**ML Gruppe Helpdesk System**

Entwicklerhandbuch - Erweiterte Edition

*Version 2.0 - 23.10.2025*

Inhalt

[1. Projektstruktur 3](#_Toc212122062)

[2. Technologie Stack 4](#_Toc212122063)

[Backend Framework 4](#_Toc212122064)

[Datenbanken 5](#_Toc212122065)

[Caching & Message Queue 5](#_Toc212122066)

[Frontend 5](#_Toc212122067)

[Python Libraries 5](#_Toc212122068)

[3. Installation und Setup 5](#_Toc212122069)

[Entwicklungsumgebung einrichten 5](#_Toc212122070)

[4. Datenbankstruktur (Detailliert) 7](#_Toc212122071)

[Entity-Relationship Diagramm (Logisch) 7](#_Toc212122072)

[Datenbankschema - SQL Struktur 8](#_Toc212122073)

[Beziehungen zwischen Tabellen 10](#_Toc212122074)

[5. Models - Tiefgehende Analyse 10](#_Toc212122075)

[User Model (apps/accounts/models.py) 10](#_Toc212122076)

[Ticket Model (apps/tickets/models.py) 11](#_Toc212122077)

[TicketComment Model 13](#_Toc212122078)

[KnowledgeArticle Model 13](#_Toc212122079)

[6. Views und URL-Routing 15](#_Toc212122080)

[URL Routing Struktur (urls.py) 15](#_Toc212122081)

[View Funktion Beispiel 15](#_Toc212122082)

[7. Formularverarbeitung 17](#_Toc212122083)

[Form Klassen (forms.py) 17](#_Toc212122084)

[11. Neue Django Apps erstellen 19](#_Toc212122085)

[Schritt-für-Schritt Anleitung 19](#_Toc212122086)

[1. App Gerüst erstellen 19](#_Toc212122087)

[2. App in INSTALLED\_APPS registrieren 19](#_Toc212122088)

[3. Models definieren 19](#_Toc212122089)

[4. Admin registrieren 20](#_Toc212122090)

[5. Forms erstellen 20](#_Toc212122091)

[6. Views erstellen 20](#_Toc212122092)

[7. URLs konfigurieren 21](#_Toc212122093)

[8. Migrations erstellen 22](#_Toc212122094)

[9. Templates erstellen 22](#_Toc212122095)

[12. Testing und Debugging 23](#_Toc212122096)

[Unit Tests schreiben 23](#_Toc212122097)

[Tests ausführen 24](#_Toc212122098)

[13. Performance Optimierung 25](#_Toc212122099)

[Database Queries optimieren 25](#_Toc212122100)

[Caching implementieren 25](#_Toc212122101)

[Indexierung 25](#_Toc212122102)

[14. Deployment 26](#_Toc212122103)

[Production Checklist 26](#_Toc212122104)

[15. Best Practices 26](#_Toc212122105)

[Code Style und Struktur 26](#_Toc212122106)

