I want to build this project backend with Django (DRF) API but the registers users will be able to add or create other users who will have different levels of access to the module they are subscribed too

HOw do I set thet django project up

� LUMINA Care OS – Optimized SaaS Architecture Overview Vision To build an AI-orchestrated, modular home care SaaS platform that automates and enhances workforce recruitment, care coordination, training, compliance, and client engagement—at scale. CORE PLATFORM PILLARS (Modular Microservices) Pillar Modules (Grouped by Capability) Description Highlights 1. Talent Engine Recruitment CRM, AI Matching, Interview Scheduling, Compliance End-to-end intelligent recruitment pipeline with AI CV matching, gap detection, DBS/API checks, references, and training tracking. 2. Compliance & Docs Document Hub, GDPR Automation, Vetting, Audit Trails Audit-ready compliance engine with auto-alerts, expiry tracking, vetting integrations, and document lifecycle control. 3. Training OS E-learning LMS, Onboarding Gating, Webinar Integration Role-based learning path engine with webinar integration and contract/document handling tied to training gates. 4. Care Coordination Client Portal, AI Care Plan Builder, eMAR, Risk Flags, Rostering Smart rostering, risk-based planning, care plan automation, mobile notes-to-logs AI conversion, CQC audit prep tools. 5. Workforce OS Staff Mobile Portal, Asset Management, Self-Service Dashboards 2FA mobile access, asset tracking, incident/body maps, leave/shifts, integrated performance and compliance tracking. 6. Analytics & Insights AI Workload Balancer, KPI Dashboards, Predictive Governance Intelligent dashboards with executive summaries, predictive alerts, bottleneck detection, and strategic recommendations. 7. Integration Hub RESTful APIs, SSO/OAuth, 3rd-party Full integration layer with hooks for HR/payroll systems, compliance APIs, and app store extensibility. Pillar Modules (Grouped by Capability) Description Highlights integrations (HRIS, Xero, DBS) FUNCTIONAL MODULE MAP (by Lifecycle) Recruitment & Onboarding • Smart job posts (multi-tenant + API) • CV parsing, AI scoring, duplicate check • Candidate timeline with gap history and risk scoring • Interview scheduling + AI performance analytics • DBS/ID/Home Office integrations • Reference automation (sentiment analysis) • Offer/contract auto-generation • Onboarding tasks & e-learning Workforce Lifecycle • Mobile dashboard (leave, rota, feedback, asset) • Real-time workload AI • Asset issuance, MOT, insurance, return • Offboarding with exit interviews, handover, P45, data cleanup • Smart alerts for burnout, compliance gaps Client & Care Coordination • AI-generated CQC-compliant care plans • eMAR with missed-med alerting • Voice note AI-to-text • Live incident flagging + risk detection • Family portal with visit/mood logs • Predictive client care engine (e.g., UTI risk, sleep timing) Governance & Compliance • GDPR data tools (auto-retention, requests) • ISO 27001 ready audit logging • Training compliance & AI enforcement • Dual-approval governance for changes • CQC KLOE audit dashboard Reporting & AI Insights • Role-based dashboards • KPI funnels, hiring metrics, compliance status • Predictive churn, risk, underperformance alerts • Executive summaries & smart recommendations SAAS DELIVERY MODEL Feature Approach Multi-tenancy Fully isolated per client; shared infrastructure with per-tenant RBAC Deployment Cloud-native (AWS/Azure), containerized (Docker/Kubernetes) Mobile Strategy React Native app with shared codebase; PWA fallback Internationalisation i18n-ready UI/UX with per-tenant locale config APIs RESTful with OAuth2, SSO (Azure AD, Google, Okta) Security RBAC, MFA, encrypted storage, ISO 27001-aligned Data Retention Configurable 6–20 years with alerts/export tools Integrations DVLA, DBS, Xero/Sage, job boards, Zoom, Teams KEY TECHNICAL TO-DOs / GAPS Item Description 1. Data Model Unification Define entity schema across tenants (candidates, carers, assets, care plans). 2. Microservice Mesh Setup Service discovery, inter-service auth, logging standardization. 