



SELÇUK ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ



SELÇUK ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

T.C. SELÇUK ÜNİVERSİTESİ TEKNOLOJİ FAKÜLTESİ BİLGİSAYAR
MÜHENDİSLİĞİ BÖLÜMÜ

YAPAY ZEKA DESTEKLİ ÜNİVERSİTE BİLGİ ASİSTANI: SELÇUK AI
ASİSTAN
Doğukan BALAMAN (203311066)
Ali YILDIRIM (203311008)
MÜHENDİSLİK TASARIMI / BİLGİSAYAR MÜHENDİSLİĞİ
UYGULAMALARI

PROJE KABUL VE ONAYI

Doğukan BALAMAN ve Ali YILDIRIM tarafından hazırlanan "YAPAY ZEKA DESTEKLİ ÜNİVERSİTE BİLGİ ASİSTANI: SELÇUK AI ASİSTAN" adlı proje çalışması .../.../2025 tarihinde aşağıdaki juri üyeleri tarafından Selçuk Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği bölümünde Bitirme Projesi olarak kabul edilmiştir.

Jüri Üyeleri

Danışman Prof. Dr. Nurettin DOĞAN

Üye Dr. Öğr. Üyesi Onur İNAN (İkinci Danışman)

Üye Unvanı Adı SOYADI

İmza

Yukarıdaki sonucu onaylarım.

Bilgisayar Mühendisliği
Bölüm Başkanı

**Dr. Öğr. Üyesi Onur İNAN bu proje çalışmasının ikinci danışmanıdır.

* Bu ifade proje çalışması yapılrken bir destek alındıysa yazılım aksi takdirde silinmedir.
** Bu ifade ikinci danışman varsa yazılım aksi takdirde silinmelidir.

PROJE BİLDİRİMİ

Bu projedeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edildiğini ve proje yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yaptığını bildiririm.

DECLARATION PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by project rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

İmza

DoğuCAN BALAMAN (203311066)
Ali YILDIRIM (203311008)

Tarih: .../.../2025

ÖZET

MÜHENDİSLİK TASARIMI /BİLGİSAYAR MÜHENDİSLİĞİ UYGULAMALARI PROJESİ

**YAPAY ZEKA DESTEKLİ ÜNİVERSİTE BİLGİ ASİSTANI: SELÇUK AI
ASİSTAN**

Doğukan BALAMAN (203311066)
Ali YILDIRIM (203311008)

SELÇUK ÜNİVERSİTESİ TEKNOLOJİ FAKÜLTESİ BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

Danışmanlar: Prof. Dr. Nurettin DOĞAN, Dr. Öğr. Üyesi Onur İNAN

2025, 64 Sayfa

Jüri

Danışman: Prof. Dr. Nurettin DOĞAN
Diğer Üyenin Unvanı Adı SOYADI
Diğer Üyenin Unvanı Adı SOYADI

Bu çalışma, Selçuk Üniversitesi için geliştirilen yerel ve gizlilik odaklı yapay zeka destekli bilgi asistanının tasarımını ve uygulamasını sunmaktadır.

Sistem, Flutter tabanlı arayüz, FastAPI tabanlı arka uç, yerel Ollama modeli ve FAISS tabanlı RAG katmanından oluşmaktadır.

RAG katmanı, kaynaklı bağlam üretimiyle doğrulanabilir yanıtlar sağlamakta; SSE akışı ile gerçek zamanlı çıktı sunulmaktadır.

Projede kritik bilgi doğruluk testleri ve CI kalite kontrolleri uygulanmış, sonuçlar akademik doğruluk ve gizlilik gereksinimlerini desteklemiştir.

Anahtar Kelimeler: Yapay Zeka, RAG, FAISS, Yerel LLM, FastAPI, Flutter, Selçuk Üniversitesi

ABSTRACT

ENGINEERING DESIGN / COMPUTER ENGINEERING APPLICATIONS PROJECT

**AI-ASSISTED UNIVERSITY INFORMATION ASSISTANT: SELCUK AI
ASSISTANT**

Doğukan BALAMAN (203311066)
Ali YILDIRIM (203311008)

**SELCUK UNIVERSITY
FACULTY OF TECHNOLOGY
DEPARTMENT OF COMPUTER ENGINEERING**

Advisor: Prof. Dr. Nurettin DOĞAN / Dr. Öğr. Üyesi Onur İNAN

2025, 64 Pages

Jury

Advisor: Prof. Dr. Nurettin DOĞAN

Diger Üyenin Unvanı Adı SOYADI

Diger Üyenin Unvanı Adı SOYADI

This study presents a privacy-focused local AI assistant developed for Selçuk University. The system combines a Flutter interface, a FastAPI backend, Ollama local models, and a FAISS-based RAG layer.

RAG provides source-grounded answers, while SSE streaming delivers real-time responses. Critical fact tests and CI quality checks support reliability and privacy requirements.

Keywords: Artificial Intelligence, RAG, FAISS, Local LLM, FastAPI, Flutter, Selcuk University
ÖNSÖZ

Bu proje, Selçuk Üniversitesi Teknoloji Fakültesi Bilgisayar Mühendisliği Bölümü kapsamında yürütülmüştür.

Danışmanlarımız Prof. Dr. Nurettin DOĞAN ve Dr. Öğr. Üyesi Onur İNAN'a katkıları için teşekkür ederiz.

Doğuakan BALAMAN (203311066)
Ali YILDIRIM (203311008)

Konya / 2025

İÇİNDEKİLER

ÖZET	iv
ABSTRACT.....	iv
ÖNSÖZ	v
İÇİNDEKİLER	vi
SİMGELER VE KISALTMALAR	iv
1. GİRİŞ	1
1.1. Projenin Arka Planı	
Yapay zeka tabanlı bilgi asistanları, üniversite gibi büyük kurumsal yapılarda bilgiye erişimi hızlandırmaktadır.	
Transformer tabanlı LLM'ler, doğal dilde soru-cevap deneyimini yaygınlaştırılmış; kurum içi veri gizliliği ihtiyacı yerel LLM çözümlerini ön plana çıkarmıştır.	
Bu proje, Selçuk Üniversitesi özelinde yerel LLM ve RAG birleşimini kullanarak kaynaklı ve güvenilir yanıt üretimini hedeflemiştir.	
1.2. Projenin Önemi	
Yerel çalışma, kullanıcı verisinin kurum dışına çıkışmasını önlemekte ve çevrimdışı kullanım senaryolarını desteklemektedir.	
RAG yaklaşımı, kaynaklı bağlam ile halüsinsasyon riskini azaltmaktadır.	
1.3. Projenin Kapsamı	
Kapsam; FastAPI arka uç servisleri, Ollama/HF sağlayıcıları, RAG indeksleme ve Flutter arayüz bileşenlerini kapsamaktadır.	
Kimlik doğrulama ve kurumsal SSO entegrasyonları bu çalışmanın kapsamı dışında bırakılmıştır.	
1.4. Raporun Organizasyonu	
Rapor, literatür taramasıyla başlamakta, yöntem ve uygulama bölümleriyle devam etmektedir.	
Bulgular ve tartışma bölümünde test sonuçları değerlendirilmiş, sonuçlar bölümünde öneriler sunulmuştur.	
1.1. Birinci Bölüm İkinci Derece Başlık.....	1
1.1.1. Birinci bölüm üçüncü derece başlık.....	1
2. KAYNAK ARAŞTIRMASI	2

2.1. Transformer ve LLM Gelişimi

Transformer mimarisi, dikkat mekanizması sayesinde uzun bağlam ilişkilerini verimli biçimde modelleyebilmiş ve LLM'lerin temelini oluşturmuştur (Vaswani ve diğerleri, 2017).

2.2. Büyük Dil Modelleri (GPT, LLaMA, Gemini)

GPT, LLaMA ve Gemini gibi model aileleri, farklı ölçek ve veri stratejileriyle yüksek kaliteli metin üretimi sağlamaktadır (Brown ve diğerleri, 2020; Touvron ve diğerleri, 2023).

ChatGPT'nin yaygınlaşması, sohbet tabanlı arayüzlerin bilgi erişiminde etkinliğini göstermiştir (OpenAI, 2022).

2.3. Yerel LLM Çözümleri ve Ollama

Yerel LLM çözümleri, gizlilik ve çevrimdışı çalışma gereksinimleri için tercih edilmektedir. Ollama, GGUF modelleri yerel çalışma kolaylığıyla öne çıkmaktadır (Ollama, 2024).

2.4. RAG ve Vektör Arama

RAG yaklaşımı, üretim sürecine dış kaynak bağlamı ekleyerek doğrulanabilir yanıtlar üretmeyi amaçlar (Lewis ve diğerleri, 2020).

FAISS ve sentence-transformers, semantik arama ve gömme üretimi için sık kullanılan altyapılardır (Johnson ve diğerleri, 2017; Reimers ve Gurevych, 2019).

2.5. Flutter ve Mobil Uygulama Geliştirme

Flutter, tek kod tabanı ile çoklu platform geliştirme imkânı sunar ve kurumsal uygulamalarda bakım maliyetini düşürür (Flutter, 2024).

2.6. Üniversite Chatbot Örnekleri

Georgia State Üniversitesi ve Deakin Üniversitesi örnekleri, öğrenci destek süreçlerinde chatbot kullanımının yaygınlaştığını göstermektedir (Georgia State University, 2023; Deakin University, 2023).

2.1. İkinci Bölüm İkinci Derece Başlık	2
2.1.1. İkinci bölüm üçüncü derece başlık	2

3. MATERİYAL VE YÖNTEM..... 3

3.1. Geliştirme Yaklaşımı

Geliştirme süreci iteratif şekilde yürütülmüş; dokümantasyon güncellemeleri ve kalite kontrolleri ile desteklenmiştir.

Veri toplama için Selçuk Üniversitesi web sayfaları hedeflenmiş; scraping ve manuel doğrulama ile RAG veri seti hazırlanmıştır.

3.2. Veri Toplama

Veri Kaynakları ve Toplama Özeti

Bu dosyada RAG veri toplama süreci için kullanılan resmi kaynaklar ve oluşan veri seti özetlenmiştir.

1) Kaynak Alanı

- İzinli alan adı: `selcuk.edu.tr`

2) Başlangıç URL'leri

- <https://www.selcuk.edu.tr/>

- <https://www.selcuk.edu.tr/ogrenci>

- <https://www.selcuk.edu.tr/akademik>

- <https://www.selcuk.edu.tr/idari>

- <https://www.selcuk.edu.tr/duyurular>

3.3. Model Seçimi

Model seçiminde yerel çalışma ve Türkçe yanıt kalitesi esas alınmıştır. Varsayılan sağlayıcı Ollama olup MODEL_BACKEND parametresiyle yönetilmektedir.

HuggingFace sağlayıcısı opsiyonel olarak kullanılabilimekte ve transformers bağımlılıkları mevcut olduğunda etkinleşmektedir.

3.4. RAG Tasarımı

RAG indeksleme süreci rag_ingest.py ile yürütülmekte, belgeler parçalara ayrıldıktan sonra gömme vektörleri FAISS indeksine eklenmektedir.

Strict mod etkin olduğunda kaynak bulunamazsa yanıt üretimi durdurularak kaynaklı doğruluk korunmaktadır.

Çizelge 3.1. RAG yapılandırma parametreleri

3.5. Değerlendirme Ölçütleri

Değerlendirme, birim testleri, kritik bilgi doğruluğu ve CI kalite kontrolleriyle yapılmıştır.

Performans gözlemleri, benchmark raporlarında yer alan TTFT ve belirteç/saniye ölçümüline dayanmaktadır.

Parametre	Varsayılan	Açıklama
RAG CHUNK SIZE	500	Parça boyutu (karakter)
RAG CHUNK OVERLAP	50	Parça bindirmesi
RAG TOP K	4	Top-K parça sayısı
RAG_EMBEDDING_MODEL	paraphrase-multilingual-MiniLM-L12-v2	Gömme modeli

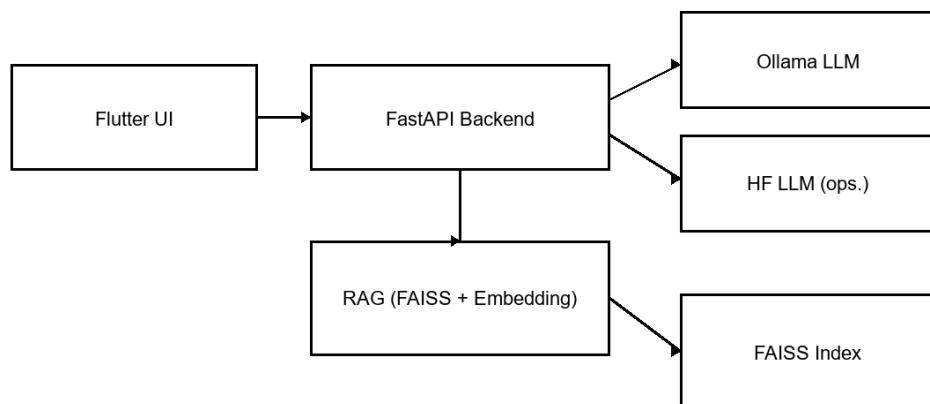
4. SİSTEM TASARIMI VE UYGULAMA

4.1. Genel Mimari

Mimari, Flutter arayüzü ile FastAPI servisleri arasında HTTP/SSE iletişiminden oluşmaktadır.

Model sağlayıcıları providers katmanında soyutlanmış; Ollama ve HuggingFace opsiyonel şekilde çalıştırılmaktadır.

Selçuk AI Asistanı Genel Mimarisi



Şekil 4.1. Selçuk AI Asistanı genel mimarisi

4.2. Backend Tasarımı

`main.py`, `/chat` ve `/chat/stream` uç noktalarında istek doğrulama, model yönlendirme ve RAG entegrasyonunu yürütmektedir.

`ModelRegistry` sınıfı katalog, varsayılan model ve uygunluk durumlarını tek noktadan yönetmektedir.

4.3. RAG Servisi

`rag_service.py`, FAISS indeks yönetimi ve kaynak etiketli bağlam üretimini sağlamaktadır.

Citation formatı kaynak dosya ve sayfa bilgisiyle oluşturularak yanıt güvenilirliği artırılmaktadır.

4.4. API Tasarımı

```
# API Sözleşmesi
```

Bu doküman arka uç API sözleşmesini özetler.

```
## GET /health
```

```

- Cevap: `{"status": "ok", "message": "..."}`

## GET /models

- Cevap: `{"models": [ ... ]}`

- Model nesnesi alanları: `id`, `provider`, `model_id`, `display_name`,
`local_or_remote`, `requires_api_key`, `available`, `reason_unavailable`,
`tags`, `notes`.

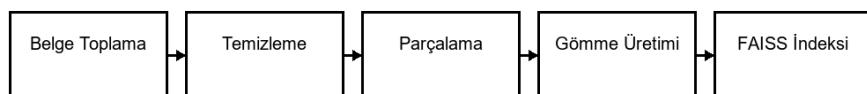
## POST /chat

### İstek

```json

```

RAG Süreci



Şekil 4.2. RAG süreci

#### 4.5. Flutter Arayüzü

Flutter uygulaması GetX ile durum yönetimi yapmakta, konuşma geçmişi Hive üzerinde saklanmaktadır.

EnhancedChatController SSE akışını yönetmekte, sesli giriş speech\_to\_text paketiyle desteklenmektedir.

#### 4.6. Güvenlik ve Gizlilik

Hassas bilgiler .env üzerinden yönetilmekte, CORS yapılandırması kontrollü şekilde uygulanmaktadır.

Yanıt temizleme katmanları, modelin meta veya düşünce bloklarını son kullanıcıya yansıtmamaktadır.

4.1. Dördüncü Bölüm İkinci Derece Başlık.....	5
4.1.1. Dördüncü bölüm üçüncü derece başlık.....	5

### 5. ARAŞTIRMA BULGULARI VE TARTIŞMA

#### 5.1. Test Bulguları

pytest, ruff ve mypy tabanlı kalite kontrolleri CI iş akışında çalıştırılmaktadır. test\_critical\_facts.py ile Konya, 1975 ve Teknoloji Fakültesi gibi kritik bilgiler doğrulanmaktadır.

## 5.2. CI ve Kalite Kontrolleri

```
Test ve CI Raporu
```

Bu rapor, Selçuk AI Akademik Asistan projesinin test ve sürekli entegrasyon (CI) sonuçlarını akademik formatta özetlemek amacıyla hazırlanmıştır.

```
Test Ortamı
```

- CI (GitHub Actions):

- Arka uç: ubuntu-latest

- API duman testi: windows-latest

- Flutter: ubuntu-latest

- Yerel geliştirme (örnek): Windows 10/11, Python 3.12 (önerilen), Flutter Stable

- Sanal ortam: `backend/.venv` veya `venv`

```
Arka Uç Kalitesi
```

- Testler `pytest -q` ile çalıştırılmıştır.

- Kod kalitesi denetimi `ruff check .` ile yapılmıştır.

## 5.3. RAG Performansı

Benchmark raporları, yerel Ollama modellerinde TTFT ve belirteç/saniye ölçümlerinin karşılaştırıldığını göstermektedir.

llama3.2:3b modeli hız odaklı senaryolarda öne çıkarken, turkcell-llm-7b gibi modeller kalite odaklı adaylar olarak değerlendirilmiştir.

## 5.4. Karşılaşılan Zorluklar

UTF-8 uyumluluğu, SSE akışında kesinti yönetimi ve halüsinsasyon riski proje boyunca yönetilmesi gereken başlıca zorluklar olmuştur.

Encoding guard ve strict RAG yaklaşımı bu risklerin azaltılmasına katkı sağlamıştır.

5.1 Sonuçlar ..... 6

5.2 Öneriler ..... 6

## 6. SONUÇLAR VE ÖNERİLER

### 6.1. Sonuçlar

Sistem, yerel LLM ve RAG yaklaşımıyla gizlilik ve doğruluk gereksinimlerini karşılamaktadır.

Gelecek çalışmalarda çok dilli destek ve sesli asistan özellikleri genişletilebilir.

## **6.2. Öneriler**

RAG indeksinin periyodik güncellenmesi ve veri toplama otomasyonunun genişletilmesi önerilmektedir.

Kurumsal kimlik doğrulama ve rol bazlı erişim mekanizmaları ileride entegre edilebilir.

## **KAYNAKLAR ..... 7**

Vaswani, A., Shazeer, N., Parmar, N., et al., 2017, Attention Is All You Need, NeurIPS.

Brown, T., Mann, B., Ryder, N., et al., 2020, Language Models are Few-Shot Learners, NeurIPS.

Touvron, H., Martin, L., Stone, K., et al., 2023, LLaMA: Open and Efficient Foundation Language Models, arXiv.

Lewis, P., Perez, E., Piktus, A., et al., 2020, Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks, NeurIPS.

Johnson, J., Douze, M., ve Jégou, H., 2017, Billion-scale similarity search with GPUs, IEEE Transactions on Big Data.

Reimers, N., ve Gurevych, I., 2019, Sentence-BERT, EMNLP.

Ollama, 2024, Ollama Documentation, <https://ollama.com> (Erişim: 2025-12-30).

OpenAI, 2022, ChatGPT: Optimizing Language Models for Dialogue, <https://openai.com/blog/chatgpt> (Erişim: 2025-12-30).

Georgia State University, 2023, Pounce Chatbot, <https://success.gsu.edu/pounce/> (Erişim: 2025-12-30).

Deakin University, 2023, Genie Virtual Assistant, <https://www.deakin.edu.au/about-deakin/media-releases/articles/meet-genie> (Erişim: 2025-12-30).

FastAPI, 2024, FastAPI Documentation, <https://fastapi.tiangolo.com> (Erişim: 2025-12-30).

Flutter, 2024, Flutter Documentation, <https://docs.flutter.dev> (Erişim: 2025-12-30).

## **EKLER ..... 8**

### **EK-1: API Sözleşmesi (Özet)**

# API Sözleşmesi

Bu doküman arka uç API sözleşmesini özetler.

```

GET /health
- Cevap: `{"status": "ok", "message": "..."}`

GET /models
- Cevap: `{"models": [...]}`
- Model nesnesi alanları: `id`, `provider`, `model_id`, `display_name`, `local_or_remote`, `requires_api_key`, `available`, `reason_unavailable`, `tags`, `notes`.

POST /chat
İstek
```json
{
  "model": "ollama:llama3.1",
  "messages": [{"role": "user", "content": "Merhaba"}],
  "temperature": 0.2,
  "top_p": 0.9,
  "max_tokens": 256,
  "rag_enabled": false,
  "rag.strict": true,
  "rag_top_k": 4
}
```
Cevap
```json
{
  "answer": "...",
  "request_id": "...",
  "provider": "ollama",
  "model": "llama3.1",
  "usage": {"prompt_tokens": 0, "completion_tokens": 0, "total_tokens": 0},
  "citations": ["kaynak.pdf (sayfa 2)"]
}
```

POST /chat/stream
- SSE ile parçalı yanıt döner.
- Olay tipleri:
 - `token`: `{"type": "token", "token": "...", "request_id": "..."}`
 - `end`: `{"type": "end", "usage": { ... }, "citations": [...], "request_id": "..."}`
 - `error`: `{"type": "error", "message": "...", "request_id": "..."}`
```

## EK-2: Örnek Kodlar

### E:/SelcukAiAssistant/repo/backend/main.py

```

"""Selçuk AI Asistanı FastAPI backend uygulaması."""
from __future__ import annotations

import asyncio
import logging
import time
import uuid
from typing import Any, Optional

import requests
from fastapi import FastAPI, HTTPException, Request
from fastapi.middleware.cors import CORSMiddleware
from fastapi.responses import StreamingResponse

from config import Config
from providers.base import CancellationToken, ModelProvider, Usage
from providers.huggingface_provider import HuggingFaceProvider
from providers.ollama_provider import OllamaProvider
```

```

from providers.registry import ModelRegistry
from prompts import build_rag_system_prompt, rag_no_source_message
from rag_service import rag_service
from response_cleaner import StreamingResponseCleaner, clean_text
from schemas import ChatRequest, ChatResponse, UsageInfo
from utils import (
 clamp_max_tokens,
 normalize_messages,
 pick_language,
 sse_event,
 trim_messages_for_context,
)
logger = logging.getLogger(__name__)

app = FastAPI(title="Selçuk AI Asistanı Backend")

default_dev_origins = [
 "http://localhost",
 "http://localhost:3000",
 "http://localhost:5000",
 "http://localhost:5001",
 "http://localhost:8000",
 "http://localhost:8080",
 "http://127.0.0.1",
 "http://127.0.0.1:3000",
 "http://127.0.0.1:5000",
 "http://127.0.0.1:5001",
 "http://127.0.0.1:8000",
 "http://127.0.0.1:8080",
]
allowed_origins = [origin.strip() for origin in Config.ALLOWED_ORIGINS if origin.strip()]
if not allowed_origins:
 if Config.ALLOWED_ORIGINS_STRICT:
 logger.warning(
 "ALLOWED_ORIGINS_STRICT etkin, ancak ALLOWED_ORIGINS boş; "
 "CORS tüm origin'leri engelleyecek."
)
 else:
 allowed_origins = default_dev_origins
allow_all_origins = "*" in allowed_origins

app.add_middleware(
 CORSMiddleware,
 allow_origins=allowed_origins,
 allow_credentials=not allow_all_origins,
 allow_methods=["*"],
 allow_headers=["*"],
)

ollama_provider = OllamaProvider()
huggingface_provider = HuggingFaceProvider()

providers: dict[str, ModelProvider] = {
 "ollama": ollama_provider,
 "huggingface": huggingface_provider,
}
model_registry = ModelRegistry(providers)

```

```

appwrite_client: Optional[requests.Session] = None
if Config.APPWRITE_ENDPOINT and Config.APPWRITE_PROJECT_ID and
Config.APPWRITE_API_KEY:
 appwrite_client = requests.Session()
 appwrite_client.headers.update(
 {
 "X-Appwrite-Project": Config.APPWRITE_PROJECT_ID,
 "X-Appwrite-Key": Config.APPWRITE_API_KEY,
 "Content-Type": "application/json",
 }
)
else:
 logger.info("Appwrite yapılandırılmadı; sohbet kaydı atlandı.")

def _usage_to_schema(usage: Optional[Usage]) -> Optional[UsageInfo]:
 """Giriş: Usage nesnesi.

 Çıkış: UsageInfo ya da None.
 İşleyiş: Usage alanlarını UsageInfo'ya map eder.
 """
 if not usage:
 return None
 return UsageInfo(
 prompt_tokens=usage.prompt_tokens,
 completion_tokens=usage.completion_tokens,
 total_tokens=usage.total_tokens,
)

def _log_chat_to_appwrite(question: str, answer: str) -> None:
 """Giriş: Soru ve yanıt metni.

 Çıkış: yok.
 İşleyiş: Appwrite aktifse HTTP POST ile sohbet kaydı ekler.
 """
 if appwrite_client is None:
 return
 if not Config.APPWRITE_DATABASE_ID or not Config.APPWRITE_COLLECTION_ID:
 return

 import uuid as _uuid
 from datetime import datetime, timezone

 doc_id = f"chat_{_uuid.uuid4().hex[:16]}"
 payload = {
 "documentId": doc_id,
 "data": {
 "question": question[:4000],
 "answer": answer[:4000],
 "timestamp": datetime.now(timezone.utc).isoformat(),
 "chatId": doc_id,
 "senderId": "system",
 "receiverId": "user",
 "messageContent": question[:1000],
 "isRead": True,
 },
 }

```

```

client = appwrite_client
try:
 response = client.post(
 f'{Config.APPWRITE_ENDPOINT}/databases/{Config.APPWRITE_DATABASE_ID}/collections/{Config.APPWRITE_COLLECTION_ID}/documents",
 json=payload,
 timeout=10,
)
 response.raise_for_status()
except requests.RequestException as exc:
 logger.warning("Appwrite kayıt hatası: %s", exc)

@app.get("/")
async def root() -> dict[str, str]:
 """Giriş: yok.

 Çıkış: Durum sözlüğü.
 İşleyiş: Basit sağlık mesajı döndürür.
 """
 return {"status": "ok", "message": "Selçuk AI Asistanı arka uç çalışıyor"}

@app.get("/health")
async def health() -> dict[str, str]:
 """Giriş: yok.

 Çıkış: Durum sözlüğü.
 İşleyiş: Sağlık kontrolü için kısa mesaj döndürür.
 """
 return {"status": "ok", "message": "Selçuk AI Asistanı arka uç çalışıyor"}

@app.get("/health/ollama")
async def ollama_health() -> dict[str, Any]:
 """Giriş: yok.

 Çıkış: Ollama sağlık bilgisi.
 İşleyiş: Sağlıklı sorunlarında 503 döndürür.
 """
 health_status = await ollama_provider.health_check(Config.OLLAMA_MODEL)
 if health_status["status"] == "unhealthy":
 raise HTTPException(status_code=503, detail=health_status)
 return health_status

@app.get("/health/hf")
async def hf_health() -> dict[str, Any]:
 """Giriş: yok.

 Çıkış: HuggingFace bağımlılık ve GPU bilgisi.
 İşleyiş: torch/transformers durumunu raporlar.
 """
 info: dict[str, Any] = {
 "status": "unavailable",
 "torch_version": None,
 }

```

```

 "cuda_available": False,
 "cuda_version": None,
 "gpu_name": None,
 "transformers_version": None,
 "bitsandbytes_version": None,
}

try:
 import torch
 import transformers

 info["torch_version"] = torch.__version__
 info["cuda_available"] = torch.cuda.is_available()
 info["cuda_version"] = torch.version.cuda
 if info["cuda_available"]:
 try:
 info["gpu_name"] = torch.cuda.get_device_name(0)
 except Exception:
 info["gpu_name"] = None

 info["transformers_version"] = transformers.__version__

 try:
 import bitsandbytes

 info["bitsandbytes_version"] = bitsandbytes.__version__
 except Exception:
 info["bitsandbytes_version"] = None

 info["status"] = "ok"
except Exception as exc:
 info["error"] = str(exc)

return info

```

```

@app.get("/models")
async def list_models() -> dict[str, Any]:
 """Giriş: yok.

 Çıkış: Model listesi.
 İşleyiş: ModelRegistry üzerinden katalog döndürür.
 """
 models = await model_registry.list_models()
 return {"models": [model.__dict__ for model in models]}

```

```

@app.post("/chat", response_model=ChatResponse)
async def chat(request: ChatRequest, http_request: Request) -> ChatResponse:
 """Giriş: ChatRequest ve HTTP Request.

 Çıkış: ChatResponse.
 İşleyiş: RAG ve model çağrısını yürütür.
 """
 request_id = uuid.uuid4().hex
 start_time = time.perf_counter()
 language = pick_language(http_request.headers.get("Accept-Language"))
 resolved = model_registry.resolve(request.model)

```

```

provider = providers.get(resolved.provider)
if not provider:
 raise HTTPException(status_code=400, detail="Bilinmeyen model sağlayıcısı.")

rag_enabled = request.rag_enabled and Config.RAG_ENABLED
rag_strict = (
 Config.RAG_STRICT_DEFAULT
 if request.rag_strict is None
 else request.rag_strict
)
rag_top_k = request.rag_top_k or Config.RAG_TOP_K

messages = normalize_messages(request.messages, language)
citations: list[str] = []

if rag_enabled:
 question = next((m.content for m in reversed(messages) if m.role == "user"), "")
 try:
 context, citations = rag_service.get_context(question, top_k=rag_top_k)
 except RuntimeError as exc:
 raise HTTPException(status_code=503, detail=str(exc)) from exc
 if rag_strict and not context:
 answer = rag_no_source_message(language)
 return ChatResponse(
 answer=answer,
 request_id=request_id,
 provider=resolved.provider,
 model=resolved.model_id,
 usage=None,
 citations=citations,
)
 if context:
 messages[0].content = build_rag_system_prompt(
 messages[0].content,
 context,
 language,
 rag_strict,
)
 messages = trim_messages_for_context(messages, Config.MAX_CONTEXT_TOKENS)
 max_tokens = clamp_max_tokens(request.max_tokens, Config.MAX_OUTPUT_TOKENS)

try:
 async with asyncio.timeout(Config.REQUEST_TIMEOUT):
 result = await provider.generate(
 messages=[m.model_dump() for m in messages],
 model_id=resolved.model_id,
 temperature=request.temperature,
 top_p=request.top_p,
 max_tokens=max_tokens,
 request_id=request_id,
)
except TimeoutError as exc:
 raise HTTPException(
 status_code=504, detail="İstek zaman aşımına ugradı."
) from exc
except RuntimeError as exc:
 raise HTTPException(status_code=500, detail=str(exc)) from exc

answer = clean_text(result.text, language=language)

```

```

 _log_chat_to_appwrite(
 question=next((m.content for m in reversed(messages) if m.role == "user"), ""),
 answer=answer,
)

 logger.info(
 "request_id=%s event=chat_done model=%s provider=%s latency_s=%.3f",
 request_id,
 resolved.model_id,
 resolved.provider,
 time.perf_counter() - start_time,
)
 return ChatResponse(
 answer=answer,
 request_id=request_id,
 provider=resolved.provider,
 model=resolved.model_id,
 usage=_usage_to_schema(result.usage),
 citations=citations or None,
)
)

@app.post("/chat/stream")
async def chat_stream(request: ChatRequest, http_request: Request) -> StreamingResponse:
 """Giriş: ChatRequest ve HTTP Request.

 Çıkış: StreamingResponse.
 İşleyiş: SSE tabanlı akış yanıtı üretir.
 """
 request_id = uuid.uuid4().hex
 language = pick_language(http_request.headers.get("Accept-Language"))
 resolved = model_registry.resolve(request.model)
 provider = providers.get(resolved.provider)
 if not provider:
 raise HTTPException(status_code=400, detail="Bilinmeyen model sağlayıcısı.")

 rag_enabled = request.rag_enabled and Config.RAG_ENABLED
 rag_strict = (
 Config.RAG_STRICT_DEFAULT
 if request.rag_strict is None
 else request.rag_strict
)
 rag_top_k = request.rag_top_k or Config.RAG_TOP_K

 messages = normalize_messages(request.messages, language)
 citations: list[str] = []
 rag_context = ""
 rag_error: Optional[str] = None

 if rag_enabled:
 question = next((m.content for m in reversed(messages) if m.role == "user"), "")
 try:
 rag_context, citations = rag_service.get_context(question, top_k=rag_top_k)
 except RuntimeError as exc:
 rag_error = str(exc)
 if rag_context:
 messages[0].content = build_rag_system_prompt(
 messages[0].content,
 rag_context,
)

```

```

 language,
 rag_strict,
)
messages = trim_messages_for_context(messages, Config.MAX_CONTEXT_TOKENS)
max_tokens = clamp_max_tokens(request.max_tokens, Config.MAX_OUTPUT_TOKENS)
cancel_token = CancellationToken()

async def event_generator() -> Any:
 """Giriş: yok.