# 1. Projektstruktur

mini-helpdesk/  
├── manage.py # Django Management Skript  
├── requirements.txt # Python Abhängigkeiten  
├── .gitignore # Git Ignore Datei  
├── README.md # Projekt Dokumentation  
│  
├── helpdesk/ # Django Projekteinstellungen  
│ ├── \_\_init\_\_.py  
│ ├── settings.py # Haupt-Konfigurationsdatei  
│ ├── urls.py # Haupt-URL Router  
│ ├── asgi.py # ASGI Interface  
│ ├── wsgi.py # WSGI Interface  
│ └── celery.py # Celery Konfiguration  
│  
├── apps/ # Django Anwendungen  
│ ├── accounts/ # Benutzerverwaltung  
│ │ ├── migrations/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── admin.py  
│ │ ├── apps.py  
│ │ ├── forms.py  
│ │ ├── models.py  
│ │ ├── tests.py  
│ │ ├── urls.py  
│ │ └── views.py  
│ │  
│ ├── tickets/ # Ticketmanagement  
│ │ ├── migrations/  
│ │ ├── management/commands/ # Custom Django Commands  
│ │ ├── \_\_init\_\_.py  
│ │ ├── admin.py  
│ │ ├── apps.py  
│ │ ├── forms.py  
│ │ ├── models.py  
│ │ ├── signals.py  
│ │ ├── tests.py  
│ │ ├── urls.py  
│ │ ├── views.py  
│ │ ├── ai\_service.py # Claude AI Integration  
│ │ └── [other utilities]  
│ │  
│ ├── knowledge/ # Knowledge Base  
│ │ ├── migrations/  
│ │ ├── \_\_init\_\_.py  
│ │ ├── admin.py  
│ │ ├── apps.py  
│ │ ├── forms.py  
│ │ ├── models.py  
│ │ ├── tests.py  
│ │ ├── urls.py  
│ │ └── views.py  
│ │  
│ └── main/ # Dashboard & Übersicht  
│ ├── migrations/  
│ ├── \_\_init\_\_.py  
│ ├── admin.py  
│ ├── apps.py  
│ ├── tests.py  
│ ├── urls.py  
│ └── views.py  
│  
├── templates/ # HTML Templates  
│ ├── base.html # Basis Template  
│ ├── accounts/  
│ ├── tickets/  
│ ├── knowledge/  
│ └── main/  
│  
├── static/ # CSS, JavaScript, Bilder  
│ ├── css/  
│ ├── js/  
│ ├── images/  
│ └── vendor/  
│  
├── media/ # Benutzerdateien (Uploads)  
│  
└── docs/ # Dokumentation  
 ├── Benutzerhandbuch\_ML\_Helpdesk.docx  
 ├── Administrationshandbuch\_ML\_Helpdesk.docx  
 └── Entwicklerhandbuch\_ML\_Helpdesk.docx

# 2. Technologie Stack

## Backend Framework

Django 5.0+: Web Framework mit ORM, Admin, Authentication

Python 3.11+: Programmiersprache

## Datenbanken

SQLite: Entwicklung (Standard)

PostgreSQL: Produktion (empfohlen)

MySQL/MariaDB: Alternative für Produktion

## Caching & Message Queue

Redis: Cache und Session Store

Celery: Asynchrone Task Queue

RabbitMQ: Message Broker (optional)

## Frontend

Bootstrap 5: CSS Framework

HTML5: Markup

JavaScript (Vanilla): Client-side Logik

TinyMCE: Rich Text Editor

## Python Libraries

* django-crispy-forms: Form Rendering
* pillow: Bildverarbeitung
* python-docx: Word Document Generation
* requests: HTTP Requests
* python-decouple: Umgebungsvariablen
* gunicorn: WSGI Server
* psycopg2: PostgreSQL Adapter
* mysqlclient: MySQL Adapter
* django-redis: Redis Cache Backend
* celery: Task Queue
* python-dateutil: Datum/Zeit Utilities

# 3. Installation und Setup

## Entwicklungsumgebung einrichten

# 1. Repository klonen  
git clone <repo-url>  
cd mini-helpdesk  
  
# 2. Virtual Environment erstellen  
python3 -m venv .venv  
  
# 3. Aktivieren  
# Windows:  
.venv\Scripts\activate  
# Linux/Mac:  
source .venv/bin/activate  
  
# 4. Dependencies installieren  
pip install -r requirements.txt  
  
# 5. Environment Variablen setzen  
cp .env.example .env  
# Bearbeiten Sie .env mit Ihren Werten  
  
# 6. Datenbank initialisieren  
python manage.py migrate  
  
# 7. Superuser erstellen  
python manage.py createsuperuser  
  
# 8. Testdaten laden (optional)  
python manage.py loaddata fixtures/initial\_data.json  
  
# 9. Static Files sammeln  
python manage.py collectstatic --noinput  
  
# 10. Development Server starten  
python manage.py runserver

# 4. Datenbankstruktur (Detailliert)

## Entity-Relationship Diagramm (Logisch)

User (auth\_user + Erweiterung)  
├── email (UNIQUE)  
├── password (gehashed)  
├── first\_name  
├── last\_name  
├── phone  
├── role (customer, support\_agent, admin)  
├── support\_level (1-4, nur für Agents)  
├── is\_active  
├── created\_at  
└── force\_password\_change  
  
Category (tickets\_category)  
├── name (UNIQUE)  
├── description  
├── is\_active  
└── created\_at  
  
