dgroove
Tecnologia emozionale.

ICT Training Center

Il tuo partner per la Formazione e la Trasformazione digitale della tua azienda

Spring AI

Generative Artificial Intelligence con Java

Simone Scannapieco

Corso avanzato per Venis S.p.A, Venezia, Italia

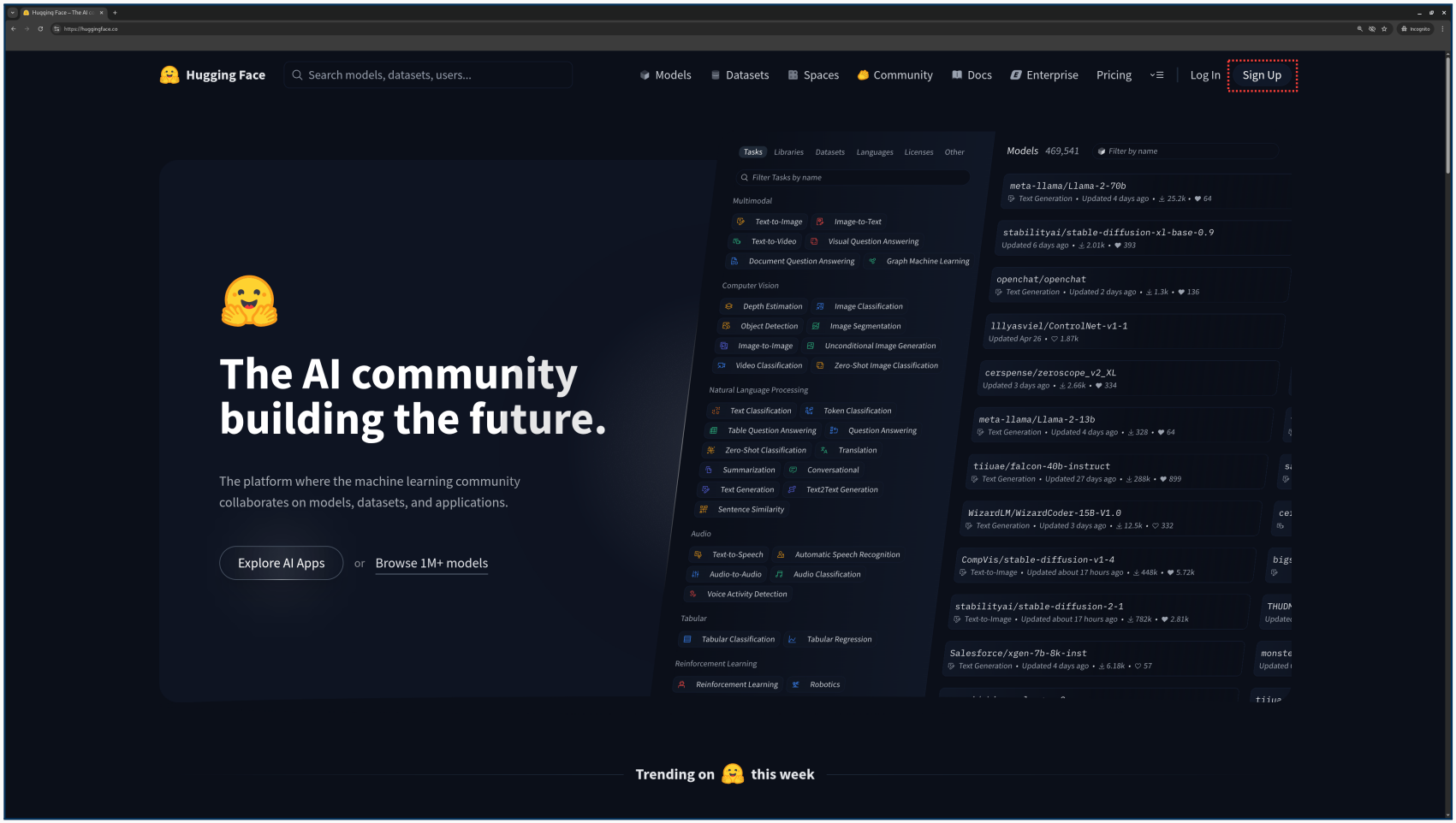
Novembre 2025

🔧 Retrieval Augmented Generation

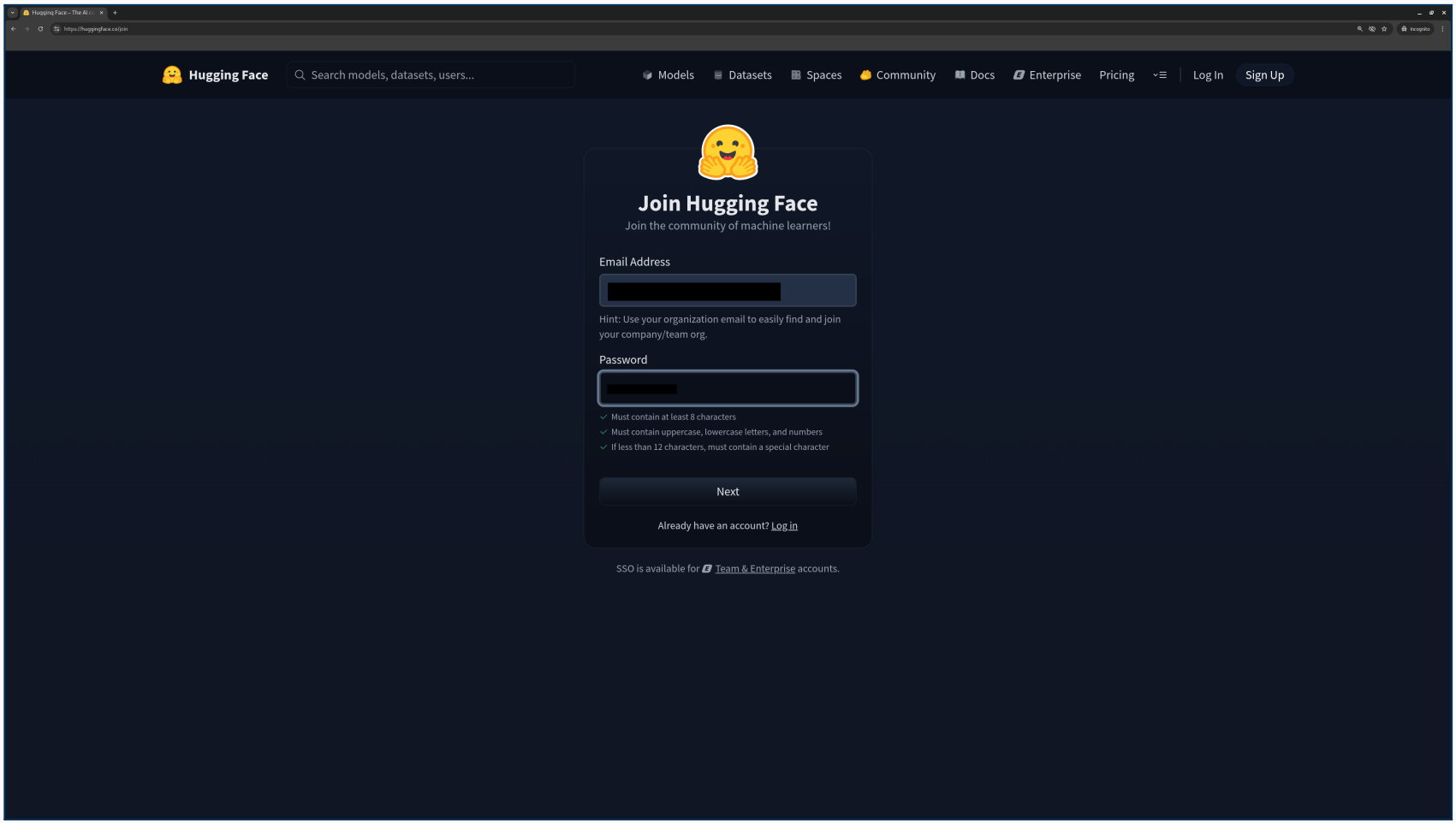
Parte 1 - Approccio text to vector store

- *Chatbot* Ollama-CV e Gemini-Venis
- Approccio *naive* (informazione testuale estratta tramite GENAI-*assisted web scraping/summarization*)
  1. Iscrizione al portale HuggingFace e creazione *access token*