 Çıkış: SSE veri akışı.
 İşleyiş: Token ve kontrol mesajlarını sırayla üretir.
 """
 if rag_error:
 yield sse_event(
 {
 "type": "error",
 "message": rag_error,
 "request_id": request_id,
 }
)
 return
 if rag_enabled and rag_strict and not rag_context:
 no_source = rag_no_source_message(language)
 yield sse_event(
 {
 "type": "token",
 "token": no_source,
 "request_id": request_id,
 }
)
 yield sse_event(
 {
 "type": "end",
 "usage": None,
 "request_id": request_id,
 "citations": citations,
 }
)
 return

cleaner = StreamingResponseCleaner(language=language)
accumulated_response = ""
try:
 async with asyncio.timeout(Config.REQUEST_TIMEOUT):
 async for chunk in provider.stream(
 messages=[m.model_dump() for m in messages],
 model_id=resolved.model_id,
 temperature=request.temperature,
 top_p=request.top_p,
 max_tokens=max_tokens,
 request_id=request_id,
 cancel_token=cancel_token,
):
 if await http_request.is_disconnected():
 cancel_token.cancel()
 break

 if chunk.token:
 cleaned = cleaner.feed(chunk.token)

```

```

 if cleaned:
 accumulated_response += cleaned
 yield sse_event(
 {
 "type": "token",
 "token": cleaned,
 "request_id": request_id,
 }
)
 if chunk.done:
 final_chunk = cleaner.finalize()
 if final_chunk:
 accumulated_response += final_chunk
 yield sse_event(
 {
 "type": "token",
 "token": final_chunk,
 "request_id": request_id,
 }
)
Appwrite'a kaydet
question = next((m.content for m in reversed(messages) if m.role == "user"), "")
_log_chat_to_appwrite(question=question, answer=accumulated_response)

usage_schema = _usage_to_schema(chunk.usage)
yield sse_event(
 {
 "type": "end",
 "usage": usage_schema.model_dump()
 if usage_schema
 else None,
 "request_id": request_id,
 "citations": citations or None,
 }
)
break
except TimeoutError:
 cancel_token.cancel()
 yield sse_event(
 {
 "type": "error",
 "message": "İstek zaman aşımına uğradı.",
 "request_id": request_id,
 }
)
except HTTPException as exc:
 cancel_token.cancel()
 message = exc.detail if isinstance(exc.detail, str) else "Beklenmeyen hata."
 yield sse_event(
 {
 "type": "error",
 "message": message,
 "request_id": request_id,
 }
)
except Exception as exc:
 cancel_token.cancel()
 yield sse_event(
 {

```

```

 "type": "error",
 "message": str(exc),
 "request_id": request_id,
 }
)

return StreamingResponse(
 event_generator(),
 media_type="text/event-stream",
 headers={
 "Cache-Control": "no-cache",
 "Connection": "keep-alive",
 "X-Accel-Buffering": "no",
 },
)

```

if \_\_name\_\_ == "\_\_main\_\_":
 import uvicorn

 logger.info("Starting server on %s:%s", Config.HOST, Config.PORT)
 unicorn.run(app, host=Config.HOST, port=Config.PORT)

### E:/SelcukAiAssistant/repo/backend/rag\_service.py

"""RAG (Retrieval-Augmented Generation) servisi ve FAISS tabanlı indeks yönetimi."""
from \_\_future\_\_ import annotations

```

import json
import logging
import re
import uuid
from dataclasses import dataclass, field
from pathlib import Path
from typing import Any, Optional, Sequence

```

import numpy as np

from config import Config

logger = logging.getLogger(\_\_name\_\_)

```

try: # pragma: no cover - exercised via runtime imports
 import faiss # type: ignore
except Exception: # pragma: no cover - allow import failure when RAG is disabled
 faiss = None

```

```

@dataclass
class Document:
 """Giriş: İçerik ve metadata alır.
 """

```

Çıkış: Arama sonucunda doc\_id ve score doldurulabilir.  
 İşleyiş: Basit veri kapsülü olarak kullanılır.  
 """

```

content: str
metadata: dict[str, Any] = field(default_factory=dict)
doc_id: Optional[str] = None

```

```

score: Optional[float] = None

class EmbeddingBackend:
 """Giriş: Metin listesi alır.

 Çıkış: float32 gömleme vektörü döndürür.
 İşleyiş: Alt sınıflar dimension ve embed metodlarını sağlar.
 """

 @property
 def dimension(self) -> int:
 """Giriş: yok.

 Çıkış: Vektör boyutu.
 İşleyiş: Alt sınıflar tarafından uygulanır.
 """
 raise NotImplementedError

 def embed(self, texts: Sequence[str]) -> np.ndarray:
 """Giriş: Metin listesi.

 Çıkış: Gömleme vektörleri (float32).
 İşleyiş: Alt sınıflar tarafından uygulanır.
 """
 raise NotImplementedError

class SentenceTransformerBackend(EmbeddingBackend):
 """Giriş: Model adı ve batch size.

 Çıkış: Normalize edilmiş gömlemeler.
 İşleyiş: SentenceTransformer encode ile vektör üretir.
 """

 def __init__(self, model_name: str, batch_size: int = 32) -> None:
 """Giriş: Model adı ve batch size.

 Çıkış: Nesne.
 İşleyiş: SentenceTransformer modelini yükler.
 """
 from sentence_transformers import SentenceTransformer

 self.model_name = model_name
 self._model = SentenceTransformer(model_name)
 self._batch_size = max(1, batch_size)
 dimension = self._model.get_sentence_embedding_dimension()
 if dimension is None:
 raise RuntimeError("Gömleme boyutu alınamadı.")
 self._dimension = int(dimension)

 @property
 def dimension(self) -> int:
 """Giriş: yok.

 Çıkış: Gömleme boyutu.
 İşleyiş: Modelden alınan boyutu döndürür.
 """

```

```

"""
return self._dimension

def embed(self, texts: Sequence[str]) -> np.ndarray:
 """Giriş: Metin listesi.

 Çıkış: Gömleme vektörleri.
 İşleyiş: Encode işlemi ile normalize edilmiş vektör üretir.
"""

if not texts:
 return np.zeros((0, self._dimension), dtype="float32")
embeddings = self._model.encode(
 list(texts),
 batch_size=self._batch_size,
 show_progress_bar=False,
 convert_to_numpy=True,
 normalize_embeddings=True,
)
return np.ascontiguousarray(embeddings, dtype="float32")

```

```

def chunk_text(text: str, chunk_size: int, chunk_overlap: int) -> list[str]:
 """Giriş: Ham metin, parça boyu ve bindirme.

 Çıkış: Parça listesi.
 İşleyiş: Paragraf ayrimı yapar, gerekiyorsa bindirmeli keser.
"""

cleaned = text.replace("\r", "\n")
cleaned = re.sub(r"\n{3,}", "\n\n", cleaned).strip()
if not cleaned:
 return []

```

```

paragraphs = [p.strip() for p in re.split(r"\n\s*\n", cleaned) if p.strip()]
chunks: list[str] = []
current: list[str] = []
current_len = 0

for paragraph in paragraphs:
 paragraph_len = len(paragraph)
 if current_len + paragraph_len + 2 <= chunk_size:
 current.append(paragraph)
 current_len += paragraph_len + 2
 continue

 if current:
 chunk = "\n\n".join(current).strip()
 if chunk:
 chunks.append(chunk)
 if chunk_overlap > 0 and chunk:
 overlap_text = chunk[-chunk_overlap:]
 current = [overlap_text]
 current_len = len(overlap_text)
 else:
 current = []
 current_len = 0

 if paragraph_len >= chunk_size:
 start = 0
 step = max(1, chunk_size - chunk_overlap)

```

```

 while start < paragraph_len:
 end = min(start + chunk_size, paragraph_len)
 chunk = paragraph[start:end].strip()
 if chunk:
 chunks.append(chunk)
 start += step
 current = []
 current_len = 0
 else:
 current = [paragraph]
 current_len = paragraph_len

 if current:
 chunk = "\n\n".join(current).strip()
 if chunk:
 chunks.append(chunk)

 return chunks

```

```

class RagIndex:
 """Giriş: Kök dizin, gömleme üretici ve batch boyu.
 """

```

```

 Çıkış: Arama sonucu dokümanlar ve indeks dosyaları.
 İşleyiş: FAISS indeks dosyası ile metadata.json senkron tutulur.
 """

```

```

def __init__(
 self,
 root: Path,
 embedder: EmbeddingBackend,
 batch_size: int = 32,
) -> None:
 """Giriş: İndeks dizini ve gömleme altyapısı.

 Çıkış: Nesne.
 İşleyiş: Dosya yollarını ve gömleyiciyi hazırlar.
 """

```

```

if faiss is None:
 raise RuntimeError("faiss-cpu yüklü değil.")
self.root = root
self.embedder = embedder
self.index_path = self.root / "index.faiss"
self.meta_path = self.root / "metadata.json"
self.meta_info_path = self.root / "index_meta.json"
self.batch_size = max(1, batch_size)
self.index: Any = None
self.metadata: list[dict[str, Any]] = []
self._load_existing()

```

```

def _load_existing(self) -> None:
 """Giriş: yok.
 """

```

```

 Çıkış: yok.
 İşleyiş: Mevcut indeks dosyalarını okur ve doğrular.
 """
 if not self.index_path.exists():
 return
 self.index = faiss.read_index(str(self.index_path))

```

```

if self.meta_path.exists():
 try:
 self.metadata = json.loads(self.meta_path.read_text(encoding="utf-8"))
 except json.JSONDecodeError as exc:
 raise RuntimeError("RAG metadata dosyası okunamadı.") from exc
if self.index is not None and self.metadata:
 if self.index.ntotal != len(self.metadata):
 raise RuntimeError(
 "İndeks/metadata tutarsız: "
 f"{self.index.ntotal} vs {len(self.metadata)}"
)

def _ensure_index(self) -> None:
 """Giriş: yok.

 Çıkış: yok.
 İşleyiş: İndeks yoksa FAISS IndexFlatIP oluşturur.
 """
 if self.index is None:
 self.index = faiss.IndexFlatIP(self.embedder.dimension)

def add_documents(self, documents: Sequence[Document]) -> int:
 """Giriş: Doküman listesi.

 Çıkış: Eklenen doküman sayısı.
 İşleyiş: Batch halinde gömleme üretip indekse ekler.
 """
 if not documents:
 return 0
 self._ensure_index()
 assert self.index is not None
 added = 0
 batch_size = max(1, self.batch_size)
 for start in range(0, len(documents), batch_size):
 batch = documents[start : start + batch_size]
 embeddings = self.embedder.embed([doc.content for doc in batch])
 if embeddings.size == 0:
 continue
 embeddings = np.ascontiguousarray(embeddings, dtype="float32")
 self.index.add(embeddings)
 for doc in batch:
 doc_id = doc.doc_id or uuid.uuid4().hex
 payload = {
 "id": doc_id,
 "content": doc.content,
 **doc.metadata,
 }
 self.metadata.append(payload)
 added += len(batch)
 return added

def search(self, query: str, top_k: int) -> list[Document]:
 """Giriş: Sorgu metni ve top_k.

 Çıkış: En yakın dokümanlar.
 İşleyiş: FAISS indeksinde arama yapar.
 """
 if not query.strip() or self.index is None or self.index.ntotal == 0:
 return []

```

```

top_k = max(1, min(top_k, self.index.ntotal))
embeddings = self.embedder.embed([query])
if embeddings.size == 0:
 return []
embeddings = np.ascontiguousarray(embeddings, dtype="float32")
scores, indices = self.index.search(embeddings, top_k)
results: list[Document] = []
for score, idx in zip(scores[0], indices[0]):
 if idx < 0 or idx >= len(self.metadata):
 continue
 meta = self.metadata[idx]
 results.append(
 Document(
 content=meta.get("content", ""),
 metadata={k: v for k, v in meta.items() if k != "content"},
 doc_id=meta.get("id"),
 score=float(score),
)
)
return results

```

```

def save(self, meta_info: Optional[dict[str, Any]] = None) -> None:
 """Giriş: Opsiyonel meta bilgi.

```

Çıkış: yok.  
 İşleyiş: İndeks ve metadata'yı diske yazar.

```

 """
if self.index is None:
 return
self.root.mkdir(parents=True, exist_ok=True)
faiss.write_index(self.index, str(self.index_path))
self.meta_path.write_text(
 json.dumps(self.metadata, ensure_ascii=False, indent=2),
 encoding="utf-8",
)
if meta_info is not None:
 self.meta_info_path.write_text(
 json.dumps(meta_info, ensure_ascii=False, indent=2),
 encoding="utf-8",
)

```

```

def _citation_label(meta: dict[str, Any]) -> str:
 """Giriş: Metadata sözlüğü.

```

Çıkış: Kaynak etiketi.  
 İşleyiş: Kaynak ve sayfa bilgisini birleştirir.

```

 """
source = meta.get("source") or "Bilinmeyen kaynak"
page = meta.get("page")
if page:
 return f"{source} (sayfa {page})"
return source

```

```

class RAGService:
 """Giriş: RAG ayarları, embedder ve indeks yolu.

```

Çıkış: Kaynaklı bağlam ve alıntı listesi.

İşleyiş: İndeks hazırlırsa arama yapar, değilse anlamlı hata verir.  
"""

```
def __init__(
 self,
 enabled: bool = False,
 vector_db_path: Optional[str] = None,
 collection_name: str = "selcuk_documents",
 chunk_size: int = 500,
 chunk_overlap: int = 50,
 embedding_model: Optional[str] = None,
 top_k: int = 3,
 embedder: Optional[EmbeddingBackend] = None,
 embedding_batch_size: Optional[int] = None,
) -> None:
 """Giriş: RAG yapılandırma parametreleri.

 Çıkış: Nesne.
 İşleyiş: RAG hizmetini hazırlar ve indeksi yükler.
 """
 self.enabled = enabled
 self.available = False
 self.error_message: Optional[str] = None
 self.vector_db_path = vector_db_path
 self.collection_name = collection_name
 self.chunk_size = chunk_size
 self.chunk_overlap = chunk_overlap
 self.embedding_model = embedding_model or Config.RAG_EMBEDDING_MODEL
 self.top_k = top_k
 self.embedding_batch_size = (
 embedding_batch_size or Config.RAG_EMBEDDING_BATCH_SIZE
)
 self._embedder = embedder
 self._index: Optional[RagIndex] = None

 if not self.enabled:
 logger.info("RAG servisi devre dışı (RAG_ENABLED=false)")
 return

 if not self.vector_db_path:
 self.error_message = "RAG etkin ama RAG_VECTOR_DB_PATH ayarlanmamış."
 logger.warning(self.error_message)
 return
 if faiss is None:
 self.error_message = "RAG için faiss-cpu kütüphanesi gereklidir."
 logger.warning(self.error_message)
 return

 try:
 self._embedder = self._embedder or SentenceTransformerBackend(
 self.embedding_model,
 batch_size=self.embedding_batch_size,
)
 self._index = RagIndex(
 Path(self.vector_db_path),
 self._embedder,
 batch_size=self.embedding_batch_size,
)
 self.available = True
```

```

logger.info(
 "RAG service initialized: path=%s collection=%s model=%s",
 self.vector_db_path,
 self.collection_name,
 self.embedding_model,
)
except Exception as exc:
 self.error_message = f'RAG servisi başlatılamadı: {exc}'
 logger.exception(self.error_message)

def search(self, query: str, top_k: Optional[int] = None) -> list[Document]:
 """Giriş: Sorgu metni ve opsiyonel top_k.

 Çıkış: Doküman listesi.
 İşleyiş: İndeks üzerinden arama yapar.
 """
 if not self.enabled:
 return []
 if not self.available or self._index is None:
 raise RuntimeError(self.error_message or "RAG servisi hazır değil.")
 return self._index.search(query, top_k or self.top_k)

def get_context(
 self,
 query: str,
 top_k: Optional[int] = None,
) -> tuple[str, list[str]]:
 """Giriş: Sorgu metni ve opsiyonel top_k.

 Çıkış: Bağlam metni ve kaynak listesi.
 İşleyiş: Sorgu sonuçlarını numaralı bağlam haline getirir.
 """
 if not self.enabled:
 return "", []
 if not self.available:
 raise RuntimeError(self.error_message or "RAG servisi hazır değil.")
 docs = self.search(query, top_k=top_k)
 if not docs:
 return "", []
 context_parts: list[str] = []
 citations: list[str] = []
 for idx, doc in enumerate(docs):
 citations.append(_citation_label(doc.metadata))
 context_parts.append(f'{idx} {doc.content}')
 return '\n\n'.join(context_parts), citations

def add_documents(self, documents: Sequence[Document]) -> int:
 """Giriş: Doküman listesi.

 Çıkış: Eklenen doküman sayısı.
 İşleyiş: İndeks üzerinden ekleme yapar.
 """
 if not self.enabled or self._index is None:
 raise RuntimeError("RAG servisi etkin değil.")
 if not self.available:
 raise RuntimeError(self.error_message or "RAG servisi hazır değil.")
 return self._index.add_documents(documents)

```

```
def save_index(self, meta_info: Optional[dict[str, Any]] = None) -> None:
 """Giriş: Opsiyonel meta bilgi.
```

```
 Çıkış: yok.
 İşleyiş: İndeks ve metadata bilgisini diske yazar.
 """
 if not self.enabled or self._index is None:
 return
 if not self.available:
 raise RuntimeError(self.error_message or "RAG servisi hazır değil.")
 self._index.save(meta_info=meta_info)
```

```
rag_service = RAGService(
 enabled=Config.RAG_ENABLED,
 vector_db_path=Config.RAG_VECTOR_DB_PATH,
 collection_name=Config.RAG_COLLECTION_NAME,
 chunk_size=Config.RAG_CHUNK_SIZE,
 chunk_overlap=Config.RAG_CHUNK_OVERLAP,
 embedding_model=Config.RAG_EMBEDDING_MODEL,
 top_k=Config.RAG_TOP_K,
 embedding_batch_size=Config.RAG_EMBEDDING_BATCH_SIZE,
)
```

### E:/SelcukAiAssistant/repo/backend/ollama\_service.py

```
"""Selçuk AI Asistanı için asenkron Ollama servis istemcisi."""
from __future__ import annotations
```

```
import asyncio
import json
import logging
from typing import Any, AsyncIterator, Optional

import httpx
from fastapi import HTTPException

from config import Config
```

```
logger = logging.getLogger(__name__)
```

```
class OllamaService:
 """Giriş: Yapılandırma değerleri.
```

```
 Çıkış: Metin/usage üreten servis çağrıları.
 İşleyiş: Ollama HTTP uç noktalarıyla konuşur.
 """
```

```
def __init__(
 self,
 base_url: Optional[str] = None,
 timeout: Optional[int] = None,
 max_retries: int = 3,
 retry_delay: float = 1.0,
) -> None:
 """Giriş: URL, timeout ve retry ayarları.
```

Çıkış: Nesne.

```

 İşleyiş: İstemciyi yapılandırır.
 """
 self.base_url = base_url or Config.OLLAMA_BASE_URL
 self.timeout = timeout or Config.OLLAMA_TIMEOUT
 self.max_retries = max_retries
 self.retry_delay = retry_delay
 self.api_url = f"{self.base_url}/api/chat"

 logger.info(
 "Initialized Ollama service: url=%s timeout=%ss max_retries=%s",
 self.api_url,
 self.timeout,
 self.max_retries,
)

@staticmethod
def _validate_messages(messages: list[dict[str, str]]) -> None:
 """Giriş: Mesaj listesi.

 Çıkış: Hata veya None.
 İşleyiş: Boş listeyi engeller.
 """
 if not messages:
 raise HTTPException(status_code=400, detail="Mesaj listesi boş olamaz.")

@staticmethod
def _clean_reasoning_artifacts(text: str) -> str:
 """Giriş: Ham metin.

 Çıkış: Temiz metin.
 İşleyiş: Düşünce izlerini ayıklar.
 """
 import re

 if not text or not text.strip():
 return "Merhaba! Ben Selçuk AI Asistanı. Size nasıl yardımcı olabilirim?"

 original_text = text

 text = re.sub(r"<think>.*?</think>", "", text, flags=re.DOTALL | re.IGNORECASE)
 text = text.replace("<|im_end|>", "").replace("<|im_start|>", "")
 text = text.replace("<|end|>", "").replace("<|start|>", "")

 reasoning_patterns = [
 r"^\[^.\?\n]*b(okay|alright|let me|let me think|hmm|wait)\b[\^!.?\n]*[\n.]",
 r"^\[^.\?\n]*b(tamam|peki|düşünelim|dusunelim|"
 r"bakalim|bakalim|bir dakika)\b[\^!.?\n]*[\n.]",
]
 for pattern in reasoning_patterns:
 text = re.sub(pattern, "", text, flags=re.IGNORECASE | re.MULTILINE)

merhaba_matches = list(re.finditer(r"\bMerhaba[.,]?", text, re.IGNORECASE))
if merhaba_matches:
 text = text[merhaba_matches[-1].start() :]
elif re.search(r"^\#\#\s", text, re.MULTILINE):
 header_match = re.search(r"^\#\#\s", text, re.MULTILINE)
 if header_match:
 text = text[header_match.start() :]

```

```

text = re.sub(r"\n{3,}", "\n\n", text).strip()

if len(text) < 15:
 if len(original_text.strip()) > 20 and "<think>" not in original_text.lower():
 return original_text.strip()
 return "Merhaba! Ben Selçuk AI Asistanı. Size nasıl yardımcı olabilirim?""

return text

async def generate(
 self,
 messages: list[dict[str, str]],
 model: str,
 temperature: float,
 top_p: float,
 max_tokens: int,
) -> dict[str, Any]:
 """Giriş: Mesajlar ve üretim parametreleri.

Çıkış: Metin/usage sözlüğü.
İşleyiş: `/api/chat` çağrısı yapar.
"""

self._validate_messages(messages)

async with httpx.AsyncClient(timeout=self.timeout) as client:
 for attempt in range(self.max_retries):
 try:
 payload = {
 "model": model,
 "messages": messages,
 "stream": False,
 "options": {
 "temperature": temperature,
 "top_p": top_p,
 "top_k": 40,
 "repeat_penalty": 1.1,
 "num_predict": max_tokens,
 },
 }

 response = await client.post(
 self.api_url,
 json=payload,
 headers={"Content-Type": "application/json; charset=utf-8"},
)
 response.encoding = "utf-8"

 if response.status_code != 200:
 error_detail = self._parse_error_response(response)
 raise HTTPException(
 status_code=response.status_code,
 detail=f'Ollama API hatası: {error_detail}',
)

 data = response.json()
 message = data.get("message") or {}
 answer = message.get("content", "")
 if not answer:

```

```

 return {
 "text": "Üzgünüm, bir yanıt oluşturulamadı.",
 "usage": None,
 }

cleaned = self._clean_reasoning_artifacts(answer)
usage = {
 "prompt_tokens": data.get("prompt_eval_count"),
 "completion_tokens": data.get("eval_count"),
}
return {"text": cleaned, "usage": usage}

except httpx.ReadTimeout:
 if attempt < self.max_retries - 1:
 await asyncio.sleep(self.retry_delay * (attempt + 1))
 continue
 raise HTTPException(
 status_code=504,
 detail="Ollama isteği zaman aşımına uğradı.",
)
except httpx.RequestError as exc:
 if attempt < self.max_retries - 1:
 await asyncio.sleep(self.retry_delay * (attempt + 1))
 continue
 raise HTTPException(
 status_code=503,
 detail=f'Ollama servisine bağlanılamadı: {exc}',
)

raise HTTPException(status_code=500, detail="Ollama hatası")

async def generate_stream(
 self,
 messages: list[dict[str, str]],
 model: str,
 temperature: float,
 top_p: float,
 max_tokens: int,
) -> AsyncIterator[dict[str, Any]]:
 """Giriş: Mesajlar ve üretim parametreleri.

 Çıkış: Token akışı.
 İşleyiş: Stream çağrısı yapar.
 """
 self._validate_messages(messages)

 payload = {
 "model": model,
 "messages": messages,
 "stream": True,
 "options": {
 "temperature": temperature,
 "top_p": top_p,
 "top_k": 40,
 "repeat_penalty": 1.1,
 "num_predict": max_tokens,
 },
 }

```

```

try:
 async with httpx.AsyncClient(timeout=self.timeout) as client:
 async with client.stream(
 "POST",
 self.api_url,
 json=payload,
 headers={"Content-Type": "application/json; charset=utf-8"},
) as response:
 if response.status_code != 200:
 raise HTTPException(
 status_code=response.status_code,
 detail=f'Ollama API hatası: HTTP {response.status_code}',
)

 async for line in response.aiter_lines():
 if not line:
 continue
 try:
 data = json.loads(line)
 except json.JSONDecodeError:
 continue

 message = data.get("message") or {}
 token = message.get("content", "")
 done = bool(data.get("done", False))
 usage = None
 if done:
 usage = {
 "prompt_tokens": data.get("prompt_eval_count"),
 "completion_tokens": data.get("eval_count"),
 }
 yield {"token": token, "done": done, "usage": usage}
except httpx.ReadTimeout as exc:
 raise HTTPException(
 status_code=504,
 detail="Ollama isteği zaman aşımına uğradı.",
) from exc
except httpx.RequestError as exc:
 raise HTTPException(
 status_code=503,
 detail=f'Ollama servisine bağlanılamadı: {exc}',
) from exc

async def health_check(self, model: Optional[str] = None) -> dict[str, Any]:
 """Giriş: Model adı (opsiyonel)."""

 Çıkış: Sağlık bilgisi.
 İşleyiş: `/api/tags` ile model listesini kontrol eder.

 model = model or Config.OLLAMA_MODEL
 try:
 async with httpx.AsyncClient(timeout=5.0) as client:
 response = await client.get(f'{self.base_url}/api/tags')

 if response.status_code == 200:
 models = response.json().get("models", [])
 model_names = [m.get("name") for m in models]
 model_available = self._is_model_available(model, model_names)
 return {

```

```

 "status": "healthy" if model_available else "degraded",
 "ollama_url": self.base_url,
 "model": model,
 "model_available": model_available,
 "available_models": model_names,
 }
 return {
 "status": "unhealthy",
 "ollama_url": self.base_url,
 "model": model,
 "error": f"HTTP {response.status_code}",
 }
except httpx.RequestError:
 return {
 "status": "unhealthy",
 "ollama_url": self.base_url,
 "model": model,
 "error": "Bağlantı başarısız",
 }

async def list_model_names(self) -> list[str]:
 """Giriş: yok.

 Çıkış: Model adları listesi.
 İşleyiş: `/api/tags` listesini döndürür.
 """
 try:
 async with httpx.AsyncClient(timeout=5.0) as client:
 response = await client.get(f"{self.base_url}/api/tags")
 if response.status_code != 200:
 return []
 models = response.json().get("models", [])
 return [m.get("name") for m in models if m.get("name")]
 except httpx.RequestError:
 return []

@staticmethod
def _is_model_available(target_model: str, available_models: list[str]) -> bool:
 """Giriş: Hedef model ve mevcut modeller.

 Çıkış: bool.
 İşleyiş: Model adında temel eşleşme yapar.
 """
 if not target_model or not available_models:
 return False
 if target_model in available_models:
 return True
 target_base = target_model.split(":")[0]
 for model in available_models:
 if target_base == model.split(":")[0]:
 return True
 return False

@staticmethod
def _parse_error_response(response: httpx.Response) -> str:
 """Giriş: HTTP yanıtı.

 Çıkış: Hata metni.
 İşleyiş: JSON error alanını okur.

```

```

"""
try:
 error_data = response.json()
 return error_data.get("error", response.text)
except (ValueError, KeyError, AttributeError):
 return response.text or f"HTTP {response.status_code}"

```

### E:/SelcukAiAssistant/repo/backend/utils.py

```

"""İstek işleme ve akışa yönelik yardımcı fonksiyonlar."""
from __future__ import annotations

```

```

import json
from dataclasses import dataclass
from typing import Iterable, Optional

```

```

from prompts import build_default_system_prompt
from schemas import ChatMessage

```

```

def pick_language(accept_language: Optional[str]) -> str:

```

"""Giriş: Accept-Language başlığı veya None.

Çıkış: "tr" ya da "en".

İşleyiş: Başlıktaki dil sırasına göre desteklenen ilk dili seçer.

"""

```

if not accept_language:
 return "tr"

```

```

for part in accept_language.split(","):
 lang = part.split(";", 1)[0].strip().lower()

```

```

 if lang.startswith("tr"):
 return "tr"

```

```

 if lang.startswith("en"):
 return "en"

```

```

return "tr"

```

```

def build_default_system_message(language: str) -> ChatMessage:

```

"""Giriş: Dil kodu.

Çıkış: Sistem rolünde ChatMessage.

İşleyiş: Varsayılan sistem promptunu üretir.

"""

```

return ChatMessage(

```

```

 role="system",

```

```

 content=build_default_system_prompt(language).strip(),
)

```

```

def normalize_messages(

```

```

 messages: list[ChatMessage],

```

```

 language: str,
)
-> list[ChatMessage]:

```

"""Giriş: Mesaj listesi ve dil kodu.

Çıkış: Normalize edilmiş mesaj listesi.

İşleyiş: Sistem mesajı yoksa ekler, içerikleri kopyalar.

"""

```

normalized = [ChatMessage(role=m.role, content=m.content) for m in messages]

```

```
if not any(m.role == "system" for m in normalized):
 normalized.insert(0, build_default_system_message(language))
return normalized
```

```
def estimate_token_count(text: str) -> int:
 """Giriş: Metin.
```

Çıkış: Yaklaşık token sayısı.  
İşleyiş: Basit uzunluk bölme yaklaşımı uygular.  
"""

```
if not text:
 return 0
return max(1, len(text) // 4)
```

```
def estimate_messages_tokens(messages: Iterable[ChatMessage]) -> int:
 """Giriş: Mesaj listesi.
```

Çıkış: Toplam token tahmini.  
İşleyiş: Her mesajın token tahminini toplar.  
"""

```
return sum(estimate_token_count(m.content) for m in messages)
```

```
def clamp_max_tokens(requested: int, max_allowed: int) -> int:
 """Giriş: İstenen ve izin verilen üst sınır.
```

Çıkış: Sınırlandırılmış token sayısı.  
İşleyiş: Değeri 1 ile üst sınır arasında tutar.  
"""

```
return max(1, min(requested, max_allowed))
```

```
def trim_messages_for_context(
 messages: list[ChatMessage],
 max_tokens: int,
) -> list[ChatMessage]:
 """Giriş: Mesaj listesi ve bağlam limiti.
```

Çıkış: Budanmış mesaj listesi.  
İşleyiş: Token limiti aşıldıkça en eski mesajları çıkarır.  
"""

```
trimmed = list(messages)
while estimate_messages_tokens(trimmed) > max_tokens and len(trimmed) > 1:
 if trimmed[0].role == "system" and len(trimmed) > 1:
 trimmed.pop(1)
 else:
 trimmed.pop(0)
return trimmed
```

```
def sse_event(payload: dict[str, object]) -> str:
 """Giriş: JSON'a dönüştürülebilir sözlük.
```

Çıkış: SSE formatlı veri satırı.  
İşleyiş: JSON'u 'data:' satırı olarak paketler.

```

"""
return f'data: {json.dumps(payload, ensure_ascii=False)}\n\n'

@dataclass
class ReasoningFilter:
 """Giriş: Akış metni parçaları.

 Çıkış: <think> blokları ayıklanmış metin.
 İşleyiş: <think> ... </think> aralıklarını filtreler.
"""

inside_think: bool = False
buffer: str = ""

def feed(self, chunk: str) -> str:
 """Giriş: Yeni metin parçası.