3. Frontend Frameworks Decide on React + Tailwind (admin, mobile PWA, family portal). 4. DevOps Pipeline CI/CD for services, auto-scaling policies, rollback strategies. 5. AI Models & Pipelines Model training/selection for scoring, sentiment, care suggestions. 6. Documentation & SDKs API documentation (Swagger/OpenAPI), sandbox environment. 7. Marketplace Strategy App extensions (e.g., external LMS, rostering plugins). ISO 9001 Evidence-Generating Layer (Integrated Across Platform) Goal: To enable automated or on-demand generation of auditable evidence to demonstrate conformance to ISO 9001:2015 requirements—by tracking, recording, and reporting key operational processes within the system. ISO 9001 Clause Coverage & Software Integration ISO 9001 Clause Evidence via Lumina OS Modules 4. Context of the Organisation Company setup forms, role hierarchies, access logs, stakeholder mapping in admin dashboard 5. Leadership & Commitment Audit logs of Super Admin approvals, policy updates, feedback engine, training participation logs 6. Planning (Risks & Opportunities) AI risk alerts (e.g. missed meds, staff burnout), strategic dashboards, care incident flags 7. Support (Resources, Competence) Asset tracker, training completion logs, job-role learning matrix, license/MOT expiry alerts ISO 9001 Clause Evidence via Lumina OS Modules 8. Operation (Service Delivery Control) Rostering engine, mobile care logs, real-time KPI dashboards, task checklists, compliance triggers 9. Performance Evaluation Multi-role dashboards, audit reports, exit interview analytics, training evaluation summaries 10. Improvement (Corrective Action) Feedback engine with tracked resolutions, AI recommendations, offboarding insight dashboards Smart ISO 9001 Audit Dashboard (New Feature) A new “ISO 9001 Evidence Console” will centralize and export all data needed for an audit in human- and auditor-readable formats: Features: • Clause-Based Report Builder – Pre-configured with ISO 9001 structure (4–10) • Filter by Date, Department, Region, or Process • Attach Evidence Files – e.g., training certificates, policies, asset logs • Export Formats – PDF, DOCX, XLSX for auditors • Audit Trail Embed – Time-stamped logs proving ongoing quality oversight Integrations Across Modules Module Evidence it Produces Training & Onboarding Staff competence matrix, completion dates, quiz scores, webinar attendance Recruitment CRM Hiring process consistency, standardisation, CV scoring logs Compliance Engine Document expiry reports, audit logs, automated vetting trails Incident/Risk Logs eMAR records, falls tracking, safeguarding escalations with supervisor signoff Asset Management Vehicle logs, issuance/return trails, MOT/insurance schedule, condition change history Module Evidence it Produces Dashboards Time-to-hire, training compliance %, staff turnover, client satisfaction feedback Feedback Engine Client/family/staff feedback reports, root cause logging, follow-up actions tracked Next Steps to Implement 1. Define Report Templates for each ISO clause 2. Map All Platform Events to QMS Elements (via log tagging or workflow hooks) 3. Design ISO Dashboard UI (Filters, Uploads, Exports) 4. Enable Auto-Scheduling of Evidence Reports (quarterly/annual for audits) 5. Admin Alerts for missing documentation or unresolved issues You're absolutely right—while the current plan is structured and strategic, it needs more engineering-ready specificity to guide developers from concept to implementation. To fully equip developers, we'll now expand the specification with the missing critical elements, such as: Additions to Make the Plan Developer-Ready 1. Data Architecture Include core entities, relationships, and example data models: • Carer, Client, Shift, TrainingModule, Incident, Document, etc. • Specify whether multi-tenancy is row- or schema-level 2. Key Workflows (End-to-End Examples) Detail 4–5 primary user workflows, such as: • Candidate recruitment journey (from job post to onboarding) • Carer training + assessment lifecycle (with tutor/IQA touchpoints) • Shift