  2. Verifica e modifica variabili di ambiente
  3. Creazione configurazione ambiente Docker Qdrant
  4. Modifica dipendenze di progetto
  5. Modifica configurazione e profilo applicativo
  6. Creazione *prompt templates* per strategia RAG
  7. Configurazione *vector store* per Ollama e Gemini *embeddings*
  8. Creazione *component* per popolamento *vector store* (dati Venis e CV)
  9. Creazione interfaccia e implementazione del servizio
  10. Modifica controllore MVC
  11. *Test* delle funzionalità con Postman/Insomnia

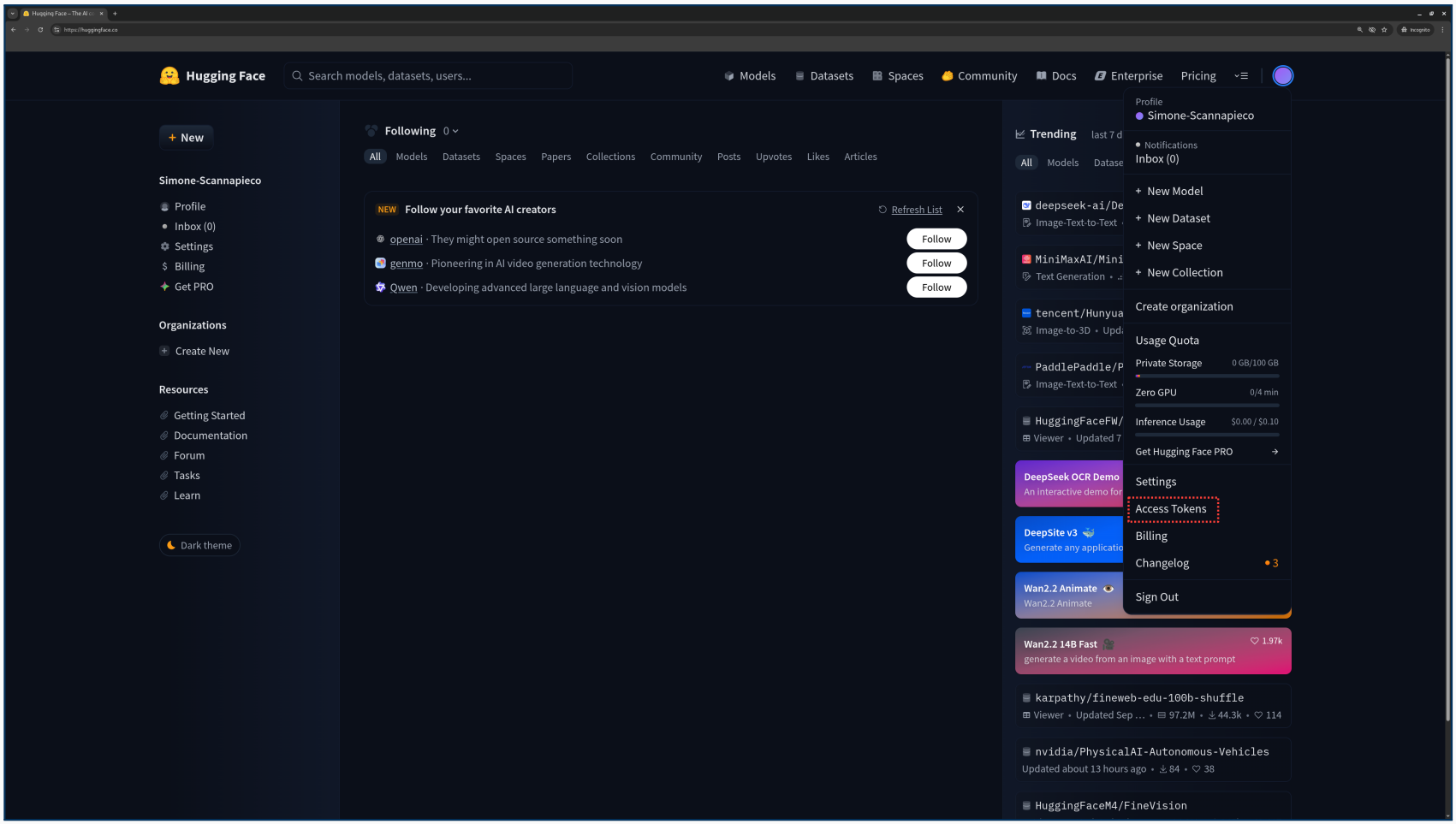
1. Accedere al portale `https://huggingface.co/` e cliccare il pulsante `Sign up`