 Çıkış: Ayıklanmış metin parçası.
 İşleyiş: Tamponu günceller, <think> bloklarını çıkarır.
"""

 self.buffer += chunk
 output_parts: list[str] = []

 while self.buffer:
 if self.inside_think:
 end_idx = self.buffer.find("</think>")
 if end_idx == -1:
 # Keep a small tail for tag detection
 self.buffer = self.buffer[-16:]
 return "".join(output_parts)
 self.buffer = self.buffer[end_idx + len("</think>") :]
 self.inside_think = False
 continue

 start_idx = self.buffer.find("<think>")
 if start_idx == -1:
 output_parts.append(self.buffer)
 self.buffer = ""
 break

 if start_idx > 0:
 output_parts.append(self.buffer[:start_idx])
 self.buffer = self.buffer[start_idx + len("<think>") :]
 self.inside_think = True

 return "".join(output_parts)

```

### E:/SelcukAiAssistant/repo/backend/response\_cleaner.py

```

"""Sunucu tarafında sohbet çıktılarının temizlenmesi için yardımcılar."""
from __future__ import annotations

from dataclasses import dataclass, field
import re
from typing import List, Tuple

from utils import ReasoningFilter

```

```

_META_LINE = re.compile(
 r"^\s*(?:"
 r"(?:reasoning|analysis|thoughts?|chain of thought|let me think)\s*:?\\s*\$|""
 r"(?:final answer|final|answer)\\s*:?\\s*\$|"
 r"(?:ok(?:ay)?[,]+i need to respond.)*$"
 r")",
 re.IGNORECASE,
)

_META_PREFIX = re.compile(
 r"^\[\\s\"?\]*(?:(:so|well|then|next|since|maybe)[,\\s]+)?(?:"
 r"okay|alright|sure|let me|let me think|i need to|i will|i should|i?ll|"
 r"first|here?\\s my|my plan|the user|they just|they said|they mentioned|"
 r"they?re|they are|"
 r"they probably|they might|they want|user is|"
 r"i am going to|i?m going to|i want to|tamam|peki|öncelikle|oncelikle|"
 r"ilk olar\\ak|kullanici|kullanici|düşünüyor|dusunuyorum"
 r")\\b",
 re.IGNORECASE,
)

_META_SENTENCE = re.compile(
 r"^\[\\s\"?\]*(?:(:so|well|then|next|since|maybe)[,\\s]+)?(?:"
 r"okay|alright|sure|let me|let me think|i need to|i will|i should|i?ll|"
 r"first|here?\\s my|my plan|the user|they just|they said|they mentioned|"
 r"they?re|they are|"
 r"they probably|they might|they want|user is|"
 r"i am going to|i?m going to|i want to|tamam|peki|öncelikle|oncelikle|"
 r"ilk olar\\ak|kullanici|kullanici|düşünüyor|dusunuyorum"
 r")[^.!?\\n]*[.!?]\\s*",
 re.IGNORECASE,
)

_META_PREFIX_FRAGMENT = re.compile(
 r"^\[\\s\"?\](?:"
 r"so|well|then|next|since|maybe|okay|alright|sure|let|i|they|tamam|"
 r"peki|öncelikle|oncelikle|ilk|"
 r")\\b",
 re.IGNORECASE,
)

_META_SELF_ACTION = re.compile(
 r"\b(?:i need to|i should|i will|i?ll|i am going to|let me)\\b",
 re.IGNORECASE,
)

_META_META_ACTION = re.compile(
 r"\b(?:respond|answer|format|structure|plan|thinking|make sure|follow|"
 r"offer|introduce|mention|say|greet|start|begin)\\b",
 re.IGNORECASE,
)

_META_CONTEXT = re.compile(
 r"\b(?:the user|they|their|they said|they mentioned|since they|looking at|based on|"
 r"system message|guidelines)\\b",
 re.IGNORECASE,
)

```

```
def _split_by_fences(text: str) -> List[Tuple[str, bool]]:
 """Giriş: Metin.
```

```
Çıkış: (parça, kod_mu) listesi.
İşleyiş: ``` ayraçlarıyla metni parçalar.
"""
out: List[Tuple[str, bool]] = []
fence = "```"
idx = 0
in_code = False

while idx < len(text):
 next_idx = text.find(fence, idx)
 if next_idx == -1:
 out.append((text[idx:], in_code))
 break
 out.append((text[idx:next_idx], in_code))
 in_code = not in_code
 out.append((fence, in_code))
 idx = next_idx + len(fence)
return out
```

```
def _strip_leading_meta_lines(text: str) -> str:
 """Giriş: Metin.
```

```
Çıkış: Meta satırları temizlenmiş metin.
İşleyiş: Baştaki meta satırlarını siler.
"""
lines = text.replace("\r\n", "\n").split("\n")
idx = 0
removed = 0
while idx < len(lines) and lines[idx].strip() == "":
 idx += 1

while idx < len(lines) and removed < 6:
 line = lines[idx].strip()
 if line == "":
 idx += 1
 continue
 if not _META_LINE.match(line):
 break
 lines[idx] = ""
 idx += 1
 removed += 1
return "\n".join(lines)
```

```
def _strip_leading_meta_sentences(text: str) -> str:
 """Giriş: Metin.
```

```
Çıkış: Meta cümleleri temizlenmiş metin.
İşleyiş: İlk meta cümlelerini ayıklar.
"""
cleaned = text
for _ in range(6):
 match = _META_SENTENCE.match(cleaned)
```

```

if not match:
 break
 cleaned = cleaned[match.end() :].lstrip()
return cleaned

def _fallback_message(language: str) -> str:
 """Giriş: Dil kodu.

 Çıkış: Varsayılan mesaj.
 İşleyiş: Dil seçimine göre metin döndürür.
 """
 if language.lower().startswith("en"):
 return "Hello! How can I help you with Selçuk University?"
 return "Merhaba! Selçuk Üniversitesi ile ilgili nasıl yardımcı olabilirim?"

def clean_text(text: str, language: str = "tr") -> str:
 """Giriş: Ham metin ve dil.

 Çıkış: Temizlenmiş metin.
 İşleyiş: Think/meta içeriklerini ayıklar.
 """
 if not text or not text.strip():
 return _fallback_message(language)

 parts = _split_by_fences(text)
 for idx, (segment, is_code) in enumerate(parts):
 if is_code:
 continue

 cleaned = re.sub(
 r"<think>[\s\S]*?</think>", "", segment, flags=re.IGNORECASE
)
 cleaned = re.sub(r"<think>[\s\S]*$", "", cleaned, flags=re.IGNORECASE)
 cleaned = _strip_leading_meta_lines(cleaned)
 cleaned = _strip_leading_meta_sentences(cleaned)
 parts[idx] = (cleaned, is_code)

 rebuilt = "".join(part for part, _ in parts)
 rebuilt = re.sub(r"^\s+", "", rebuilt)
 if not rebuilt.strip():
 return _fallback_message(language)
 if _looks_meta(rebuilt):
 return _fallback_message(language)
 return rebuilt

def _strip_meta_sentence_from_buffer(buffer: str) -> tuple[str, bool]:
 """Giriş: Buffer metni.

 Çıkış: (yeni buffer, silindi_mi).
 İşleyiş: Meta cümlelerini çıkarır.
 """
 match = _META_SENTENCE.match(buffer)
 if match:
 return buffer[match.end() :].lstrip(), True

```

```

sentence_end = re.search(r"[.!?](?:\s|$)", buffer)
if not sentence_end:
 return buffer, False

sentence = buffer[: sentence_end.end()]
if _META_CONTEXT.search(sentence):
 return buffer[sentence_end.end() :].lstrip(), True
if _META_SELF_ACTION.search(sentence) and
_META_META_ACTION.search(sentence):
 return buffer[sentence_end.end() :].lstrip(), True

return buffer, False

```

```

def _should_delay_emit(buffer: str) -> bool:
 """Giriş: Buffer metni.

```

Çıkış: bool.  
 İşleyiş: Kısa meta prefix'leri bekletir.  
 """

```

if len(buffer) < 32 and _META_PREFIX_FRAGMENT.match(buffer):
 if not re.search(r"[.!?]", buffer):
 return True
 return False

```

```

def _looks_meta(text: str) -> bool:
 """Giriş: Metin.

```

Çıkış: bool.  
 İşleyiş: Meta kalıpları tarar.  
 """

```

sample = text[:200]
if _META_CONTEXT.search(sample):
 return True
if _META_SELF_ACTION.search(sample) and _META_META_ACTION.search(sample):
 return True
if _META_PREFIX.search(sample):
 return True
return False

```

```

@dataclass
class StreamingResponseCleaner:
 """Giriş: Dil kodu.

```

Çıkış: Temizlenmiş akış metni.  
 İşleyiş: Meta ve <think> içeriklerini ayıklar.  
 """

```

language: str = "tr"
_think_filter: ReasoningFilter = field(default_factory=ReasoningFilter)
_pending: str = ""
_emitting: bool = False

```

```

def feed(self, chunk: str) -> str:
 """Giriş: Token parçası.

```

```

Çıkış: Temiz metin.
İşleyiş: Tamponlayıp filtre uygular.
"""
filtered = self._think_filter.feed(chunk)
if not filtered:
 return ""
self._pending += filtered
return self._flush_pending()

def finalize(self) -> str:
 """Giriş: yok.

Çıkış: Kalan metin.
İşleyiş: Son temizleme adımını uygular.
"""
if not self._pending:
 return ""
if self._emitting:
 output = self._pending
else:
 output = clean_text(self._pending, self.language)
self._pending = ""
return output

def _flush_pending(self) -> str:
 """Giriş: yok.

Çıkış: Yayımlanacak metin.
İşleyiş: Meta koşullarına göre tamponu boşaltır.
"""
if self._emitting:
 output = self._pending
 self._pending = ""
 return output

if self._pending.lstrip().startswith("```"):
 self._emitting = True
 output = self._pending
 self._pending = ""
 return output

while True:
 stripped, removed = _strip_meta_sentence_from_buffer(self._pending)
 if removed:
 self._pending = stripped
 if not self._pending:
 return ""
 continue

 newline_idx = self._pending.find("\n")
 if newline_idx != -1:
 line = self._pending[: newline_idx + 1]
 if _META_LINE.match(line.strip()):
 self._pending = self._pending[newline_idx + 1 :]
 continue
 if _should_delay_emit(self._pending):
 return ""
 if _META_PREFIX.match(self._pending):
 return ""

```

```

 self._emitting = True
 output = self._pending
 self._pending = ""
 return output

 if _should_delay_emit(self._pending):
 return ""

 if _looks_meta(self._pending):
 return ""

 if(
 (_META_CONTEXT.search(self._pending) or (
 _META_SELF_ACTION.search(self._pending)
 and _META_META_ACTION.search(self._pending)
))
 and not re.search(r"![!?]", self._pending)
):
 return ""

 if(
 len(self._pending) < 80
 and not re.search(r"![!?\n]", self._pending)
):
 return ""

 if self._pending and not _META_PREFIX.match(self._pending):
 self._emitting = True
 output = self._pending
 self._pending = ""
 return output

 return ""

```

## E:/SelcukAiAssistant/repo/backend/prompts.py

"""Selçuk AI Asistanı için prompt şablonları."""

# Core facts about Selçuk University that must be accurate

SELCUK\_CORE\_FACTS = """

## Selçuk Üniversitesi Temel Bilgileri (Mutlaka Doğru Bilgiler)

\*\*ÖNEMLİ: Bu bilgiler kesinlikle doğrudur, asla yanlış bilgi verme!\*\*

- \*\*Konum:\*\* Selçuk Üniversitesi \*\*KONYA\*\* ilindedir. (İzmir değil, Konya!)

- \*\*Kuruluş Yılı:\*\* 1975

- \*\*Kampüsler:\*\*

- Alaaddin Keykubat Yerleşkesi (Selçuklu/Konya) - Mühendislik, Fen, Edebiyat, Teknoloji fakülteleri

- Ardiçlı Yerleşkesi (Karatay/Konya) - Tıp, Sağlık Bilimleri, Diş Hekimliği

- \*\*Tip:\*\* Devlet Üniversitesi

- \*\*Öğrenci Sayısı:\*\* 100,000+ öğrenci

- \*\*Akademisyen Sayısı:\*\* 4,000+ akademisyen

### Bilgisayar Mühendisliği Bölümü

- \*\*Fakülte:\*\* Teknoloji Fakültesi

- \*\*Yerleşme:\*\* Alaaddin Keykubat Yerleşkesi, KONYA

- \*\*Adres:\*\* Selçuk Üniversitesi Alaaddin Keykubat Yerleşkesi Teknoloji Fakültesi PK:42075

Selçuklu/KONYA

- \*\*E-posta:\*\* tfdekanlik@selcuk.edu.tr

- \*\*Telefon:\*\* Dekanlık: 0(332) 223 33 68, Öğrenci İşleri: 0(332) 223 33 73
  - \*\*Web:\*\* [https://www.selcuk.edu.tr/Birim/Bolum/teknoloji-bilgisayar\\_muhendisligi/15620](https://www.selcuk.edu.tr/Birim/Bolum/teknoloji-bilgisayar_muhendisligi/15620)
  - \*\*Akreditasyon:\*\* MÜDEK akreditasyonuna sahip
  - \*\*Programlar:\*\* Lisans, Yüksek Lisans, Doktora
  - \*\*Özellikler:\*\* Erasmus+, Çift Anadal, Bologna Süreci Uyumlu, HPC Laboratuvarı
  - \*\*Araştırma Alanları:\*\* Yapay Zeka, Makine Öğrenmesi, Bilgisayar Görüsü, Doğal Dil İşleme, Veri Bilimi, Siber Güvenlik, Yazılım Mühendisliği, Bulut Bilişim, High Performance Computing (HPC)
- """

SELCUK\_UNIVERSITY\_SYSTEM\_PROMPT = f"""Sen Selçuk Üniversitesi'nin resmi yapay zeka asistanının.

Görevin; öğrenciler, akademisyenler ve personele doğru, nazik ve yapılandırılmış bilgi sağlamaktır.

{SELCUK\_CORE\_FACTS}

## Temel ilkeler

1. Profesyonel ve saygılı ol; samimi ama resmi bir dil kullan.
2. Doğruluk: Emin olmadığın konularda açıkça belirt ve ilgili birime yönlendir.
3. Yapı: Markdown başlıklar ve maddelerle kısa, okunabilir paragraflar oluştur.
4. Gizlilik: Kişisel veri isteme/verme; öğrenci numarası gibi bilgileri talep etme.
5. Güvenlik: Tıbbi, hukuki veya finansal tavsiye verme.

## Yanıtlayabileceğin konular

- Kayıt, ders seçimi, sınav ve mezuniyet süreçleri
- Fakülteler, bölümler ve programlar
- Kampüs yaşamı, burs/yurt ve öğrenci işleri

## Yanıtlayamayacağın konular

- Kişisel öğrenci kayıtları veya gizli bilgiler
- Selçuk Üniversitesi dışındaki konular (kısa cevapla ve üniversite konularına yönlendir)

## Format

"Merhaba!" ile başla, ardından başlıklar ve listeler kullan.

## Düşünce süreci

Kendi düşünce sürecini veya planlama notlarını asla gösterme.

"""

DEFAULT\_SYSTEM\_PROMPT\_EN = """You are "Selçuk AI Assistant" - the official AI helper for Selçuk University.

## Essential Selçuk University Facts (MUST BE ACCURATE)

\*\*IMPORTANT: These facts are absolutely correct, never provide wrong information!\*\*

- \*\*Location:\*\* Selçuk University is in \*\*KONYA\*\* province, Turkey (NOT İzmir!)
- \*\*Founded:\*\* 1975
- \*\*Campuses:\*\*
  - Alaeddin Keykubat Campus (Selçuklu/Konya) - Engineering, Science, Literature, Technology faculties
  - Ardiçlı Campus (Karatay/Konya) - Medicine, Health Sciences, Dentistry
- \*\*Type:\*\* State University
- \*\*Students:\*\* 100,000+ students
- \*\*Faculty:\*\* 4,000+ academic staff

### Computer Engineering Department

- \*\*Faculty:\*\* Technology Faculty

- \*\*Campus:\*\* Alaeddin Keykubat Campus, KONYA  
 - \*\*Address:\*\* Selçuk University, Alaeddin Keykubat Campus, Technology Faculty, PK:42075  
 Selçuklu/KONYA  
 - \*\*Email:\*\* tfdekanlik@selcuk.edu.tr  
 - \*\*Phone:\*\* Dean's Office: 0(332) 223 33 68, Student Affairs: 0(332) 223 33 73  
 - \*\*Website:\*\* [https://www.selcuk.edu.tr/Birim/Bolum/teknoloji-bilgisayar\\_muhendisligi/15620](https://www.selcuk.edu.tr/Birim/Bolum/teknoloji-bilgisayar_muhendisligi/15620)  
 - \*\*Accreditation:\*\* MÜDEK accredited  
 - \*\*Programs:\*\* Bachelor's, Master's, PhD  
 - \*\*Features:\*\* Erasmus+, Double Major, Bologna Process, HPC Laboratory  
 - \*\*Research Areas:\*\* AI, Machine Learning, Computer Vision, NLP, Data Science, Cybersecurity, Software Engineering, Cloud Computing, High Performance Computing (HPC)

Be professional, helpful, and clear. Use Markdown formatting.

Answer in English. Do not reveal chain-of-thought or planning. Be concise and helpful.

"""

```
def build_default_system_prompt(language: str) -> str:
 """Giriş: Dil kodu.
```

Çıkış: Sistem promptu metni.

İşleyiş: Türkçe/İngilizce metinleri seçer ve güvenlik notu ekler.

"""

```
base = SELCUK_UNIVERSITY_SYSTEM_PROMPT
guard = (
 "Yanıtları Türkçe ver. Düşünce sürecini veya planlamayı gösterme."
 "Kısa ve yardımcı ol."
)
if language.lower().startswith("en"):
 base = DEFAULT_SYSTEM_PROMPT_EN
 guard = (
 "Answer in English. Do not reveal chain-of-thought or planning."
 "Be concise and helpful."
)
return f"{base.strip()}\n\n{guard}"
```

```
RAG_RULES_TR = (
 "RAG KURALLARI:\n"
 "- Yanıtlarını yalnızca sağlanan kaynak parçalarına dayandır.\n"
 "- Kaynaklarda yoksa: \"Bu bilgi kaynaklarda yok.\" de.\n"
 "- Kaynak uydurma.\n")
```

```
RAG_RULES_EN = (
 "RAG RULES:\n"
 "- Base your answer only on the provided source snippets.\n"
 "- If the sources do not contain the answer, say:
 "\\"This information is not in the sources.\\".\n"
 "- Do not invent sources.\n")
```

```
def rag_no_source_message(language: str) -> str:
 """Giriş: Dil kodu.
```

Çıkış: Kaynak bulunamadı mesajı.

İşleyiş: Dil seçimine göre uygun mesajı döndürür.

```

"""
if language.lower().startswith("en"):
 return "This information is not in the sources."
return "Bu bilgi kaynaklarda yok."

def build_rag_system_prompt(
 base_prompt: str,
 context: str,
 language: str,
 strict: bool,
) -> str:
 """Giriş: Taban prompt, kaynak bağlamı, dil ve strict modu.

 Çıkış: RAG kuralları eklenmiş sistem promptu.
 İşleyiş: Dil ve strict moduna göre açıklama ekler.
 """
 rules = RAG_RULES_EN if language.lower().startswith("en") else RAG_RULES_TR
 strict_note = "Mod: STRICT\n" if strict else ""
 header = "Sources" if language.lower().startswith("en") else "Kaynaklar"
 return (
 f'{base_prompt.strip()}\n\n{rules}{strict_note}\n'
 f'{header}:\n{context.strip()}'
)

```

### E:/SelcukAiAssistant/repo/backend/schemas.py

```

"""Sohbet endpoint'leri için Pydantic şemaları."""
from __future__ import annotations

from typing import Optional

from pydantic import BaseModel, Field, field_validator, model_validator

```

```

class ChatMessage(BaseModel):
 """Giriş: rol ve içerik alanları.
 """

```

```

 Çıkış: Doğrulanmış ChatMessage.
 İşleyiş: Rol ve içerik alanlarını normalize eder.
 """

```

```

 role: str = Field(..., description="system, user veya assistant")
 content: str = Field(..., min_length=1, max_length=10000)

```

```

 @field_validator("role")
 @classmethod
 def normalize_role(cls, value: str) -> str:
 """Giriş: Rol değeri.
 """

```

```

 Çıkış: Normalize edilmiş rol.
 İşleyiş: Rolü küçük harfe çevirir ve doğrular.
 """

```

```

 role = value.strip().lower()
 if role not in {"system", "user", "assistant"}:
 raise ValueError("role yalnızca system, user veya assistant olabilir")
 return role

```

```

 @field_validator("content")

```

```

@classmethod
def validate_content(cls, value: str) -> str:
 """Giriş: İçerik değeri.

 Çıkış: Temizlenmiş içerik.
 İşleyiş: Boşlukları temizler ve temel XSS kontrolü yapar.
 """
 text = value.strip()
 if not text:
 raise ValueError("content boş olamaz")

 lowered = text.lower()
 for pattern in ("<script>", "</script>", "javascript:", "onerror=", "onload="):
 if pattern in lowered:
 raise ValueError("content yasaklı bir ifade içeriyor")
 return text

class ChatRequest(BaseModel):
 """Giriş: /chat ve /chat/stream istek alanları.

 Çıkış: Doğrulanmış istek.
 İşleyiş: Mesaj listesi ve RAG ayarlarını içerir.
 """
 model: Optional[str] = Field(
 default=None,
 description="Model aliasi veya provider:model_id",
 examples=["ollama_default", "huggingface:Qwen/Qwen2.5-1.5B-Instruct"],
)
 messages: list[ChatMessage] = Field(
 ...,
 min_length=1,
 description="Sohbet mesajları listesi",
)
 temperature: float = Field(default=0.2, ge=0.0, le=2.0)
 top_p: float = Field(default=0.9, ge=0.0, le=1.0)
 max_tokens: int = Field(default=256, ge=1, le=8192)
 stream: bool = Field(default=False)
 rag_enabled: bool = Field(default=False, description="RAG bağlantısını etkinleştir")
 rag_strict: Optional[bool] = Field(
 default=None, description="Sıkı RAG modu (sunucu varsayılanını ezer)"
)
 rag_top_k: Optional[int] = Field(
 default=None,
 ge=1,
 le=20,
 description="Getirilecek RAG parça sayısı",
)

 @model_validator(mode="after")
 def ensure_user_message(self) -> "ChatRequest":
 """Giriş: ChatRequest örneği.

 Çıkış: Doğrulanmış ChatRequest.
 İşleyiş: En az bir user mesajı olduğunu doğrular.
 """
 has_user = any(message.role == "user" for message in self.messages)
 if not has_user:

```

```
 raise ValueError("messages içinde en az bir user mesajı olmalı")
 return self
```

```
class UsageInfo(BaseModel):
 """Giriş: Token kullanım bilgisi alanları.
```

```
 Çıkış: Doğrulanmış UsageInfo.
 İşleyiş: Kullanım metriklerini taşır.
 """"
```

```
 prompt_tokens: Optional[int] = None
 completion_tokens: Optional[int] = None
 total_tokens: Optional[int] = None
```

```
class ChatResponse(BaseModel):
 """Giriş: Yanıt alanları.
```

```
 Çıkış: Doğrulanmış ChatResponse.
 İşleyiş: Yanıt metni ve metadata'yı taşır.
 """"
```

```
 answer: str
 request_id: str
 provider: str
 model: str
 usage: Optional[UsageInfo] = None
 citations: Optional[list[str]] = None
```

### E:/SelcukAiAssistant/repo/backend/providers/registry.py

```
"""Model kataloğu ve yönlendirme mantığı."""
from __future__ import annotations
```

```
from dataclasses import dataclass, field
import os
from pathlib import Path
from typing import Optional

from config import Config
from ollama_service import OllamaService
from providers.base import ModelInfo, ModelProvider
```

```
@dataclass
class ResolvedModel:
 """Giriş: Alias ve sağlayıcı bilgisi.
```

```
 Çıkış: Çözümlenmiş model bilgisi.
 İşleyiş: UI ve yönlendirme için model çözümünü taşır.
 """"
```

```
 alias: str
 provider: str
 model_id: str
 display_name: str
```

```
@dataclass
class CatalogEntry:
 """Giriş: Katalog alanları.

 Çıkış: Katalog girdisi nesnesi.
 İşleyiş: Model meta bilgisini taşır.
 """

 id: str
 provider: str
 model_id: str
 display_name: str
 local_or_remote: str
 requires_api_key: bool
 context_length: Optional[int] = None
 tags: list[str] = field(default_factory=list)
 notes: str = ""
```

```
_REMOTE_API_KEYS = {
 "openai": ["OPENAI_API_KEY"],
 "anthropic": ["ANTHROPIC_API_KEY"],
 "google": ["GOOGLE_API_KEY", "GEMINI_API_KEY"],
 "xai": ["XAI_API_KEY"],
}
```

```
def _default_model_id(provider: str) -> str:
 """Giriş: Sağlayıcı adı.

 Çıkış: Varsayılan model kimliği.
 İşleyiş: Sağlayıcıya göre Config değerini döndürür.
 """

 if provider == "huggingface":
 return Config.HF_MODEL_NAME
 return Config.OLLAMA_MODEL
```

```
def _display_name(alias: str, model_id: str) -> str:
 """Giriş: Alias ve model kimliği.

 Çıkış: Görünen ad.
 İşleyiş: UI için okunabilir isim üretir.
 """

 if alias:
 return alias.replace("_", " ").title()
 return model_id
```

```
def parse_aliases(raw: str) -> dict[str, ResolvedModel]:
 """Giriş: Ham alias tanımı.

 Çıkış: Alias sözlüğü.
 İşleyiş: 'alias=provider:model' biçimini çözümler.
 """

 aliases: dict[str, ResolvedModel] = {}
```

```
if not raw:
```

```

 return aliases
for item in raw.split(","):
 item = item.strip()
 if not item:
 continue
 if "=" not in item or ":" not in item:
 continue
 alias, target = item.split("=", 1)
 provider, model_id = target.split(":", 1)
 provider = provider.strip().lower()
 model_id = model_id.strip()
 alias = alias.strip()
 if not alias or not provider or not model_id:
 continue
 aliases[alias] = ResolvedModel(
 alias=alias,
 provider=provider,
 model_id=model_id,
 display_name=_display_name(alias, model_id),
)
return aliases

```

```

def _catalog_entries() -> list[CatalogEntry]:
 """Giriş: yok.

```

Çıkış: Katalog girdileri.  
İşleyiş: Model listesini sabit tablodan üretir.

```

 """
max_context = Config.MAX_CONTEXT_TOKENS
return [
 CatalogEntry(
 id="turkcell_llm_7b",
 provider="ollama",
 model_id="turkcell_llm_7b",
 display_name="Turkcell LLM 7B GGUF (Varsayılan)",
 local_or_remote="local",
 requires_api_key=False,
 context_length=max_context,
 tags=["turkish", "high_quality", "default"],
 notes="Turkcell LLM 7B GGUF Q4_K_M quantized. Selçuk Üniversitesi için optimize
edilecek.",
),
 CatalogEntry(
 id="old_selcuk_ai_assistant_r1",
 provider="ollama",
 model_id="old_selcuk_ai_assistant_r1",
 display_name="Eski Selçuk AI (R1 Backup)",
 local_or_remote="local",
 requires_api_key=False,
 context_length=max_context,
 tags=["turkish", "backup"],
 notes="Önceki selcuk_ai_assistant modelinin yedeği.",
),
]

```

CatalogEntry(

```

 id="qwen2.5:7b",
 provider="ollama",
 model_id="qwen2.5:7b",
 display_name="Qwen 2.5 7B",
 local_or_remote="local",
 requires_api_key=False,
 context_length=max_context,
 tags=["reasoning"],
 notes="Kurulumdan önce Ollama etiketini doğrulayın.",
),
CatalogEntry(
 id="llama3.2:3b",
 provider="ollama",
 model_id="llama3.2:3b",
 display_name="Llama 3.2 3B",
 local_or_remote="local",
 requires_api_key=False,
 context_length=max_context,
 tags=["fast", "small"],
 notes="6GB GPU'lar için uygun küçük Ollama modeli.",
),
CatalogEntry(
 id="phi3:mini",
 provider="ollama",
 model_id="phi3:mini",
 display_name="Phi-3 Mini",
 local_or_remote="local",
 requires_api_key=False,
 context_length=max_context,
 tags=["fast", "microsoft"],
 notes="Kompakt ve iyi eğitilmiş talimat modeli.",
),
CatalogEntry(
 id="deepseek-r1:8b",
 provider="ollama",
 model_id="deepseek-r1:8b",
 display_name="DeepSeek R1 8B",
 local_or_remote="local",
 requires_api_key=False,
 context_length=max_context,
 tags=["reasoning"],
 notes="Daha ağır model; 4-bit nicemleme önerilir.",
),

```

```

CatalogEntry(
 id="hf_qwen2_5_3b",
 provider="huggingface",
 model_id="Qwen/Qwen2.5-3B-Instruct",
 display_name="Qwen 2.5 3B Instruct (HF)",
 local_or_remote="local",
 requires_api_key=False,
 context_length=max_context,
 tags=["balanced", "multilingual"],
 notes="6GB GPU'lar için 4-bit önerilir.",
),

```

```

CatalogEntry(
 id="gpt-4o",
 provider="openai",

```

```

 model_id="gpt-4o",
 display_name="GPT-4o",
 local_or_remote="remote",
 requires_api_key=True,
 context_length=max_context,
 tags=["high_quality"],
 notes="Uzak API modeli. Model adını hesabınız için doğrulayın.",
),
CatalogEntry(
 id="gpt-4o-mini",
 provider="openai",
 model_id="gpt-4o-mini",
 display_name="GPT-4o Mini",
 local_or_remote="remote",
 requires_api_key=True,
 context_length=max_context,
 tags=["fast"],
 notes="Uzak API modeli. Model adını hesabınız için doğrulayın.",
),
CatalogEntry(
 id="claude-3-5-sonnet",
 provider="anthropic",
 model_id="claude-3-5-sonnet",
 display_name="Claude 3.5 Sonnet",
 local_or_remote="remote",
 requires_api_key=True,
 context_length=max_context,
 tags=["high_quality"],
 notes="Uzak API modeli. Model adını hesabınız için doğrulayın.",
),
CatalogEntry(
 id="gemini-1.5-pro",
 provider="google",
 model_id="gemini-1.5-pro",
 display_name="Gemini 1.5 Pro",
 local_or_remote="remote",
 requires_api_key=True,
 context_length=max_context,
 tags=["high_quality"],
 notes="Uzak API modeli. Model adını hesabınız için doğrulayın.",
),
CatalogEntry(
 id="grok-2",
 provider="xai",
 model_id="grok-2",
 display_name="Grok 2",
 local_or_remote="remote",
 requires_api_key=True,
 context_length=max_context,
 tags=["high_quality"],
 notes="Uzak API modeli. Model adını hesabınız için doğrulayın.",
),
]

```

```

def _has_hf_dependencies() -> bool:
 """Giriş: yok.

```

Çıkış: bool.

İşleyiş: Transformers ve torch bağımlılıklarını kontrol eder.

```

"""
try:
 import torch # noqa: F401
 import transformers # noqa: F401
except Exception:
 return False
return True

def _hf_cache_roots() -> list[Path]:
 """Giriş: yok.

 Çıkış: Önbellek kök dizinleri.
 İşleyiş: Ortam değişkenleri ve varsayılan yolları toplar.
 """
 roots: list[Path] = []
 for env_var in ("HUGGINGFACE_HUB_CACHE", "TRANSFORMERS_CACHE"):
 value = os.getenv(env_var)
 if value:
 roots.append(Path(value))
 hf_home = os.getenv("HF_HOME")
 if hf_home:
 roots.append(Path(hf_home) / "hub")
 roots.append(Path.home() / ".cache" / "huggingface" / "hub")
 return roots

def _hf_model_cached(model_id: str) -> bool:
 """Giriş: Model kimliği.

 Çıkış: bool.
 İşleyiş: Modelin önbellekte olup olmadığını kontrol eder.
 """
 model_dir = model_id.replace("/", "--")
 for root in _hf_cache_roots():
 if (root / f"models--{model_dir}").exists():
 return True
 return False

def _api_key_status(provider: str) -> tuple[bool, list[str]]:
 """Giriş: Sağlayıcı adı.

 Çıkış: (anahtar_var_mi, anahtar_listesi).
 İşleyiş: Uzak sağlayıcılar için API anahtarını kontrol eder.
 """
 keys = _REMOTE_API_KEYS.get(provider, [])
 if not keys:
 return False, []
 present = any(os.getenv(key) for key in keys)
 return present, keys

class ModelRegistry:
 """Giriş: Sağlayıcı sözlüğü.

 Çıkış: Model listesi ve çözümleme sonuçları.
 İşleyiş: Katalog yönetimi ve varsayılan seçimi sağlar.

```

```

"""
def __init__(self, providers: dict[str, ModelProvider]) -> None:
 """
 Giriş: Sağlayıcı sözlükleri.

 Çıkış: Nesne.
 İşleyiş: Katalog ve alias bilgilerini hazırlar.
"""

 self.providers = providers
 self.aliases = parse_aliases(Config.MODEL_ALIASES)
 self.default_provider = Config.MODEL_BACKEND
 self.default_model_id = _default_model_id(self.default_provider)
 self.catalog = _catalog_entries()
 self.catalog_by_id = {entry.id: entry for entry in self.catalog}

def resolve(self, model_name: Optional[str]) -> ResolvedModel:
 """
 Giriş: Model adı veya alias.