creation and AI-based rostering workflow (incl. gap fill logic) • CQC audit prep: how data is pulled and validated • GDPR request flow (SAR → download → deletion) 3. Permissions & Roles Matrix Explicit breakdown of: • Admin, HR, Carer, Client, Family, Auditor, Tutor, Assessor, IQA, EQA • What each role can View / Create / Edit / Delete per module 4. API Contracts (Examples or OpenAPI Spec Outline) Define endpoints and required fields for: • Job posting API • Shift allocation service • Document upload / verification • Care plan creation 5. State Machines for Key Models For example: • Shift State: Scheduled → Accepted → In Progress → Completed → Missed → Reallocated • Compliance Document State: Pending → Verified → Expired → Renewed • Training Module State: Assigned → In Progress → Complete → Expired 6. AI Logic & Trigger Points Make AI tasks clear and pluggable: • Predictive care gaps → Triggered weekly by care note sentiment • Roster gaps → Triggered by cancellation or unfilled shift flag • Candidate matching → Triggered after CV parsing 7. Notification System Define what triggers a notification, its recipient, and delivery channel: • Email / SMS / In-app / Push (via Firebase or OneSignal) • E.g., “Shift Reassigned”, “DBS Expiring”, “Training Overdue” ⍰⍰ COMPLETE COVERAGE AREAS The document covers all essential pillars for a scalable SaaS healthcare platform: Area Coverage Notes Vision & Business Logic ⍰⍰ Clear, outcome-driven Includes objectives, users, and deployment model Modules ⍰⍰ All major care operations HR, onboarding, compliance, rostering, care planning, analytics Compliance ⍰⍰ Deep ISO 9001, CQC, GDPR, DBS, RTW, training audits User Roles & Permissions ⍰⍰ Defined in matrix Maps responsibilities and privileges clearly Architecture ⍰⍰ Industry standard Microservices, async AI orchestration, message queues, CI/CD Data & Workflows ⍰⍰ Well-defined Entities, relationships, states, triggers, notifications AI & Automation ⍰⍰ Embedded at multiple levels CV parsing, rostering, call analysis, reporting summaries Executive Visibility ⍰⍰ Strategic reporting & board tools Includes monthly summaries and live dashboards Mobile & Self-Service ⍰⍰ Carer portal & app detailed Training, schedules, performance, and shift tools STRATEGIC GAPS & LOGICAL ENHANCEMENTS TO INCLUDE To ensure the system is truly developer-ready and future-proof, I recommend addressing the following: 1. Fallback Procedures / Manual Overrides Why? AI systems can fail or misallocate resources. Add: • Manual shift override for rota admins • Manual approval of AI-extracted tasks from calls • Toggle between auto-allocate and manual assign modes 2. Disaster Recovery & Data Redundancy Why? No system is resilient without DR. Add: • Daily backups (cold + hot) • Restore-from-snapshot capability • Uptime commitment (e.g., 99.9%) 3. Multi-Timezone & Locale Support Why? For international scaling. Add: • UTC-first time tracking • Timezone awareness in rostering and reporting • Locale-specific settings (date formats, language) 4. Accessibility & Inclusion Why? Many carers are neurodiverse or older. Add: • WCAG-compliant design • Text scaling, screen reader compatibility • Voice interaction for carers on mobile (for shift actions) 5. AI Explainability & Transparency Why? For legal, ethical, and audit purposes. Add: • Display AI decision confidence (e.g., 88% match) • Log model version + logic used per AI decision • User override + feedback loop 6. Tenant-specific Customisation Why? Each care agency may need flexibility. Add: • Tenant override config (logo, forms, policies) • Role-based UI toggles • Optional modules per subscription tier 7. Care Outcome Tracking Why? Value-based care is emerging. Add: • Track metrics like improved mobility, fewer falls • Link outcomes to care plan revisions • Export for funding bodies or regulators 8. Change History / Versioning Why? Key for compliance and error tracing. Add: • Care plan version tracking • Rostering change logs • Shift assignment edit history SaaS-Oriented System Requirements Specification (SRS) Modular AI-Integrated