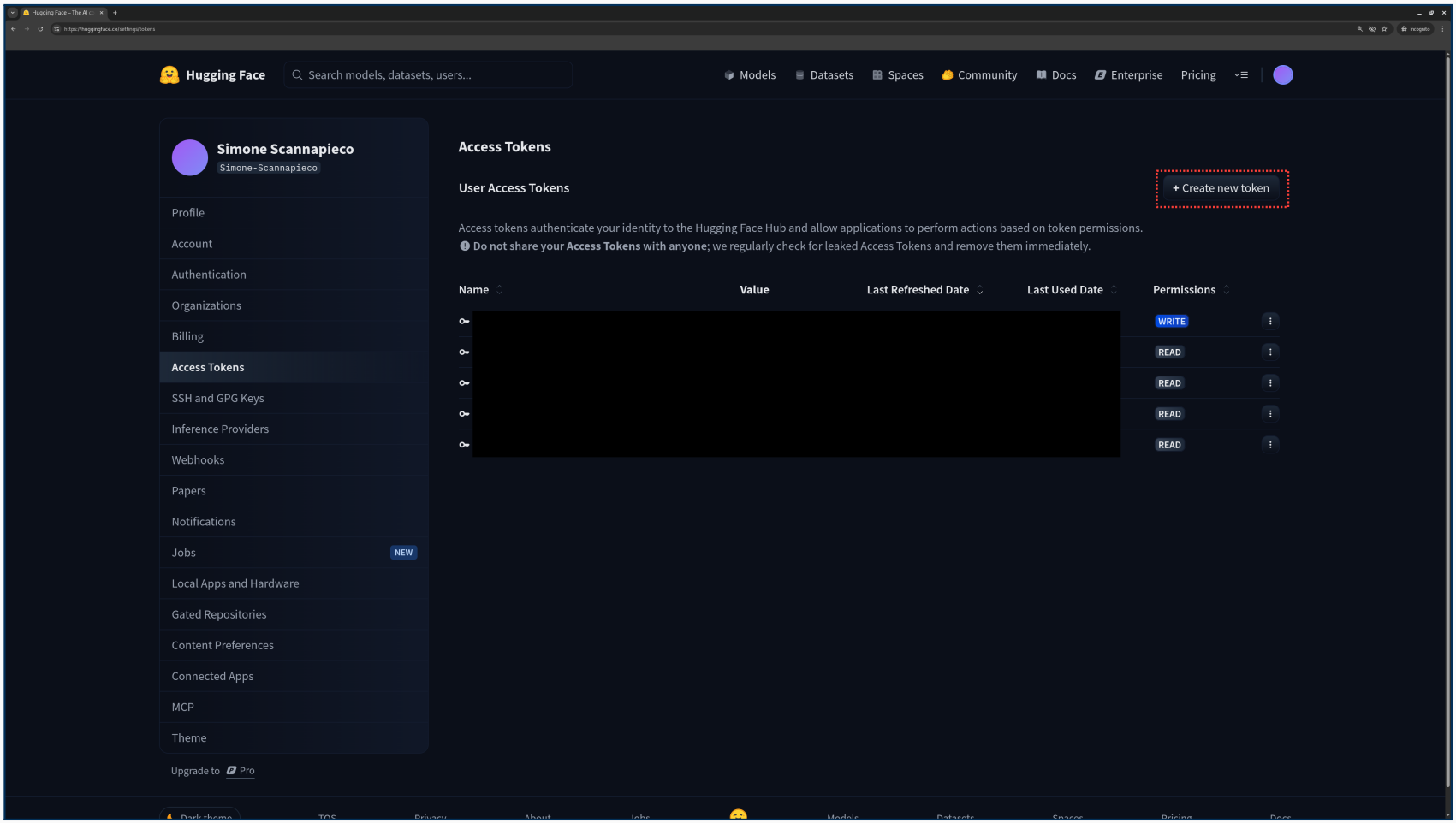
2 Creare una nuova utenza



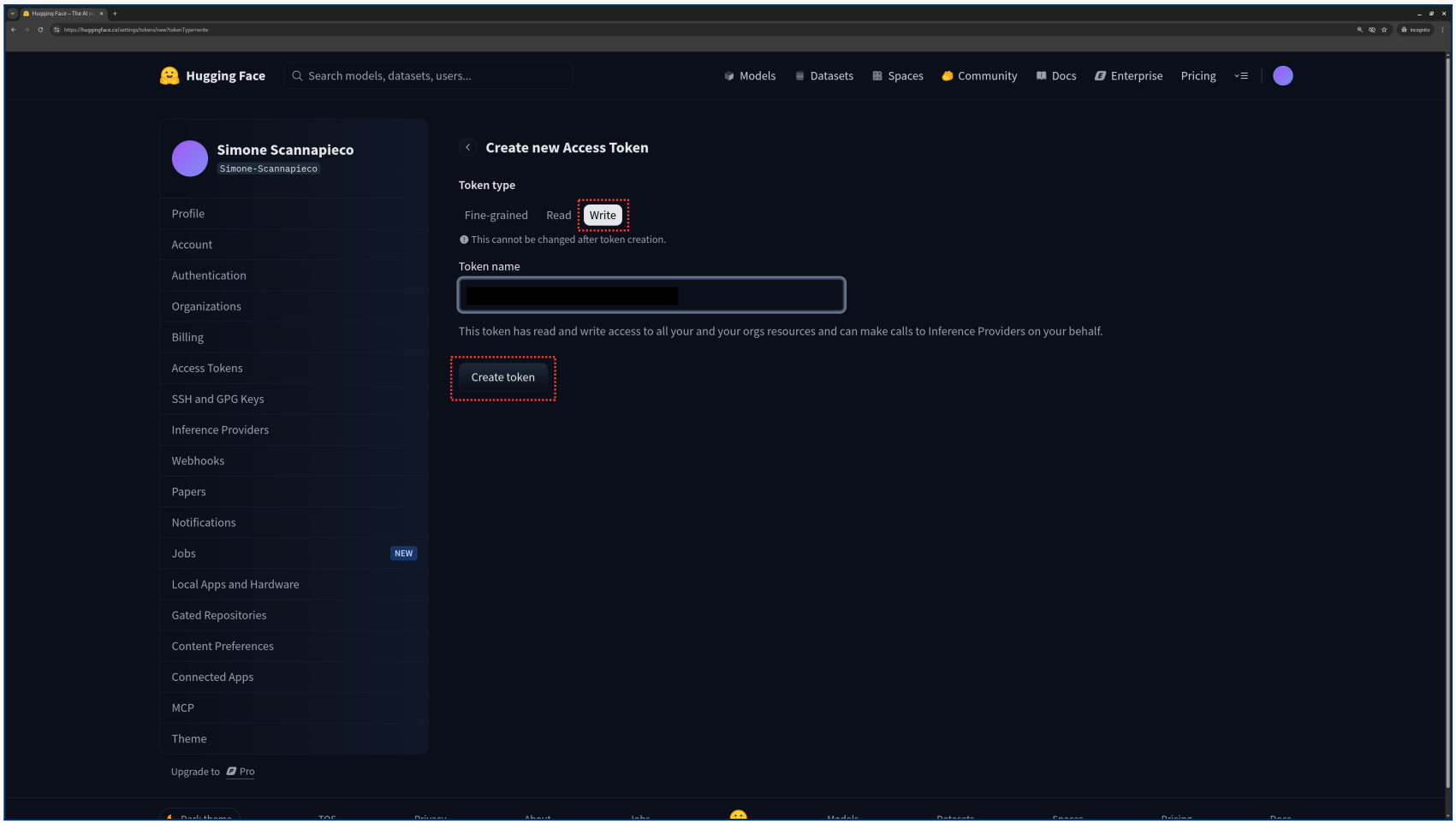
3 Accedere al portale e