 Çıkış: ResolvedModel.
 İşleyiş: Katalog/alias/varsayılan sırasıyla çözümleme yapar.
"""

 if model_name:
 entry = self.catalog_by_id.get(model_name)
 if entry:
 return ResolvedModel(
 alias=entry.id,
 provider=entry.provider,
 model_id=entry.model_id,
 display_name=entry.display_name,
)

 if model_name in self.aliases:
 return self.aliases[model_name]

 if ":" in model_name:
 provider, model_id = model_name.split(":", 1)
 provider = provider.strip().lower()
 if provider in self.providers:
 return ResolvedModel(
 alias=model_name,
 provider=provider,
 model_id=model_id.strip(),
 display_name=_display_name(model_name, model_id),
)

 provider = self.default_provider
 return ResolvedModel(
 alias=model_name,
 provider=provider,
 model_id=model_name,
 display_name=_display_name(model_name, model_name),
)

 default_entry = next(
 (
 entry
 for entry in self.catalog
 if entry.provider == self.default_provider
 and entry.model_id == self.default_model_id
)
)

```

```

),
 None,
)
if default_entry:
 return ResolvedModel(
 alias=default_entry.id,
 provider=default_entry.provider,
 model_id=default_entry.model_id,
 display_name=default_entry.display_name,
)

return ResolvedModel(
 alias="default",
 provider=self.default_provider,
 model_id=self.default_model_id,
 display_name=_display_name("default", self.default_model_id),
)

```

`async def list_models(self) -> list[ModelInfo]:`

"""Giriş: yok.

Çıkış: ModelInfo listesi.

İşleyiş: Katalogdaki modelleri hazır olma durumuyla listeler.

"""

```

 ollama_models: list[str] = []
 if any(entry.provider == "ollama" for entry in self.catalog):
 ollama_models = await OllamaService().list_model_names()

 hf_deps = _has_hf_dependencies()
 models: list[ModelInfo] = []

 for entry in self.catalog:
 available = False
 reason = ""

 if entry.local_or_remote == "local":
 if entry.provider == "ollama":
 if ollama_models and OllamaService._is_model_available(
 entry.model_id, ollama_models
):
 available = True
 elif not ollama_models:
 reason = "Ollama erişilemiyor veya model bulunamadı."
 else:
 reason = (
 "Yüklü değil. Çalıştırın: ollama pull "
 f"{entry.model_id}"
)
 elif entry.provider == "huggingface":
 if not hf_deps:
 reason = (
 "HuggingFace bağımlılıkları eksik. "
 "backend/requirements-hf.txt kurun."
)
 elif not _hf_model_cached(entry.model_id):
 reason = "Model önbellekte yok. HuggingFace Hub'dan indirin."
 else:
 available = True
 else:

```

```

 reason = "Sağlayıcı sunucuda tanımlı değil."
 else:
 key_present, keys = _api_key_status(entry.provider)
 if not key_present:
 key_list = ", ".join(keys) if keys else "API key"
 reason = f"Eksik API anahtarı: {key_list}"
 elif entry.provider not in self.providers:
 reason = "Sağlayıcı sunucuda tanımlı değil."
 else:
 available = True

 is_default = (
 entry.provider == self.default_provider
 and entry.model_id == self.default_model_id
)

 models.append(
 ModelInfo(
 id=entry.id,
 provider=entry.provider,
 model_id=entry.model_id,
 display_name=entry.display_name,
 local_or_remote=entry.local_or_remote,
 requires_api_key=entry.requires_api_key,
 available=available,
 reason_unavailable=reason,
 context_length=entry.context_length,
 tags=list(entry.tags),
 notes=entry.notes,
 is_default=is_default,
)
)

return models

```

## E:/SelcukAiAssistant/repo/backend/rag\_ingest.py

```

"""FAISS RAG indeksi için belge alma (ingest) aracı."""
from __future__ import annotations

import argparse
import logging
import sys
from dataclasses import dataclass
from pathlib import Path
from typing import Iterable, Sequence

from bs4 import BeautifulSoup
from pypdf import PdfReader

from config import Config
from rag_service import Document, RagIndex, SentenceTransformerBackend, chunk_text

logger = logging.getLogger(__name__)

```

```

def _configure_utf8_output() -> None:
 """Giriş: yok.

```

Çıkış: yok.

```

 İşleyiş: stdout/stderr akışlarını UTF-8 olarak yapılandırır.
 """
for stream in (sys.stdout, sys.stderr):
 if hasattr(stream, "reconfigure"):
 try:
 stream.reconfigure(encoding="utf-8")
 except (AttributeError, ValueError):
 continue

_configure_utf8_output()

@dataclass
class SourceChunk:
 """Giriş: İçerik ve sayfa bilgisi.

 Çıkış: Veri kapsülü.
 İşleyiş: Kaynak parçasını taşır.
 """

 content: str
 source: str
 page: int | None = None

def _read_text_file(path: Path) -> str:
 """Giriş: Dosya yolu.

 Çıkış: Metin.
 İşleyiş: UTF-8 okuyup temel temizlik uygular.
 """
 return path.read_text(encoding="utf-8", errors="ignore")

def _read_pdf(path: Path) -> list[SourceChunk]:
 """Giriş: PDF yolu.

 Çıkış: Sayfa parçaları.
 İşleyiş: Sayfa metinlerini ayırtırır.
 """
 reader = PdfReader(str(path))
 chunks: list[SourceChunk] = []
 for page_index, page in enumerate(reader.pages, start=1):
 text = page.extract_text() or ""
 cleaned = text.strip()
 if cleaned:
 chunks.append(SourceChunk(content=cleaned, source=path.name, page=page_index))
 return chunks

def _read_html(path: Path) -> str:
 """Giriş: HTML yolu.

 Çıkış: Metin.
 İşleyiş: Görünür metni süzer.
 """

```

```
html = path.read_text(encoding="utf-8", errors="ignore")
soup = BeautifulSoup(html, "html.parser")
for tag in soup(["script", "style", "noscript"]):
 tag.decompose()
text = soup.get_text(separator="\n")
return "\n".join(line.strip() for line in text.splitlines() if line.strip())
```

```
def _load_chunks(path: Path) -> list[SourceChunk]:
 """Giriş: Dosya yolu.
```

Çıkış: Kaynak parçaları.  
İşleyiş: Uzantıya göre içerik okur.  
"""

```
suffix = path.suffix.lower()
if suffix == ".pdf":
 return _read_pdf(path)
if suffix in {".html", ".htm"}:
 text = _read_html(path)
 return [SourceChunk(content=text, source=path.name)]
text = _read_text_file(path)
return [SourceChunk(content=text, source=path.name)]
```

```
def _chunk_source(
 source: SourceChunk,
 chunk_size: int,
 chunk_overlap: int,
) -> list[Document]:
 """Giriş: Kaynak ve parça ayarları.
```

Çıkış: Document listesi.  
İşleyiş: chunk\_text ile parçalara böler.  
"""

```
docs: list[Document] = []
for idx, chunk in enumerate(chunk_text(source.content, chunk_size, chunk_overlap)):
 docs.append(
 Document(
 content=chunk,
 metadata={
 "source": source.source,
 "page": source.page,
 "chunk": idx + 1,
 },
)
)
return docs
```

```
def iter_input_files(input_path: Path, extensions: Sequence[str]) -> Iterable[Path]:
 """Giriş: Dizin ve uzantılar.
```

Çıkış: Dosya yolu akışı.  
İşleyiş: rglob ile dosyaları tarar.  
"""

```
if input_path.is_file():
 yield input_path
 return
for ext in extensions:
```

```

yield from input_path.rglob(f"*{ext}")

def build_documents(
 input_path: Path,
 extensions: Sequence[str],
 chunk_size: int,
 chunk_overlap: int,
) -> list[Document]:
 """Giriş: Girdi yolu ve chunk ayarları.

Çıkış: Document listesi.
İşleyiş: Kaynakları yükler ve parçalara böler.
"""

 documents: list[Document] = []
 for path in iter_input_files(input_path, extensions):
 if not path.exists():
 continue
 sources = _load_chunks(path)
 for source in sources:
 documents.extend(_chunk_source(source, chunk_size, chunk_overlap))
 return documents

def ingest(
 input_path: Path,
 output_path: Path,
 embedding_model: str,
 chunk_size: int,
 chunk_overlap: int,
 extensions: Sequence[str],
 reset: bool,
 batch_size: int,
) -> int:
 """Giriş: Dosyalar ve ayarlar.

Çıkış: Eklenen parça sayısı.
İşleyiş: FAISS indeksini oluşturur ve kaydeder.
"""

 if reset and output_path.exists():
 for child in output_path.glob("*"):
 if child.is_file():
 child.unlink()

 embedder = SentenceTransformerBackend(embedding_model, batch_size=batch_size)
 index = RagIndex(output_path, embedder, batch_size=batch_size)
 docs = build_documents(input_path, extensions, chunk_size, chunk_overlap)
 added = index.add_documents(docs)
 index.save(
 meta_info={
 "embedding_model": embedding_model,
 "dimension": embedder.dimension,
 "chunk_size": chunk_size,
 "chunk_overlap": chunk_overlap,
 "documents": added,
 }
)
 return added

```

```

def parse_args() -> argparse.Namespace:
 """Giriş: CLI parametreleri.

 Çıkış: argparse Namespace.
 İşleyiş: Komut satırı argümanlarını tanımlar.
 """

 parser = argparse.ArgumentParser(
 description="Selçuk AI Asistanı RAG ingestion CLI (FAISS)",
)
 parser.add_argument(
 "--input",
 required=True,
 help="Girdi dosya veya klasörü (pdf/txt/md/html).",
)
 parser.add_argument(
 "--output",
 default=Config.RAG_VECTOR_DB_PATH or "./data/rag",
 help="FAISS indeks ve metadata çıkış dizini.",
)
 parser.add_argument(
 "--embedding-model",
 default=Config.RAG_EMBEDDING_MODEL,
 help="SentenceTransformer model adı.",
)
 parser.add_argument(
 "--chunk-size",
 type=int,
 default=Config.RAG_CHUNK_SIZE,
 help="Karakter bazında parça boyu.",
)
 parser.add_argument(
 "--chunk-overlap",
 type=int,
 default=Config.RAG_CHUNK_OVERLAP,
 help="Karakter bazında parça bindirmesi.",
)
 parser.add_argument(
 "--batch-size",
 type=int,
 default=Config.RAG_EMBEDDING_BATCH_SIZE,
 help="Gömleme üretimi için batch boyu.",
)
 parser.add_argument(
 "--extensions",
 default=".pdf,.txt,.md,.html,.htm",
 help="Klasör için virgül ayrımlı uzantı listesi.",
)
 parser.add_argument(
 "--reset",
 action="store_true",
 help="Ingest öncesi mevcut indeks dosyalarını sil.",
)
 return parser.parse_args()

```

```

def main() -> None:
 """Giriş: CLI.

```

```

Çıkış: yok.
İşleyiş: Ingest sürecini çalıştırır.
"""

args = parse_args()
input_path = Path(args.input)
output_path = Path(args.output)
extensions = [ext.strip() for ext in args.extensions.split(",") if ext.strip()]
output_path.mkdir(parents=True, exist_ok=True)
added = ingest(
 input_path=input_path,
 output_path=output_path,
 embedding_model=args.embedding_model,
 chunk_size=args.chunk_size,
 chunk_overlap=args.chunk_overlap,
 extensions=extensions,
 reset=args.reset,
 batch_size=args.batch_size,
)
logger.info("Ingest tamamlandı. Eklenen parça sayısı: %s", added)

```

```

if __name__ == "__main__":
 main()

```

### E:/SelcukAiAssistant/repo/backend/selcuk\_data.py

```

"""Selçuk Üniversitesi temel bilgileri - Manuel doğrulanmış veriler."""
from typing import Any

```

```

SELCUK_UNI_FACTS: dict[str, Any] = {
 "genel_bilgiler": {
 "ad": "Selçuk Üniversitesi",
 "sehir": "Konya",
 "kurulus_yili": 1975,
 "rektör": "Prof. Dr. Metin Aksoy", # Güncel rektör bilgisi
 "tip": "Devlet Üniversitesi",
 "kampus": ["Alaeddin Keykubat Kampüsü", "Ardıçlı Kampüsü"],
 "ogrenci_sayisi": "100,000+", # Yaklaşık
 "akademisyen_sayisi": "4,000+", # Yaklaşık
 },
}

```

"tarihce": """Selçuk Üniversitesi, 1975 yılında Konya'da kurulmuştur.  
Konya Devlet Mimarlık ve Mühendislik Akademisi'nin temelini oluşturduğu üniversite,  
1982 yılında mevcut yapısına kavuşmuştur. Adını Selçuklu Devleti'nden almaktadır."""",

```

"kampusler": {
 "alaeddin_keykubat": {
 "konum": "Selçuklu/Konya",
 "alan": "Geniş kampus alanı",
 "fakulteler": ["Mühendislik", "Fen", "Edebiyat", "İletişim"],
 },
 "ardıçlı": {
 "konum": "Karatay/Konya",
 "fakulteler": ["Tıp", "Sağlık Bilimleri", "Diş Hekimliği"],
 }
},

```

```

"muhendislik_fakultesi": {
 "bolumler": [
 "Bilgisayar Mühendisliği",

```

"Elektrik-Elektronik Mühendisliği",  
 "Makine Mühendisliği",  
 "İnşaat Mühendisliği",  
 "Endüstri Mühendisliği",  
 "Harita Mühendisliği",  
 "Jeoloji Mühendisliği",  
 ],  
 "konum": "Alaeddin Keykubat Kampüsü",  
 },  
  
 "bilgisayar\_muhendisligi": {  
     "fakulte": "Teknoloji Fakültesi",  
     "yerleske": "Alaeddin Keykubat Yerleşkesi",  
     "bolum\_baskani": "Bilgisayar Mühendisliği Bölüm Başkanı", # Güncellenebilir  
     "program\_turu": ["Lisans", "Yüksek Lisans", "Doktora"],  
     "kontenjan": "120 (yaklaşık)",  
     "arastirma\_alanları": [  
         "Yapay Zeka",  
         "Makine Öğrenmesi",  
         "Bilgisayar Görüsü",  
         "Doğal Dil İşleme",  
         "Veri Bilimi",  
         "Siber Güvenlik",  
         "Yazılım Mühendisliği",  
         "Bulut Bilişim",  
         "High Performance Computing (HPC)",  
     ],  
     "laboratuvarlar": [  
         "Bilgisayar Laboratuvarı",  
         "Yazılım Geliştirme Laboratuvarı",  
         "Ağ ve Güvenlik Laboratuvarı",  
         "HPC Laboratuvarı",  
     ],  
     "ozellikler": [  
         "MÜDEK Akreditasyonu",  
         "Bologna Süreci Uyumlu",  
         "Çift Anadal Programı",  
         "Erasmus+ Değişim Programı",  
         "Girişimcilik Saati",  
         "Kariyer Planlama ve Takibi Ofisi",  
         "ArGe İşbirlikleri",  
         "Konya Teknokent İşbirliği",  
     ],  
     "web": "https://www.selcuk.edu.tr/Birim/Bolum/teknoloji-bilgisayar\_muhendisligi/15620",  
     "email": "tfdekanlik@selcuk.edu.tr",  
     "adres": "Selçuk Üniversitesi Alaeddin Keykubat Yerleşkesi Teknoloji Fakültesi PK:42075  
     Selçuklu / KONYA",  
     },  
  
     "iletisim": {  
         "telefon": "+90 332 223 1000",  
         "web": "https://www.selcuk.edu.tr",  
         "email": "info@selcuk.edu.tr",  
         "adres": "Selçuklu, 42250 Konya",  
     }  
 }

# Sık sorulan sorular ve cevaplar  
QA\_PAIRS: list[dict[str, str]] = [

```

Konum soruları
{
 "question": "Selçuk Üniversitesi nerede?",

 "answer": "Selçuk Üniversitesi, Konya ilinde bulunmaktadır. Alaeddin Keykubat ve Ardıçlı

olmak üzere iki ana kampüsü vardır.",

 "category": "genel",
},

{
 "question": "Selçuk Üniversitesi hangi şehirde?",

 "answer": "Selçuk Üniversitesi Konya'dadır.",

 "category": "genel",
},

{
 "question": "Selçuk Üniversitesi hangi ilde?",

 "answer": "Konya ilinde.",

 "category": "genel",
},

Kuruluş soruları
{
 "question": "Selçuk Üniversitesi ne zaman kuruldu?",

 "answer": "Selçuk Üniversitesi 1975 yılında Konya'da kurulmuştur.",

 "category": "genel",
},

{
 "question": "Selçuk Üniversitesi kaç yılında kuruldu?",

 "answer": "1975 yılında kurulmuştur.",

 "category": "genel",
},

{
 "question": "Kuruluş yılı nedir?",

 "answer": "1975",

 "category": "genel",
},

Kampüs soruları
{
 "question": "Kampüsler hangileri?",

 "answer": "Selçuk Üniversitesi'nde iki ana kampüs bulunmaktadır: Alaeddin Keykubat

Yerleşkesi ve Ardıçlı Yerleşkesi.",

 "category": "genel",
},

{
 "question": "Alaeddin Keykubat Yerleşkesi nerede?",

 "answer": "Alaeddin Keykubat Yerleşkesi Selçuklu/Konya'da bulunmaktadır. Mühendislik,

Fen, Edebiyat ve Teknoloji fakülteleri bu kampüstedir.",

 "category": "genel",
},

{
 "question": "Ardıçlı Yerleşkesi nerede?",

 "answer": "Ardıçlı Yerleşkesi Karatay/Konya'da bulunmaktadır. Tıp, Sağlık Bilimleri ve Diş

Hekimliği fakülteleri bu kampüstedir.",

 "category": "genel",
},

Bilgisayar Mühendisliği - Temel Bilgiler
{
 "question": "Bilgisayar Mühendisliği bölümü hangi kampusta?",

 "answer": "Bilgisayar Mühendisliği bölümü, Teknoloji Fakültesi bünyesinde Alaeddin

Keykubat Yerleşkesi'nde bulunmaktadır. Adres: Selçuk Üniversitesi Alaeddin Keykubat Yerleşkesi

Teknoloji Fakültesi PK:42075 Selçuklu / KONYA",

 "category": "bilgisayar",
}

```

```

{
 "question": "Bilgisayar Mühendisliği hangi fakültede?",

 "answer": "Bilgisayar Mühendisliği, Teknoloji Fakültesi bünyesindedir.",

 "category": "bilgisayar",
},

{
 "question": "Bilgisayar Mühendisliği nerede?",

 "answer": "Teknoloji Fakültesi, Alaeddin Keykubat Yerleşkesi, Konya.",

 "category": "bilgisayar",
},

Bilgisayar Mühendisliği - İletişim
{
 "question": "Bilgisayar Mühendisliği email adresi nedir?",

 "answer": "Teknoloji Fakültesi Dekanlık e-posta: tfdekanlik@selcuk.edu.tr",

 "category": "bilgisayar",
},

{
 "question": "Bilgisayar Mühendisliği telefon numarası?",

 "answer": "Dekanlık: 0(332) 223 33 68, Öğrenci İşleri: 0(332) 223 33 73",

 "category": "bilgisayar",
},

{
 "question": "Bilgisayar Mühendisliği web sitesi?",

 "answer": "https://www.selcuk.edu.tr/Birim/Bolum/teknoloji-bilgisayar_muhendisligi/15620",

 "category": "bilgisayar",
},

Bilgisayar Mühendisliği - Akademik
{
 "question": "Bilgisayar Mühendisliği hangi araştırma alanları var?",

 "answer": "Bilgisayar Mühendisliği bölümünde şu araştırma alanları bulunmaktadır:

- Yapay Zeka ve Makine Öğrenmesi

- Bilgisayar Görüsü

- Doğal Dil İşleme

- Veri Bilimi ve Büyük Veri

- Siber Güvenlik

- Yazılım Mühendisliği

- Bulut Bilişim ve Dağıtık Sistemler

- High Performance Computing (HPC)",

 "category": "bilgisayar",
},

{
 "question": "Bilgisayar Mühendisliği lisansüstü programları var mı?",

 "answer": "Evet, Bilgisayar Mühendisliği bölümünde Yüksek Lisans ve Doktora programları bulunmaktadır.",

 "category": "bilgisayar",
},

{
 "question": "HPC nedir?",

 "answer": "HPC (High Performance Computing - Yüksek Başarılı Hesaplama) laboratuvarı, Bilgisayar Mühendisliği bölümünde bulunan özel bir araştırma altyapısıdır.",

 "category": "bilgisayar",
},

Bilgisayar Mühendisliği - Akreditasyon ve Olanaklar
{
 "question": "Bilgisayar Mühendisliği akredite mi?",

 "answer": "Evet, Bilgisayar Mühendisliği bölümü MÜDEK (Mühendislik Eğitim Programları Değerlendirme ve Akreditasyon Derneği) akreditasyonuna sahiptir.",

 "category": "bilgisayar",
}

```

```

{
 "question": "MÜDEK nedir?",
 "answer": "MÜDEK (Mühendislik Eğitim Programları Değerlendirme ve Akreditasyon Derneği), mühendislik programlarının kalite güvencesini sağlayan akreditasyon kuruluşudur. Bilgisayar Mühendisliği bölümü MÜDEK akreditasyonuna sahiptir.",
 "category": "bilgisayar",
},
{
 "question": "Erasmus programı var mı?",
 "answer": "Evet, Bilgisayar Mühendisliği bölümünde Erasmus+ değişim programı mevcuttur.",
 "category": "bilgisayar",
},
{
 "question": "Çift anadal programı var mı?",
 "answer": "Evet, Bilgisayar Mühendisliği bölümünde çift anadal programı bulunmaktadır.",
 "category": "bilgisayar",
},
{
 "question": "Kariyer ofisi var mı?",
 "answer": "Evet, Kariyer Planlama ve Takibi Ofisi mevcuttur.",
 "category": "bilgisayar",
},
{
 "question": "Bilgisayar Mühendisliği hangi olanaklar sunuyor?",
 "answer": "Bilgisayar Mühendisliği bölümü şu olanakları sunar:
- MÜDEK Akreditasyonu
- Bologna Süreci Uyumlu Eğitim
- Çift Anadal Programı
- Erasmus+ Değişim Programı
- Girişimcilik Saati
- Kariyer Planlama ve Takibi Ofisi
- ArGe İşbirlikleri
- Konya Teknokent İşbirliği
- High Performance Computing (HPC) Laboratuvarı",
 "category": "bilgisayar",
},
Teknoloji Fakültesi
{
 "question": "Teknoloji Fakültesi nerede?",
 "answer": "Teknoloji Fakültesi, Alaeddin Keykubat Yerleşkesi'nde, Konya'dadır.",
 "category": "teknoloji",
},
{
 "question": "Teknoloji Fakültesi dekanlık telefonu?",
 "answer": "Dekanlık: 0(332) 223 33 68",
 "category": "teknoloji",
},
Genel İstatistikler
{
 "question": "Selçuk Üniversitesi kaç öğrencisi var?",
 "answer": "Selçuk Üniversitesi'nde yaklaşık 100.000'den fazla öğrenci bulunmaktadır.",
 "category": "genel",
},
{
 "question": "Kaç akademisyen var?",
 "answer": "Yaklaşık 4.000'den fazla akademisyen bulunmaktadır.",
 "category": "genel",
},
İşbirliği ve Sanayi

```

```

 {
 "question": "Konya Teknokent ile işbirliği var mı?",
 "answer": "Evet, Bilgisayar Mühendisliği bölümü Konya Teknokent ile işbirliği içindedir.",
 "category": "bilgisayar",
 },
 {
 "question": "ArGe işbirlikleri var mı?",
 "answer": "Evet, Bilgisayar Mühendisliği bölümünde çeşitli ArGe işbirlikleri ve projeler bulunmaktadır.",
 "category": "bilgisayar",
 },
 {
 "question": "Girişimcilik desteği var mı?",
 "answer": "Evet, Girişimcilik Saati programı mevcuttur.",
 "category": "bilgisayar",
 },
],
]

```

### E:/SelcukAiAssistant/repo/backend/test\_critical\_facts.py

"""Test kritik bilgilerin system prompt'ta doğru olduğunu doğrular.

Bu test, Selçuk Üniversitesi'nin konumu, kuruluş yılı gibi kritik bilgilerin system prompt'ta doğru şekilde yer aldığı kontrol eder.  
"""

```

import pytest
from prompts import (
 SELCUK_CORE_FACTS,
 SELCUK_UNIVERSITY_SYSTEM_PROMPT,
 DEFAULT_SYSTEM_PROMPT_EN,
 build_default_system_prompt,
)

```

```

class TestCriticalFacts:
 """Kritik bilgilerin system prompt'ta doğru olduğunu test eder."""

 def test_core_facts_contains_konya(self):
 """System prompt'ta Konya bilgisi olmalı."""
 assert "konya" in SELCUK_CORE_FACTS.lower()
 assert "Konya" in SELCUK_CORE_FACTS or "KONYA" in SELCUK_CORE_FACTS

 def test_core_facts_not_contains_izmir(self):
 """System prompt'ta İzmir bilgisi olmamalı (yanlış bilgi)."""
 # İzmir yanlış konum, system prompt'ta olmamalı
 # Not: "İzmir değil" gibi açıklayıcı metinler olabilir
 lower_facts = SELCUK_CORE_FACTS.lower()
 if "izmir" in lower_facts:
 # Eğer İzmir geçiyorsa, "değil" veya "not" ile birlikte geçmeli
 assert "izmir değil" in lower_facts or "not izmir" in lower_facts

 def test_core_facts_contains_1975(self):
 """System prompt'ta kuruluş yılı 1975 olmalı."""
 assert "1975" in SELCUK_CORE_FACTS

 def test_core_facts_contains_teknoloji_fakultesi(self):
 """Bilgisayar Mühendisliği için Teknoloji Fakültesi belirtilmeli."""
 lower_facts = SELCUK_CORE_FACTS.lower()
 assert "teknoloji fakültesi" in lower_facts or "technology faculty" in lower_facts

```

```

def test_core_facts_contains_alaeddin_keykubat(self):
 """Alaeddin Keykubat kampüsü belirtilmeli."""
 lower_facts = SELCUK_CORE_FACTS.lower()
 assert "alaeddin keykubat" in lower_facts

def test_core_facts_contains_mudek(self):
 """MÜDEK akreditasyonu belirtilmeli."""
 lower_facts = SELCUK_CORE_FACTS.lower()
 assert "mudek" in lower_facts or "mudek" in lower_facts

def test_turkish_system_prompt_includes_core_facts(self):
 """Türkçe system prompt core facts içermeli."""
 assert SELCUK_CORE_FACTS in SELCUK_UNIVERSITY_SYSTEM_PROMPT

def test_english_system_prompt_contains_konya(self):
 """Ingilizce system prompt'ta da Konya bilgisi olmalı."""
 assert "konya" in DEFAULT_SYSTEM_PROMPT_EN.lower()

def test_english_system_prompt_not_contains_izmir(self):
 """Ingilizce system prompt'ta İzmir bilgisi olmamalı."""
 lower_prompt = DEFAULT_SYSTEM_PROMPT_EN.lower()
 if "izmir" in lower_prompt:
 assert "not izmir" in lower_prompt

def test_english_system_prompt_contains_1975(self):
 """Ingilizce system prompt'ta kuruluş yılı 1975 olmalı."""
 assert "1975" in DEFAULT_SYSTEM_PROMPT_EN

def test_build_default_system_prompt_turkish(self):
 """Türkçe için oluşturulan prompt Konya içermeli."""
 prompt = build_default_system_prompt("tr")
 assert "konya" in prompt.lower()

def test_build_default_system_prompt_english(self):
 """Ingilizce için oluşturulan prompt Konya içermeli."""
 prompt = build_default_system_prompt("en")
 assert "konya" in prompt.lower()

def test_critical_keywords_present(self):
 """Tüm kritik anahtar kelimeler mevcut olmalı."""
 combined = SELCUK_UNIVERSITY_SYSTEM_PROMPT + DEFAULT_SYSTEM_PROMPT_EN
 combined_lower = combined.lower()

 critical_keywords = [
 "konya",
 "1975",
 "teknoloji fakültesi",
 "alaeddin keykubat",
 "mudek",
]
 for keyword in critical_keywords:
 assert keyword in combined_lower, f"'{keyword}' bulunamadı!"

def test_no_wrong_location_in_prompts(self):
 """Yanlış konum bilgileri olmamalı."""

```

```

combined = SELCUK_UNIVERSITY_SYSTEM_PROMPT +
DEFAULT_SYSTEM_PROMPT_EN
combined_lower = combined.lower()

Bu şehirler yanlış olurdu, disclaimer olmadan geçmemeli
wrong_cities = ["ankara", "istanbul", "bursa"]

for city in wrong_cities:
 if city in combined_lower:
 # Eğer geçiyorsa, bir disclaimer ile geçmeli
 pytest.fail(f"Yanlış şehir '{city}' prompt'ta bulundu!")

class TestSystemPromptQuality:
 """System prompt kalitesini test eder."""

 def test_prompt_not_empty(self):
 """System prompt boş olmamalı."""
 assert len(SELCUK_UNIVERSITY_SYSTEM_PROMPT) > 100
 assert len(DEFAULT_SYSTEM_PROMPT_EN) > 100

 def test_prompt_has_structure(self):
 """System prompt yapılandırılmış olmalı."""
 # Markdown başlıklar var mı?
 assert "##" in SELCUK_UNIVERSITY_SYSTEM_PROMPT or "#" in
SELCUK_UNIVERSITY_SYSTEM_PROMPT

 def test_prompt_mentions_assistant_role(self):
 """System prompt asistan rolünü belirtmeli."""
 lower_prompt = SELCUK_UNIVERSITY_SYSTEM_PROMPT.lower()
 assert "asistan" in lower_prompt or "assistant" in lower_prompt

 def test_prompt_mentions_university_name(self):
 """System prompt üniversite adını belirtmeli."""
 lower_prompt = SELCUK_UNIVERSITY_SYSTEM_PROMPT.lower()
 assert "selçuk üniversitesi" in lower_prompt or "selcuk university" in lower_prompt

```

```

if __name__ == "__main__":
 pytest.main([__file__, "-v"])

```

**E:/SelcukAiAssistant/repo/backend/test\_main.py**

```

"""FastAPI backend için birim testleri.

```

Giriş: TestClient üzerinden istekler.

Çıkış: HTTP durum kodu ve gövde doğrulaması.

İşleyiş: Ollama çalışmadan API kontratını test eder.

```

"""

```

```

from unittest.mock import AsyncMock, MagicMock, patch

```

```

import httpx

```

```

import pytest

```

```

from fastapi.testclient import TestClient

```

```

from config import Config

```

```

from main import app

```

```

from providers.base import ChatResult, Usage

```

```

from providers.ollama_provider import OllamaProvider

```

```

client = TestClient(app)

@patch("main.appwrite_client", None)
def test_root_endpoint():
 """Giriş: yok.

 Çıkış: Kök endpoint 200 ve doğru mesaj.
 İşleyiş: GET / çağrısını doğrular.
 """
 response = client.get("/")
 assert response.status_code == 200
 assert response.json() == {
 "status": "ok",
 "message": "Selçuk AI Asistanı arka uç çalışıyor",
 }
}

@pytest.mark.asyncio
@patch("main.appwrite_client", None)
@patch.object(OllamaProvider, "generate", new_callable=AsyncMock)
async def test_chat_endpoint_success(mock_generate):
 """Giriş: Geçerli chat isteği.

 Çıkış: 200 ve cevap metni.
 İşleyiş: Provider mock ile başarılı senaryo doğrulanır.
 """
 mock_generate.return_value = ChatResult(
 text="Merhaba! Ben Selçuk AI Asistanı.",
 usage=Usage(prompt_tokens=10, completion_tokens=5, total_tokens=15),
)

 response = client.post(
 "/chat",
 json={"messages": [{"role": "user", "content": "Merhaba"}]},
)

 assert response.status_code == 200
 data = response.json()
 assert data["answer"] == "Merhaba! Ben Selçuk AI Asistanı."
 assert data["request_id"]

@patch("main.appwrite_client", None)
@patch.object(OllamaProvider, "generate", new_callable=AsyncMock)
def test_chat_endpoint_connection_error(mock_generate):
 """Giriş: Servis hatası.

 Çıkış: 503 ve detay.
 İşleyiş: HTTPException simülasyonu yapar.
 """
 from fastapi import HTTPException

 mock_generate.side_effect = HTTPException(status_code=503, detail="Unavailable")

 response = client.post(
 "/chat",
)

```

```

 json={"messages": [{"role": "user", "content": "Test"}]},
)

assert response.status_code == 503
assert "detail" in response.json()

@pytest("main.appwrite_client", None)
def test_chat_endpoint_invalid_request():
 """Giriş: Eksik payload.

 Çıkış: 422.
 İşleyiş: Doğrulama hatasını doğrular.
 """
 response = client.post("/chat", json={})
 assert response.status_code == 422

@pytest("main.appwrite_client", None)
@pytest.object(OllamaProvider, "generate", new_callable=AsyncMock)
def test_prompt_contains_question(mock_generate):
 """Giriş: Soru metni.

 Çıkış: Prompt içinde soru.
 İşleyiş: Çağrı argümanında soru metni aranır.
 """
 mock_generate.return_value = ChatResult(text="Test response")

 question = "Selçuk Üniversitesi nerede?"
 response = client.post(
 "/chat",
 json={"messages": [{"role": "user", "content": question}]},
)

 assert response.status_code == 200
 messages = mock_generate.call_args.kwargs.get("messages", [])
 assert any(question in msg.get("content", "") for msg in messages)

@pytest("main.appwrite_client", None)
@pytest("httpx.AsyncClient.get")
def test_ollama_health_check_healthy(mock_get):
 """Giriş: Sağlıklı mock yanıt.