Ticket (tickets\_ticket)  
├── ticket\_number (UNIQUE)  
├── title  
├── description  
├── status (open, in\_progress, resolved, closed)  
├── priority (low, medium, high, critical)  
├── created\_by (FK -> User)  
├── assigned\_to (FK -> User, nullable)  
├── category (FK -> Category)  
├── mobile\_classroom (FK -> MobileClassroom, nullable)  
├── created\_at  
├── updated\_at  
├── resolved\_at (nullable)  
├── closed\_at (nullable)  
└── sla\_due\_at  
  
TicketComment (tickets\_ticketcomment)  
├── ticket (FK -> Ticket)  
├── author (FK -> User)  
├── content  
├── is\_internal (boolean)  
├── created\_at  
└── updated\_at  
  
MobileClassroom (tickets\_mobileclassroom)  
├── name  
├── location (FK -> MobileClassroomLocation)  
├── is\_active  
└── created\_at  
  
MobileClassroomLocation (tickets\_mobileclassroomlocation)  
├── name  
├── address  
└── is\_active  
  
KnowledgeArticle (knowledge\_knowledgearticle)  
├── title  
├── slug (UNIQUE)  
├── content (HTML)  
├── category (FK -> KnowledgeCategory)  
├── is\_published (boolean)  
├── created\_by (FK -> User)  
├── created\_at  
├── updated\_at  
└── view\_count  
  
KnowledgeCategory (knowledge\_knowledgecategory)  
├── name  
├── description  
└── is\_active

## Datenbankschema - SQL Struktur

-- Benutzer (erweitert)  
CREATE TABLE auth\_user (  
 id INTEGER PRIMARY KEY,  
 username VARCHAR(150) UNIQUE NOT NULL,  
 email VARCHAR(254) UNIQUE NOT NULL,  
 password VARCHAR(128) NOT NULL,  
 first\_name VARCHAR(150),  
 last\_name VARCHAR(150),  
 phone VARCHAR(20),  
 role VARCHAR(20) DEFAULT 'customer',  
 support\_level INTEGER DEFAULT 1,  
 is\_active BOOLEAN DEFAULT TRUE,  
 force\_password\_change BOOLEAN DEFAULT FALSE,  
 created\_at TIMESTAMP DEFAULT NOW(),  
 updated\_at TIMESTAMP DEFAULT NOW()  
);  
  
-- Kategorien  
CREATE TABLE tickets\_category (  
 id INTEGER PRIMARY KEY,  
 name VARCHAR(100) UNIQUE NOT NULL,  
 description TEXT,  
 is\_active BOOLEAN DEFAULT TRUE,  
 created\_at TIMESTAMP DEFAULT NOW()  
);  
  
-- Tickets  
CREATE TABLE tickets\_ticket (  
 id INTEGER PRIMARY KEY,  
 ticket\_number VARCHAR(20) UNIQUE NOT NULL,  
 title VARCHAR(200) NOT NULL,  
 description TEXT NOT NULL,  
 status VARCHAR(20) DEFAULT 'open',  
 priority VARCHAR(20) DEFAULT 'medium',  
 created\_by\_id INTEGER NOT NULL REFERENCES auth\_user(id),  
 assigned\_to\_id INTEGER REFERENCES auth\_user(id),  
 category\_id INTEGER REFERENCES tickets\_category(id),  
 mobile\_classroom\_id INTEGER REFERENCES tickets\_mobileclassroom(id),  
 created\_at TIMESTAMP DEFAULT NOW(),  
 updated\_at TIMESTAMP DEFAULT NOW(),  
 resolved\_at TIMESTAMP,  
 closed\_at TIMESTAMP,  
 sla\_due\_at TIMESTAMP  
);  
  
-- Ticket Kommentare  
CREATE TABLE tickets\_ticketcomment (  
 id INTEGER PRIMARY KEY,  
 ticket\_id INTEGER NOT NULL REFERENCES tickets\_ticket(id),  
 author\_id INTEGER NOT NULL REFERENCES auth\_user(id),  
 content TEXT NOT NULL,  
 is\_internal BOOLEAN DEFAULT FALSE,  
 created\_at TIMESTAMP DEFAULT NOW(),  
 updated\_at TIMESTAMP DEFAULT NOW()  
);  
  