selezionare `Access Tokens` nel menu in alto a destra



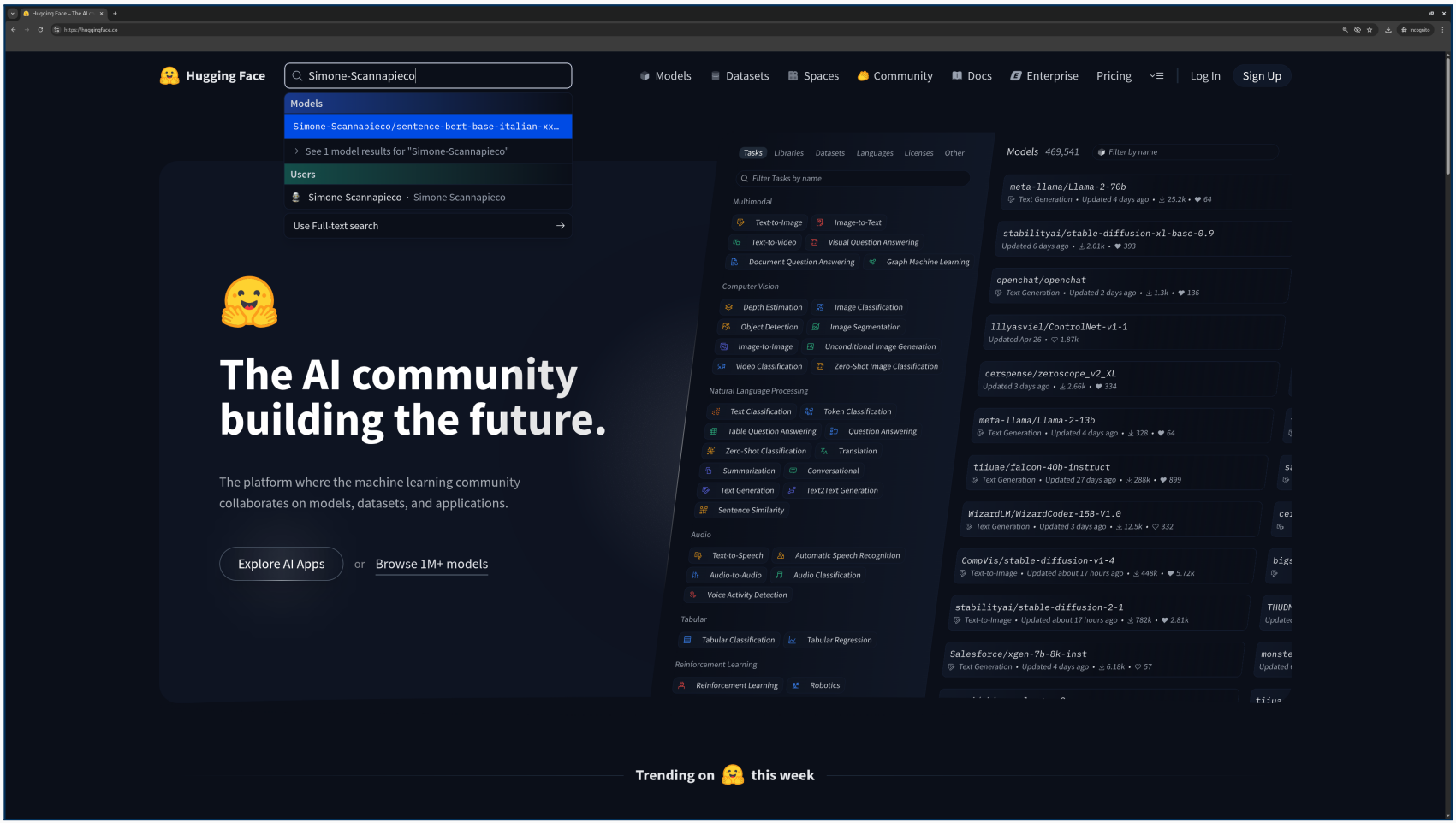
4 Premere sul pulsante `Create new token`