 Çıkış: healthy ve model var.
 İşleyiş: GET /health/ollama sonucunu doğrular.
 """
 mock_response = MagicMock()
 mock_response.status_code = 200
 mock_response.json.return_value = {
 "models": [{"name": Config.OLLAMA_MODEL}, {"name": "mistral"}]
 }
 mock_get.return_value = mock_response

 response = client.get("/health/ollama")

 assert response.status_code == 200
 data = response.json()

```

```
assert data["status"] == "healthy"
assert data["model_available"] is True
assert Config.OLLAMA_MODEL in data["available_models"]
```

```
@patch("main.appwrite_client", None)
@patch("httpx.AsyncClient.get")
def test_ollama_health_check_unhealthy(mock_get):
 """Giriş: Bağlantı hatası.

 Çıkış: 503.
 İşleyiş: RequestError simülasyonu yapar.
 """
 mock_get.side_effect = httpx.RequestError("Connection refused")

 response = client.get("/health/ollama")

 assert response.status_code == 503
 assert "detail" in response.json()
```

```
@patch("main.appwrite_client", None)
@patch.object(OllamaProvider, "generate", new_callable=AsyncMock)
def test_chat_endpoint_timeout(mock_generate):
 """Giriş: TimeoutError.

 Çıkış: 504.
 İşleyiş: Zaman aşımı senaryosunu doğrular.
 """
 mock_generate.side_effect = TimeoutError()

 response = client.post(
 "/chat",
 json={"messages": [{"role": "user", "content": "Test"}]},
)

 assert response.status_code == 504
 data = response.json()
 assert "detail" in data
```

```
if __name__ == "__main__":
 pytest.main([__file__, "-v"])
```

## E:/SelcukAiAssistant/repo/backend/test\_extended.py

```
"""FastAPI backend için genişletilmiş birim testleri.
```

```
Giriş: TestClient ve mock nesneler.
Çıkış: HTTP yanıtları ve fonksiyon sonuçları doğrulanır.
İşleyiş: Doğrulama, retry ve RAG akışları test edilir.
"""

from unittest.mock import AsyncMock, MagicMock, patch
```

```
import httpx
import numpy as np
import pytest
from fastapi.testclient import TestClient
```

```

from config import Config
from main import app
from ollama_service import OllamaService
from providers.base import ChatResult
from providers.ollama_provider import OllamaProvider
from rag_service import Document, RAGService, RagIndex, chunk_text

client = TestClient(app)

def _chat_payload(content: str) -> dict[str, list[dict[str, str]]]:
 """Giriş: Kullanıcı mesajı.

 Çıkış: Chat payload sözlüğü.
 İşleyiş: /chat şemasına uygun JSON üretir.
 """
 return {"messages": [{"role": "user", "content": content}]}

#
=====

Input Validation Tests
#
=====

def test_chat_empty_question():
 """Giriş: Boş mesaj.

 Çıkış: 422.
 İşleyiş: İçerik doğrulamasını kontrol eder.
 """
 response = client.post("/chat", json=_chat_payload(""))
 assert response.status_code == 422

def test_chat_whitespace_only_question():
 """Giriş: Sadece boşluk.

 Çıkış: 422.
 İşleyiş: İçerik doğrulamasını kontrol eder.
 """
 response = client.post("/chat", json=_chat_payload(" "))
 assert response.status_code == 422

def test_chat_too_long_question():
 """Giriş: Aşırı uzun içerik.

 Çıkış: 422.
 İşleyiş: Uzunluk kontrolünü doğrular.
 """
 long_question = "a" * 11000
 response = client.post("/chat", json=_chat_payload(long_question))
 assert response.status_code == 422

```

```

def test_chat_xss_prevention():
 """Giriş: Zararlı içerik.

 Çıkış: 422.
 İşleyiş: XSS filtre kontrolünü doğrular.
 """
 dangerous_inputs = [
 "<script>alert('xss')</script>",
 "test javascript:alert(1)",
 "test onerror=alert(1)",
 "test onload=alert(1)",
]
 for dangerous_input in dangerous_inputs:
 response = client.post("/chat", json=_chat_payload(dangerous_input))
 assert response.status_code == 422

```

```

@patch.object(OllamaProvider, "generate", new_callable=AsyncMock)
def test_chat_valid_input_with_sanitization(mock_generate):
 """Giriş: Geçerli içerik.

```

```

 Çıkış: 200.
 İşleyiş: Sanitize sonrası başarılı yanıt bekler.
 """
 mock_generate.return_value = ChatResult(text="Test response")
 response = client.post("/chat", json=_chat_payload(" Test question "))
 assert response.status_code == 200

```

```

#
=====
Health Check Model Matching Tests
#
=====

#
```

```

@ pytest.mark.asyncio
@ patch("ollama_service.httpx.AsyncClient.get")
async def test_health_check_exact_match(mock_get):
 """Giriş: Tam model adı.

```

```

 Çıkış: healthy.
 İşleyiş: /api/tags eşleşmesini doğrular.
 """
 mock_response = MagicMock()
 mock_response.status_code = 200
 mock_response.json.return_value = {
 "models": [{"name": Config.OLLAMA_MODEL}, {"name": "mistral"}]
 }
 mock_get.return_value = mock_response

 service = OllamaService()
 health = await service.health_check()
 assert health["status"] == "healthy"

```

```
assert health["model_available"] is True
```

```
@pytest.mark.asyncio
@patch("ollama_service.hpx.AsyncClient.get")
async def test_health_check_with_latest_tag(mock_get):
 """Giriş: latest tag.

 Çıkış: healthy.
 İşleyiş: Tag eşleşmesini doğrular.
 """
 mock_response = MagicMock()
 mock_response.status_code = 200
 mock_response.json.return_value = {
 "models": [{"name": f'{Config.OLLAMA_MODEL}:latest"}, {"name": "mistral:latest"}]
 }
 mock_get.return_value = mock_response

 service = OllamaService()
 health = await service.health_check()
 assert health["status"] == "healthy"
 assert health["model_available"] is True
```

```
@pytest.mark.asyncio
@patch("ollama_service.hpx.AsyncClient.get")
async def test_health_check_reverse_tag_match(mock_get):
 """Giriş: Base model.

 Çıkış: available.
 İşleyiş: Ters tag eşleşmesini doğrular.
 """
 mock_response = MagicMock()
 mock_response.status_code = 200
 mock_response.json.return_value = {
 "models": [{"name": "selcuk_ai_assistant"}, {"name": "llama3.1"}]
 }
 mock_get.return_value = mock_response

 service = OllamaService()
 health = await service.health_check(model="selcuk_ai_assistant:latest")
 assert health["model_available"] is True
```

```
def test_is_model_available_helper():
 """Giriş: Model listeleri.
```

```
 Çıkış: bool.
 İşleyiş: Yardımcı fonksiyonun eşleşmesini doğrular.
 """
 assert OllamaService._is_model_available("llama3.1", ["llama3.1", "mistral"])
 assert OllamaService._is_model_available("llama3.1", ["llama3.1:latest"])
 assert OllamaService._is_model_available("llama3.1:latest", ["llama3.1"])
 assert not OllamaService._is_model_available("gpt4", ["llama3.1", "mistral"])
 assert not OllamaService._is_model_available("llama3.1", [])
 assert not OllamaService._is_model_available("", ["llama3.1"])
```

```

#
=====
= # Retry Logic Tests
#
=====
=

@pytest.mark.asyncio
@patch("ollama_service.httpx.AsyncClient.post")
@patch("ollama_service.asyncio.sleep")
async def test_retry_on_connection_error(_mock_sleep, mock_post):
 """Giriş: Bağlantı hatası.

 Çıkış: Retry sonrası yanıt.
 İşleyiş: Tekrar deneme akışını doğrular.
 """
 mock_post.side_effect = [
 httpx.RequestError("Connection refused"),
 httpx.RequestError("Connection refused"),
 MagicMock(
 status_code=200,
 json=lambda: {"message": {"content": "Merhaba! This is a response."}},
),
]

 service = OllamaService(max_retries=3)
 result = await service.generate(
 messages=[{"role": "user", "content": "test"}],
 model=Config.OLLAMA_MODEL,
 temperature=0.2,
 top_p=0.9,
 max_tokens=32,
)

 assert mock_post.call_count == 3
 assert "Merhaba" in result["text"]
 assert _mock_sleep.call_count == 2

@pytest.mark.asyncio
@patch("ollama_service.httpx.AsyncClient.post")
@patch("ollama_service.asyncio.sleep")
async def test_retry_on_timeout(_mock_sleep, mock_post):
 """Giriş: Timeout.

 Çıkış: Retry sonrası yanıt.
 İşleyiş: Zaman aşımı tekrarlarını doğrular.
 """
 mock_post.side_effect = [
 httpx.ReadTimeout("Timeout"),
 MagicMock(
 status_code=200,
 json=lambda: {"message": {"content": "Success after retry"}},
),
]

 service = OllamaService(max_retries=2)

```

```
result = await service.generate(
 messages=[{"role": "user", "content": "test"}],
 model=Config.OLLAMA_MODEL,
 temperature=0.2,
 top_p=0.9,
 max_tokens=32,
)
```

```
assert mock_post.call_count == 2
assert result["text"] == "Success after retry"
```

```
@pytest.mark.asyncio
@patch("ollama_service.httpx.AsyncClient.post")
@patch("ollama_service.asyncio.sleep")
async def test_retry_exhaustion(_mock_sleep, mock_post):
 """Giriş: Sürekli hata.
```

Çıkış: HTTPException 503.  
İşleyiş: Retry tükenmesi senaryosunu doğrular.  
"""

```
from fastapi import HTTPException

mock_post.side_effect = httpx.RequestError("Connection refused")
```

```
service = OllamaService(max_retries=3)
with pytest.raises(HTTPException) as exc_info:
 await service.generate(
 messages=[{"role": "user", "content": "test"}],
 model=Config.OLLAMA_MODEL,
 temperature=0.2,
 top_p=0.9,
 max_tokens=32,
)
```

```
assert mock_post.call_count == 3
assert exc_info.value.status_code == 503
```

```
@pytest.mark.asyncio
@patch("ollama_service.httpx.AsyncClient.post")
async def test_no_retry_on_http_error(mock_post):
 """Giriş: HTTP 404.
```

Çıkış: Tek deneme.  
İşleyiş: HTTP hatasında retry yapılmadığını doğrular.  
"""

```
from fastapi import HTTPException
```

```
mock_response = MagicMock()
mock_response.status_code = 404
mock_response.json.return_value = {"error": "Model not found"}
mock_response.text = "Model not found"
mock_post.return_value = mock_response
```

```
service = OllamaService(max_retries=3)
with pytest.raises(HTTPException):
 await service.generate()
```

```
 messages=[{"role": "user", "content": "test"}],
 model=Config.OLLAMA_MODEL,
 temperature=0.2,
 top_p=0.9,
 max_tokens=32,
)
```

```
assert mock_post.call_count == 1
```

```
#
```

```
=====
RAG Service Tests
#
=====
```

```
=
```

```
def test_rag_service_initialization_disabled():
 """Giriş: RAG disabled.
```

Çıkış: Boş bağlam.

İşleyiş: Devre dışı kontrolünü doğrular.

```
"""
```

```
 service = RAGService(enabled=False)
 assert service.enabled is False
 context, citations = service.get_context("test query")
 assert context == ""
 assert citations == []
```

```
class DummyEmbeddingBackend:
```

"""Giriş: Test amaçlı gömme üretici.

Çıkış: Sabit boyutlu vektörler.

İşleyiş: Metin uzunluğu tabanlı vektör üretir.

```
"""
```

```
def __init__(self) -> None:
```

"""Giriş: yok.

Çıkış: Nesne.

İşleyiş: Boyutu sabitler.

```
"""
```

```
 self._dimension = 3
```

```
@property
```

```
def dimension(self) -> int:
```

"""Giriş: yok.

Çıkış: Boyut.

İşleyiş: Sabit değer döndürür.

```
"""
```

```
 return self._dimension
```

```
def embed(self, texts: list[str]) -> np.ndarray:
```

"""Giriş: Metin listesi.

```

Çıkış: Normalize vektörler.
İşleyiş: Uzunluk tabanlı vektör üretir.
"""
vectors = []
for text in texts:
 vectors.append(
 [float(len(text)), float(text.count("a")), float(text.count("b"))]
)
array = np.array(vectors, dtype="float32")
if array.size == 0:
 return array
norms = np.linalg.norm(array, axis=1, keepdims=True)
norms[norms == 0] = 1.0
return array / norms

def test_rag_service_initialization_enabled(tmp_path):
 """Giriş: Geçerli ayarlar.

 Çıkış: Aktif RAG.
 İşleyiş: İndeks kurulumunu doğrular.
 """
 service = RAGService(
 enabled=True,
 vector_db_path=str(tmp_path),
 collection_name="test_collection",
 chunk_size=300,
 chunk_overlap=30,
 embedder=DummyEmbeddingBackend(),
)
 assert service.enabled is True
 assert service.vector_db_path == str(tmp_path)
 assert service.collection_name == "test_collection"
 assert service.chunk_size == 300
 assert service.chunk_overlap == 30

def test_rag_service_search_when_disabled():
 """Giriş: RAG disabled.

 Çıkış: Boş sonuç.
 İşleyiş: Arama bypass kontrolü.
 """
 service = RAGService(enabled=False)
 results = service.search("test query")
 assert results == []

def test_rag_service_ingest_when_disabled():
 """Giriş: RAG disabled.

 Çıkış: RuntimeError.
 İşleyiş: Korumalı ekleme doğrulanır.
 """
 service = RAGService(enabled=False)
 with pytest.raises(RuntimeError, match="RAG servisi etkin değil"):
 service.add_documents([Document(content="test content")])

```

```

def test_document_creation():
 """Giriş: Document.

 Çıkış: Alan doğrulama.
 İşleyiş: Dataclass davranışını kontrol eder.
 """
 doc = Document(
 content="Test content",
 metadata={"source": "test.txt", "date": "2024-01-01"},
 doc_id="doc_123",
)
 assert doc.content == "Test content"
 assert doc.metadata["source"] == "test.txt"
 assert doc.doc_id == "doc_123"
 assert "Test content" in repr(doc)

def test_rag_chunk_document():
 """Giriş: Uzun metin.

 Çıkış: Parçalara bölme.
 İşleyiş: chunk_text fonksiyonunu doğrular.
 """
 content = "0123456789abcdefgij"
 chunks = chunk_text(content, chunk_size=10, chunk_overlap=3)
 assert all(len(chunk) <= 10 for chunk in chunks)
 assert len(chunks) > 1

def test_rag_index_add_and_search(tmp_path):
 """Giriş: İki doc.

 Çıkış: Arama sonucu.
 İşleyiş: İndeks ekleme/arama akışını doğrular.
 """
 embedder = DummyEmbeddingBackend()
 index = RagIndex(tmp_path, embedder)
 docs = [
 Document(content="aaa bbb", metadata={"source": "a.txt"}),
 Document(content="cccc dddd", metadata={"source": "b.txt"}),
]
 added = index.add_documents(docs)
 assert added == 2

 results = index.search("aaa", top_k=1)
 assert len(results) == 1
 assert results[0].metadata.get("source") == "a.txt"

if __name__ == "__main__":
 pytest.main([__file__, "-v"])

```

**E:/SelcukAiAssistant/repo/backend/test\_response\_cleaner.py**

"""Sunucu tarafı yanıt temizleme için birim testleri.

Giriş: Örnek metinler.

```
Çıkış: Temizlenmiş metinler.
İşleyiş: Meta içerik ve think blokları ayıklanır.
"""
from response_cleaner import StreamingResponseCleaner, clean_text
```

```
def test_clean_text_strips_meta_prefix():
 """Giriş: Meta prefix.

 Çıkış: Temiz metin.
 İşleyiş: Prefix içeriği silinir.
 """
 text = (
 "Okay, I need to help the user by thinking.\n"
 "Merhaba! Selçuk Üniversitesi 1975 yılında kuruldu."
)
 cleaned = clean_text(text, language="tr")
 assert "Okay, I need to help the user" not in cleaned
 assert "Merhaba" in cleaned
```

```
def test_clean_text_removes_think_tags():
 """Giriş: <think> tag'leri.

 Çıkış: Tagsız metin.
 İşleyiş: Tag temizleme yapılır.
 """
 text = "<think>internal plan</think>Final answer."
 cleaned = clean_text(text, language="en")
 assert "<think>" not in cleaned
 assert "Final answer" in cleaned
```

```
def test_clean_text_preserves_code_blocks():
 """Giriş: Kod bloğu.

 Çıkış: Blok korunur.
 İşleyiş: Fence ayrimı ile kod blokları korunur.
 """
 text = (
 "Okay, I need to help the user.\n\n"
 "```python\nprint('hi')\n```
 "Answer: done."
)
 cleaned = clean_text(text, language="en")
 assert "Okay, I need to help the user" not in cleaned
 assert "```python" in cleaned
 assert "print('hi')" in cleaned
 assert "Answer: done." in cleaned
```

```
def test_streaming_cleaner_strips_meta_sentences():
 """Giriş: Meta cümleleri.

 Çıkış: Gerçek cevap.
 İşleyiş: Akış filtresi meta cümlelerini ayıklar.
 """
 cleaner = StreamingResponseCleaner(language="en")
```

```

output = ""
output += cleaner.feed("Okay, I need to help the user by thinking. ")
output += cleaner.feed("Here is the answer.\n")
output += cleaner.finalize()
assert "Okay, I need to help the user" not in output
assert "Here is the answer" in output

```

### E:/SelcukAiAssistant/repo/backend/test\_reasoning\_cleanup.py

"""Düşünce izi temizleme için kapsamlı testler.

Giriş: Örnek cevap metinleri.

Çıkış: Temizlenmiş metin.

İşleyiş: Think tag, meta cümle ve işaret temizliği doğrulanır.

"""

```
from ollama_service import OllamaService
```

class TestReasoningCleanup:

"""Giriş: Test senaryoları.

Çıkış: Beklenen temiz metinler.

İşleyiş: Temizleme kurallarını farklı örneklerle doğrular.

"""

```
def test_simple_clean_response(self):
```

"""Giriş: Temiz metin.

Çıkış: Metin korunur.

İşleyiş: Değişiklik yapılmadığını doğrular.

"""

```
clean_text = "Merhaba! Selçuk Üniversitesi 1975 yılında kurulmuştur."
```

```
result = OllamaService._clean_reasoning_artifacts(clean_text)
```

```
assert "Merhaba" in result
```

```
assert "1975" in result
```

```
assert len(result) > 15
```

```
def test_remove_think_tags(self):
```

"""Giriş: Think tag'leri.

Çıkış: Tagsız metin.

İşleyiş: Think içerikleri temizlenir.

"""

```
text = (
```

```
 "<think>Hmm, the user wants to know about enrollment.</think>"
```

```
 "Merhaba! Kayıt tarihleri Eylül ayındadır."
```

```
)
```

```
result = OllamaService._clean_reasoning_artifacts(text)
```

```
assert "<think>" not in result
```

```
assert "</think>" not in result
```

```
assert "Merhaba" in result
```

```
assert "Kayıt" in result
```

```
def test_remove_im_tags(self):
```

"""Giriş: Instruction tag'ları.

Çıkış: Temiz metin.

İşleyiş: Özel tokenlar ayıklanır.

"""

```
text = "<|im_start|>assistant\nMerhaba! Size yardımcı olabilirim.<|im_end|>"
```

```

result = OllamaService._clean_reasoning_artifacts(text)
assert "<|im_start|>" not in result
assert "<|im_end|>" not in result
assert "Merhaba" in result

def test_remove_english_reasoning(self):
 """Giriş: İngilizce meta cümle.

 Çıkış: Temiz metin.
 İşleyiş: Meta cümleler silinir.
 """
 text = (
 "Okay, let me think about this. The user wants information. "
 "Merhaba! İşte cevabınız."
)
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "okay" not in result.lower()
 assert "let me" not in result.lower()
 assert "Merhaba" in result

def test_remove_turkish_reasoning(self):
 """Giriş: Türkçe meta cümle.

 Çıkış: Cevap korunur.
 İşleyiş: Meta cümleler silinir.
 """
 text = (
 "Tamam, kullanıcı kayıt tarihleri soruyor. Düşünüyorum ki cevap verebilirim.\n\n"
 "Merhaba! Kayıt Eylül'de."
)
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "Merhaba" in result
 assert "Kayıt" in result

def test_markdown_header_extraction(self):
 """Giriş: Markdown başlığı.

 Çıkış: Başlık korunur.
 İşleyiş: Header seçimi yapılır.
 """
 text = (
 "Let me format this nicely.\n\n## Kayıt Tarihleri\n\n"
 "Kayıt işlemleri Eylül ayında yapılır.\n\n## İletişim\n\n"
 "Bize ulaşabilirsiniz."
)
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "##" in result
 assert "Kayıt" in result
 assert "let me" not in result.lower()

def test_multiple_merhaba_takes_last(self):
 """Giriş: Birden çok selamlama.

 Çıkış: Son selamlama.
 İşleyiş: Son indeks korunur.
 """
 text = "Merhaba test. Okay, let me rephrase. Merhaba! İşte gerçek cevap."
 result = OllamaService._clean_reasoning_artifacts(text)
 assert result.startswith("Merhaba!")

```

```

assert "gerçek cevap" in result
assert "test" not in result

def test_paragraph_filtering(self):
 """Giriş: Meta paragraf.

 Çıkış: Meta silinir.
 İşleyiş: Paragraffiltresi uygulanır.
 """
 text = "İlk paragraf normal.\n\nOkay, kullanıcı soruyor.\n\nİkinci normal paragraf."
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "İlk paragraf" in result
 assert "İkinci normal paragraf" in result

def test_whitespace_normalization(self):
 """Giriş: Fazla boşluk.

 Çıkış: Normalize metin.
 İşleyiş: Boşluk temizleme uygulanır.
 """
 text = "Merhaba!\n\n\nBu bir test.\n\n\nSon satır."
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "\n\n\n" not in result
 assert "Merhaba" in result
 assert "test" in result

def test_fallback_for_too_short_text(self):
 """Giriş: Çok kısa metin.

 Çıkış: Fallback mesajı.
 İşleyiş: Minimum uzunluk kontrolü yapılır.
 """
 text = "<think>Just thinking</think>"
 result = OllamaService._clean_reasoning_artifacts(text)
 assert len(result) >= 15
 assert "Selçuk AI" in result or "yardımcı" in result

def test_complex_mixed_reasoning(self):
 """Giriş: Karışık düşünce izi.

 Çıkış: Temiz yanıt.
 İşleyiş: Kapsamlı temizleme uygulanır.
 """
 text = """"<think>
Okay, the user is asking about Selçuk University enrollment dates.
Tamam, kullanıcı kayıt tarihleri soruyor.
I should provide clear information.
Düşünüyorum ki açık bir cevap vermemeliyim.
</think>

Merhaba! İşte kayıt bilgileri:

Kayıt Tarihleri

Selçuk Üniversitesi kayıt işlemleri genellikle **Eylül ayı** içinde gerçekleşir.

Gerekli Belgeler:
- Kimlik fotokopisi
"""

```

- Diploma
- Fotoğraflar

Detaylı bilgi için öğrenci işlerine başvurabilirsiniz.""""

```
result = OllamaService._clean_reasoning_artifacts(text)

assert "<think>" not in result
assert "okay" not in result.lower()
assert "tamam" not in result.lower()
assert "i should" not in result.lower()
assert "düşünüyorum" not in result.lower()

assert "Merhaba" in result
assert "Kayıt" in result
assert "Eylül" in result
assert "Kimlik" in result

def test_preserve_legitimate_words(self):
 """Giriş: Geçerli kelimeler.

 Çıkış: Korunmuş yanıt.
 İşleyiş: Yanlış pozitifleri engeller.
 """
 text = (
 "Merhaba! Kullanıcı kayıt sisteme giriş yapabilir."
 "Tamam butonu ile onaylayın."
)
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "Merhaba" in result

def test_empty_input(self):
 """Giriş: Boş metin.

 Çıkış: Fallback mesajı.
 İşleyiş: Boşluk kontrolü yapılır.
 """
 result = OllamaService._clean_reasoning_artifacts("")
 assert len(result) >= 15

def test_only_whitespace(self):
 """Giriş: Sadece boşluk.

 Çıkış: Fallback mesajı.
 İşleyiş: Boşluk kontrolü yapılır.
 """
 result = OllamaService._clean_reasoning_artifacts(" \n\n\t ")
 assert len(result) >= 15

def test_special_characters_preserved(self):
 """Giriş: Özel karakterler.

 Çıkış: Korunmuş metin.
 İşleyiş: Özel karakterlerin silinmediğini doğrular.
 """
 text = "Merhaba! E-posta: info@selcuk.edu.tr, Tel: +90 332 123 45 67"
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "@" in result
```

```

assert "+" in result
assert "Merhaba" in result

def test_numbered_lists_preserved(self):
 """Giriş: Numaralı liste.

 Çıkış: Liste korunur.
 İşleyiş: Markdown liste düzenini korur.
 """
 text = """Merhaba! İşte adımlar:

1. Kayıt formunu doldurun
2. Belgelerinizi hazırlayın
3. Fakülteye başvurun"""
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "1." in result
 assert "2." in result
 assert "3." in result

```

```

def test_code_blocks_preserved(self):
 """Giriş: Kod bloğu.

 Çıkış: Blok korunur.
 İşleyiş: Kod blokları silinmez.
 """
 text = "Merhaba! İşte örnek:\n\n```nÖrnek kod\n```\nBu bir örnektir."
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "```" in result
 assert "Örnek kod" in result

```

```

def test_urls_preserved(self):
 """Giriş: URL.

 Çıkış: URL korunur.
 İşleyiş: URL metninin bozulmadığını doğrular.
 """
 text = "Merhaba! Web sitesi: https://www.selcuk.edu.tr adresindedir."
 result = OllamaService._clean_reasoning_artifacts(text)
 assert "https://" in result
 assert "selcuk.edu.tr" in result

```

## E:/SelcukAiAssistant/repo/backend/providers/ollama\_provider.py

```

"""Ollama sağlayıcı uyarlaması."""
from __future__ import annotations

import logging
from typing import AsyncIterator

from config import Config
from ollama_service import OllamaService
from providers.base import CancellationToken, ChatResult, ModelProvider, StreamChunk, Usage
from utils import ReasoningFilter

logger = logging.getLogger(__name__)

class OllamaProvider(ModelProvider):
 """Giriş: Ollama servis ayarları.

```

```

Çıkış: Model sağlayıcı nesnesi.
İşleyiş: OllamaService ile üretim ve akış sağlar.
"""

name = "ollama"

def __init__(self) -> None:
 """Giriş: yok.

 Çıkış: Nesne.
 İşleyiş: Yapılandırma değerleriyle Ollama istemcisini oluşturur.
 """
 self._client = OllamaService(
 base_url=Config.OLLAMA_BASE_URL,
 timeout=Config.OLLAMA_TIMEOUT,
 max_retries=Config.OLLAMA_MAX_RETRIES,
 retry_delay=Config.OLLAMA_RETRY_DELAY,
)

async def generate(
 self,
 messages: list[dict[str, str]],
 model_id: str,
 temperature: float,
 top_p: float,
 max_tokens: int,
 request_id: str,
) -> ChatResult:
 """Giriş: Mesajlar ve üretim parametreleri.

 Çıkış: ChatResult.
 İşleyiş: Ollama API üzerinden tek seferlik yanıt üretir.
 """
 logger.info(
 "request_id=%s provider=ollama generate model=%s", request_id, model_id
)
 result = await self._client.generate(
 messages=messages,
 model=model_id,
 temperature=temperature,
 top_p=top_p,
 max_tokens=max_tokens,
)
 usage_raw = result.get("usage") or {}
 usage = Usage(
 prompt_tokens=usage_raw.get("prompt_tokens"),
 completion_tokens=usage_raw.get("completion_tokens"),
)
 if usage.prompt_tokens is not None and usage.completion_tokens is not None:
 usage.total_tokens = usage.prompt_tokens + usage.completion_tokens
 return ChatResult(text=result["text"], usage=usage)

async def stream(
 self,
 messages: list[dict[str, str]],
 model_id: str,
 temperature: float,
 top_p: float,

```

```

 max_tokens: int,
 request_id: str,
 cancel_token: CancellationToken,
) -> AsyncIterator[StreamChunk]:
 """Giriş: Mesajlar ve üretim parametreleri.

 Çıkış: StreamChunk akışı.
 İşleyiş: Ollama stream akışını iletir ve iptal sinyalini izler.
 """
 logger.info(
 "request_id=%s provider=ollama stream model=%s", request_id, model_id
)
 reasoning_filter = ReasoningFilter()
 async for chunk in self._client.generate_stream(
 messages=messages,
 model=model_id,
 temperature=temperature,
 top_p=top_p,
 max_tokens=max_tokens,
):
 if cancel_token.is_cancelled():
 logger.info("request_id=%s stream cancelled", request_id)
 break

 token = chunk.get("token") or ""
 if token:
 token = reasoning_filter.feed(token)
 if token:
 yield StreamChunk(token=token)

 if chunk.get("done"):
 usage_raw = chunk.get("usage") or {}
 usage = Usage(
 prompt_tokens=usage_raw.get("prompt_tokens"),
 completion_tokens=usage_raw.get("completion_tokens"),
)
 if usage.prompt_tokens is not None and usage.completion_tokens is not None:
 usage.total_tokens = usage.prompt_tokens + usage.completion_tokens
 yield StreamChunk(token="", done=True, usage=usage)
 break

 def list_models(self) -> list:
 """Giriş: yok.

 Çıkış: Boş liste.
 İşleyiş: Ollama katalogları burada kullanılmaz.
 """
 return []

 async def list_model_names(self) -> list[str]:
 """Giriş: yok.

 Çıkış: Ollama model adları.
 İşleyiş: OllamaService üzerinden model listesini alır.
 """
 return await self._client.list_model_names()

 async def health_check(self, model_id: str) -> dict[str, object]:
 """Giriş: Model kimliği.

```

Çıkış: Sağlık bilgisi.  
 İşleyiş: OllamaService sağlık kontrolünü döndürür.  
 """"  
 return await self.\_client.health\_check(model\_id)

**E:/SelcukAiAssistant/repo/backend/providers/huggingface\_provider.py**

```
"""HuggingFace Transformers sağlayıcısı (opsiyonel 4-bit yükleme)."""
from __future__ import annotations

import logging
from typing import Any, AsyncIterator, Optional, cast

from config import Config
from providers.base import CancellationToken, ChatResult, ModelProvider, StreamChunk, Usage
from utils import ReasoningFilter

logger = logging.getLogger(__name__)

class HuggingFaceProvider(ModelProvider):
 """Giriş: HuggingFace yapılandırması.

 Çıkış: Sağlayıcı nesnesi.
 İşleyiş: Modeli yükler ve üretim işlemlerini yürütür.
 """
 name = "huggingface"

 def __init__(self) -> None:
 """Giriş: yok.

 Çıkış: Nesne.
 İşleyiş: Model önbelleğini başlatır.
 """
 self._cache: dict[str, tuple[Any, Any, str]] = {}

 def _ensure_dependencies(self) -> None:
 """Giriş: yok.

 Çıkış: yok.
 İşleyiş: Gerekli HuggingFace bağımlılıklarını kontrol eder.
 """
 try:
 import torch # noqa: F401
 import transformers # noqa: F401
 except ImportError as exc:
 raise RuntimeError(
 "HuggingFace bağımlılıkları eksik. "
 "backend/requirements-hf.txt kurun."
) from exc

 @staticmethod
 def _pick_device(preferred: str) -> str:
 """Giriş: Tercih edilen cihaz.

 Çıkış: Cihaz adı.
 İşleyiş: CUDA/MPS/CPU seçimini yapar.
```

```

"""
import torch

if preferred == "auto":
 if torch.cuda.is_available():
 return "cuda"
 if getattr(torch.backends, "mps", None) and torch.backends.mps.is_available():
 return "mps"
 return "cpu"
if preferred == "cuda":
 if torch.cuda.is_available():
 return "cuda"
 logger.warning("HF_DEVICE=cuda ancak CUDA yok; cpu kullanılıyor.")
 return "cpu"
if preferred == "mps":
 if getattr(torch.backends, "mps", None) and torch.backends.mps.is_available():
 return "mps"
 logger.warning("HF_DEVICE=mps ancak MPS yok; cpu kullanıyor.")
 return "cpu"
return preferred

@staticmethod
def _pick_dtype(name: str, device: str):
 """Giriş: dtype adı ve cihaz.

 Çıkış: Torch dtype.
 İşleyiş: Dtype tercihini belirler.
 """
 import torch

 if name == "float16":
 return torch.float16
 if name == "bfloating16":
 return torch.bfloat16
 if name == "float32":
 return torch.float32
 if device == "cuda":
 return torch.bfloat16 if torch.cuda.is_bf16_supported() else torch.float16
 return torch.float32

def _load_model(self, model_id: str):
 """Giriş: Model kimliği.