-- Knowledge Base  
CREATE TABLE knowledge\_knowledgearticle (  
 id INTEGER PRIMARY KEY,  
 title VARCHAR(200) NOT NULL,  
 slug VARCHAR(200) UNIQUE NOT NULL,  
 content TEXT NOT NULL,  
 category\_id INTEGER REFERENCES knowledge\_knowledgecategory(id),  
 is\_published BOOLEAN DEFAULT FALSE,  
 created\_by\_id INTEGER NOT NULL REFERENCES auth\_user(id),  
 created\_at TIMESTAMP DEFAULT NOW(),  
 updated\_at TIMESTAMP DEFAULT NOW(),  
 view\_count INTEGER DEFAULT 0  
);  
  
-- Indices für Performance  
CREATE INDEX idx\_ticket\_created\_by ON tickets\_ticket(created\_by\_id);  
CREATE INDEX idx\_ticket\_assigned\_to ON tickets\_ticket(assigned\_to\_id);  
CREATE INDEX idx\_ticket\_status ON tickets\_ticket(status);  
CREATE INDEX idx\_ticket\_priority ON tickets\_ticket(priority);  
CREATE INDEX idx\_ticket\_created\_at ON tickets\_ticket(created\_at);  
CREATE INDEX idx\_comment\_ticket ON tickets\_ticketcomment(ticket\_id);  
CREATE INDEX idx\_comment\_author ON tickets\_ticketcomment(author\_id);  
CREATE INDEX idx\_article\_published ON knowledge\_knowledgearticle(is\_published);

## Beziehungen zwischen Tabellen

1:N Beziehungen:

* User -> Ticket (created\_by): Ein User kann viele Tickets erstellen
* User -> Ticket (assigned\_to): Ein Agent kann viele Tickets bearbeiten
* Ticket -> TicketComment: Ein Ticket kann viele Kommentare haben
* Category -> Ticket: Eine Kategorie kann viele Tickets haben
* User -> KnowledgeArticle: Ein User kann viele Artikel schreiben

# 5. Models - Tiefgehende Analyse

## User Model (apps/accounts/models.py)

from django.contrib.auth.models import AbstractUser  
  
class User(AbstractUser):  
 ROLE\_CHOICES = [  
 ('customer', 'Customer'),  
 ('support\_agent', 'Support Agent'),  
 ('admin', 'Administrator'),  
 ]  
  
 SUPPORT\_LEVEL\_CHOICES = [  
 (1, 'Level 1 - Junior'),  
 (2, 'Level 2 - Senior'),  
 (3, 'Level 3 - Expert'),  
 (4, 'Level 4 - Team Lead'),  
 ]  
  
 # Zusätzliche Felder  
 phone = models.CharField(max\_length=20, blank=True)  
 role = models.CharField(max\_length=20, choices=ROLE\_CHOICES, default='customer')  
 support\_level = models.IntegerField(choices=SUPPORT\_LEVEL\_CHOICES, default=1)  
 force\_password\_change = models.BooleanField(default=False)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
  
 class Meta:  
 db\_table = 'auth\_user'  
 ordering = ['created\_at']  
  
 def \_\_str\_\_(self):  
 return f"{self.get\_full\_name()} ({self.get\_role\_display()})"  
  
 @property  
 def full\_name(self):  
 return f"{self.first\_name} {self.last\_name}".strip()  
  
 def can\_access\_ticket(self, ticket):  
 """Check if user can access this ticket"""  
 if self.role == 'admin':  
 return True  
 if self.role == 'support\_agent':  
 return True  
 # Customers can only see their own tickets  
 return ticket.created\_by == self

## Ticket Model (apps/tickets/models.py)

class Ticket(models.Model):  
 STATUS\_CHOICES = [  
 ('open', 'Open'),  
 ('in\_progress', 'In Progress'),  
 ('resolved', 'Resolved'),  
 ('closed', 'Closed'),  
 ]  
  
 PRIORITY\_CHOICES = [  
 ('low', 'Low'),  
 ('medium', 'Medium'),  
 ('high', 'High'),  
 ('critical', 'Critical'),  
 ]  
  
 # Wichtige Felder  
 ticket\_number = models.CharField(max\_length=20, unique=True)  
 title = models.CharField(max\_length=200)  
 description = models.TextField()  
 status = models.CharField(max\_length=20, choices=STATUS\_CHOICES, default='open')  
 priority = models.CharField(max\_length=20, choices=PRIORITY\_CHOICES, default='medium')  
  