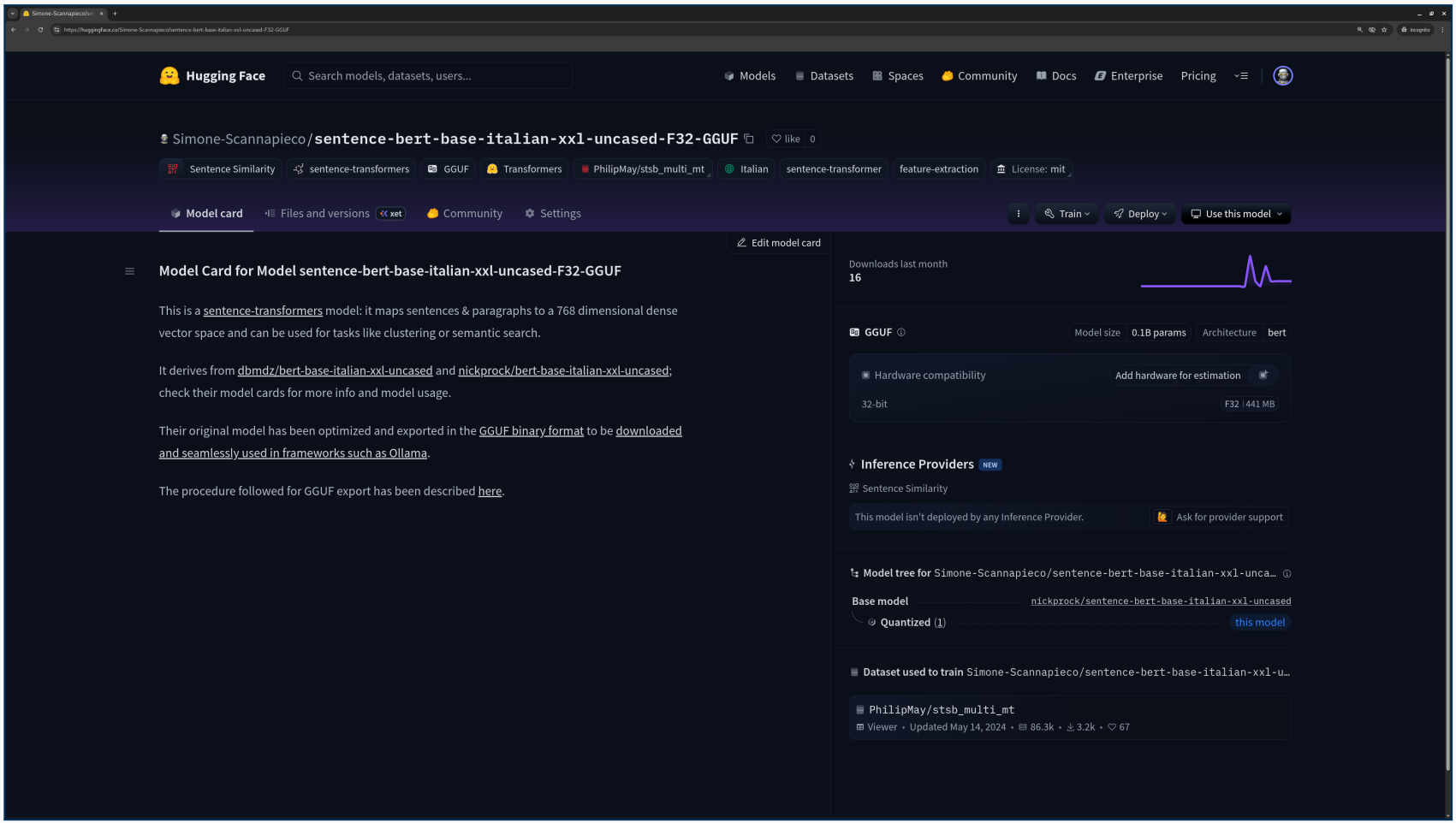
5 Selezionare *token* di tipo `Write`, denominare il *token* e premere `Create token`