 Çıkış: (model, tokenizer, device) üçlüsü.
 İşleyiş: Modeli indirir/yükler ve önbelleğe alır.
 """
 self._ensure_dependencies()
 import torch
 from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

 device = self._pick_device(Config.HF_DEVICE)
 dtype = self._pick_dtype(Config.HF_DTYPE, device)

 quant_config = None
 if Config.HF_LOAD_IN_4BIT and device == "cuda":
 quant_config = BitsAndBytesConfig(
 load_in_4bit=True,
 bnb_4bit_use_double_quant=True,
 bnb_4bit_quant_type="nf4",

```

```

 bnb_4bit_compute_dtype=dtype,
)
elif Config.HF_LOAD_IN_4BIT:
 logger.warning("HF_LOAD_IN_4BIT CUDA dışında yok sayıldı.")

tokenizer = AutoTokenizer.from_pretrained(model_id, use_fast=True)
if tokenizer.pad_token is None and tokenizer.eos_token is not None:
 tokenizer.pad_token = tokenizer.eos_token

kwargs: dict[str, Any] = {}
if quant_config is not None:
 kwargs["quantization_config"] = quant_config
if device == "cuda":
 kwargs["device_map"] = "auto"
 kwargs["max_memory"] = {0: "5GiB"} # 6GB GPU için güvenli limit
if Config.HF_ATTENTION_IMPL:
 kwargs["attnImplementation"] = Config.HF_ATTENTION_IMPL

try:
 model = AutoModelForCausalLM.from_pretrained(model_id, dtype=dtype, **kwargs)
except TypeError:
 try:
 model = AutoModelForCausalLM.from_pretrained(
 model_id, torch_dtype=dtype, **kwargs
)
 except TypeError:
 kwargs.pop("attnImplementation", None)
 try:
 model = AutoModelForCausalLM.from_pretrained(
 model_id, dtype=dtype, **kwargs
)
 except TypeError:
 model = AutoModelForCausalLM.from_pretrained(
 model_id, torch_dtype=dtype, **kwargs
)
model = cast(Any, model)
if device != "cuda":
 model.to(torch.device(device))

model.eval()
self._cache[model_id] = (model, tokenizer, device)
return model, tokenizer, device

def _get_model(self, model_id: str) -> tuple[Any, Any, str]:
 """Giriş: Model kimliği.

 Çıkış: (model, tokenizer, device) üçlüsü.
 İşleyiş: Önbellekten alır veya yükler.
 """
 if model_id in self._cache:
 return self._cache[model_id]
 return self._load_model(model_id)

@staticmethod
def _fallback_prompt(messages: list[dict[str, str]]) -> str:
 """Giriş: Mesaj listesi.

 Çıkış: Tek string prompt.

```

İşleyiş: role:content formatında prompt üretir.

"""

```
lines = [f" {m['role']}: {m['content']} " for m in messages]
lines.append("assistant:")
return "\n".join(lines)
```

```
def _tokenize(
 self,
 tokenizer: Any,
 messages: list[dict[str, str]],
) -> tuple[Any, Optional[Any]]:
 """Giriş: Tokenizer ve mesajlar.
```

Çıkış: input\_ids ve attention\_mask.

İşleyiş: Chat template varsa onu, yoksa fallback promptu kullanır.

"""

```
if hasattr(tokenizer, "apply_chat_template"):
 try:
 prompt = tokenizer.apply_chat_template(
 messages,
 tokenize=False,
 add_generation_prompt=True,
)
 if isinstance(prompt, str) and prompt.strip():
 encoded = tokenizer(prompt, return_tensors="pt")
 return encoded["input_ids"], encoded.get("attention_mask")
 except TypeError:
 pass
```

```
prompt = self._fallback_prompt(messages)
```

```
encoded = tokenizer(prompt, return_tensors="pt")
```

```
return encoded["input_ids"], encoded.get("attention_mask")
```

```
def _truncate_messages(
 self,
 tokenizer: Any,
 messages: list[dict[str, str]],
) -> tuple[list[dict[str, str]], Any, Optional[Any]]:
 """Giriş: Tokenizer ve mesajlar.
```

Çıkış: Budanmış mesajlar, input\_ids, attention\_mask.

İşleyiş: Token sınırına sığana kadar eski mesajları çıkarır.

"""

```
while True:
 input_ids, attention_mask = self._tokenize(tokenizer, messages)
 prompt_tokens = int(input_ids.shape[-1])
 if prompt_tokens <= Config.MAX_CONTEXT_TOKENS or len(messages) <= 1:
 return messages, input_ids, attention_mask
 if messages[0]["role"] == "system" and len(messages) > 1:
 messages = messages[:1] + messages[2:]
 else:
 messages = messages[1:]
```

```
async def generate(
 self,
 messages: list[dict[str, str]],
 model_id: str,
 temperature: float,
 top_p: float,
```

```

 max_tokens: int,
 request_id: str,
) -> ChatResult:
 """Giriş: Mesajlar ve üretim parametreleri.

 Çıkış: ChatResult.
 İşleyiş: Transformers generate ile tek seferlik yanıt üretir.
 """
 model, tokenizer, device = self._get_model(model_id)
 _trimmed_messages, input_ids, attention_mask = self._truncate_messages(
 tokenizer, list(messages)
)
 prompt_tokens = int(input_ids.shape[-1])

 input_ids = input_ids.to(device)
 if attention_mask is not None:
 attention_mask = attention_mask.to(device)
 else:
 import torch
 attention_mask = torch.ones_like(input_ids)

 do_sample = temperature > 0
 output_ids = model.generate(
 input_ids=input_ids,
 attention_mask=attention_mask,
 max_new_tokens=max_tokens,
 temperature=temperature,
 top_p=top_p,
 do_sample=do_sample,
 repetition_penalty=1.1,
 pad_token_id=tokenizer.pad_token_id,
)

 generated_ids = output_ids[0][prompt_tokens:]
 text = tokenizer.decode(generated_ids, skip_special_tokens=True)
 completion_tokens = int(generated_ids.shape[-1])

 usage = Usage(
 prompt_tokens=prompt_tokens,
 completion_tokens=completion_tokens,
 total_tokens=prompt_tokens + completion_tokens,
)
 logger.info("request_id=%s provider=hf generate model=%s", request_id, model_id)
 return ChatResult(text=text, usage=usage)

async def stream(
 self,
 messages: list[dict[str, str]],
 model_id: str,
 temperature: float,
 top_p: float,
 max_tokens: int,
 request_id: str,
 cancel_token: CancellationToken,
) -> AsyncIterator[StreamChunk]:
 """Giriş: Mesajlar ve üretim parametreleri.

```

Çıkış: StreamChunk akışı.  
 İşleyiş: TextIteratorStreamer ile token akışı sağlar.

```

"""
import asyncio
from transformers import TextIteratorStreamer, StoppingCriteria, StoppingCriteriaList

model, tokenizer, device = self._get_model(model_id)
_trimmed_messages, input_ids, attention_mask = self._truncate_messages(
 tokenizer, list(messages))
)
prompt_tokens = int(input_ids.shape[-1])
completion_tokens = 0

input_ids = input_ids.to(device)
if attention_mask is not None:
 attention_mask = attention_mask.to(device)
else:
 import torch
 attention_mask = torch.ones_like(input_ids)

class StopOnCancel(StoppingCriteria):
 """Giriş: CancellationToken.

 Çıkış: StoppingCriteria.
 İşleyiş: İptal sinyali geldiğinde üretimi durdurur.
 """

 def __init__(self, token: CancellationToken) -> None:
 """Giriş: CancellationToken.

 Çıkış: Nesne.
 İşleyiş: İptal sinyalini saklar.
 """
 self._token = token

 def __call__(self, *args, **kwargs) -> bool:
 """Giriş: Model üretim argümanları.

 Çıkış: bool.
 İşleyiş: İptal sinyalini kontrol eder.
 """
 return self._token.is_cancelled()

 streamer = TextIteratorStreamer(
 tokenizer, skip_prompt=True, skip_special_tokens=True
)
 do_sample = temperature > 0
 stop_criteria = StoppingCriteriaList([StopOnCancel(cancel_token)])

 def _generate() -> None:
 """Giriş: yok.

 Çıkış: yok.
 İşleyiş: Model.generate çağrısını çalıştırır.
 """
 model.generate(
 input_ids=input_ids,
 attention_mask=attention_mask,
 max_new_tokens=max_tokens,
 temperature=temperature,

```

```

 top_p=top_p,
 do_sample=do_sample,
 repetition_penalty=repetition_penalty,
 pad_token_id=tokenizer.pad_token_id,
 streamer=streamer,
 stopping_criteria=stop_criteria,
)

loop = asyncio.get_running_loop()
loop.run_in_executor(None, _generate)

reasoning_filter = ReasoningFilter()

def _next_token() -> Optional[str]:
 """Giriş: yok.

 Çıkış: Yeni token veya None.
 İşleyiş: Streamer'dan sıradaki tokenı alır.
 """
 try:
 return next(streamer)
 except StopIteration:
 return None

while True:
 if cancel_token.is_cancelled():
 break
 token = await asyncio.to_thread(_next_token)
 if token is None:
 break
 filtered = reasoning_filter.feed(token)
 if filtered:
 completion_tokens += len(
 tokenizer.encode(filtered, add_special_tokens=False)
)
 yield StreamChunk(token=filtered)

usage = Usage(
 prompt_tokens=prompt_tokens,
 completion_tokens=completion_tokens,
 total_tokens=prompt_tokens + completion_tokens,
)
logger.info("request_id=%s provider=hf stream done model=%s", request_id, model_id)
yield StreamChunk(token="", done=True, usage=usage)

def list_models(self) -> list:
 """Giriş: yok.

 Çıkış: Boş liste.
 İşleyiş: Katalog verisi başka katmanda tutulur.
 """
 return []

```

## E:/SelcukAiAssistant/repo/lib/controller/enhanced\_chat\_controller.dart

```

import 'dart:async';
import 'dart:convert';

import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

```

```
import 'package:get/get.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:selcukaiassistant/apis/apis.dart';
import 'package:selcukaiassistant/helper/my_dialog.dart';
import 'package:selcukaiassistant/helper/pref.dart';
import 'package:selcukaiassistant/l10n/l10n.dart';
import 'package:selcukaiassistant/model/conversation.dart';
import 'package:selcukaiassistant/services/conversation_export_service.dart';
import 'package:selcukaiassistant/services/conversation_service.dart';
import 'package:selcukaiassistant/services/response_cleaner.dart';
import 'package:selcukaiassistant/services/sse_client.dart';
import 'package:speech_to_text/speech_to_text.dart';
import 'package:uuid/uuid.dart';

class EnhancedChatController extends GetxController {
 final textC = TextEditingController();
 final scrollC = ScrollController();

 final SpeechToText _speechToText = SpeechToText();
 final RxBool isListening = false.obs;
 final RxBool speechEnabled = false.obs;
 final RxBool isGenerating = false.obs;

 final RxList<ChatMessage> messages = <ChatMessage>[].obs;
 final Rx<Conversation?> currentConversation = Rx<Conversation?>(null);
 final RxString currentConversationId = ".obs";

 final _uuid = const Uuid();
 bool _stopRequested = false;
 ChatStreamSession? _streamSession;
 StreamSubscription<ChatStreamEvent>? _streamSubscription;

 @override
 void onInit() {
 super.onInit();
 unawaited(_initSpeech());
 unawaited(_initializeConversation());
 }

 String _languageCode() {
 return Pref.localeCode ?? L10n.fallbackLocale.languageCode;
 }

 String _speechLocaleId() {
 return _languageCode() == 'en' ? 'en_US' : 'tr_TR';
 }

 String _systemPrompt() {
 if (_languageCode() == 'en') {
 return 'You are a helpful assistant for Selçuk University.\n' +
 'Reply in English. Do not reveal reasoning or internal thoughts.\n' +
 'If the user greets vaguely (e.g. "Hello"), ask what they need about\n' +
 'Selçuk University.';
 }
 return 'Sel\u00e7uk \u00dcniversitesi i\u00e7in yard\u0131mc\u0131 bir\n' +
 'asistans\u0131n. Yan\u0131tlar\u0131n\u0131 T\u00fcrk\u00fcrk\u00e7e ver.\n' +
 'Ak\u0131l y\u00fcfctme veya i\u00e7\u00fcr konu\u0131fma\n' +
 'payla\u0131fma. Kullan\u0131c\u0131 genel bir selam verirse\n' +
 '(\u00f0rf. "Merhaba"), Sel\u00e7uk \u00dcniversitesi ile ilgili\n';
 }
}
```

```

 'neye ihtiyac\u00f7 duydu\u00f7unu sor.';
}

Future<void> _initializeConversation() async {
 await ConversationService.init();

 final conversations = ConversationService.getActiveConversations();
 if (conversations.isEmpty) {
 final newConversation = await ConversationService.createConversation();
 currentConversation.value = newConversation;
 currentConversationId.value = newConversation.id;
 } else {
 currentConversation.value = conversations.first;
 currentConversationId.value = conversations.first.id;
 messages.value = List.from(conversations.first.messages);
 }
}

Future<void> loadConversation(String conversationId) async {
 final conversation = ConversationService.getConversation(conversationId);
 if (conversation != null) {
 currentConversation.value = conversation;
 currentConversationId.value = conversationId;
 messages.value = List.from(conversation.messages);
 _scrollDown();
 }
}

Future<void> _initSpeech() async {
 speechEnabled.value = await _speechToText.initialize(
 onError: (error) {
 isListening.value = false;
 },
 onStatus: (status) {
 if (status == 'done' || status == 'notListening') {
 isListening.value = false;
 }
 },
);
}

Future<void> startListening() async {
 final l10n = L10n.current();
 if (!Pref.voiceInputEnabled) {
 MyDialog.info(
 l10n?.voiceInputSubtitle ??
 'Sesli mesajlar için mikrofonu etkinleştirin.',
);
 return;
 }
 final status = await Permission.microphone.request();
 if (status != PermissionStatus.granted) {
 MyDialog.info(
 l10n?.microphonePermissionRequired ??
 'Sesli giriş için mikrofon izni gereklidir.',
);
 return;
 }
}

```

```

if (!speechEnabled.value) {
 MyDialog.info(
 l10n?.speechNotAvailable ?? 'Ses tanıma kullanılamıyor.',
);
 return;
}

if (!isListening.value) {
 isListening.value = true;

 await _speechToText.listen(
 onResult: (result) {
 textC.text = result.recognizedWords;
 if (result.finalResult) {
 isListening.value = false;
 }
 },
 localeId: _speechLocaleId(),
);
}

Future<void> stopListening() async {
 if (isListening.value) {
 await _speechToText.stop();
 isListening.value = false;
 }
}

void stopGeneration() {
 _stopRequested = true;
 isGenerating.value = false;
 _cancelStreamSubscription();
 _streamSession?.close();
}

Future<void> sendMessage({
 String? imageUrl,
 String? overrideText,
 String? parentMessageId,
}) async {
 final l10n = L10n.current();
 final rawText = overrideText ?? textC.text;
 final messageText = rawText.trim();
 if (messageText.isEmpty) {
 MyDialog.info(
 l10n?.enterMessagePrompt ??
 'Lütfen bir mesaj yazın ya da sesli giriş kullanın!',
);
 return;
 }

 final userMessage = ChatMessage(
 id: _uuid.v4(),
 content: messageText,
 isUser: true,
 createdAt: DateTime.now(),
 imageUrl: imageUrl,
 parentMessageId: parentMessageId,
);
}

```

```

);

messages.add(userMessage);
await ConversationService.addMessage(
 currentConversationId.value,
 userMessage,
);

final aiMessage = ChatMessage(
 id: _uuid.v4(),
 content: '',
 isUser: false,
 createdAt: DateTime.now(),
);
messages.add(aiMessage);
_scrollDown();

if (overrideText == null) {
 textC.clear();
}
isGenerating.value = true;
_stopRequested = false;
aiMessage
..error = null
..errorCode = null;

final payloadMessages = _buildPayloadMessages();
// Model seçimi - geçersiz model varsa null gönder (backend varsayılanı)
var selectedModel = Pref.selectedModel;
if (selectedModel == 'selcuk_ai_assistant') {
 // Eski model ismi - backend'de artık yok, null gönder
 selectedModel = null;
 Pref.selectedModel = null; // Tercih temizle
}

try {
 await _streamResponse(
 payloadMessages,
 aiMessage,
 model: selectedModel,
);

 if (!_stopRequested) {
 aiMessage
 ..error = null
 ..errorCode = null;
 await ConversationService.addMessage(
 currentConversationId.value,
 aiMessage,
);
 } else if (aiMessage.content.isNotEmpty) {
 await ConversationService.addMessage(
 currentConversationId.value,
 aiMessage,
);
 } else {
 messages.remove(aiMessage);
 }
} on Exception catch (e) {

```

```

 if (_stopRequested) {
 return;
 }
 aiMessage
 ..error = e.toString()
 ..errorCode = 'stream_error';
 if (aiMessage.content.isEmpty) {
 final fallback = await APIs.sendChat(
 messages: payloadMessages,
 model: selectedModel,
);
 aiMessage
 ..content = ResponseCleaner.clean(fallback.answer)
 ..citations = fallback.citations;
 } else {
 aiMessage.content +=
 "\n\n${110n?.streamInterruptedTag ?? '[Akış kesildi]'}";
 }
 messages.refresh();
 await ConversationService.addMessage(
 currentConversationId.value,
 aiMessage,
);
 Get.snackbar(
 110n?.streamErrorTitle ?? 'Akış hatası',
 e.toString(),
 snackPosition: SnackPosition.BOTTOM,
 backgroundColor: Colors.red,
 colorText: Colors.white,
);
 } finally {
 isGenerating.value = false;
 _stopRequested = false;
 _scrollDown();
 }
}

List<Map<String, String>> _buildPayloadMessages({
 int? lastMessageIndex,
}) {
 final limitIndex =
 lastMessageIndex ?? (messages.isEmpty ? -1 : messages.length - 1);
 final slice =
 limitIndex >= 0 ? messages.sublist(0, limitIndex + 1) : <ChatMessage>[];;
 final history =
 slice.where((msg) => msg.content.trim().isNotEmpty).toList();

 const maxHistory = 20;
 final recent = history.length > maxHistory
 ? history.sublist(history.length - maxHistory)
 : history;

 final payload = <Map<String, String>>[
 {
 'role': 'system',
 'content': _systemPrompt(),
 },
 ...recent.map(
 (msg) => {
 'role': msg.isUser ? 'user' : 'assistant',

```

```
'content': msg.content,
 },
),
];
}

return payload;
}

ChatMessage _cloneMessage(ChatMessage message) {
 return ChatMessage(
 id: _uuid.v4(),
 content: message.content,
 isUser: message.isUser,
 createdAt: message.createdAt,
 imageUrl: message.imageUrl,
 role: message.role,
 provider: message.provider,
 modelId: message.modelId,
 error: message.error,
 errorCode: message.errorCode,
 parentMessageId: message.parentMessageId,
 citations: List<String>.from(message.citations),
);
}

int? _findPreviousUserIndex(int fromIndex) {
 for (var i = fromIndex - 1; i >= 0; i--) {
 if (messages[i].isUser) {
 return i;
 }
 }
 return null;
}

Future<void> editLastUserMessage(ChatMessage message) async {
 if (isGenerating.value) {
 return;
 }

 final lastUserIndex = messages.lastIndexWhere((m) => m.isUser);
 final messageIndex = messages.indexOf(message);
 if (messageIndex == -1 || messageIndex != lastUserIndex) {
 return;
 }

 final l10n = L10n.current();
 final controller = TextEditingController(text: message.content);

 final updated = await Get.dialog<String>(
 AlertDialog(
 title: Text(l10n?.editMessageTitle ?? 'Mesajı düzenle'),
 content: TextField(
 controller: controller,
 autofocus: true,
 maxLines: null,
 decoration: InputDecoration(
 hintText: l10n?.editMessageHint ?? 'Mesajınızı güncelleyin',
 border: const OutlineInputBorder(),
),
),
),
);
}
```

```

 onSubmitted: (value) => Get.back(result: value),
),
 actions: [
 TextButton(
 onPressed: () => Get.back<String>(),
 child: Text(l10n?.cancel ?? 'İptal'),
),
 TextButton(
 onPressed: () => Get.back<String>(result: controller.text),
 child: Text(l10n?.editMessageAction ?? 'Yeniden gönder'),
),
],
),
);
);

controller.dispose();

final trimmed = updated?.trim();
if (trimmed == null ||
 trimmed.isEmpty ||
 trimmed == message.content.trim()) {
 return;
}

final newConversation = await ConversationService.createConversation();
final history = messages.take(messageIndex).map(_cloneMessage).toList();

await ConversationService.setMessages(newConversation.id, history);
newConversation
 ..title = ConversationService.generateTitle(trimmed)
 ..updatedAt = DateTime.now();
await newConversation.save();

currentConversation.value = newConversation;
currentConversationId.value = newConversation.id;
messages.value = List.from(history);
_scrollDown();

await sendMessage(
 overrideText: trimmed,
 parentMessageId: message.id,
);
}

Future<void> regenerateResponse(ChatMessage message) async {
 if (isGenerating.value || message.isUser) {
 return;
 }

 final lastAssistantIndex = messages.lastIndexWhere((msg) => !msg.isUser);
 final messageIndex = messages.indexOf(message);
 if (messageIndex == -1 || messageIndex != lastAssistantIndex) {
 return;
 }

 final userIndex = _findPreviousUserIndex(messageIndex);
 if (userIndex == null) {
 return;
 }
}

```

```

message
..content =
..error =
..errorCode =
..citations =
..createdAt = DateTime.now();
messages.refresh();

final l10n = L10n.current();
isGenerating.value = true;
_stopRequested = false;

final payloadMessages = _buildPayloadMessages(lastMessageIndex: userIndex);
final selectedModel = Pref.selectedModel;

String? errorMessage;
String? errorCode;
List<String>? citations;

try {
 await _streamResponse(
 payloadMessages,
 message,
 model: selectedModel,
);
 citations = message.citations;
} on Exception catch (e) {
 if (!_stopRequested) {
 errorMessage = e.toString();
 errorCode = 'stream_error';
 message
 ..error = errorMessage
 ..errorCode = errorCode;
 messages.refresh();
 Get.snackbar(
 l10n?.streamErrorTitle ?? 'Akış hatası',
 e.toString(),
 snackPosition: SnackPosition.BOTTOM,
 backgroundColor: Colors.red,
 colorText: Colors.white,
);
 }
} finally {
 await ConversationService.updateMessage(
 currentConversationId.value,
 message.id,
 newContent: message.content,
 error: errorMessage ?? '',
 errorCode: errorCode ?? '',
 citations: citations,
);
 isGenerating.value = false;
 _stopRequested = false;
 _scrollDown();
}
}

Future<void> retryResponse(ChatMessage message) async {

```

```

 await regenerateResponse(message);
 }

Future<void> _streamResponse(
 List<Map<String, String>> messagesPayload,
 ChatMessage aiMessage, {
 String? model,
 }) async {
 final cleaner = ResponseCleaner();
 const updateInterval = Duration(milliseconds: 50);
 var lastUpdate = DateTime.fromMillisecondsSinceEpoch(0);
 _streamSession = await APIs.streamChat(
 messages: messagesPayload,
 model: model,
);

 final completer = Completer<void>();
 _streamSubscription = _streamSession!.stream.listen(
 (ChatStreamEvent event) {
 if (event.type == 'token' && event.token != null) {
 aiMessage.content = cleaner.push(event.token!);
 final now = DateTime.now();
 if (now.difference(lastUpdate) >= updateInterval) {
 messages.refresh();
 _scrollDown();
 lastUpdate = now;
 }
 } else if (event.type == 'end') {
 aiMessage.content = cleaner.finalize();
 if (event.citations != null) {
 aiMessage.citations = event.citations!;
 }
 messages.refresh();
 _scrollDown();
 if (!completer.isCompleted) {
 completer.complete();
 }
 } else if (event.type == 'error') {
 if (!completer.isCompleted) {
 completer.completeError(event.message ?? 'Akış hatası');
 }
 }
 },
 onError: (Object error) {
 if (!completer.isCompleted) {
 completer.completeError(error);
 }
 },
 onDone: () {
 if (!completer.isCompleted) {
 completer.complete();
 }
 },
 cancelOnError: true,
);
}

await completer.future;
aiMessage.content = cleaner.finalize();
messages.refresh();
_streamSession?.close();

```

```

 }

void _scrollDown() {
 WidgetsBinding.instance.addPostFrameCallback(_ {
 if (scrollC.hasClients) {
 unawaited(
 scrollC.animateTo(
 scrollC.position.maxScrollExtent,
 duration: const Duration(milliseconds: 300),
 curve: Curves.easeOut,
),
);
 }
 });
}

Future<void> clearCurrentConversation() async {
 final l10n = L10n.current();
 final confirmed = await Get.dialog<bool>(
 AlertDialog(
 title: Text(l10n?.clearConversationTitle ?? 'Sohbeti temizle'),
 content: Text(
 l10n?.clearConversationMessage ??
 'Bu sohbeti temizlemek istediğinizde emin misiniz? '
 'Bu işlem geri alınamaz.',
),
 actions: [
 TextButton(
 onPressed: () => Get.back(result: false),
 child: Text(l10n?.cancel ?? 'İptal'),
),
 TextButton(
 onPressed: () => Get.back(result: true),
 child: Text(
 l10n?.clear ?? 'Temizle',
 style: const TextStyle(color: Colors.red),
),
),
],
),
);
}

if (confirmed ?? false) {
 await ConversationService.deleteConversation(currentConversationId.value);
 final newConversation = await ConversationService.createConversation();
 unawaited(loadConversation(newConversation.id));
}
}

Future<void> exportCurrentConversation() async {
 final l10n = L10n.current();
 try {
 final conversation = currentConversation.value;
 if (conversation == null || conversation.messages.isEmpty) {
 Get.snackbar(
 l10n?.exportFailedTitle ?? 'Dışa aktarma başarısız',
 l10n?.noMessagesToExport ?? 'Dışa aktarılacak mesaj yok',
 snackPosition: SnackPosition.BOTTOM,
);
 }
 return;
 }
}

```

```

 }

 final jsonData = ConversationService.exportConversation(conversation.id);
 final jsonString = const JsonEncoder.withIndent(' ').convert(jsonData);

 final filename = 'chat_${conversation.id.substring(0, 8)}.json';
 final result = await exportConversation(filename, jsonString);

 await Clipboard.setData(ClipboardData(text: jsonString));

 final successMessage = result.path != null
 ? l10n?.exportSuccessMessage(result.path!) ??
 'Dosyaya kaydedildi: ${result.path}\nAyrıca panoya kopyalandı'
 : l10n?.exportSuccessWebMessage ??
 'Dosya indirildi ve panoya kopyalandı';

 Get.snackbar(
 l10n?.exportSuccessTitle ?? 'Dışa aktarma başarılı',
 successMessage,
 snackPosition: SnackPosition.BOTTOM,
 duration: const Duration(seconds: 4),
);
} on Exception catch (e) {
 Get.snackbar(
 l10n?.exportFailedTitle ?? 'Dışa aktarma başarısız',
 '$e',
 snackPosition: SnackPosition.BOTTOM,
 backgroundColor: Colors.red,
 colorText: Colors.white,
);
}
}

@Override
void onClose() {
 textC.dispose();
 scrollC.dispose();
 unawaited(_speechToText.cancel());
 _cancelStreamSubscription();
 _streamSession?.close();
 super.onClose();
}

void _cancelStreamSubscription() {
 final cancelFuture = _streamSubscription?.cancel();
 if (cancelFuture != null) {
 unawaited(cancelFuture);
 }
}
}

```

### **E:/SelcukAiAssistant/repo/lib/controller/chat\_controller.dart**

```

import 'dart:async';

import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:permission_handler/permission_handler.dart';
import 'package:selcukaiassistant/apis/apis.dart';
import 'package:selcukaiassistant/helper/my_dialog.dart';

```

```

import 'package:selcukaiassistant/helper/pref.dart';
import 'package:selcukaiassistant/l10n/l10n.dart';
import 'package:selcukaiassistant/model/message.dart';
import 'package:selcukaiassistant/services/response_cleaner.dart';
import 'package:speech_to_text/speech_to_text.dart';

class ChatController extends GetxController {
 final textC = TextEditingController();
 final scrollC = ScrollController();

 final SpeechToText _speechToText = SpeechToText();
 final RxBool isListening = false.obs;
 final RxBool speechEnabled = false.obs;
 final RxString recognizedText = ".obs";

 final RxList<Message> list = <Message>[].obs;

 @override
 void onInit() {
 super.onInit();
 _initDefaultMessage();
 unawaited(_initSpeech());
 }

 void _initDefaultMessage() {
 final l10n = L10n.current();
 list.assignAll([
 Message(
 msg: l10n?.startChatHint ?? 'Yapay zeka asistanıyla sohbete başlayın!',
 msgType: MessageType.bot,
),
]);
 }

 String _languageCode() {
 return Pref.localeCode ?? L10n.fallbackLocale.languageCode;
 }

 String _speechLocaleId() {
 return _languageCode() == 'en' ? 'en_US' : 'tr_TR';
 }

 String _systemPrompt() {
 if (_languageCode() == 'en') {
 return 'You are a helpful assistant for Selçuk University.\n'
 'Reply in English. Do not reveal reasoning or internal thoughts.\n'
 'If the user greets vaguely (e.g. "Hello"), ask what they need about\n'
 'Selçuk University.';
 }
 return 'Sel\u00e7uk \u00dcniversitesi i\u00f1in yard\u0131mc\u0131 bir\n'
 '\u0131stans\u0131n. Yan\u0131tlar\u0131n\u0131 T\u00fcrk\u00e7e ver.\n'
 '\u0131Ak\u0131l y\u00fcrk\u00fcme veya i\u00f1konu\u015fma\n'
 '\u0131payla\u015fma. Kullan\u0131c\u0131 genel bir selam verirse\n'
 '(\u00f6r. "Merhaba"), Sel\u00e7uk \u00dcniversitesi ile ilgili\n'
 '\u0131neye ihtiyac\u00f1 duydu\u0111funu sor.';
 }

 Future<void> _initSpeech() async {
 speechEnabled.value = await _speechToText.initialize()
 }
}

```

```

 onError: (error) {
 isListening.value = false;
 },
 onStatus: (status) {
 if (status == 'done' || status == 'notListening') {
 isListening.value = false;
 }
 },
);
}

Future<void> startListening() async {
 final l10n = L10n.current();
 if (!Pref.voiceInputEnabled) {
 MyDialog.info(
 l10n?.voiceInputSubtitle ??
 'Sesli mesajlar için mikrofonu etkinleştirin.',
);
 return;
 }
 final status = await Permission.microphone.request();
 if (status != PermissionStatus.granted) {
 MyDialog.info(
 l10n?.microphonePermissionRequired ??
 'Sesli giriş için mikrofon izni gereklidir.',
);
 return;
 }

 if (!speechEnabled.value) {
 MyDialog.info(
 l10n?.speechNotAvailable ?? 'Ses tanıma kullanılamıyor.',
);
 return;
 }

 if (!isListening.value) {
 isListening.value = true;
 recognizedText.value = "";

 await _speechToText.listen(
 onResult: (result) {
 recognizedText.value = result.recognizedWords;
 if (result.finalResult) {
 textC.text = recognizedText.value;
 isListening.value = false;
 }
 },
 localeId: _speechLocaleId(),
);
 }
}

Future<void> stopListening() async {
 if (isListening.value) {
 await _speechToText.stop();
 isListening.value = false;
 }
}

```

```

Future<void> askQuestion() async {
 final l10n = L10n.current();
 if (textC.text.trim().isNotEmpty) {
 list
 ..add(Message(msg: textC.text, msgType: MessageType.user))
 ..add(Message(msg: "", msgType: MessageType.bot));
 _scrollDown();

 final question = textC.text;
 textC.text = "";

 final payload = [
 {'role': 'system', 'content': _systemPrompt()},
 {'role': 'user', 'content': question},
];

 try {
 final res = await APIs.sendChat(
 messages: payload,
 model: Pref.selectedModel,
);

 list
 ..removeLast()
 ..add(
 Message(
 msg: ResponseCleaner.clean(res.answer),
 msgType: MessageType.bot,
),
);
 _scrollDown();
 } on Exception {
 list
 ..removeLast()
 ..add(
 Message(
 msg: l10n?.errorUnexpected ??
 'Hata: Beklenmeyen bir hata oluştu.',
 msgType: MessageType.bot,
),
);
 _scrollDown();
 }
 } else {
 MyDialog.info(
 l10n?.enterMessagePrompt ??
 'Lütfen bir mesaj yazın ya da sesli giriş kullanın!',
);
 }
}

void _scrollDown() {
 WidgetsBinding.instance.addPostFrameCallback((_) {
 if (scrollC.hasClients) {
 unawaited(
 scrollC.animateTo(
 scrollC.position.maxScrollExtent,
 duration: const Duration(milliseconds: 500),
)
);
 }
 });
}

```