 # Beziehungen  
 created\_by = models.ForeignKey(User, on\_delete=models.PROTECT, related\_name='created\_tickets')  
 assigned\_to = models.ForeignKey(User, on\_delete=models.SET\_NULL, null=True, blank=True,  
 related\_name='assigned\_tickets')  
 category = models.ForeignKey(Category, on\_delete=models.SET\_NULL, null=True, blank=True)  
 mobile\_classroom = models.ForeignKey(MobileClassroom, on\_delete=models.SET\_NULL,  
 null=True, blank=True)  
  
 # Zeitstempel  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 resolved\_at = models.DateTimeField(null=True, blank=True)  
 closed\_at = models.DateTimeField(null=True, blank=True)  
 sla\_due\_at = models.DateTimeField(null=True, blank=True)  
  
 class Meta:  
 db\_table = 'tickets\_ticket'  
 ordering = ['-created\_at']  
 indexes = [  
 models.Index(fields=['status']),  
 models.Index(fields=['priority']),  
 models.Index(fields=['created\_at']),  
 ]  
  
 def \_\_str\_\_(self):  
 return f"{self.ticket\_number} - {self.title}"  
  
 def set\_priority\_based\_sla(self):  
 """Set SLA due date based on priority"""  
 from datetime import timedelta  
 sla\_hours = {'low': 24, 'medium': 8, 'high': 4, 'critical': 1}  
 self.sla\_due\_at = self.created\_at + timedelta(hours=sla\_hours[self.priority])  
  
 def get\_history\_as\_text(self):  
 """Get ticket history for email"""  
 history = f"Ticket {self.ticket\_number}: {self.title}\n"  
 history += f"Status: {self.get\_status\_display()}\n"  
 # ... weitere Details ...  
 return history

## TicketComment Model

class TicketComment(models.Model):  
 ticket = models.ForeignKey(Ticket, on\_delete=models.CASCADE, related\_name='comments')  
 author = models.ForeignKey(User, on\_delete=models.PROTECT)  
 content = models.TextField()  
 is\_internal = models.BooleanField(default=False)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
  
 class Meta:  
 db\_table = 'tickets\_ticketcomment'  
 ordering = ['created\_at']  
  
 def \_\_str\_\_(self):  
 return f"Comment on {self.ticket.ticket\_number} by {self.author}"

## KnowledgeArticle Model

class KnowledgeArticle(models.Model):  
 title = models.CharField(max\_length=200)  
 slug = models.SlugField(unique=True)  
 content = models.TextField() # HTML Content from TinyMCE  
 category = models.ForeignKey(KnowledgeCategory, on\_delete=models.SET\_NULL,  
 null=True, blank=True)  
 is\_published = models.BooleanField(default=False)  
 created\_by = models.ForeignKey(User, on\_delete=models.PROTECT)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
 view\_count = models.IntegerField(default=0)  
  
 class Meta:  
 db\_table = 'knowledge\_knowledgearticle'  
 ordering = ['-created\_at']  
  
 def \_\_str\_\_(self):  
 return self.title  
  
 def save(self, \*args, \*\*kwargs):  
 """Auto-generate slug from title"""  
 if not self.slug:  
 from django.utils.text import slugify  
 self.slug = slugify(self.title)  
 super().save(\*args, \*\*kwargs)  
  
 def increment\_views(self):  
 """Increment view counter"""  
 self.view\_count += 1  
 self.save(update\_fields=['view\_count'])

# 6. Views und URL-Routing

## URL Routing Struktur (urls.py)

# helpdesk/urls.py (Haupt URL Router)  
from django.contrib import admin  
from django.urls import path, include  
from django.conf import settings  
from django.conf.urls.static import static  
  
urlpatterns = [  
 path('admin/', admin.site.urls),  
 path('', include('apps.main.urls', namespace='main')),  
 path('accounts/', include('apps.accounts.urls', namespace='accounts')),  
 path('tickets/', include('apps.tickets.urls', namespace='tickets')),  
 path('kb/', include('apps.knowledge.urls', namespace='knowledge')),  
]  
  
if settings.DEBUG:  
 urlpatterns += static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)  
 urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)  
  
# apps/tickets/urls.py (Ticket App URLs)  
from django.urls import path  
from . import views  
  
app\_name = 'tickets'  
  
urlpatterns = [  
 path('', views.ticket\_list, name='list'),  
 path('create/', views.ticket\_create, name='create'),  
 path('<int:pk>/', views.ticket\_detail, name='detail'),  
 path('<int:pk>/assign/', views.ticket\_assign, name='assign'),  
 path('<int:pk>/escalate/', views.ticket\_escalate, name='escalate'),  
 path('<int:pk>/close/', views.ticket\_close, name='close'),  
 path('api/search-customers/', views.search\_customers\_api, name='search\_customers\_api'),  
]

