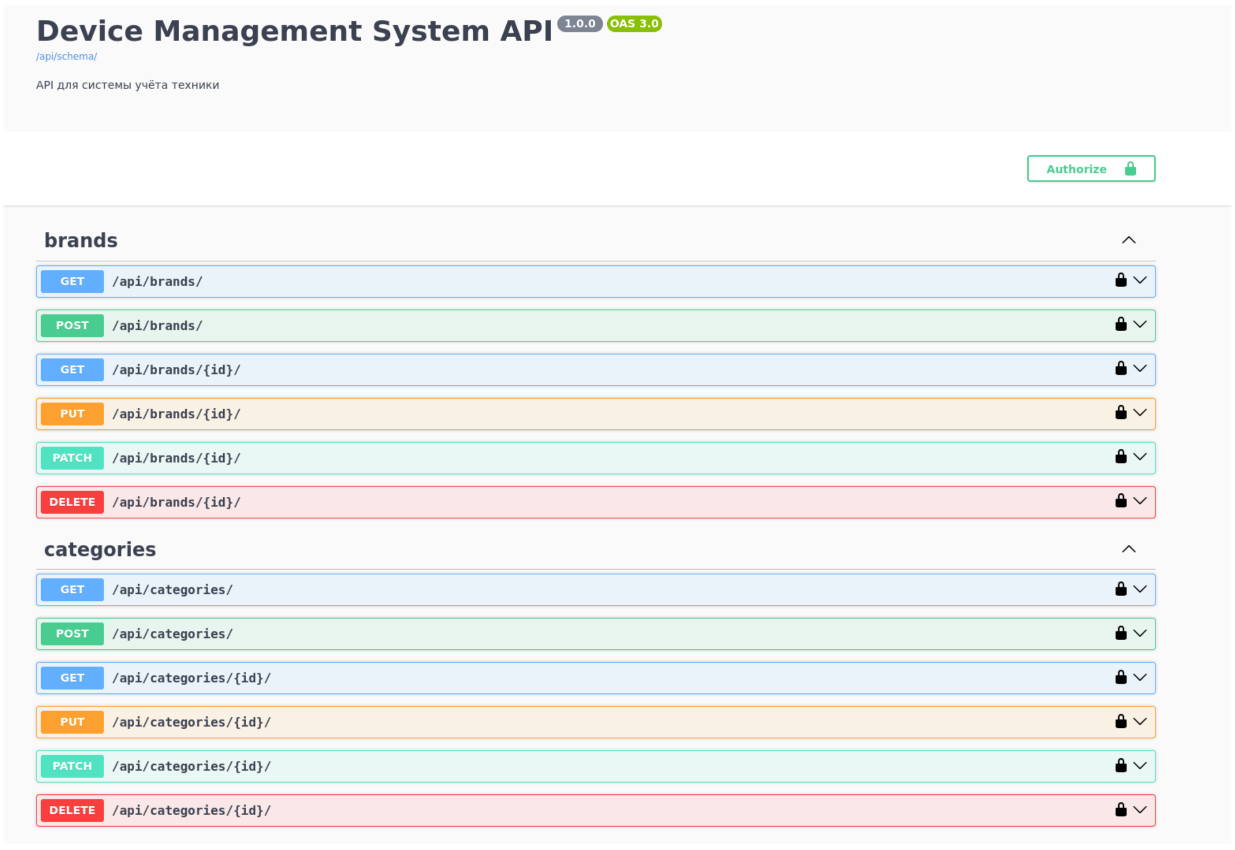
```python
from rest_framework import viewsets, filters
from .models import Role, UserProfile
from .serializers import RoleSerializer, UserProfileSerializer, UserSerializer
from django.contrib.auth.models import User


class RoleViewSet(viewsets.ModelViewSet):
    queryset = Role.objects.all()
    serializer_class = RoleSerializer
    filter_backends = [filters.SearchFilter, filters.OrderingFilter]
    search_fields = ["name", "description"]
    ordering_fields = ["name", "created_at"]
    ordering = ["name"]


class UserProfileViewSet(viewsets.ModelViewSet):
    queryset = UserProfile.objects.all()
    serializer_class = UserProfileSerializer
    filter_backends = [filters.SearchFilter, filters.OrderingFilter]
    search_fields = ["user__username", "phone", "department"]
    ordering_fields = ["created_at"]
    ordering = ["-created_at"]


class UserViewSet(viewsets.ReadOnlyModelViewSet):
    queryset = User.objects.all()
    serializer_class = UserSerializer
    filter_backends = [filters.SearchFilter, filters.OrderingFilter]
    search_fields = ["username", "email", "first_name", "last_name"]
    ordering_fields = ["username", "date_joined"]
    ordering = ["username"]
```

Результат работы:



Device Management System API 1.0.0 OAS 3.0

/api/schema/

API для системы учёта техники

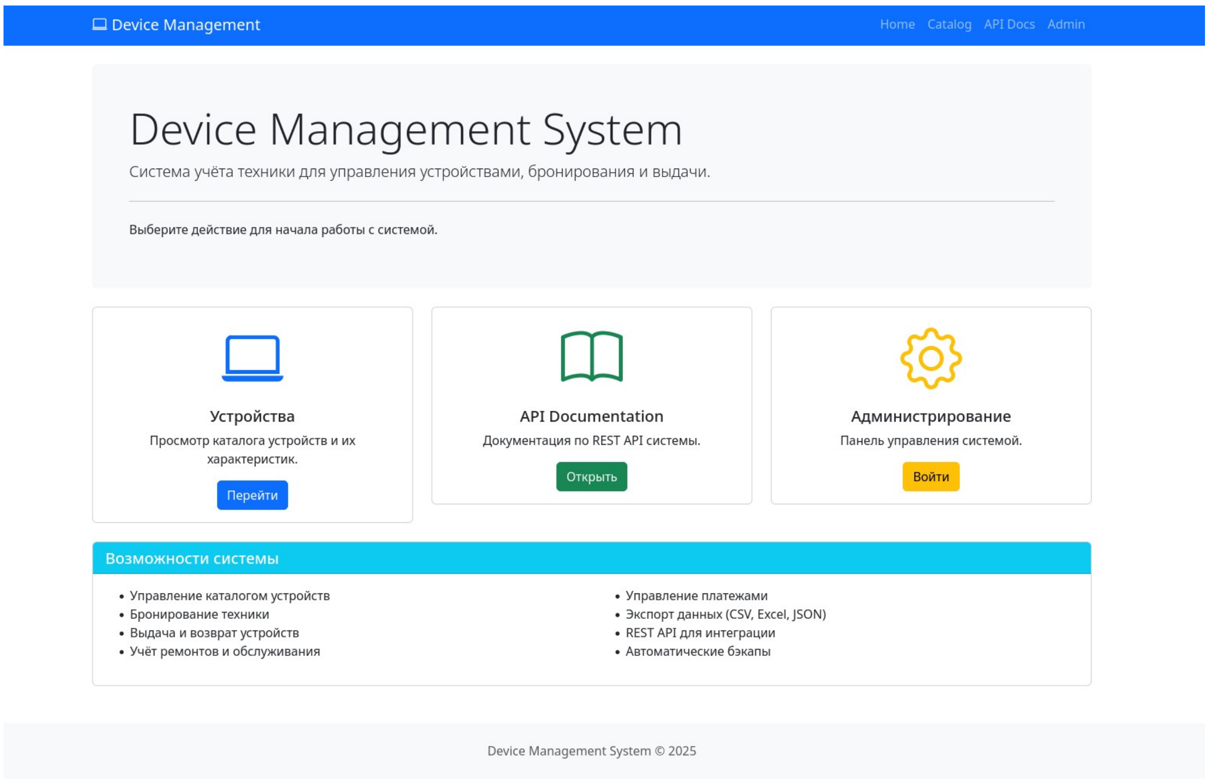
Authorize

brands

GET /api/brands/
POST /api/brands/
GET /api/brands/{id}/
PUT /api/brands/{id}/
PATCH /api/brands/{id}/
DELETE /api/brands/{id}/

categories

GET /api/categories/
POST /api/categories/
GET /api/categories/{id}/
PUT /api/categories/{id}/
PATCH /api/categories/{id}/
DELETE /api/categories/{id}/



Device Management — Home Catalog API Docs Admin

Device Management System

Система учёта техники для управления устройствами, бронирования и выдачи.

Выберите действие для начала работы с системой.

Устройства
Просмотр каталога устройств и их характеристик.
Перейти

API Documentation
Документация по REST API системы.
Открыть

Администрирование
Панель управления системой.
Войти

Возможности системы

- Управление каталогом устройств
- Бронирование техники
- Выдача и возврат устройств
- Учёт ремонтов и обслуживания
- Управление платежами
- Экспорт данных (CSV, Excel, JSON)
- REST API для интеграции
- Автоматические бэкапы

Device Management System © 2025

Вывод: В ходе выполнения курсовой работы была спроектирована и реализована информационная система учета оборудования с использованием фреймворка Django и СУБД PostgreSQL. Были изучены и применены на практике механизмы управления версиями схемы данных через Django Migrations, а также принципы построения гибкого программного интерфейса с помощью Django REST Framework. Особое внимание было уделено углубленному изучению технологии ORM, что позволило реализовать сложные выборки и обеспечить ссылочную целостность данных на уровне приложения без написания прямого SQL-кода. Дополнительно были освоены особенности администрирования PostgreSQL и настроена автоматизированная система резервного копирования для обеспечения надежности хранения информации.