Recruitment CRM for SaaS Deployment Overview A scalable, multi-tenant, SaaS-ready CRM platform for AI-enhanced recruitment, compliance automation, care coordination, workforce management, HR oversight, and financial operations. The system is modular, microservice-based, and cloud-native. It supports internationalisation, mobile-first interfaces, role-based access, integration APIs, audit trails, HR tools, and advanced analytics. Core Modules 1. Job Management • Multi-tenant job posting and vacancy pipeline manager • REST APIs and webhook triggers for job boards and branded portals • Location-aware and department-filtered publishing controls 2. AI Candidate Intelligence • AI CV parsing, deduplication, gap analysis, and smart match scoring • Candidate behavioural profile builder and retention prediction • Career journey tracking and machine learning feedback loop 3. Interview & Offer Management • Cross-platform interview scheduling and AI-assisted transcription • Interview scoring rubric, notes, and role-based approvals • Automated conditional offer letters with version tracking 4. Compliance & Vetting • End-to-end compliance lifecycle engine • Integrations with DBS, Home Office, right to work, identity verification • Custom rules-based workflow builder with audit logs and document tagging 5. Onboarding & Training • Smart onboarding sequences based on role and job type • Mandatory training assignment (eLearning, webinar, face-to-face) • Confirmation gating, policy dissemination, and e-signature capture 6. Document Hub • Document templates, builder, lifecycle management, and audit trail • Secure e-signature support and expirable download links • Version control, smart categorisation, and metadata tagging 7. Workflow Engine (360° Recruitment) • Trigger-based workflow automation and onboarding event tracking • Role-sensitive task allocation with escalation rules and reminders • Full candidate journey mapping from attraction to placement 8. AI Allocation & Business Insights • AI workload distribution and performance optimisation engine • KPI dashboards with predictive alerts and AI recommendation layers • Forecasted demand, cost impact, and capacity analytics 9. Dashboards & Analytics • Cross-module dashboard framework with widget customisation • ISO 9001, CQC, GDPR, and financial KPI compliance built-in • Data warehouse-ready analytics stream with drill-downs, exports • Forecasting, planning, and outcome tracking for executives 10. Integrations • Public API (GraphQL/REST) with webhook broadcasting and token auth • Standard integrations: HRIS, ERP, CRM, communication, job boards • SSO with SAML2/OAuth2 (Okta, Azure AD, Google Workspace) 11. Security & Permissions • Full-role hierarchy builder with RBAC, team scopes, and audit actions • Temporary role escalation with expiry settings and MFA override • Geo-fencing, logging, data sovereignty enforcement, retention policies 12. Offboarding & Termination • Task-based exit workflows for HR, Payroll, IT, Compliance • Integrated with asset return, training lockout, and system deactivation • Digital archive for reference history and compliance logs 13. Reference Automation • Multi-template reference generation (character, performance, placement) • API-triggered requests with automated reminders and scoring criteria 14. Career Gap Analysis • AI-enhanced analysis of employment and education history • Gap justifications, timeline rendering, and risk assessment tagging 15. Asset & Resource Tracking • Asset inventory with condition tracking and QR scan logs • Insurance, MOT, allocation, return, and depreciation reporting • Linked to onboarding, offboarding, and compliance workflows 16. Carer/Employee Dashboard (Mobile First) • Secure mobile dashboard with care feed and task interaction • Personal compliance status, care logs, shift response, and training • Onboarding tracker, skills matrix, and allocated clients panel • Real-time calendar sync, incident reporting, care logs • Driver/non-driver filter, location awareness, live route planner • Inbox for task management, notification centre, and support chat 17. Feedback Engine • Structured and free-text feedback collection • Review routing, visibility rules, and optional publication workflow 18. Smart Application Forms • Multi-step application builder with dynamic logic • Configurable fields for NI number, address, references, etc. • Mobile submission support and file upload integration 19. Financial Management Suite • Payroll calculator with tiered pay, tax, and holiday rules • Real-time expense tracking, mileage logs, approvals, and receipts • Invoice builder (client-specific rates, task-mapped) • Income vs cost analytics, forecasting, P&L, audit-ready exports 20. Human Resources (HR) Management • Centralised HR dashboard with access by team, location, and contract type • Real-time view of employee performance including feedback, training, task completion, punctuality, and incident logs • Automated contract issuance with digital signing and archiving • Equipment issuance, tracking, and digital acceptance forms • Company policy publishing with required-read status and completion tracking • Escalation alerts and warnings for overdue documents, missed performance reviews, or compliance gaps • Executive-level analytics module tracking people trends, retention, skills coverage, absenteeism, and cost of workforce Client & Care Coordination Suite 21. Client Management Portal • Comprehensive client record: demographics, setting, permissions • Configurable assessment modules (mobility, risk, meds, fluids) • AI-powered care plan builder and goal tracker • Document import via email parsing, with AI profile extraction • Outcome planning and review tracker • Care Circle management: relatives, social worker, GP, etc. • Client portal for document sharing, status updates, visit logs • Role-based access visibility and exportable client reports 22. Rostering & Care Task Automation • Smart roster engine (auto/manual override, AI grouping) • Shift logic: time blocks, distance, driver/non-driver, priority • Medication plan auto-linkage, visit alerts, escalation rules • Task auto-generation from AI-assessed care plan • Live handover, task progress, and incident report logging •