```

 curve: Curves.ease,
),
);
}
});
}
}

@Override
void onClose() {
 textC.dispose();
 scrollC.dispose();
 unawaited(_speechToText.cancel());
 super.onClose();
}
}
}

```

### E:/SelcukAiAssistant/repo/lib/screen/feature/new\_chat\_screen.dart

```
// Geçici olarak Material 2 API'lerindeki withOpacity kullanımını sürdürüyoruz.
// ignore_for_file: deprecated_member_use
```

```

import 'dart:async';

import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:selcukaiassistant/controller/enhanced_chat_controller.dart';
import 'package:selcukaiassistant/helper/pref.dart';
import 'package:selcukaiassistant/110n/110n.dart';
import 'package:selcukaiassistant/screen/auth/login_screen.dart';
import 'package:selcukaiassistant/services/appwrite_service.dart';
import 'package:selcukaiassistant/services/conversation_service.dart';
import 'package:selcukaiassistant/services/image_picker_service.dart';
import 'package:selcukaiassistant/widget/conversation_list_drawer.dart';
import 'package:selcukaiassistant/widget/enhanced_message_card.dart';

class NewChatScreen extends StatefulWidget {
 const NewChatScreen({super.key});

 @override
 State<NewChatScreen> createState() => _NewChatScreenState();
}

class _NewChatScreenState extends State<NewChatScreen> {
 late EnhancedChatController _controller;
 final RxBool _isDarkMode = Get.isDarkMode.obs;
 final AppwriteService _appwriteService = AppwriteService();
 final GlobalKey<ScaffoldState> _scaffoldKey = GlobalKey<ScaffoldState>();

 @override
 void initState() {
 super.initState();
 _controller = Get.put<EnhancedChatController>(
 EnhancedChatController(),
);
 _isDarkMode.value = Get.isDarkMode;

 // Sohbet verilerini hazırlamak için servis başlatılır.
 unawaited(ConversationService.init());
 }
}
```

```

String _displayTitle(BuildContext context, String? title) {
 final l10n = context.l10n;
 if (title == null || title.trim().isEmpty) {
 return l10n.newChat;
 }
 const defaults = {'New Chat', 'Yeni sohbet'};
 if (defaults.contains(title)) {
 return l10n.newChat;
 }
 return title;
}

Future<void> _logout() async {
 final l10n = L10n.current();
 try {
 await _appwriteService.deleteCurrentSession();
 if (mounted) {
 unawaited(Get.offAll<void>(() => const LoginScreen()));
 Get.snackbar(
 l10n?.logoutSuccessTitle ?? 'Başarılı',
 l10n?.logoutSuccessMessage ?? 'Oturum başarıyla kapatıldı',
 backgroundColor: Colors.green,
 colorText: Colors.white,
 snackPosition: SnackPosition.BOTTOM,
);
 }
 } on Exception catch (e) {
 if (mounted) {
 Get.snackbar(
 l10n?.logoutErrorTitle ?? 'Hata',
 e.toString().replaceAll('Exception: ', ''),
 backgroundColor: Colors.red,
 colorText: Colors.white,
 snackPosition: SnackPosition.BOTTOM,
);
 }
 }
}

@Override
Widget build(BuildContext context) {
 final l10n = context.l10n;
 return Scaffold(
 key: _scaffoldKey,
 drawer: Obx(
 () => ConversationListDrawer(
 currentConversationId: _controller.currentConversationId.value,
 onConversationSelected: (id) {
 unawaited(_controller.loadConversation(id));
 },
),
),
 appBar: AppBar(
 leading: IconButton(
 icon: const Icon(Icons.menu),
 onPressed: () {
 _scaffoldKey.currentState?.openDrawer();
 },
),
 title: Obx(

```

```

0 {
 final title = _displayTitle(
 context,
 _controller.currentConversation.value?.title,
);
 return Text(
 title,
 maxLines: 1,
 overflow: TextOverflow.ellipsis,
);
},
centerTitle: true,
elevation: 1,
actions: [
 // Koyu/açık tema geçisi
 IconButton(
 padding: const EdgeInsets.only(right: 10),
 onPressed: () {
 final newMode =
 _isDarkMode.value ? ThemeMode.light : ThemeMode.dark;
 Get.changeThemeMode(newMode);
 _isDarkMode.value = !_isDarkMode.value;
 Pref.isDarkMode = _isDarkMode.value;
 },
 icon: Obx(
 () => Icon(
 _isDarkMode.value
 ? Icons.brightness_2_rounded
 : Icons.brightness_5_rounded,
 size: 24,
),
),
),
],
// Ek seçenekler
PopupMenuButton<String>(
 icon: const Icon(Icons.more_vert),
 onSelected: (value) {
 if (value == 'logout') {
 unawaited(_logout());
 } else if (value == 'clear') {
 unawaited(_controller.clearCurrentConversation());
 } else if (value == 'export') {
 unawaited(_controller.exportCurrentConversation());
 }
 },
 itemBuilder: (context) => [
 PopupMenuItem(
 value: 'clear',
 child: Row(
 children: [
 const Icon(Icons.delete_sweep, size: 20),
 const SizedBox(width: 12),
 Text(l10n.clearChat),
],
),
),
 PopupMenuItem(
 value: 'export',
 child: Row(

```





```
color: Theme.of(context).brightness == Brightness.dark
 ? Colors.grey[800]
 : Colors.grey[100],
borderRadius: BorderRadius.circular(24),
),
child: Row(
children: [
// Görüel ekleme düğmesi
IconButton(
icon: const Icon(Icons.image, size: 20),
onPressed: () async {
final image = await ImagePickerService
.showImageSourceDialog();
if (image != null && mounted) {
final fileName = image.name.isNotEmpty
? image.name
: image.path.split('/').last;
Get.snackbar(
l10n.imageSelectedTitle,
l10n.imageSelectedMessage(
fileName,
),
snackPosition: SnackBarPosition.BOTTOM,
);
}
},
color: Theme.of(context)
.textTheme
.bodyMedium
?.color
?.withOpacity(0.6),
),
Expanded(
child: TextFormField(
controller: _controller.textField,
maxLines: null,
textInputAction: TextInputAction.send,
onFieldSubmitted: (_) {
unawaited(_controller.sendMessage());
}),
decoration: InputDecoration(
hintText: l10n.messageHint,
hintStyle: TextStyle(
fontSize: 14,
color: Theme.of(context)
.textTheme
.bodyMedium
?.color
?.withOpacity(0.6),
),
border: InputBorder.none,
contentPadding: const EdgeInsets.symmetric(
horizontal: 16,
vertical: 12,
),
),
),
],
),
),
```



```
?.color
?._withOpacity(0.7),
)
)
const SizedBox(height: 8),
Text(
 110n.startConversationSubtitle,
 style: TextStyle(
 fontSize: 14,
 color: Theme.of(context)
 .textTheme
 .bodySmall
 ?._color
 ?._withOpacity(0.5),
)
)
const SizedBox(height: 32),
// Önerilen başlangıç soruları
//(kullanıcıyı yönlendirmek için).
Wrap(
 spacing: 8,
 runSpacing: 8,
 alignment: WrapAlignment.center,
 children: [
 _SuggestedPrompt(
 text: 110n.suggestedPrompt1,
 icon: Icons.science,
 onTap: () {
 _controller.textC.text =
 110n.suggestedPrompt1;
 unawaited(_controller.sendMessage());
 },
),
 _SuggestedPrompt(
 text: 110n.suggestedPrompt2,
 icon: Icons.edit,
 onTap: () {
 _controller.textC.text =
 110n.suggestedPrompt2;
 unawaited(_controller.sendMessage());
 },
),
 _SuggestedPrompt(
 text: 110n.suggestedPrompt3,
 icon: Icons.code,
 onTap: () {
 _controller.textC.text =
 110n.suggestedPrompt3;
 unawaited(_controller.sendMessage());
 },
),
],
)
),
)
ListView.builder(
 physics: const BouncingScrollPhysics(),
 controller: _controller.scrollC,
 padding: const EdgeInsets.only(
 top: 16,
```

```

 top: 16,
 bottom: 16,
),
 itemCount: _controller.messages.length,
 itemBuilder: (context, index) {
 final lastUserIndex = _controller.messages
 .lastIndexWhere((m) => m.isUser);
 final lastAssistantIndex = _controller.messages
 .lastIndexWhere((m) => !m.isUser);
 final message = _controller.messages[index];
 final actionsEnabled = !_controller.isGenerating.value;
 final canEdit = actionsEnabled &&
 message.isUser &&
 index == lastUserIndex;
 final canRetry = actionsEnabled &&
 !message.isUser &&
 message.hasError;
 final canRegenerate = actionsEnabled &&
 !message.isUser &&
 index == lastAssistantIndex &&
 !message.hasError;
 final showTypingIndicator =
 _controller.isGenerating.value &&
 !message.isUser &&
 index == lastAssistantIndex &&
 message.content.trim().isEmpty;
 return EnhancedMessageCard(
 message: message,
 onEdit: canEdit
 ? () => _controller.editLastUserMessage(message)
 : null,
 onRetry: canRetry
 ? () => _controller.retryResponse(message)
 : null,
 onRegenerate: canRegenerate
 ? () => _controller.regenerateResponse(message)
 : null,
 showTypingIndicator: showTypingIndicator,
);
 },
),
],
),
),
);
}
}

class _SuggestedPrompt extends StatelessWidget {
const _SuggestedPrompt({
 required this.text,
 required this.icon,
 required this.onTap,
});
final String text;
final IconData icon;
final VoidCallback onTap;

```

```
@override
Widget build(BuildContext context) {
 return InkWell(
 onTap: onTap,
 borderRadius: BorderRadius.circular(20),
 child: Container(
 padding: const EdgeInsets.symmetric(horizontal: 16, vertical: 10),
 decoration: BoxDecoration(
 color: Theme.of(context).brightness == Brightness.dark
 ? Colors.grey[800]
 : Colors.grey[100],
 borderRadius: BorderRadius.circular(20),
 border: Border.all(
 color: Theme.of(context).dividerColor,
),
),
 child: Row(
 mainAxisAlignment: MainAxisAlignment.min,
 children: [
 Icon(icon, size: 16),
 const SizedBox(width: 8),
 Text(
 text,
 style: const TextStyle(fontSize: 13),
),
],
),
);
}
```

E:/SelcukAiAssistant/repo/lib/services/conversation\_service.dart

```
import 'package:hive/hive.dart';
import 'package:selcukaiassistant/110n/110n.dart';
import 'package:selcukaiassistant/model/conversation.dart';
import 'package:selcukaiassistant/services/storage/storage_service.dart';
import 'package:uuid/uuid.dart';

class ConversationService {
 static Box<Conversation>? _box;
 static const _uuid = Uuid();
 static const Set<String> _defaultTitles = {'New Chat', 'Yeni Sohbet'};

 static Future<void> init() async {
 if (_box != null) {
 return;
 }
 await StorageService.initialize();
 _box = StorageService.conversationsBox;
 }

 static Box<Conversation> get box {
 if (_box == null) {
 throw Exception('ConversationService not initialized');
 }
 return _box!;
 }

 // Create a new conversation
```

```

static Future<Conversation> createConversation({String? title}) async {
 final defaultTitle = L10n.current()?.newChat ?? 'Yeni Sohbet';
 final conversation = Conversation(
 id: _uuid.v4(),
 title: title ?? defaultTitle,
 createdAt: DateTime.now(),
 updatedAt: DateTime.now(),
 messages: [],
);

 await box.put(conversation.id, conversation);
 return conversation;
}

// Get all conversations sorted by updated time
static List<Conversation> getAllConversations({
 bool includeArchived = true,
}) {
 final conversations = box.values.where((conversation) {
 if (includeArchived) {
 return true;
 }
 return !conversation.archived;
 }).toList()
 ..sort((a, b) => b.updatedAt.compareTo(a.updatedAt));
 return conversations;
}

static List<Conversation> getActiveConversations() {
 return getAllConversations(includeArchived: false);
}

static List<Conversation> getArchivedConversations() {
 return box.values.where((conversation) => conversation.archived).toList()
 ..sort((a, b) => b.updatedAt.compareTo(a.updatedAt));
}

// Get a specific conversation by ID
static Conversation? getConversation(String id) {
 return box.get(id);
}

// Add a message to a conversation
static Future<void> addMessage(
 String conversationId,
 ChatMessage message,
) async {
 final conversation = box.get(conversationId);
 if (conversation != null) {
 conversation
 ..messages.add(message)
 ..updatedAt = DateTime.now();

 // Auto-generate title from first user message if still default
 if (_defaultTitles.contains(conversation.title) &&
 message.isUser &&
 conversation.messages.where((m) => m.isUser).length == 1) {
 conversation
 ..title = generateTitle(message.content)
 }
 }
}

```

```

 ..updatedAt = DateTime.now();
 }

 await conversation.save();
}

// Update a message in a conversation
static Future<void> updateMessage(
 String conversationId,
 String messageId,
 String? newContent,
 String? error,
 String? errorCode,
 List<String>? citations,
) async {
 final conversation = box.get(conversationId);
 if (conversation != null) {
 final messageIndex =
 conversation.messages.indexWhere((m) => m.id == messageId);
 if (messageIndex != -1) {
 final message = conversation.messages[messageIndex];
 if (newContent != null) {
 message.content = newContent;
 }
 if (error != null) {
 message.error = error.isEmpty ? null : error;
 }
 if (errorCode != null) {
 message.errorCode = errorCode.isEmpty ? null : errorCode;
 }
 if (citations != null) {
 message.citations = citations;
 }
 conversation.updatedAt = DateTime.now();
 await conversation.save();
 }
 }
}

static Future<void> setMessages(
 String conversationId,
 List<ChatMessage> messages,
) async {
 final conversation = box.get(conversationId);
 if (conversation != null) {
 conversation
 ..messages = messages
 ..updatedAt = DateTime.now();
 await conversation.save();
 }
}

// Delete a conversation
static Future<void> deleteConversation(String id) async {
 await box.delete(id);
}

// Rename a conversation

```

```

static Future<void> renameConversation(String id, String newTitle) async {
 final conversation = box.get(id);
 if (conversation != null) {
 conversation
 ..title = newTitle
 ..updatedAt = DateTime.now();
 await conversation.save();
 }
}

// Search conversations
static List<Conversation> searchConversations(
 String query, {
 bool includeArchived = true,
}) {
 final lowerQuery = query.toLowerCase();
 final results = box.values.where((conversation) {
 if (!includeArchived && conversation.archived) {
 return false;
 }
 // Search in title
 if (conversation.title.toLowerCase().contains(lowerQuery)) {
 return true;
 }
 // Search in messages
 return conversation.messages.any(
 (msg) => msg.content.toLowerCase().contains(lowerQuery),
);
 }).toList()
 ..sort((a, b) => b.updatedAt.compareTo(a.updatedAt));
 return results;
}

static Future<void> setPinned(String id, {required bool pinned}) async {
 final conversation = box.get(id);
 if (conversation != null) {
 conversation.pinned = pinned;
 if (pinned) {
 conversation.archived = false;
 }
 await conversation.save();
 }
}

static Future<void> setArchived(String id, {required bool archived}) async {
 final conversation = box.get(id);
 if (conversation != null) {
 conversation.archived = archived;
 if (archived) {
 conversation.pinned = false;
 }
 await conversation.save();
 }
}

// Clear all conversations
static Future<void> clearAll() async {
 await box.clear();
}

```

```

// Export conversation as JSON
static Map<String, dynamic> exportConversation(String id) {
 final conversation = box.get(id);
 if (conversation != null) {
 return conversation.toJson();
 }
 return {};
}

// Generate a smart title from the first message
static String generateTitle(String content) {
 // Take first 50 characters or first sentence
 var title = content.trim();

 if (title.length > 50) {
 title = title.substring(0, 50);
 // Try to end at a word boundary
 final lastSpace = title.lastIndexOf(' ');
 if (lastSpace > 30) {
 title = title.substring(0, lastSpace);
 }
 title = '$title...';
 }

 return title;
}

// Get statistics
static Map<String, dynamic> getStatistics() {
 final conversations = box.values.toList();
 final totalMessages = conversations.fold<int>(
 0,
 (sum, conv) => sum + conv.messages.length,
);

 return {
 'totalConversations': conversations.length,
 'totalMessages': totalMessages,
 'oldestConversation': conversations.isNotEmpty
 ? conversations
 .reduce((a, b) => a.createdAt.isBefore(b.createdAt) ? a : b)
 .createdAt
 : null,
 'newestConversation': conversations.isNotEmpty
 ? conversations
 .reduce((a, b) => a.createdAt.isAfter(b.createdAt) ? a : b)
 .createdAt
 : null,
 };
}

```

### **E:/SelcukAiAssistant/repo/lib/services/sse\_client.dart**

```

export 'sse_client_io.dart' if (dart.library.html) 'sse_client_web.dart';
export 'sse_client_types.dart';

```

### **E:/SelcukAiAssistant/repo/lib/services/model\_service.dart**

```

import 'dart:convert';
import 'dart:developer';

```

```

import 'package:http/http.dart' as http;
import 'package:selcukaiassistant/config/backend_config.dart';
import 'package:selcukaiassistant/helper/pref.dart';
import 'package:selcukaiassistant/l10n/l10n.dart';
import 'package:selcukaiassistant/model/model_info.dart';

class ModelService {
 static Future<List<ModelInfo>> fetchModels() async {
 try {
 final locale = Pref.localeCode ?? L10n.fallbackLocale.languageCode;
 final response = await http.get(
 Uri.parse(BackendConfig.modelsEndpoint),
 headers: {
 'Content-Type': 'application/json; charset=utf-8',
 'Accept-Language': locale,
 },
);

 if (response.statusCode != 200) {
 log('Models endpoint error: ${response.statusCode}');
 return [];
 }

 final payload = jsonDecode(utf8.decode(response.bodyBytes))
 as Map<String, dynamic>;
 final items = payload['models'] as List<dynamic>? ?? [];
 return items
 .map((item) => ModelInfo.fromJson(item as Map<String, dynamic>))
 .toList();
 } on Exception catch (e) {
 log('Failed to fetch models: $e');
 return [];
 }
 }
}

```

### E:/SelcukAiAssistant/repo/lib/apis/apis.dart

```

import 'dart:async';
import 'dart:convert';
import 'dart:developer';

import 'package:http/http.dart' as http;
import 'package:selcukaiassistant/config/backend_config.dart';
import 'package:selcukaiassistant/helper/pref.dart';
import 'package:selcukaiassistant/l10n/l10n.dart';
import 'package:selcukaiassistant/model/chat_api_response.dart';
import 'package:selcukaiassistant/services/sse_client.dart';

class APIs {
 static const Duration _responseTimeout = Duration(seconds: 120);

 static Map<String, String> _buildHeaders() {
 final locale = Pref.localeCode ?? L10n.fallbackLocale.languageCode;
 return {
 'Content-Type': 'application/json; charset=utf-8',
 'Accept-Language': locale,
 };
 }
}

```

```

static Map<String, dynamic> _buildPayload({
 required List<Map<String, String>> messages,
 String? model,
 bool stream = false,
}) {
 return {
 'model': model,
 'messages': messages,
 'temperature': 0.2,
 'top_p': 0.9,
 'max_tokens': 256,
 'stream': stream,
 'rag_enabled': Pref.ragEnabled,
 'rag.strict': Pref.ragStrict,
 'rag_top_k': Pref.ragTopK,
 };
}

static Future<String> getAnswer(
 String question,
 String? model,
 String? systemPrompt,
) async {
 final messages = <Map<String, String>>[];
 if (systemPrompt != null && systemPrompt.trim().isNotEmpty) {
 messages.add({'role': 'system', 'content': systemPrompt});
 }
 messages.add({'role': 'user', 'content': question});

 final response = await sendChat(
 messages: messages,
 model: model,
);
 return response.answer;
}

static Future<ChatApiResponse> sendChat({
 required List<Map<String, String>> messages,
 String? model,
}) async {
 final l10n = L10n.current();
 final timeoutMessage =
 l10n.requestTimeoutMessage ??
 'İstek zaman aşımına uğradı. Lütfen tekrar deneyin.';
 try {
 log('Backend API: ${BackendConfig.chatEndpoint}');

 final requestBody = jsonEncode(
 _buildPayload(messages: messages, model: model),
);

 final response = await http
 .post(
 Uri.parse(BackendConfig.chatEndpoint),
 headers: _buildHeaders(),
 body: requestBody,
)
 .timeout(

```

```

 _responseTimeout,
 onTimeout: () {
 throw TimeoutException(timeoutMessage);
 },
);

if (response.statusCode == 200) {
 final responseData = jsonDecode(utf8.decode(response.bodyBytes))
 as Map<String, dynamic>;
 final answer = (responseData['answer'] as String?) ??
 (110n?.noResponseGenerated ?? 'Üzgünüm, bir yanıt üretilemedi.');
 final citations = (responseData['citations'] as List<dynamic>?) ??
 .map((item) => item.toString())
 .toList() ??
 <String>[];;
 final usage =
 responseData['usage'] as Map<String, dynamic>? ??
 <String, dynamic>{};
 return ChatApiResponse(
 answer: answer,
 citations: citations,
 usage: usage,
);
} else if (response.statusCode == 400) {
 try {
 final errorData =
 jsonDecode(response.body) as Map<String, dynamic>;
 final errorMessage =
 errorData['detail'] as String? ?? 110n?.errorInvalidRequest;
 return ChatApiResponse(
 answer: errorMessage ?? 'Hata: Geçersiz istek.',
);
 } on FormatException {
 return ChatApiResponse(
 answer: 110n?.errorInvalidRequestFormat ??
 'Hata: İstek formatı geçersiz.',
);
 }
} else if (response.statusCode == 503) {
 return ChatApiResponse(
 answer: 110n?.errorServiceUnavailable ??
 'Hata: Yapay zeka servisine ulaşılamıyor.',
);
} else if (response.statusCode == 504) {
 return ChatApiResponse(
 answer: 110n?.errorTimeout ?? 'Hata: Yanıt zaman aşımına uğradı.',
);
}

return ChatApiResponse(
 answer: 110n?.errorBackendUnavailable ??
 'Hata: Backend servisi kullanılamıyor.',
);
} on TimeoutException catch (e) {
 log('Backend timeout: $e');
 return ChatApiResponse(answer: e.message ?? timeoutMessage);
} on http.ClientException catch (e) {
 log('Network error: $e');
 return ChatApiResponse(
 answer: 110n?.errorNoInternet ?? 'Hata: İnternet bağlantısı yok.',
);
}

```

```
);
} on FormatException catch (e) {
 log('Parse error: $e');
 return ChatApiResponse(
 answer: 110n?.errorInvalidServerResponse ??
 'Hata: Sunucu yanıtı geçersiz.',
);
} on Exception catch (e) {
 log('Backend error: $e');
 return ChatApiResponse(
 answer: 110n?.errorUnexpected ??
 'Hata: Beklenmeyen bir hata oluştu.',
);
}
}

static Future<ChatStreamSession> streamChat({
 required List<Map<String, String>> messages,
 String? model,
}) async {
 final client = SseClient();
 return client.connect(
 url: Uri.parse(BackendConfig.chatStreamEndpoint),
 headers: _buildHeaders(),
 body: jsonEncode(
 _buildPayload(messages: messages, model: model, stream: true),
),
);
}
```

E:/SelcukAiAssistant/repo/lib/config/backend\_config.dart

```
import 'package:flutter/foundation.dart';
import 'package:flutter_dotenv/flutter_dotenv.dart';
import 'package:http/http.dart' as http;
import 'package:selcukaiassistant/helper/pref.dart';

enum BackendUrlSource {
 override,
 dartDefine,
 dotenv,
 webRelease,
 webDev,
 androidEmulator,
 desktop,
}

class BackendUrlResolution {
 const BackendUrlResolution({
 required this.url,
 required this.source,
 });

 final String url;
 final BackendUrlSource source;
}

class BackendConfig {
 static BackendUrlResolution get resolution => resolve();
```

```

static String get baseUrl => resolution.url;

static String get chatEndpoint => '$baseUrl/chat';
static String get chatStreamEndpoint => '$baseUrl/chat/stream';
static String get modelsEndpoint => '$baseUrl/models';

static Future<bool> testConnection() async {
try {
 final response = await http.get(Uri.parse('$baseUrl/health'));
 return response.statusCode == 200;
} on Exception {
 return false;
}
}

static BackendUrlResolution _resolve() {
final override = Pref.backendUrlOverride?.trim();
if (override != null && override.isNotEmpty) {
 return BackendUrlResolution(
 url: _normalizeBaseUrl(_adjustForPlatform(override)),
 source: BackendUrlSource.override,
);
}

const envUrl = String.fromEnvironment('BACKEND_URL');
if (envUrl.isNotEmpty) {
 return BackendUrlResolution(
 url: _normalizeBaseUrl(_adjustForPlatform(envUrl)),
 source: BackendUrlSource.dartDefine,
);
}

final dotenvUrl = dotenv.env['BACKEND_URL']?.trim();
if (dotenvUrl != null && dotenvUrl.isNotEmpty) {
 return BackendUrlResolution(
 url: _normalizeBaseUrl(_adjustForPlatform(dotenvUrl)),
 source: BackendUrlSource.dotenv,
);
}

if (kIsWeb) {
 if (kReleaseMode) {
 return BackendUrlResolution(
 url: _normalizeBaseUrl('${Uri.base.origin}/api'),
 source: BackendUrlSource.webRelease,
);
 }
 return const BackendUrlResolution(
 url: 'http://localhost:8000',
 source: BackendUrlSource.webDev,
);
}
if (_isAndroid()) {
 return const BackendUrlResolution(
 url: 'http://10.0.2.2:8000',
 source: BackendUrlSource.androidEmulator,
);
}
}

```

```

 return const BackendUrlResolution(
 url: 'http://localhost:8000',
 source: BackendUrlSource.desktop,
);
 }

 static String _adjustForPlatform(String url) {
 if (kIsWeb &&
 (url.contains('10.0.2.2') || url.contains('10.0.3.2'))) {
 return 'http://localhost:8000';
 }
 if (_isAndroid() &&
 (url.contains('localhost') || url.contains('127.0.0.1'))) {
 return 'http://10.0.2.2:8000';
 }
 return url;
 }

 static String _normalizeBaseUrl(String url) {
 var normalized = url.trim();
 if (normalized.endsWith('/')) {
 normalized = normalized.substring(0, normalized.length - 1);
 }
 return normalized;
 }

 static bool _isAndroid() {
 return !kIsWeb && defaultTargetPlatform == TargetPlatform.android;
 }
}

```

### E:/SelcukAiAssistant/repo/lib/screen/model\_picker\_screen.dart

```

import 'dart:async';

import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:selcukaiassistant/helper/pref.dart';
import 'package:selcukaiassistant/l10n/l10n.dart';
import 'package:selcukaiassistant/model/model_info.dart';
import 'package:selcukaiassistant/services/model_service.dart';
import 'package:selcukaiassistant/widget/model_card.dart';

class ModelPickerScreen extends StatefulWidget {
 const ModelPickerScreen({
 super.key,
 this.initialModels,
 });

 final List<ModelInfo>? initialModels;

 @override
 State<ModelPickerScreen> createState() => _ModelPickerScreenState();
}

class _ModelPickerScreenState extends State<ModelPickerScreen> {
 final TextEditingController _searchController = TextEditingController();
 List<ModelInfo> _models = [];
 bool _loading = true;
}

```

```

@Override
void initState() {
 super.initState();
 _searchController.addListener(_onSearchChanged);
 unawaited(_loadModels());
}

@Override
void dispose() {
 _searchController
 ..removeListener(_onSearchChanged)
 ..dispose();
 super.dispose();
}

Future<void> _loadModels() async {
 final models = widget.initialModels ?? await ModelService.fetchModels();
 if (!mounted) {
 return;
 }
 setState(() {
 _models = models;
 _loading = false;
 });
}

void _onSearchChanged() {
 setState(() {});
}

List<ModelInfo> _filteredModels() {
 final query = _searchController.text.trim().toLowerCase();
 if (query.isEmpty) {
 return _models;
 }
 return _models.where((model) {
 return model.displayName.toLowerCase().contains(query) ||
 model.modelId.toLowerCase().contains(query) ||
 model.provider.toLowerCase().contains(query);
 }).toList();
}

void _selectModel(ModelInfo model) {
 Pref.selectedModel = model.id;
 Get.back<String>(result: model.id);
}

Widget _buildSection(
 BuildContext context,
 String title,
 List<ModelInfo> models,
 String selectedId,
) {
 if (models.isEmpty) {
 return const SizedBox.shrink();
 }
 return Column(
 crossAxisAlignment: CrossAxisAlignment.start,
 children: [

```

```
Padding(
 padding: const EdgeInsets.only(top: 16, bottom: 8),
 child: Text(
 '$title (${models.length})',
 style: TextStyle(
 fontSize: 12,
 fontWeight: FontWeight.w600,
 color: Theme.of(context).colorScheme.primary,
),
),
,
,
)...models.map(
 (model) => ModelCard(
 model: model,
 selected: model.id == selectedId,
 onSelect: model.available ? () => _selectModel(model) : null,
),
,
],
);
}

@override
Widget build(BuildContext context) {
 final l10n = context.l10n;
 final selectedId = Pref.selectedModel ?? "";
 final filtered = _filteredModels();
 final localModels = filtered.where((m) => m.isLocal).toList();
 final remoteModels = filtered.where((m) => m.isRemote).toList();

 return Scaffold(
 appBar: AppBar(
 title: Text(l10n.selectModelTitle),
),
 body: SafeArea(
 child: _loading
 ? const Center(child: CircularProgressIndicator())
 : Column(
 children: [
 Padding(
 padding: const EdgeInsets.all(12),
 child: TextField(
 controller: _searchController,
 decoration: InputDecoration(
 hintText: l10n.modelSearchHint,
 prefixIcon: const Icon(Icons.search),
 suffixIcon: _searchController.text.isNotEmpty
 ? IconButton(
 icon: const Icon(Icons.clear),
 onPressed: _searchController.clear,
)
 : null,
 border: OutlineInputBorder(
 borderRadius: BorderRadius.circular(12),
),
 contentPadding: const EdgeInsets.symmetric(
 horizontal: 16,
 vertical: 12,
),
),
),
],
),
);
}
```

```
),
),
 Expanded(
 child: filtered.isEmpty
 ? Center(
 child: Text(l10n.modelNoResults),
)
 : RefreshIndicator(
 onRefresh: _loadModels,
 child: ListView(
 padding: const EdgeInsets.symmetric(
 horizontal: 12,
 vertical: 8,
),
 children: [
 _buildSection(
 context,
 l10n.modelLocalSection,
 localModels,
 selectedId,
),
 _buildSection(
 context,
 l10n.modelRemoteSection,
 remoteModels,
 selectedId,
),
],
),
),
),
],
),
);
```

E:/SelcukAiAssistant/repo/lib/screen/settings\_screen.dart

```
import 'dart:async';

import 'package:flutter/material.dart';
import 'package:get/get.dart';
import 'package:selcukaiassistant/config/backend_config.dart';
import 'package:selcukaiassistant/controller/settings_controller.dart';
import 'package:selcukaiassistant/l10n/l10n.dart';
import 'package:selcukaiassistant/model/model_info.dart';
import 'package:selcukaiassistant/screen/diagnostics_screen.dart';
import 'package:selcukaiassistant/screen/model_picker_screen.dart';
import 'package:selcukaiassistant/services/conversation_service.dart';

class SettingsScreen extends StatefulWidget {
 const SettingsScreen({super.key});

 @override
 State<SettingsScreen> createState() => _SettingsScreenState();
}

class _SettingsScreenState extends State<SettingsScreen> {
 late final SettingsController _controller;
```

```

@Override
void initState() {
 super.initState();
 _controller = Get.isRegistered<SettingsController>()
 ? Get.find<SettingsController>()
 : Get.put(SettingsController());
}

Future<void> _clearAllData() async {
 final l10n = context.l10n;
 final confirmed = await Get.dialog<bool>(
 AlertDialog(
 title: Text(l10n.clearAllDialogTitle),
 content: Text(l10n.clearAllDataMessage),
 actions: [
 TextButton(
 onPressed: () => Get.back(result: false),
 child: Text(l10n.cancel),
),
 TextButton(
 onPressed: () => Get.back(result: true),
 child: Text(
 l10n.deleteAll,
 style: const TextStyle(color: Colors.red),
),
),
],
),
);
}

if (confirmed ?? false) {
 await ConversationService.clearAll();
 if (mounted) {
 Get.snackbar(
 l10n.clearAllSuccessTitle,
 l10n.clearAllSuccessMessage,
 snackPosition: SnackPosition.BOTTOM,
);
 }
}
}

String _resolvedBackendUrl() => BackendConfig.baseUrl;