## View Funktion Beispiel

from django.shortcuts import render, redirect, get\_object\_or\_404  
from django.contrib.auth.decorators import login\_required  
from django.contrib import messages  
from .models import Ticket, TicketComment  
from .forms import TicketCreateForm  
  
@login\_required  
def ticket\_detail(request, pk):  
 """View ticket details and add comments"""  
 # Security Check: Ensure user has access  
 ticket = get\_object\_or\_404(Ticket, pk=pk)  
 if not request.user.can\_access\_ticket(ticket):  
 return HttpResponseForbidden()  
  
 # Handle comment submission  
 if request.method == 'POST':  
 form = TicketCommentForm(request.POST)  
 if form.is\_valid():  
 comment = form.save(commit=False)  
 comment.ticket = ticket  
 comment.author = request.user  
 comment.save()  
 messages.success(request, 'Comment added!')  
 return redirect('tickets:detail', pk=ticket.pk)  
 else:  
 form = TicketCommentForm()  
  
 # Get visible comments based on user role  
 if request.user.role == 'customer':  
 comments = ticket.comments.filter(is\_internal=False)  
 else:  
 comments = ticket.comments.all()  
  
 context = {  
 'ticket': ticket,  
 'comments': comments,  
 'form': form,  
 }  
 return render(request, 'tickets/detail.html', context)

# 7. Formularverarbeitung

## Form Klassen (forms.py)

from django import forms  
from django.contrib.auth.forms import UserCreationForm  
from .models import Ticket, TicketComment, User  
  
class UserRegistrationForm(UserCreationForm):  
 """Form für Benutzer Registrierung"""  
 email = forms.EmailField(required=True)  
 first\_name = forms.CharField(max\_length=100, required=True)  
 last\_name = forms.CharField(max\_length=100, required=True)  
  
 class Meta:  
 model = User  
 fields = ('email', 'first\_name', 'last\_name', 'password1', 'password2')  
  
class TicketCreateForm(forms.ModelForm):  
 """Form für Ticket Erstellung durch Kunden"""  
 class Meta:  
 model = Ticket  
 fields = ['title', 'description', 'category', 'priority', 'mobile\_classroom']  
 widgets = {  
 'title': forms.TextInput(attrs={'class': 'form-control'}),  
 'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 6}),  
 'category': forms.Select(attrs={'class': 'form-control'}),  
 'priority': forms.Select(attrs={'class': 'form-control'}),  
 'mobile\_classroom': forms.Select(attrs={'class': 'form-control'}),  
 }  
  
class TicketCommentForm(forms.ModelForm):  
 """Form für Ticket Kommentare"""  
 is\_internal = forms.BooleanField(  
 required=False,  
 label='Internal comment (only visible to staff)',  
 widget=forms.CheckboxInput(attrs={'class': 'form-check-input'})  
 )  
  
 class Meta:  
 model = TicketComment  
 fields = ['content', 'is\_internal']  
 widgets = {  
 'content': forms.Textarea(attrs={'class': 'form-control', 'rows': 4}),  
 }  
  
 def clean(self):  
 """Custom validation"""  
 cleaned\_data = super().clean()  
 content = cleaned\_data.get('content')  
  
 if content and len(content.strip()) < 10:  
 raise forms.ValidationError("Comment must be at least 10 characters long")  
  
 return cleaned\_data

# 11. Neue Django Apps erstellen

## Schritt-für-Schritt Anleitung

### 1. App Gerüst erstellen

python manage.py startapp my\_new\_app

### 2. App in INSTALLED\_APPS registrieren

# helpdesk/settings.py  
INSTALLED\_APPS = [  
 'django.contrib.admin',  
 'django.contrib.auth',  
 # ...  
 'apps.accounts',  
 'apps.tickets',  
 'apps.knowledge',  
 'apps.main',  
 'apps.my\_new\_app', # Neue App hinzufügen  
]