Schedule forecast and coverage heat map Visual Workflows Candidate Journey Workflow User Role Access Chart System Implementation Roadmap Care Performance and Incident Workflow Supervisor Workflow SaaS Deployment Considerations • Containerised microservices with autoscaling and load balancing • Multi-tenant architecture with client isolation and role inheritance • Regionalised data storage, GDPR and ISO 27001 alignment • CI/CD pipelines, feature flag deployment, staging environments • Modular licensing engine to enable per-feature billing Deliverables • Entity relationship and data flow diagrams • Full API spec with SDK scaffolding for frontend/backend integration • Role-based wireframes and UI kit (mobile, desktop, kiosk) • Training packs, onboarding scripts, SLA documentation • Data lifecycle policy, audit framework, ISO/CQC/Ofsted mapping

To build a Django (Django Rest Framework) backend for the LUMINA Care OS project, where registered users can create other users with varying levels of access to subscribed modules, you need a robust, multi-tenant architecture with role-based access control (RBAC), modular permissions, and a secure API. Below is a step-by-step guide to set up the Django project, tailored to the provided specification. The setup focuses on core components like multi-tenancy, user management, permissions, and module-based access, while aligning with the microservices and SaaS architecture described.

## Step 1: Project Setup and Initial Configuration

1. **Create a Virtual Environment and Install Dependencies**

   ```
   python -m venv venv
   source venv/bin/activate  # On Windows: venv\Scripts\activate
   pip install django djangorestframework django-tenants psycopg2-binary dja
   ```

2. **Initialize Django Project**

   ```
   django-admin startproject lumina_care .
   ```

3. **Directory Structure**
   Organize the project for scalability and modularity:

   ```
   lumina_care/
   ├── lumina_care/
   │   ├── __init__.py
   ```

```
|   ├── settings.py
|   ├── urls.py
|   └── wsgi.py
├── apps/
|   ├── core/
|   ├── talent_engine/
|   ├── compliance/
|   ├── training/
|   ├── care_coordination/
|   ├── workforce/
|   ├── analytics/
|   ├── integrations/
|   └── users/
├── templates/
├── static/
├── manage.py
└── requirements.txt
```

4. **Configure Database (PostgreSQL for Multi-Tenancy)**
   In `settings.py`, configure PostgreSQL and `django-tenants` for multi-tenancy:

```
# lumina_care/settings.py
INSTALLED_APPS = [
    'django_tenants',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework_simplejwt',
    'apps.core',
    'apps.users',
    'apps.talent_engine',
    'apps.compliance',
    'apps.training',
    'apps.care_coordination',
    'apps.workforce',
    'apps.analytics',
    'apps.integrations',
]
```

```
DATABASES = {
    'default': {
        'ENGINE': 'django_tenants.postgresql_backend',
        'NAME': 'lumina_care_db',
        'USER': 'your_user',
        'PASSWORD': 'your_password',
        'HOST': 'localhost',
        'PORT': '5432',
    }
}

DATABASE_ROUTERS = ['django_tenants.routers.TenantSyncRouter']
TENANT_MODEL = "core.Tenant"
TENANT_DOMAIN_MODEL = "core.Domain"
SHARED_APPS = [
    'django_tenants',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'apps.core',
    'apps.users',
]
TENANT_APPS = [
    'django.contrib.admin',
    'rest_framework',
    'rest_framework_simplejwt',
    'apps.talent_engine',
    'apps.compliance',
    'apps.training',
    'apps.care_coordination',
    'apps.workforce',
    'apps.analytics',
    'apps.integrations',
]
```

5. **Create Core App for Tenant Models**

```
python manage.py startapp core apps/core
```

Define tenant and domain models:

```
# apps/core/models.py
from django_tenants.models import TenantMixin, DomainMixin
from django.db import models

class Tenant(TenantMixin):
    name = models.CharField(max_length=100)
    created_at = models.DateTimeField(auto_now_add=True)
    auto_create_schema = True

class Domain(DomainMixin):
    pass
```

6. **Run Migrations**

```
python manage.py makemigrations
python manage.py migrate_schemas --shared
```

## Step 2: User Management and RBAC

1. **Custom User Model**
   Create a custom user model to support roles and tenant-specific users:

```
python manage.py startapp users apps/users
```

```
# apps/users/models.py
from django.contrib.auth.models import AbstractUser
from django.db import models

class CustomUser(AbstractUser):
    ROLES = (
        ('admin', 'Admin'),
        ('hr', 'HR'),
        ('carer', 'Carer'),
        ('client', 'Client'),
        ('family', 'Family'),
        ('auditor', 'Auditor'),
        ('tutor', 'Tutor'),
        ('assessor', 'Assessor'),
        ('iqa', 'IQA'),
        ('eqa', 'EQA'),
    )
    role = models.CharField(max_length=20, choices=ROLES, default='carer'
    tenant = models.ForeignKey('core.Tenant', on_delete=models.CASCADE, r

class UserProfile(models.Model):
    user = models.OneToOneField(CustomUser, on_delete=models.CASCADE)
    modules = models.ManyToManyField('core.Module', blank=True)
```

2. **Module Model for Access Control**

```
# apps/core/models.py
class Module(models.Model):
    name = models.CharField(max_length=100)  # e.g., Talent Engine, Compl
    is_active = models.BooleanField(default=True)
    tenant = models.ForeignKey('core.Tenant', on_delete=models.CASCADE)
```

3. **Permissions Matrix**

   Define permissions for each role per module in a custom permission model:

```
# apps/core/models.py
class RolePermission(models.Model):
    role = models.CharField(max_length=20, choices=CustomUser.ROLES)
    module = models.ForeignKey(Module, on_delete=models.CASCADE)
    can_view = models.BooleanField(default=False)
    can_create = models.BooleanField(default=False)
    can_edit = models.BooleanField(default=False)
    can_delete = models.BooleanField(default=False)
    tenant = models.ForeignKey('core.Tenant', on_delete=models.CASCADE)
```