6 Cercare il modello di *embedding*, leggendone la *card*



6 Cercare il modello di *embedding*, leggendone la *card*



File `launch.json`

```
{
    // Use IntelliSense to learn about possible attributes.
    // Hover to view descriptions of existing attributes.
    // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
    "version": "0.2.0",
    "configurations": [
        {
            "type": "java",
            "name": "Launch Current File",
            "request": "launch",
            "mainClass": "${file}"
        },
        {
            "type": "java",
            "name": "DemoApplication",
            "request": "launch",
            "mainClass": "it.venis.ai.spring.demo.DemoApplication",
            "projectName": "demo",
            "env": {
                "GOOGLE_AI_API_KEY": "...",
                "HUGGING_FACE_HUB_TOKEN": "..."
            }
        }
    ]
}
```

File `settings.json`

```
{
    "java.compile.nullAnalysis.mode": "disabled",
    "java.configuration.updateBuildConfiguration": "interactive",
    "java.test.config": {
        "env": {
            "GOOGLE_AI_API_KEY": "...",
            "HUGGING_FACE_HUB_TOKEN": "..."
        }
    }
}
```

File `docker-compose.yml`

```yaml
services:
  ...
  spring-ai-vector-store:
    image: qdrant/qdrant:${QDRANT_VERSION:-latest}
    hostname: spring-ai-vector-store
    container_name: spring_ai_vector_store
    ports:
      - ${QDRANT_HTTP_PORT:-6333}:6333
      - ${QDRANT_GRCP_PORT:-6334}:6334
    volumes:
      - spring_ai_vector_store:/qdrant/storage
    restart: unless-stopped

volumes:
  ...
  spring_ai_vector_store:
    name: spring_ai_vector_store
```

File `spring-ai.env`

```
...
# qdrant configuration
QDRANT_VERSION=v1.13.0                                  # default: latest
QDRANT_HTTP_PORT=6333                                     # default: 6333
QDRANT_GRCP_PORT=6334                                     # default: 6334
```

Dipendenze di sistema aggiuntive

```
...
<dependency>
        <groupId>org.springframework.ai</groupId>
        <artifactId>spring-ai-rag</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.ai</groupId>
        <artifactId>spring-ai-advisors-vector-store</artifactId>
</dependency>
<dependency>
        <groupId>org.springframework.ai</groupId>
        <artifactId>spring-ai-starter-vector-store-qdrant</artifactId>
</dependency>
...
```

Configurazione applicativo

```yaml
spring:
    ...
    profiles:
        active: rag-text-to-vector-store
    autoconfigure:
        exclude:
        - org.springframework.ai.vectorstore.qdrant.autoconfigure.QdrantVectorStoreAutoConfiguration
        # We must disable Vector Store auto-configuration because of two different EmbeddingModel beans
        # (OpenAI-Gemini and Ollama). The goal is to have two different vector collections, one for each
        # family of LLM.
    ai:
        ollama:
            ...
            embedding:
                model: hf.co/Simone-Scannapieco/sentence-bert-base-italian-xxl-uncased-F32-GGUF
        openai:
            ...
            embedding:
                options:
                    model: gemini-embedding-001
        vectorstore:
            qdrant:
                initialize-schema: true
                host: 172.17.0.1
                port: 6334
    ...
```

File application-rag-text-to-vector-store.yml

```yaml
demo:
    rag:
        prompt:
            system:
                ita: classpath:templates/get-rag-data-system-ita-prompt.st
                eng: classpath:templates/get-rag-data-system-eng-prompt.st
            user:
                ita:
                eng:
        vectorstore:
            qdrant:
                collection-name:
                    ollama: vector_store_ttvs_ollama
                    gemini: vector_store_ttvs_gemini
```

File `erase_llm_volumes.sh`

```
#!/bin/bash

volume_name=spring_ai_llm

docker volume rm $volume_name
```

File `erase_vector_store_volumes.sh`

```
#!/bin/bash

volume_name=spring_ai_vector_store

docker volume rm $volume_name
```

File templates/get-rag-data-system-ita-prompt.st

```
Sei un assistente AI in grado di rispondere alle domande dell'utente solo in base
al contesto fornito dalla sezione DOCUMENTI.
Se la risposta non è presente nella sezione DOCUMENTI, informa l'utente di non
sapere la risposta.

DOCUMENTI: <documenti>
```

File templates/get-rag-data-system-eng-prompt.st

```
You are an helpful assistant, answering questions based on the given context in
the DOCUMENTS section and no prior knowledge.
If the answer is not in the DOCUMENTS section, then reply that you cannot answer
to the question.

DOCUMENTS: <documenti>
```

Configurazione Vector Store - I

```
package it.venis.ai.spring.demo.config;

...

@Configuration
public class RAGConfig {

    @Value("${spring.ai.vectorstore.qdrant.host:localhost}")
    private String qdrantHost;
    @Value("${spring.ai.vectorstore.qdrant.port:6334}")
    private Integer qdrantPort;
    @Value("${spring.ai.vectorstore.qdrant.use-tls:false}")
    private Boolean useTls;

    @Bean
    public QdrantClient qdrantClient() {
        QdrantGrpcClient.Builder grpcClientBuilder = QdrantGrpcClient.newBuilder(
                qdrantHost, qdrantPort, useTls);
        return new QdrantClient(grpcClientBuilder.build());
    }

    @Value("${demo.rag.vectorstore.qdrant.collection-name.gemini:vector_store_gemini}")
    private String qdrantCollectionNameGemini;
    @Value("${demo.rag.vectorstore.qdrant.collection-name.ollama:vector_store_ollama}")
    private String qdrantCollectionNameOllama;
    @Value("${spring.ai.vectorstore.qdrant.initialize-schema:false}")
    private Boolean qdrantInitializeSchema;

    ...
```