### 3. Models definieren

# apps/my\_new\_app/models.py  
from django.db import models  
from apps.accounts.models import User  
  
class MyModel(models.Model):  
 """Beispiel Model für neue App"""  
 name = models.CharField(max\_length=100)  
 description = models.TextField()  
 owner = models.ForeignKey(User, on\_delete=models.CASCADE)  
 created\_at = models.DateTimeField(auto\_now\_add=True)  
 updated\_at = models.DateTimeField(auto\_now=True)  
  
 class Meta:  
 db\_table = 'my\_new\_app\_mymodel'  
 ordering = ['-created\_at']  
  
 def \_\_str\_\_(self):  
 return self.name

### 4. Admin registrieren

# apps/my\_new\_app/admin.py  
from django.contrib import admin  
from .models import MyModel  
  
@admin.register(MyModel)  
class MyModelAdmin(admin.ModelAdmin):  
 list\_display = ('name', 'owner', 'created\_at')  
 list\_filter = ('created\_at', 'owner')  
 search\_fields = ('name', 'description')  
 date\_hierarchy = 'created\_at'  
  
 def get\_queryset(self, request):  
 """Filter basierend auf Benutzerrolle"""  
 qs = super().get\_queryset(request)  
 if not request.user.is\_superuser:  
 qs = qs.filter(owner=request.user)  
 return qs

### 5. Forms erstellen

# apps/my\_new\_app/forms.py  
from django import forms  
from .models import MyModel  
  
class MyModelForm(forms.ModelForm):  
 class Meta:  
 model = MyModel  
 fields = ['name', 'description']  
 widgets = {  
 'name': forms.TextInput(attrs={'class': 'form-control'}),  
 'description': forms.Textarea(attrs={'class': 'form-control', 'rows': 4}),  
 }

### 6. Views erstellen

# apps/my\_new\_app/views.py  
from django.shortcuts import render, redirect, get\_object\_or\_404  
from django.contrib.auth.decorators import login\_required  
from django.contrib import messages  
from .models import MyModel  
from .forms import MyModelForm  
  
@login\_required  
def my\_model\_list(request):  
 """List all MyModel objects"""  
 if request.user.role == 'admin':  
 objects = MyModel.objects.all()  
 else:  
 objects = MyModel.objects.filter(owner=request.user)  
  
 return render(request, 'my\_new\_app/list.html', {'objects': objects})  
  
@login\_required  
def my\_model\_create(request):  
 """Create new MyModel"""  
 if request.method == 'POST':  
 form = MyModelForm(request.POST)  
 if form.is\_valid():  
 obj = form.save(commit=False)  
 obj.owner = request.user  
 obj.save()  
 messages.success(request, 'Created successfully!')  
 return redirect('my\_new\_app:list')  
 else:  
 form = MyModelForm()  
  
 return render(request, 'my\_new\_app/form.html', {'form': form})

### 7. URLs konfigurieren

# apps/my\_new\_app/urls.py  
from django.urls import path  
from . import views  
  
app\_name = 'my\_new\_app'  
  
urlpatterns = [  
 path('', views.my\_model\_list, name='list'),  
 path('create/', views.my\_model\_create, name='create'),  
]  
  
# In helpdesk/urls.py hinzufügen:  
urlpatterns = [  
 # ...  
 path('my-app/', include('apps.my\_new\_app.urls', namespace='my\_new\_app')),  
]

### 8. Migrations erstellen

python manage.py makemigrations my\_new\_app

python manage.py migrate my\_new\_app

### 9. Templates erstellen

Erstelle Template Ordner: templates/my\_new\_app/

Erstelle Dateien: list.html, form.html, detail.html

# 12. Testing und Debugging

## Unit Tests schreiben

# apps/tickets/tests.py  
from django.test import TestCase, Client  
from django.contrib.auth import get\_user\_model  
from .models import Ticket, Category  
  
User = get\_user\_model()  
  
class TicketModelTest(TestCase):  
 def setUp(self):  
 """Setup test data"""  
 self.user = User.objects.create\_user(  
 email='test@example.com',  
 password='testpass123'  
 )  
 self.category = Category.objects.create(name='Test Category')  
  
 def test\_ticket\_creation(self):  
 """Test creating a ticket"""  
 ticket = Ticket.objects.create(  
 ticket\_number='TST-001',  
 title='Test Ticket',  
 description='Test description',  
 created\_by=self.user,  
 category=self.category  
 )  
 self.assertTrue(Ticket.objects.filter(ticket\_number='TST-001').exists())  
  