4. **User Creation API**

   Create an API endpoint for registered users (e.g., admins) to create other users with specific roles and module access:

```
# apps/users/serializers.py
from rest_framework import serializers
from .models import CustomUser, UserProfile
from apps.core.models import Module

class UserSerializer(serializers.ModelSerializer):
    modules = serializers.PrimaryKeyRelatedField(queryset=Module.objects.
    class Meta:
        model = CustomUser
        fields = ['id', 'username', 'email', 'role', 'modules', 'tenant']

    def create(self, validated_data):
        modules = validated_data.pop('modules', [])
        user = CustomUser.objects.create_user(**validated_data)
        profile = UserProfile.objects.create(user=user)
        profile.modules.set(modules)
        return user


# apps/users/views.py
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from .models import CustomUser
from .serializers import UserSerializer
from django_tenants.utils import tenant_context

class UserViewSet(viewsets.ModelViewSet):
    queryset = CustomUser.objects.all()
    serializer_class = UserSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        tenant = self.request.user.tenant
        with tenant_context(tenant):
            return CustomUser.objects.filter(tenant=tenant)

    def perform_create(self, serializer):
        tenant = self.request.user.tenant
        if self.request.user.role != 'admin':
            raise serializers.ValidationError("Only admins can create use
        serializer.save(tenant=tenant)


# apps/users/urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import UserViewSet

router = DefaultRouter()
router.register(r'users', UserViewSet)
```

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

5. **JWT Authentication**
   Configure DRF with JWT for secure API access:

```
# lumina_care/settings.py
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
    'DEFAULT_PERMISSION_CLASSES': (
        'rest_framework.permissions.IsAuthenticated',
    ),
}

from datetime import timedelta

SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=60),
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'AUTH_HEADER_TYPES': ('Bearer',),
}
```

   Include JWT URLs:

```
# lumina_care/urls.py
from django.contrib import admin
from django.urls import path, include
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefr

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include('apps.users.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_re
]
```

## Step 3: Module-Specific Setup

1. **Create Apps for Each Pillar**

```
python manage.py startapp talent_engine apps/talent_engine
python manage.py startapp compliance apps/compliance
python manage.py startapp training apps/training
python manage.py startapp care_coordination apps/care_coordination
python manage.py startapp workforce apps/workforce
python manage.py startapp analytics apps/analytics
python manage.py startapp integrations apps/integrations
```

2. **Example: Talent Engine Models and APIs**

```
# apps/talent_engine/models.py
from django.db import models
from apps.users.models import CustomUser
from apps.core.models import Tenant

class JobPosting(models.Model):
    title = models.CharField(max_length=200)
    description = models.TextField()
    tenant = models.ForeignKey(Tenant, on_delete=models.CASCADE)
    created_by = models.ForeignKey(CustomUser, on_delete=models.SET_NULL,
    created_at = models.DateTimeField(auto_now_add=True)

class Candidate(models.Model):
    user = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    job = models.ForeignKey(JobPosting, on_delete=models.CASCADE)
    cv_data = models.JSONField()  # Store parsed CV data
    ai_score = models.FloatField(null=True)  # AI-generated score
    tenant = models.ForeignKey(Tenant, on_delete=models.CASCADE)


# apps/talent_engine/serializers.py
from rest_framework import serializers
from .models import JobPosting

class JobPostingSerializer(serializers.ModelSerializer):
    class Meta:
        model = JobPosting
        fields = '__all__'
    def validate(self, data):
        tenant = self.context['request'].user.tenant
        if data['tenant'] != tenant:
            raise serializers.ValidationError("Cannot create job for anot
        return data


# apps/talent_engine/views.py
from rest_framework import viewsets
from rest_framework.permissions import IsAuthenticated
from .models import JobPosting
from .serializers import JobPostingSerializer
from django_tenants.utils import tenant_context

class JobPostingViewSet(viewsets.ModelViewSet):
    queryset = JobPosting.objects.all()
    serializer_class = JobPostingSerializer
    permission_classes = [IsAuthenticated]

    def get_queryset(self):
        tenant = self.request.user.tenant
        with tenant_context(tenant):
            return JobPosting.objects.filter(tenant=tenant)

    def perform_create(self, serializer):
        serializer.save(created交代
```

created_by=self.request.user)

~~~
```python
# apps/talent_engine/urls.py
from django.urls import path, include
from rest_framework.routers import DefaultRouter
from .views import JobPostingViewSet

router = DefaultRouter()
router.register(r'job-postings', JobPostingViewSet)