Configurazione Vector Store - II

```
    ...
    @Bean
    public VectorStore geminiVectorStore(QdrantClient qdrantClient, OpenAiEmbeddingModel geminiEmbeddingModel) {
        return QdrantVectorStore.builder(qdrantClient, geminiEmbeddingModel)
                .collectionName(qdrantCollectionNameGemini)
                .initializeSchema(qdrantInitializeSchema)
                .build();
    }

    @Bean
    public VectorStore ollamaVectorStore(QdrantClient qdrantClient, OllamaEmbeddingModel ollamaEmbeddingModel) {
        return QdrantVectorStore.builder(qdrantClient, ollamaEmbeddingModel)
                .collectionName(qdrantCollectionNameOllama)
                .initializeSchema(qdrantInitializeSchema)
                .build();
    }

}
```

Componente popolamento Qdrant - I

```
package it.venis.ai.spring.demo.rag;

...

@Component
@Profile("rag-text-to-vector-store")
public class TextDataLoader {

    private final VectorStore geminiVectorStore;
    private final VectorStore ollamaVectorStore;

    public TextDataLoader(@Qualifier("geminiVectorStore") VectorStore geminiVectorStore,
            @Qualifier("ollamaVectorStore") VectorStore ollamaVectorStore) {
        this.geminiVectorStore = geminiVectorStore;
        this.ollamaVectorStore = ollamaVectorStore;
    }

    @PostConstruct
    public void loadVenisInfoIntoVectorStore() {
        List<String> venisInfo = List.of(...);
        SearchRequest searchRequest = SearchRequest.builder()
                .query("Check")
                .similarityThresholdAll()
                .build();
        List<Document> similarDocs = geminiVectorStore.similaritySearch(searchRequest);
        if (similarDocs.size() == 0) {
            List<Document> documents =
            venisInfo.stream().map(Document::new).collect(Collectors.toList());
            this.geminiVectorStore.add(documents);
        }

    }
    ...
```

Componente popolamento Qdrant - II

```
    ...
    @PostConstruct
    public void loadSSCVInfoIntoVectorStore() {
        List<String> ssCVInfo = List.of(...);
        SearchRequest searchRequest = SearchRequest.builder()
                .query("Check")
                .similarityThresholdAll()
                .build();
        List<Document> similarDocs = ollamaVectorStore.similaritySearch(searchRequest);
        if (similarDocs.size() == 0) {
            List<Document> documents = ssCVInfo.stream().map(Document::new).collect(Collectors.toList());
            this.ollamaVectorStore.add(documents);
        }

    }

}
```

Interfaccia servizio

```
package it.venis.ai.spring.demo.services;

import it.venis.ai.spring.demo.model.Answer;
import it.venis.ai.spring.demo.model.QuestionRequest;

public interface RAGService {

    public Answer getGeminiRAGAnswer(QuestionRequest request);

    public Answer getOllamaRAGAnswer(QuestionRequest request);

}
```

Implementazione servizio - I

```
package it.venis.ai.spring.demo.services;

...

@Service
@Configuration
public class RAGServiceImpl implements RAGService {

    private final ChatClient geminiChatClient;
    private final ChatClient ollamaChatClient;
    private final ChatClient ollamaMemoryChatClient;
    private VectorStore geminiVectorStore;
    private VectorStore ollamaVectorStore;

    public RAGServiceImpl(
            @Qualifier("geminiChatClient") ChatClient geminiChatClient,
            @Qualifier("ollamaChatClient") ChatClient ollamaChatClient,
            @Qualifier("ollamaMemoryChatClient") ChatClient ollamaMemoryChatClient,
            @Qualifier("geminiVectorStore") VectorStore geminiVectorStore,
            @Qualifier("ollamaVectorStore") VectorStore ollamaVectorStore) {

        this.geminiChatClient = geminiChatClient;
        this.ollamaChatClient = ollamaChatClient;
        this.ollamaMemoryChatClient = ollamaMemoryChatClient;
        this.geminiVectorStore = geminiVectorStore;
        this.ollamaVectorStore = ollamaVectorStore;

    }

    ...
```

Implementazione servizio - II

```
...
@Value("${demo.rag.prompt.system.eng}")
private Resource ragDataSystemEngPrompt;

@Override
public Answer getGeminiRAGAnswer(QuestionRequest request) {

    SearchRequest searchRequest = SearchRequest.builder()
            .query(request.body().question())
            .topK(4)
            .similarityThreshold(.2)
            .build();

    List<Document> similarDocs = geminiVectorStore.similaritySearch(searchRequest);

    String similarDocsString = similarDocs.stream()
            .map(Document::getText)
            .collect(Collectors.joining(System.lineSeparator()));

    return new Answer(this.geminiChatClient.prompt()
            // .advisors(advisorSpec -> advisorSpec.param(ChatMemory.CONVERSATION_ID,
            // request.username()))
            .system(s -> s.text(this.ragDataSystemEngPrompt)
                    .params(Map.of("documenti", similarDocsString)))
            .user(request.body().question())
            .templateRenderer(StTemplateRenderer.builder().startDelimiterToken('<')
                    .endDelimiterToken('>')
                    .build())
            .call()
            .content());
}

...
```