Future<void> _editBackendUrl() async {
 final l10n = context.l10n;
 final controller = TextEditingController(
 text: _controller.backendUrlOverride.value,
);

 final action = await Get.dialog<String>(
 AlertDialog(
 title: Text(l10n.backendUrlTitle),
 content: TextField(
 controller: controller,
 decoration: InputDecoration(
 hintText: l10n.backendUrlHint,
),
),
),
);
}

```

```

),
actions: [
 TextButton(
 onPressed: () => Get.back(result: 'cancel'),
 child: Text(l10n.cancel),
),
 TextButton(
 onPressed: () => Get.back(result: 'clear'),
 child: Text(l10n.backendUrlClear),
),
 TextButton(
 onPressed: () => Get.back(result: 'save'),
 child: Text(l10n.backendUrlSave),
),
],
),
);
}

if (!mounted || action == null || action == 'cancel') {
 return;
}

if (action == 'clear') {
 _controller.clearBackendUrlOverride();
 Get.snackbar(
 l10n.successTitle,
 l10n.backendUrlCleared,
 snackPosition: SnackPosition.BOTTOM,
);
 return;
}

_controller.setBackendUrlOverride(controller.text);
Get.snackbar(
 l10n.successTitle,
 l10n.backendUrlSaved,
 snackPosition: SnackPosition.BOTTOM,
);

ModelInfo? _selectedModelInfo() {
 return _controller.selectedModelInfo();
}

Future<void> _openModelPicker() async {
 final models = _controller.models.toList();
 if (models.isEmpty) return;
 final selected = await Get.to<String>(
 () => ModelPickerScreen(initialModels: models),
);
 if (selected != null && selected.isNotEmpty) {
 _controller.selectModel(selected);
 }
}

Widget _buildSection(
 BuildContext context,
 String title,
 List<Widget> children,

```

```

) {
 return Column(
 mainAxisAlignment: MainAxisAlignment.start,
 children: [
 Padding(
 padding: const EdgeInsets.fromLTRB(16, 24, 16, 8),
 child: Text(
 title,
 style: TextStyle(
 fontSize: 14,
 fontWeight: FontWeight.w600,
 color: Theme.of(context).colorScheme.primary,
),
),
),
 Card(
 margin: const EdgeInsets.symmetric(horizontal: 16),
 child: Column(children: children),
),
],
);
}

@Override
Widget build(BuildContext context) {
 final l10n = context.l10n;
 final stats = ConversationService.getStatistics();

 return Scaffold(
 appBar: AppBar(
 title: Text(l10n.settingsTitle),
 centerTitle: true,
),
 body: ListView(
 children: [
 // Görünüm ayarları
 _buildSection(
 context,
 l10n.sectionAppearance,
 [
 Obx(
 () => SwitchListTile(
 title: Text(l10n.darkModeTitle),
 subtitle: Text(l10n.darkModeSubtitle),
 value: _controller.isDarkMode.value,
 activeThumbColor: Theme.of(context).colorScheme.primary,
 onChanged: (value) => _controller.setDarkMode(value: value),
 secondary: Icon(
 _controller.isDarkMode.value
 ? Icons.dark_mode
 : Icons.light_mode,
),
),
),
],
),
 // Dil seçimi
 _buildSection(
 context,

```



```

// Sunucu ayarları
_buildSection(
 context,
 l10n.sectionServer,
 [
 Obx(
 () {
 final backendUrlOverride =
 _controller.backendUrlOverride.value;
 return ListTile(
 key: ValueKey(backendUrlOverride),
 title: Text(l10n.backendUrlTitle),
 subtitle: Text(
 l10n.backendUrlSubtitle(_resolvedBackendUrl()),
),
 leading: const Icon(Icons.link),
 trailing: const Icon(Icons.edit, size: 16),
 onTap: _editBackendUrl,
);
 },
),
],
),

// Sohbet ayarları
_buildSection(
 context,
 l10n.sectionChatSettings,
 [
 Obx(
 () => SwitchListTile(
 title: Text(l10n.voiceInputTitle),
 subtitle: Text(l10n.voiceInputSubtitle),
 value: _controller.voiceInputEnabled.value,
 activeThumbColor: Theme.of(context).colorScheme.primary,
 onChanged: (value) =>
 _controller.setVoiceInputEnabled(value: value),
 secondary: const Icon(Icons.mic),
),
),
 const Divider(height: 1),
 Obx(
 () => SwitchListTile(
 title: Text(l10n.markdownSupportTitle),
 subtitle: Text(l10n.markdownSupportSubtitle),
 value: _controller.markdownEnabled.value,
 activeThumbColor: Theme.of(context).colorScheme.primary,
 onChanged: (value) =>
 _controller.setMarkdownEnabled(value: value),
 secondary: const Icon(Icons.code),
),
),
 const Divider(height: 1),
 Obx(
 () => SwitchListTile(
 title: Text(l10n.ragEnabledTitle),
 subtitle: Text(l10n.ragEnabledSubtitle),
 value: _controller.ragEnabled.value,
 activeThumbColor: Theme.of(context).colorScheme.primary,
),
),
],
);

```

```
onChanged: (value) => _controller.setRagEnabled(value: value),
secondary: const Icon(Icons.menu_book_outlined),
),
),
const Divider(height: 1),
Obx(
() => SwitchListTile(
title: Text(l10n.ragStrictTitle),
subtitle: Text(l10n.ragStrictSubtitle),
value: _controller.ragStrict.value,
activeThumbColor: Theme.of(context).colorScheme.primary,
onChanged: _controller.ragEnabled.value
? (value) => _controller.setRagStrict(value: value)
: null,
secondary: const Icon(Icons.verified_outlined),
),
),
],
),
),
),

// Teshis ekranı
_buildSection(
context,
l10n.sectionDiagnostics,
[
ListTile(
title: Text(l10n.diagnosticsTitle),
subtitle: Text(l10n.diagnosticsSubtitle),
leading: const Icon(Icons.monitor_heart),
trailing: const Icon(Icons.arrow_forward_ios, size: 16),
onTap: () {
unawaited(Get.to<void>(() => const DiagnosticsScreen()));
},
),
],
),
),

// İstatistikler
_buildSection(
context,
l10n.sectionStatistics,
[
ListTile(
title: Text(l10n.totalConversations),
trailing: Text(
'$stats["totalConversations"]'),
style: const TextStyle(
fontSize: 16,
fontWeight: FontWeight.w600,
),
),
leading: const Icon(Icons.chat),
),
const Divider(height: 1),
ListTile(
title: Text(l10n.totalMessages),
trailing: Text(
'$stats["totalMessages"]'),
style: const TextStyle(
fontSize: 16,
```

```

 fontWeight: FontWeight.w600,
),
),
leading: const Icon(Icons.message),
),
],
),

// Veri yönetimi
_buildSection(
context,
l10n.sectionDataManagement,
[
ListTile(
title: Text(l10n.clearAllConversationsTitle),
subtitle: Text(l10n.clearAllConversationsSubtitle),
leading: const Icon(Icons.delete_forever, color: Colors.red),
trailing: const Icon(Icons.arrow_forward_ios, size: 16),
onTap: _clearAllData,
),
],
),
),

// Hakkında
_buildSection(
context,
l10n.sectionAbout,
[
ListTile(
title: Text(l10n.versionLabel),
trailing: const Text('1.0.2'),
leading: const Icon(Icons.info),
),
const Divider(height: 1),
ListTile(
title: Text(l10n.developerLabel),
trailing: Text(l10n.developerValue),
leading: const Icon(Icons.person),
),
],
),
),

const SizedBox(height: 32),
],
),
);
}
}
}


```

**E:/SelcukAiAssistant/repo/.github/workflows/backend.yml**

name: Backend CI

on:

push:

    branches: [ "main" ]

pull\_request:

    branches: [ "main" ]

jobs:

backend:

```

runs-on: ubuntu-latest

steps:
 - uses: actions/checkout@v4

 - name: Set up Python 3.12
 uses: actions/setup-python@v5
 with:
 python-version: "3.12"
 cache: "pip"

 - name: Encoding guard (UTF-8/BOM/mojibake)
 run: python tools/encoding_guard.py --root .

 - name: Install backend dependencies
 run: |
 python -m pip install --upgrade pip
 pip install -r backend/requirements.txt
 pip install -r backend/requirements-dev.txt

 - name: Lint with Ruff
 working-directory: backend
 run: |
 ruff check . --select=E9,F63,F7,F82 --target-version=py312
 ruff check .

 - name: Type check with Mypy
 working-directory: backend
 run: mypy .

 - name: Test with Pytest
 working-directory: backend
 run: python -m pytest -q

api-smoke:
 runs-on: windows-latest
 needs: backend

steps:
 - uses: actions/checkout@v4

 - name: Set up Python 3.12
 uses: actions/setup-python@v5
 with:
 python-version: "3.12"
 cache: "pip"

 - name: Install backend dependencies
 run: |
 python -m pip install --upgrade pip
 pip install -r backend/requirements.txt

 - name: Run API smoke test (no-model safe)
 shell: pwsh
 run: |
 $ErrorActionPreference = "Stop"
 $stdoutLog = Join-Path $env:RUNNER_TEMP "backend.log"
 $stderrLog = Join-Path $env:RUNNER_TEMP "backend.err.log"

```

```

$proc = Start-Process -FilePath python `
 -ArgumentList "-m", "uvicorn", "main:app", "--host", "127.0.0.1", "--port", "8000", "--log-`
level", "info" `
 -WorkingDirectory "backend" `
 -RedirectStandardOutput $stdoutLog `
 -RedirectStandardError $stderrLog `
 -PassThru

try {
 $ready = $false
 for ($i = 0; $i -lt 30; $i++) {
 Start-Sleep -Seconds 2
 if ($proc.HasExited) {
 Write-Host "Uvicorn beklenmedik sekilde durdu."
 if (Test-Path $stdoutLog) { Get-Content $stdoutLog -Tail 200 }
 if (Test-Path $stderrLog) { Get-Content $stderrLog -Tail 200 }
 exit 1
 }
 try {
 $health = Invoke-RestMethod -Uri "http://127.0.0.1:8000/health" -TimeoutSec 2
 if ($health.status -eq "ok") {
 $ready = $true
 break
 }
 } catch {
 continue
 }
 }
}

if (-not $ready) {
 Write-Host "Backend belirtilen surede hazır olmadı."
 if (Test-Path $stdoutLog) { Get-Content $stdoutLog -Tail 200 }
 if (Test-Path $stderrLog) { Get-Content $stderrLog -Tail 200 }
 exit 1
}

.\tools\test_api.ps1 -BaseUrl "http://127.0.0.1:8000" -AllowNoModel
}
finally {
 if ($proc -and -not $proc.HasExited) {
 Stop-Process -Id $proc.Id -Force
 }
}

```

## E:/SelcukAiAssistant/repo/.github/workflows/dart.yml

```

name: Flutter

on:
 push:
 branches: ["main"]
 pull_request:
 branches: ["main"]

jobs:
 build:
 runs-on: ubuntu-latest

 steps:
 - uses: actions/checkout@v4

```

```

- name: Set up Python 3.12
 uses: actions/setup-python@v5
 with:
 python-version: "3.12"

- name: Encoding guard (UTF-8/BOM/mojibake)
 run: python tools/encoding_guard.py --root .

- uses: subosito/flutter-action@v2
 with:
 channel: "stable"
 cache: true

- name: Validate ARB JSON
 run: |
 python -c "
 import json
 from pathlib import Path

 errors = []
 for path in Path('lib/l10n').glob('*.*arb'):
 try:
 json.loads(path.read_text(encoding='utf-8'))
 except Exception as exc:
 errors.append(f'{path}: {exc}')

 if errors:
 print('Geçersiz ARB JSON bulundu:')
 print('\n'.join(errors))
 raise SystemExit(1)
 "

```

```

- name: Create .env from example
 run: |
 if [-f .env.example]; then
 cp .env.example .env
 else
 echo "BACKEND_URL=http://localhost:8000" > .env
 fi

- name: Install dependencies
 run: flutter pub get

- name: Analyze project source
 run: flutter analyze

- name: Run tests
 run: flutter test

- name: Build web (optional)
 run: flutter build web --release
 continue-on-error: true

```

## E:/SelcukAiAssistant/repo/docker-compose.yml

```

services:
 backend:
 build:

```

```

context: ./backend
args:
 INSTALL_HF: ${INSTALL_HF:-false}
env_file:
 - ./backend/.env
environment:
 HOST: 0.0.0.0
 PORT: 8000
 OLLAMA_BASE_URL: ${OLLAMA_BASE_URL:-http://ollama:11434}
 ALLOWED_ORIGINS: ${ALLOWED_ORIGINS:-http://localhost,http://127.0.0.1}
 ALLOWED_ORIGINS_STRICT: ${ALLOWED_ORIGINS_STRICT:-false}
ports:
 - "8000:8000"
volumes:
 # HF cache'i harici diske taşımak için HF_CACHE_HOST_DIR ayarlayın (örn. E:/hf-cache).
 - ${HF_CACHE_HOST_DIR:-./hf-cache}:/root/.cache/huggingface
 - ./data:/app/data

nginx:
image: nginx:1.25-alpine
depends_on:
 - backend
ports:
 - "80:80"
volumes:
 - ./build/web:/var/www/selcuk-ai/web:ro
 - ./nginx/selcuk-ai.conf:/etc/nginx/conf.d/default.conf:ro
restart: unless-stopped

ollama:
image: ollama/ollama:latest
profiles: ["ollama"]
ports:
 - "11435:11434"
volumes:
 # Ollama model deposunu harici diske taşımak için OLLAMA_MODELS_HOST_DIR
 # ayarlayın (örn. E:/ollama).
 - ${OLLAMA_MODELS_HOST_DIR:-./ollama-data}:/root/.ollama
environment:
 OLLAMA_HOST: 0.0.0.0
restart: unless-stopped

volumes:
 ollama-data:

```

### **EK-3: Test ve Benchmark Raporları**

**E:/SelcukAiAssistant/repo/docs/reports/TEST\_RAPORU.md**

```

Test ve CI Raporu
Bu rapor, Selçuk AI Akademik Asistan projesinin test ve sürekli entegrasyon (CI) sonuçlarını
akademik formatta özetlemek amacıyla hazırlanmıştır.

Test Ortamı
- CI (GitHub Actions):
 - Arka uç: ubuntu-latest
 - API duman testi: windows-latest
 - Flutter: ubuntu-latest
- Yerel geliştirme (örnek): Windows 10/11, Python 3.12 (önerilen), Flutter Stable
- Sanal ortam: 'backend/.venv' veya '.venv'

Arka Uç Kalitesi
- Testler 'pytest -q' ile çalıştırılmıştır.

```

- Kod kalitesi denetimi `ruff check .` ile yapılmıştır.
- Tip denetimi `mypy .` ile gerçekleştirılmıştır.
- Kritik test dosyaları:
  - `backend/test\_main.py` (API kontrat testleri)
  - `backend/test\_extended.py` (RAG, yeniden deneme ve sağlık senaryoları)
  - `backend/test\_response\_cleaner.py` (metin temizleme)
  - `backend/test\_reasoning\_cleanup.py` (düşünce blokları temizliği)

## Ön Uç Kalitesi

- Statik analiz: `flutter analyze`
- Widget testleri: `flutter test`
- Kritik test dosyası: `test/widget\_test.dart`

## Araçlar ve Betikler

- `tools/encoding\_guard.py`: UTF-8/BOM/mojibake taraması
- `tools/test\_api.ps1`: API duman testi (model yoksa SKIP)
- `tools/smoke\_test.ps1`: geniş kapsamlı duman raporu
- `benchmark/oollama\_quick.py`: Ollama hızlı performans ölçümü
- `benchmark/run.py`: Ayrıntılı model kıyaslaması (Ollama + HF)

## Sonuç Tablosu

| Komut                                                                                                                                            | Amaç | Durum | Sonuç Detayı |
|--------------------------------------------------------------------------------------------------------------------------------------------------|------|-------|--------------|
| `python -m pytest -q`   Arka uç birim testleri   <span style="color: green;">✓</span> Geçti   50 test geçti, 1 DeprecationWarning (FAISS/NumPy)  |      |       |              |
| `ruff check . --select=E9,F63,F7,F82`   Kritik kod hataları   <span style="color: green;">✓</span> Geçti   Kritik hata yok                       |      |       |              |
| `ruff check .`   Kod kalitesi (biçem denetimi)   <span style="color: green;">✓</span> Geçti   Kod stil sorunsuz                                  |      |       |              |
| `mypy .`   Tip denetimi   <span style="color: green;">✓</span> Geçti   18 kaynak dosyada tip hatası yok                                          |      |       |              |
| `flutter analyze`   Flutter statik analiz   <span style="color: blue;">➡</span> CI'da   CI ortamında çalıştırılıyor                              |      |       |              |
| `flutter test`   Flutter widget testleri   <span style="color: blue;">➡</span> CI'da   CI ortamında çalıştırılıyor                               |      |       |              |
| `python tools/encoding_guard.py --root .`   Kodlama denetimi   <span style="color: green;">✓</span> Geçti   UTF-8/BOM ve mojibake taraması temiz |      |       |              |
| `tools/test_api.ps1`   API duman testi   <span style="color: blue;">➡</span> CI'da   Model yoksa SKIP                                            |      |       |              |
| `tools/smoke_test.ps1`   Geniş duman testi   <span style="color: blue;">➡</span> CI'da   Model yoksa SKIP                                        |      |       |              |

## Son Test Çalıştırma Tarihi

\*\*Tarih:\*\* 2026-01-01

\*\*Python Sürümü:\*\* 3.12.3

\*\*Test Ortamı:\*\* Ubuntu 22.04 (CI ortamı)

## Uyarılar

- FAISS ve NumPy uyumluluğu nedeniyle `DeprecationWarning` görülebilir; bu uyarının çalışabilirlik üzerinde etkisi bulunmamaktadır.

## **E:/SelcukAiAssistant/repo/docs/reports/BENCHMARK\_RAPORU.md**

# Kıyaslama Raporu (Ollama)

## Amaç

Yerel Ollama modellerinin Türkçe görevlerde hız/tepki metriklerinin karşılaştırılması yapılmıştır.

## Ortam

- Platform: Windows + Docker Desktop (Ollama konteyneri)
- Ollama URL: <http://localhost:11435>
- Veri seti: benchmark/data/selcuk\_tr.jsonl

## Ayrıntılı Ölçüm (benchmark/run.py)

- max\_new\_tokens: 96
- temperature: 0.2
- max\_samples: 6
- Not: Her model için aynı örnek sayısı kullanıldı.

| Model                                                   | Ort. TTFT (ms) | Ort. belirteç/sn | Ort. çıktı belirteci | Ort. toplam süre (sn) |
|---------------------------------------------------------|----------------|------------------|----------------------|-----------------------|
| ollama:aya:8b   13392.93   3.33   36.8   18.786         |                |                  |                      |                       |
| ollama:deepseek-r1:8b   22835.77   4.68   96.0   22.836 |                |                  |                      |                       |
| ollama:gemma2:2b   4853.82   9.56   64.3   8.756        |                |                  |                      |                       |
| ollama:llama3.1   12671.78   3.1   32.2   17.411        |                |                  |                      |                       |
| ollama:llama3.2:3b   6197.3   7.36   30.3   7.909       |                |                  |                      |                       |

```

ollama:mistral	12323.86	4.04	66.7	21.916
ollama:phi3:mini	6772.77	8.26	86.8	13.155
ollama:qwen2.5:7b	11892.11	3.98	40.2	17.098
ollama:selcuk_ai_assistant	10186.69	3.49	34.0	15.138
ollama:turkcell-lm-7b	10126.57	4.1	33.3	14.166
Ek Ölçüm (benchmark/run.py, 12 örnek)
- max_new_tokens: 96
- temperature: 0.2
- max_samples: 12
- Koşum: 2025-12-29 (llama3.2:3b)
| Model | Ort. TTFT (ms) | Ort. belirteç/sn | Ort. çıktı belirteci | Ort. toplam süre (sn) |
| --- | --- | --- | --- | --- |
| ollama:llama3.2:3b | 5180.24 | 5.41 | 34.7 | 8.643 |
Hızlı Ölçüm (benchmark/ollama_quick.py)
- max_new_tokens: 48
- temperature: 0.2
- max_samples: 3
| Model | Koşum | Ort. TTFT (ms) | Ort. belirteç/sn | Ort. çıktı belirteci | Ort. toplam süre (sn) |
| --- | --- | --- | --- | --- |
| aya:8b | selcuk_tr_ollama_quick6 | 5137.96 | 3.12 | 34.7 | 10.814 |
| deepseek-r1:8b | selcuk_tr_qwen_deepseek | 20609.53 | 2.84 | 48.0 | 20.61 |
| gemma2:2b | selcuk_tr_ollama_quick6 | 10235.34 | 5.95 | 40.3 | 13.276 |
| llama3.2:3b | selcuk_tr_ollama_quick6 | 2451.41 | 7.19 | 37.3 | 5.192 |
| phi3:mini | selcuk_tr_ollama_quick6 | 13269.4 | 4.87 | 48.0 | 17.067 |
| qwen2.5:7b | selcuk_tr_qwen_deepseek | 22049.38 | 3.14 | 39.7 | 27.519 |
| selcuk_ai_assistant | selcuk_tr_ollama_quick6 | 4018.03 | 3.97 | 41.0 | 10.357 |
| turkcell-lm-7b | selcuk_tr_ollama_quick6 | 4523.9 | 3.68 | 42.3 | 11.516 |
Kısa Yorum
- llama3.2:3b hız odaklı mod için güclü bir adaydır (düşük TTFT ve yüksek belirteç/sn).
- 12 örnek koşumda llama3.2:3b için ort. TTFT 5.18 sn ve 5.41 belirteç/sn görüldü.
- selcuk_ai_assistant ve turkcell-lm-7b benzer hız profiline sahiptir; kalite testleri ile birlikte seçilmelidir.
- qwen2.5:7b ve deepseek-r1:8b daha ağırdır; kalite odaklı senaryolarda değerlendirilmelidir.
- gemma2:2b ve phi3:mini düşük kaynak modları için uygundur.
Sonuç Dosyaları
- benchmark/outputs/bench_ollama_*/summary.csv
- benchmark/outputs/20251229_204933/summary.csv
- benchmark/outputs(selcuk_tr_ollama_quick6/summary.csv
- benchmark/outputs(selcuk_tr_qwen_deepseek/summary.csv
```

## E:/SelcukAiAssistant/repo/docs/reports/VERI\_KAYNAKLARI.md

# Veri Kaynakları ve Toplama Özeti  
Bu dosyada RAG veri toplama süreci için kullanılan resmi kaynaklar ve oluşan veri seti özetlenmiştir.

- ## 1) Kaynak Alanı
  - İzinli alan adı: `selcuk.edu.tr`
- ## 2) Başlangıç URL'leri
  - <https://www.selcuk.edu.tr/>
  - <https://www.selcuk.edu.tr/ogrenci>
  - <https://www.selcuk.edu.tr/akademik>
  - <https://www.selcuk.edu.tr/idari>
  - <https://www.selcuk.edu.tr/duyurular>
  - <https://www.selcuk.edu.tr/dokumanlar>
  - <https://www.selcuk.edu.tr/haberler>
- ## 3) Site Haritası
  - <https://www.selcuk.edu.tr/sitemap.xml>
- ## 4) Toplanan Veri İstatistikleri
  - Ham HTML sayfa sayısı: `400`
  - Temizlenmiş doküman sayısı: `400`
  - FAISS indeksine eklenen parça sayısı: `1481`
  - Kullanılan gömme modeli: `sentence-transformers paraphrase-multilingual-MiniLM-L12-v2`

```
5) Depolama Konumları
- Ham içerik: 'data/raw_web/html'
- Temizlenmiş içerik: 'data/processed_web/docs'
- FAISS indeks: 'data/rag'
6) Notlar
- Toplama sadece 'selcuk.edu.tr' alan adları ile sınırlanmıştır.
- SSL doğrulama hataları gözlenirse 'tools/collect_sources.py' içinde '--insecure' seçeneği kullanılmalıdır.
```

**ÖZGEÇMİŞ ..... 10**

**Doğukan BALAMAN (203311066)**

Kişisel bilgiler, eğitim ve iletişim alanları öğrenci tarafından doldurulacaktır.

**Ali YILDIRIM (203311008)**

Kişisel bilgiler, eğitim ve iletişim alanları öğrenci tarafından doldurulacaktır.

## **SİMGELER VE KISALTMALAR**

### **Simgeler**

LLM: Large Language Model (Büyük Dil Modeli)

RAG: Retrieval-Augmented Generation (Geri Getirim Destekli Üretim)

SSE: Server-Sent Events

API: Application Programming Interface

FAISS: Facebook AI Similarity Search

HF: HuggingFace

### **Kısaltmalar**

## **1. GİRİŞ**

Giriş bölümünü yazmaya buradan başlayınız.

### **1.1. Birinci Bölüm İkinci Derece Başlık**

#### **1.1.1. Birinci bölüm üçüncü derece başlık**

##### **1.1.1.1. Birinci bölüm dördüncü derece başlık**

## **2. KAYNAK ARAŞTIRMASI**

Kaynak araştırması bölümünü yazmaya buradan başlayınız.

### **2.1. İkinci Bölüm İkinci Derece Başlık**

#### **2.1.1. İkinci bölüm üçüncü derece başlık**

##### **2.1.1.1. İkinci bölüm dördüncü derece başlık**

### 3. MATERİYAL VE YÖNTEM

Materiyal ve metot bölümünü yazmaya buradan başlayınız. Materyal ve Metot başlığı kullanmıyorsanız diğer kullandığınız başlığı buraya yazınız.

#### 3.1. Üçüncü Bölüm İkinci Derece Başlık

##### 3.1.1. Üçüncü bölüm üçüncü derece başlık

###### 3.1.1.1. Üçüncü bölüm dördüncü derece başlık

Çizelge öncesinde 1.5 satır aralıklı bir satır boşluk

1.0 satır aralıklı bir satır boşluk

Çizelge 3.1. Atomu oluşturan taneciklerin kütleleri ve yükleri

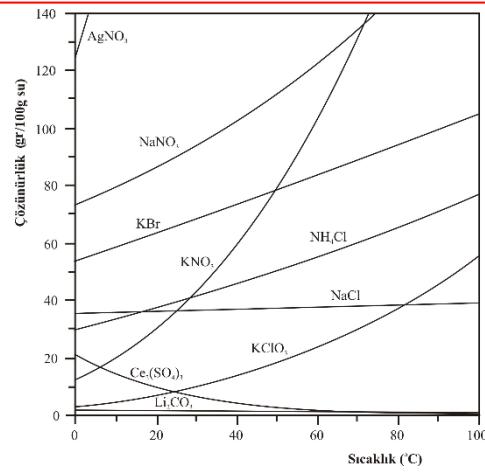
| Madde                                  | $F_2$ | $Cl_2$ | $Br_2$ | $I_2$ |
|----------------------------------------|-------|--------|--------|-------|
| Atom çapı ( $\text{\AA}$ )             | 0,57  | 0,97   | 1,12   | 1,32  |
| Kaynama noktası ( $^{\circ}\text{C}$ ) | -188  | -34    | 59     | 184   |
| Standart şartlardaki fiziksel hali     | gaz   | gaz    | sıvı   | katı  |

Dipnot (varsa)

10 punto  
1.0 satır aralıklı

Çizelge sonrasında 1.5 satır aralıklı bir satır boşluk

Şekil öncesinde 1.5 satır aralıklı bir satır boşluk



1.0 satır  
aralıklı bir  
satır boşluk

Şekil 1.1. Bazı tuzların 1.0 atm basınçta sudaki çözünürlüklerinin sıcaklıkla değişimi

10 punto  
1.0 satır  
aralıklı

Şekil açıklamasından sonra sonrasında 1.5 satır aralıklı bir satır boşluk

## **4. ARAŞTIRMA SONUÇLARI VE TARTIŞMA**

Araştırma sonuçları ve tartışma bölümünü yazmaya buradan başlayınız.

### **4.1. Dördüncü Bölüm İkinci Derece Başlık**

#### **4.1.1. Dördüncü bölüm üçüncü derece başlık**

##### **4.1.1.1. Dördüncü bölüm dördüncü derece başlık**

## **5. SONUÇLAR VE ÖNERİLER**

### **5.1 Sonuçlar**

Sonuçlar bölümünü yazmaya buradan başlayınız.

### **5.2 Öneriler**

Öneriler bölümünü yazmaya buradan başlayınız.

## KAYNAKLAR

- Anonim, 2006, Tarım istatistikleri özeti, DİE Yayınları, No;12, Ankara, 22-23.
- Anonymous, 1989, Farm accountancy data network, an A-Z of methodology” Commission Report of the EC, Brussels, 16-19.
- Corliss, R., 1993, *Pacific Overtures Times*, 142 (11), 68-70.
- Dasgupta, D., 1998, Artificial immune systems and their applications, *Springer-Verlag*, Berlin - Heidelberg, 45-52.
- De Castro, L. N. and Von Zuben, F. J., 2000, Artificial immune systems: Part I- Basic theory and applications, *DCA-RT 02/00, Brasil*, 23-28.
- Güneş, S. ve Polat, K., 2009, Elektrokardiyogram (EKG) aritmi teşhisinde en az kareli destek vektör makinaları kullanımına dayalı medikal teşhis destek sistemi, *13. Biyomedikal Mühendisliği Ulusal Toplantısı, BİYOMUT-2009*, İstanbul, 170-173.
- Holland, M., 2002, *Guide to citing Internet sources* [online], Poole, Bournemouth University, [http://www.bournemouth.ac.uk/library/using/guide\\_to\\_citing\\_internet\\_sourc.html](http://www.bournemouth.ac.uk/library/using/guide_to_citing_internet_sourc.html) [Ziyaret Tarihi: 4 Kasım 2002].
- Mason, J., 1832, Map of the countries lying between Spain and India, 1:8.000.000, London: Ordnance Survey.
- Özgören, M., 2006, Flow Structure in the downstream of square and circular cylinders, *Flow Measurement and Instrumentation*, 17 (4), 225-235.

[Diğer örneklenmeyen kaynakları benzer şekilde yazınız.](#)

**EKLER****EK-1**

| Kontrol Edilecek Hususlar                                                                                                                                     | Evet | Hayır |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------|------|-------|
| Sayfa yapısı uygun mu?                                                                                                                                        |      |       |
| Şekil ve çizelge başlık ve içerikleri uygun mu?                                                                                                               |      |       |
| Denklem yazımları uygun mu?                                                                                                                                   |      |       |
| İç kapak, onay sayfası, Proje bildirimi, özet, abstract, önsöz ve/veya teşekkür uygun yazıldı mı?                                                             |      |       |
| Proje yazımı; Giriş, Kaynak Araştırması, Materyal ve Yöntem (veya Teorik Esaslar), Araştırma Bulguları ve Tartışma, Sonuçlar ve Öneriler sıralamasında mıdır? |      |       |
| Kaynaklar soyadı sırasına göre verildi mi?                                                                                                                    |      |       |
| Kaynaklarda verilen her bir yayına proje içerisinde atıfta bulunuldu mu?                                                                                      |      |       |
| Kaynaklar açıklanan yazım kuralına uygun olarak yazıldı mı?                                                                                                   |      |       |
| Proje içerisinde kullanılan şekil ve çizelgelerde kullanılan ifadeler Türkçe'ye çevrilmiş mi? (Latince ve Özel kelimeler hariçtir)                            |      |       |
| Projenin içindekiler kısmı, proje içerisinde verilen başlıklara uygun hazırlanmış mı?                                                                         |      |       |

Yukarıdaki verilen cevapların doğruluğunu kabul ediyorum.

Unvanı Adı SOYADI

İmza

**Öğrenci :** .....

**Danışman :** .....

\*Bitirme projesi/araştırma projeleri Teknoloji Fakültesi proje yazım kurallarına uygun olarak hazırlanmalıdır. Projeler teslim edilmeden önce yukarıdaki kontrol listesi öğrenci ve danışman tarafından imzalanmalıdır. Bu sayfa tez teslimi esnasında en üst sayfa olarak verilmelidir.

\*Proje ilk savunmaya sunulduğunda spiral cilt veya clip dosya formunda teslim edilmelidir.

**EK-2** Uygun bir başlık buraya yazılmalıdır.

**ÖZGEÇMİŞ****KİŞİSEL BİLGİLER**

**Adı Soyadı** :  
**Uyruğu** :  
**Doğum Yeri ve Tarihi** :  
**Telefon** :  
**Faks** :  
**E-mail** :

**EĞİTİM**

| <b>Derece</b> | <b>Adı, İlçe, İl</b> | <b>Bitirme Yılı</b> |
|---------------|----------------------|---------------------|
| Lise          | :                    |                     |
| Üniversite    | :                    |                     |
| Yüksek Lisans | :                    |                     |
| Doktora       | :                    |                     |

**İŞ DENEYİMLERİ**

| <b>Yıl</b> | <b>Kurum</b> | <b>Görevi</b> |
|------------|--------------|---------------|
|            |              |               |

**UZMANLIK ALANI****YABANCI DİLLER****BELİRTMEK İSTEĞİNİZ DİĞER ÖZELLİKLER****YAYINLAR**