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

3. **Permission Checks for Modules**

   Add a custom permission class to restrict module access:

```
# apps/core/permissions.py
from rest_framework import permissions
from .models import RolePermission

class ModuleAccessPermission(permissions.BasePermission):
    def has_permission(self, request, view):
        user = request.user
        module_name = view.module_name  # Define module_name in view
        tenant = user.tenant
        role_perm = RolePermission.objects.filter(
            role=user.role,
            module__name=module_name,
            tenant=tenant
        ).first()
        if not role_perm:
            return False
        if request.method == 'GET':
            return role_perm.can_view
        if request.method in ['POST', 'PUT', 'PATCH']:
            return role_perm.can_create
        return role_perm.can_delete
```

   Apply to views:

```
# apps/talent_engine/views.py
class JobPostingViewSet(viewsets.ModelViewSet):
    module_name = 'Talent Engine'
    permission_classes = [IsAuthenticated, ModuleAccessPermission]
    ...
```

## Step 4: Multi-Tenancy Implementation

1. **Tenant Creation API**

```
# apps/core/views.py
from rest_framework import viewsets
from rest_framework.permissions import AllowAny
from .models import Tenant, Domain
from .serializers import TenantSerializer

class TenantViewSet(viewsets.ModelViewSet):
    queryset = Tenant.objects.all()
    serializer_class = TenantSerializer
    permission_classes = [AllowAny]

    def perform_create(self, serializer):
        tenant = serializer.save()
        Domain.objects.create(tenant=tenant, domain=f"{tenant.name}.lumir
```

```
# apps/core/serializers.py
from rest_framework import serializers
from .models import Tenant

class TenantSerializer(serializers.ModelSerializer):
    class Meta:
        model = Tenant
        fields = ['id', 'name']
```

2. **Tenant Middleware**

   Ensure requests are routed to the correct tenant schema:

```
# lumina_care/middleware.py
from django_tenants.middleware import TenantMainMiddleware

class CustomTenantMiddleware(TenantMainMiddleware):
    def get_tenant(self, request):
        hostname = request.get_host().split(':')[0]
        domain = Domain.objects.filter(domain=hostname).first()
        return domain.tenant if domain else None
```

   Update `settings.py`:

```
MIDDLEWARE = [
    ...
    'lumina_care.middleware.CustomTenantMiddleware',
]
```

## Step 5: Additional Configurations

1. **State Machines**
   Implement state machines for key models (e.g., Shift, Compliance Document) using `django-fsm`:

```
pip install django-fsm
```

```
# apps/care_coordination/models.py
from django_fsm import FSMField, transition
from django.db import models

class Shift(models.Model):
    STATE_CHOICES = (
        ('scheduled', 'Scheduled'),
        ('accepted', 'Accepted'),
        ('in_progress', 'In Progress'),
        ('completed', 'Completed'),
        ('missed', 'Missed'),
        ('reallocated', 'Reallocated'),
    )
    state = FSMField(default='scheduled', choices=STATE_CHOICES)
    carer = models.ForeignKey('users.CustomUser', on_delete=models.CASCAD
    tenant = models.ForeignKey('core.Tenant', on_delete=models.CASCADE)

    @transition(field=state, source='scheduled', target='accepted')
    def accept(self):
        pass

    @transition(field=state, source='accepted', target='in_progress')
    def start(self):
        pass
```

2. **Notification System**
   Use Django signals for notifications:

```
# apps/core/signals.py
from django.db.models.signals import post_save
from django.dispatch import receiver
from .models import Shift
from django.core.mail import send_mail

@receiver(post_save, sender=Shift)
def notify_shift_change(sender, instance, **kwargs):
    if instance.state == 'reallocated':
        send_mail(
            'Shift Reassigned',
            f'Shift reassigned to {instance.carer.username}.',
            'from@lumina-care.com',
            [instance.carer.email],
        )
```

```
# apps/core/apps.py
from django.apps import AppConfig

class CoreConfig(AppConfig):
    name = 'apps.core'

    def ready(self):
        import apps.core.signals
```

3. **API Documentation**
   Use DRF-Spectacular for OpenAPI documentation:

```
pip install drf-spectacular
```

```
# lumina_care/settings.py
INSTALLED_APPS += ['drf_spectacular']
REST_FRAMEWORK['DEFAULT_SCHEMA_CLASS'] = 'drf_spectacular.openapi.AutoSch
```

```
# lumina_care/urls.py
from drf_spectacular.views import SpectacularAPIView, SpectacularSwaggerV

urlpatterns += [
    path('api/schema/', SpectacularAPIView.as_view(), name='schema'),
    path('api/docs/', SpectacularSwaggerView.as_view(url_name='schema'),
]
```

## Step 6: DevOps and Deployment

1. **Docker Setup**

   Create a `Dockerfile`:

   ```
   FROM python:3.10
   WORKDIR /app
   COPY requirements.txt .
   RUN pip install -r requirements.txt
   COPY . .
   CMD ["gunicorn", "--bind", "0.0.0.0:8000", "lumina_care.wsgi"]
   ```