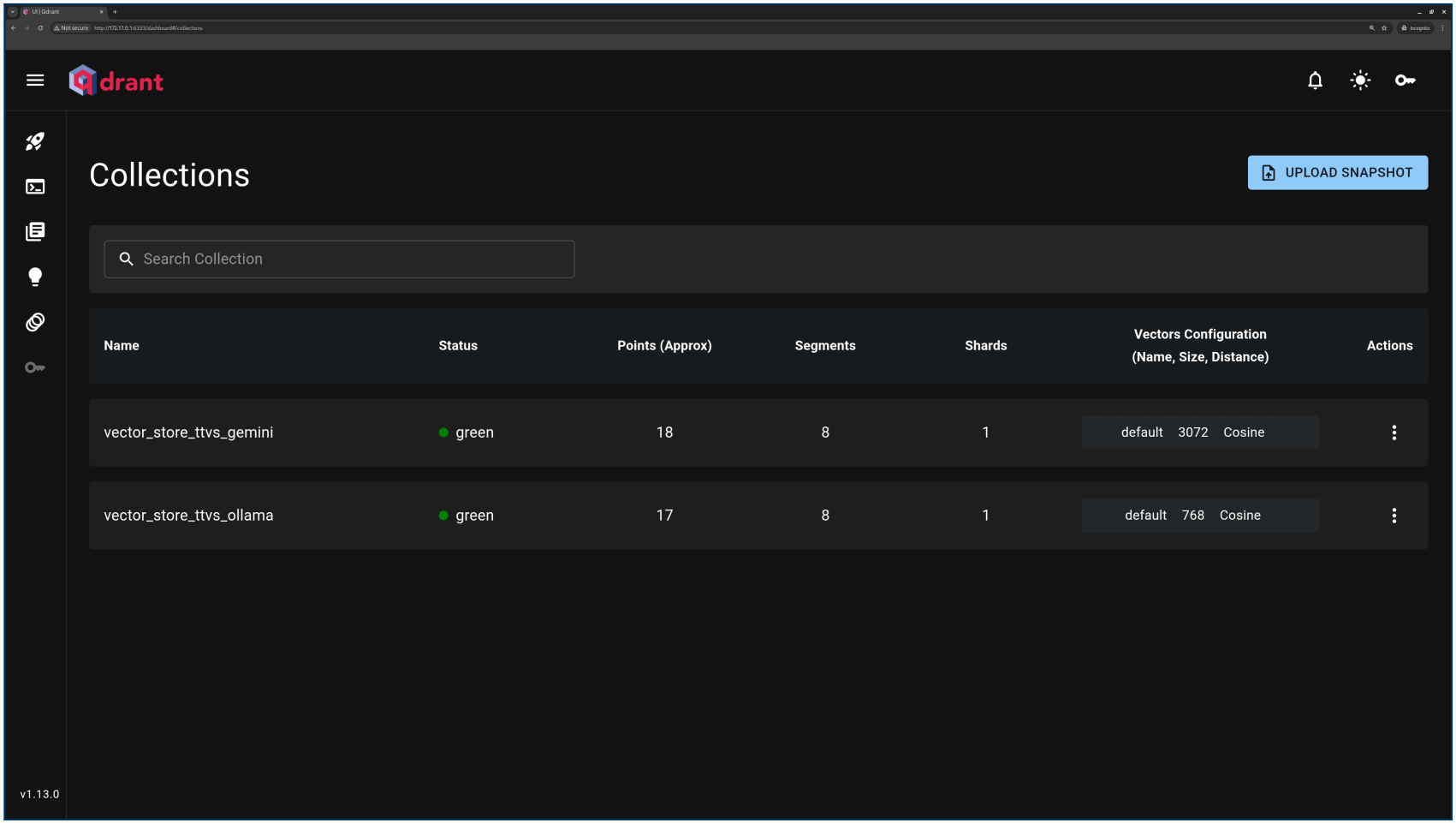
Implementazione servizio - III

```
    ...
    @Value("${demo.rag.prompt.system.ita}")
    private Resource ragDataSystemItaPrompt;

    @Override
    public Answer getOllamaRAGAnswer(QuestionRequest request) {

        SearchRequest searchRequest = SearchRequest.builder()
                .query(request.body().question())
                .topK(4)
                .similarityThreshold(.3)
                .build();

        List<Document> similarDocs = ollamaVectorStore.similaritySearch(searchRequest);

        String similarDocsString = similarDocs.stream()
                .map(Document::getText)
                .collect(Collectors.joining(System.lineSeparator()));

        return new Answer(this.ollamaMemoryChatClient.prompt()
                .advisors(advisorSpec -> advisorSpec.param(ChatMemory.CONVERSATION_ID, request.username()))
                .system(s -> s.text(this.ragDataSystemItaPrompt)
                        .params(Map.of("documenti", similarDocsString)))
                .user(request.body().question())
                .templateRenderer(StTemplateRenderer.builder().startDelimiterToken('<')
                        .endDelimiterToken('>')
                        .build())
                .call()
                .content());
    }

}
```

Implementazione controllore REST

```
package it.venis.ai.spring.demo.controllers;

...

@RestController
public class QuestionController {

    private final QuestionService service;
    private final RAGService ragService;

    public QuestionController(QuestionService service, RAGService ragService) {
        this.service = service;
        this.ragService = ragService;
    }

    ...

    @PostMapping("/gemini/ask/rag")
    public Answer getGeminiRAGAnswer(@RequestBody QuestionRequest request) {
        return this.ragService.getGeminiRAGAnswer(request);
    }

    @PostMapping("/ollama/ask/rag")
    public Answer getOllamaRAGAnswer(@RequestBody QuestionRequest request) {
        return this.ragService.getOllamaRAGAnswer(request);
    }

}
```

⚠ Verificare la *dashboard* Qdrant (`http://172.17.0.1:6333/dashboard`)



https://github.com/simonescannapieco/spring-ai-advanced-dgroove-venis-code.git

Branch: 7-spring-ai-gemini-ollama-rag-text-to-vector-store