 def test\_ticket\_sla\_calculation(self):  
 """Test SLA due date calculation"""  
 ticket = Ticket.objects.create(  
 ticket\_number='TST-002',  
 title='Test',  
 description='Test',  
 created\_by=self.user,  
 priority='high'  
 )  
 ticket.set\_priority\_based\_sla()  
 self.assertIsNotNone(ticket.sla\_due\_at)  
  
class TicketViewTest(TestCase):  
 def setUp(self):  
 self.client = Client()  
 self.user = User.objects.create\_user(  
 email='test@example.com',  
 password='testpass123'  
 )  
  
 def test\_ticket\_list\_requires\_login(self):  
 """Test that ticket list requires authentication"""  
 response = self.client.get('/tickets/')  
 self.assertEqual(response.status\_code, 302) # Redirect to login  
  
 def test\_ticket\_list\_authenticated(self):  
 """Test ticket list for authenticated user"""  
 self.client.login(email='test@example.com', password='testpass123')  
 response = self.client.get('/tickets/')  
 self.assertEqual(response.status\_code, 200)

## Tests ausführen

# Alle Tests:

python manage.py test

# Spezifische App:

python manage.py test apps.tickets

# Mit Verbosity:

python manage.py test -v 2

# 13. Performance Optimierung

## Database Queries optimieren

# SCHLECHT - N+1 Problem:  
tickets = Ticket.objects.all()  
for ticket in tickets:  
 print(ticket.created\_by.full\_name) # Extra Query für jeden Ticket!  
  
# GUT - Select Related:  
tickets = Ticket.objects.select\_related('created\_by', 'category')  
for ticket in tickets:  
 print(ticket.created\_by.full\_name) # Keine extra Queries  
  
# GUT - Prefetch Related für M2M:  
comments = TicketComment.objects.prefetch\_related('ticket')

## Caching implementieren

from django.views.decorators.cache import cache\_page  
from django.core.cache import cache  
  
# View-Level Caching (60 Sekunden)  
@cache\_page(60)  
def cached\_list(request):  
 items = MyModel.objects.all()  
 return render(request, 'list.html', {'items': items})  
  
# Manuelles Caching:  
def get\_stats():  
 cache\_key = 'stats\_cache'  
 stats = cache.get(cache\_key)  
  
 if stats is None:  
 # Calculate stats  
 stats = expensive\_calculation()  
 cache.set(cache\_key, stats, 3600) # Cache for 1 hour  
  
 return stats

## Indexierung

Füge Indizes zu häufig abgefragten Feldern hinzu:

- ticket.status

- ticket.priority

- ticket.created\_at

- ticket.created\_by\_id

# 14. Deployment

## Production Checklist

* DEBUG = False in settings.py
* SECRET\_KEY aus Umgebungsvariable
* ALLOWED\_HOSTS korrekt konfiguriert
* Datenbank: PostgreSQL oder MySQL
* Static Files sammeln: collectstatic
* HTTPS/SSL aktiviert
* Backups konfiguriert
* Logging konfiguriert
* Email Server konfiguriert
* Redis/Cache konfiguriert
* Celery Worker läuft
* Monitoring eingerichtet

# 15. Best Practices

## Code Style und Struktur

1. Verwende Black für Code Formatting:  
 pip install black  
 black apps/  
  
2. Verwende flake8 für Linting:  
 pip install flake8  
 flake8 apps/  
  
3. Django Code Style Guide befolgen:  
 - PEP 8 für Python  
 - Deskriptive Variablennamen  
 - Docstrings für Funktionen  
 - Type Hints wo möglich (Python 3.6+)  
  
4. Git Workflow:  
 - feature/\* Branches für neue Features  
 - bugfix/\* Branches für Bugfixes  
 - Aussagekräftige Commit Messages  
 - Pull Requests für Code Review  
  
5. Dokumentation:  
 - Docstrings in Funktionen  
 - Inline Comments für komplexe Logik  
 - README.md aktuell halten  
 - API Dokumentation (z.B. mit Swagger)  
  
6. Security:  
 - Input Validation  
 - SQL Injection Prevention (ORM nutzt dies automatisch)  
 - XSS Protection  
 - CSRF Protection (Django eingebaut)  
 - HTTPS nur  
 - Secrets nicht in Code speichern

*© 2025 ML Gruppe Helpdesk System - Erweitertes Entwicklerhandbuch*