2. **CI/CD with GitHub Actions**

   Example workflow:

   ```
   name: CI/CD
   on:
     push:
       branches: [main]
   jobs:
     build:
       runs-on: ubuntu-latest
       steps:
       - uses: actions/checkout@v3
       - name: Set up Python
         uses: actions/setup-python@v4
         with:
           python-version: '3.10'
       - name: Install Dependencies
         run: pip install -r requirements.txt
       - name: Run Tests
         run: python manage.py test
   ```

3. **AWS Deployment**

   - Use AWS ECS with Fargate for containerized deployment.
   - Configure RDS for PostgreSQL and Elasticache for caching.
   - Set up Route 53 for tenant-specific subdomains (e.g., `tenant.lumina-care.com`).

## Step 7: Addressing Strategic Gaps

1. **Fallback Procedures**
   Add manual override endpoints in each module's views:

```python
# apps/care_coordination/views.py
from rest_framework.decorators import action

class ShiftViewSet(viewsets.ModelViewSet):
    @action(detail=True, methods=['post'])
    def manual_override(self, request, pk=None):
        shift = self.get_object()
        shift.state = request.data.get('state')
        shift.save()
        return Response({'status': 'updated'})
```

2. **Disaster Recovery**

   - Configure AWS RDS automated backups.
   - Use S3 for cold storage of tenant data.
   - Implement health checks in ECS.

3. **Multi-Timezone Support**

```python
# lumina_care/settings.py
TIME_ZONE = 'UTC'
USE_TZ = True
```

   Add timezone field to user profile:

```python
# apps/users/models.py
class UserProfile(models.Model):
    timezone = models.CharField(max_length=50, default='UTC')
```

4. **Accessibility**
   Ensure frontend (React) uses ARIA attributes and WCAG-compliant design.
   Add voice interaction support via Web Speech API in the React Native app.

5. **AI Explainability**
   Log AI decisions in a dedicated model:

```
# apps/core/models.py
class AIDecisionLog(models.Model):
    decision_type = models.CharField(max_length=100)  # e.g., 'candidate_
    confidence_score = models.FloatField()
    model_version = models.CharField(max_length=50)
    created_at = models.DateTimeField(auto_now_add=True)
    tenant = models.ForeignKey('core.Tenant', on_delete=models.CASCADE)
```

6. **Tenant Customization**

```
# apps/core/models.py
class TenantConfig(models.Model):
    tenant = models.OneToOneField(Tenant, on_delete=models.CASCADE)
    logo = models.ImageField(upload_to='tenant_logos/')
    custom_fields = models.JSONField(default=dict)
```

7. **Care Outcome Tracking**

```
# apps/care_coordination/models.py
class CareOutcome(models.Model):
    client = models.ForeignKey('users.CustomUser', on_delete=models.CASCA
    metric = models.CharField(max_length=100)  # e.g., 'mobility_improvem
    value = models.FloatField()
    recorded_at = models.DateTimeField(auto_now_add=True)
    tenant = models.ForeignKey('core.Tenant', on_delete=models.CASCADE)
```

8. **Change History**
   Use `django-auditlog` for versioning:

```
pip install django-auditlog
```

```
# apps/care_coordination/models.py
from auditlog.registry import auditlog

auditlog.register(Shift)
```

## Step 8: Testing and Documentation

1. **Unit Tests**

```
# apps/users/tests.py
from rest_framework.test import APITestCase
from .models import CustomUser

class UserTests(APITestCase):
    def setUp(self):
        self.user = CustomUser.objects.create_user(
            username='testadmin', password='test123', role='admin'
        )
        self.client.login(username='testadmin', password='test123')

    def test_create_user(self):
        response = self.client.post('/api/users/', {
            'username': 'newuser',
            'email': 'newuser@example.com',
            'password': 'test123',
            'role': 'carer',
        })
        self.assertEqual(response.status_code, 201)
```

2. **API Documentation**
Access Swagger UI at `/api/docs/` to test and document APIs.

### Step 9: Next Steps

- **Implement AI Pipelines**: Integrate with external AI services (e.g., AWS SageMaker) for CV parsing, rostering, and care plan suggestions.
- **Frontend Integration**: Develop React/React Native frontend with API calls to DRF endpoints.
- **Compliance Dashboard**: Build ISO 9001 dashboard using aggregated data from audit logs.
- **Marketplace**: Create an app store API for third-party integrations.

This setup provides a scalable, secure, and modular Django backend for LUMINA Care OS, with multi-tenancy, RBAC, and module-based access control. Let me know if you need detailed guidance on any specific module or feature!