



UNIVERSITÀ DELLA CALABRIA

DIPARTIMENTO DI
INGEGNERIA INFORMATICA,
MODELLISTICA, ELETTRONICA
E SISTEMISTICA

DIMES

CORSO DI LAUREA MAGISTRALE IN
INGEGNERIA INFORMATICA

Progetto di Sistemi Distribuiti e Cloud Computing

UpToCloud:

**Web-app per la condivisione di file basata sui servizi di
Microsoft Azure**

Stefano Perna

Matricola: 235278

Anno Accademico 2021-2022

1	Analisi dei requisiti	4
1.1	Requisiti funzionali	4
1.2	Requisiti non funzionali	5
2	Progettazione e sviluppo	6
2.1	Architettura del sistema	6
2.2	Ambienti e tecnologie di sviluppo	7
2.3	Schema di Database	8
2.4	Microsoft Azure Services	9
2.4.1	Azure Blob Storage	9
2.4.2	Azure Cognitive Search	10
2.5	Spring Boot Web Server	11
2.5.1	Controller layer	11
2.5.2	Service layer	13
	FileService	14
	SearchService	15
2.6	Keycloak Authentication Server	17
2.7	Flutter Web App	18
3	Distribuzione della Web-App	20
3.1	Configurazione di Azure App Service	20
3.2	Docker	21
3.2.1	Configurazione di Docker	21
3.3	Nginx	22
3.4	Demo della Web-App	25
4	Considerazioni finali	34
4.1	Limiti e possibili sviluppi futuri	34

L'obiettivo di questo progetto è quello di realizzare un sistema distribuito che permetta ad un utente di caricare su Cloud documenti in diversi formati, aggiungere descrizione e tag per classificarli ed assegnare permessi di lettura ad altri utenti. Il tutto accessibile tramite un front end web che consenta di caricare e ricercare i documenti.

Il sistema, inoltre, deve utilizzare le risorse di calcolo, di storage e di virtualizzazione messe a disposizione da *Microsoft Azure*.

In questa relazione ci si sofferma inizialmente sull'analisi dei requisiti, di seguito si descrivono le fasi che hanno portato allo sviluppo della piattaforma Web, le principali scelte progettuali, l'architettura del sistema ed i servizi messi a disposizione dell'utente. Quindi, si analizzano le diverse componenti utilizzate per la distribuzione su *Azure*. Infine viene mostrata una demo della Web-App e si discutono limiti ed eventuali sviluppi.

Traccia del progetto

Implementare un sistema distribuito per la gestione e la condivisione di documenti. Il sistema deve consentire agli utenti di salvare sul Cloud documenti in diversi formati, aggiungendo una descrizione e dei tag per classificarli. L'utente potrà avere anche la possibilità di assegnare permessi di lettura ad altri utenti. Creare anche un front end web in un linguaggio a scelta per consentire il caricamento e la ricerca dei documenti.

Il sistema realizzato dovrà utilizzare le soluzioni di calcolo, storage e virtualizzazione messe a disposizione da:

- Microsoft Azure

Per il completamento del progetto, lo studente dovrà presentare una demo funzionante dell'applicazione sopra descritta.

La prima fase ha riguardato l'analisi dei requisiti. Questo step preliminare è di fondamentale importanza nello sviluppo di un sistema software ed il suo scopo è quello di definire le funzionalità che il nuovo prodotto deve offrire.

1.1 Requisiti funzionali

Dalla traccia del progetto emerge che il sistema deve soddisfare i seguenti requisiti:

- Utilizzare le risorse di calcolo, storage e virtualizzazione offerte da Microsoft Azure.
- Permettere il caricamento di documenti in diversi formati sul Cloud.
- Consentire l'inserimento di descrizione e tag per classificare i documenti.
- Permettere agli utenti di assegnare permessi di lettura ad altri utenti del sistema.
- Accesso al sistema tramite Front-end Web per il caricamento e la ricerca dei documenti.

Inoltre, anche se non esplicitati, dalle specifiche si possono dedurre ulteriori requisiti funzionali. Ad esempio, è ragionevole pensare che il sistema debba consentire:

- di registrarsi ed effettuare login;
- la visualizzazione tramite Front-end Web dei documenti già caricati o condivisi da altri utenti;
- la ricerca dei documenti sulla base di tag e descrizione;
- la modifica del titolo del documento;
- la rimozione dei documenti caricati e dei permessi di lettura.

1.2 Requisiti non funzionali

I requisiti non funzionali sono tutte quelle caratteristiche del software non richieste dal cliente, ma che influenzano il lavoro degli sviluppatori; descrivono come il sistema riesce ad eseguire certi compiti. I requisiti non funzionali individuati per il progetto sono:

- **Usabilità:** un software è considerato usabile se è facile da utilizzare dagli utenti. È una qualità soggettiva, influenzata molto dall'interfaccia grafica. In questo caso il Front-end Web che si andrà a realizzare.
- **Prestazioni:** sono qualità influenzate dall'efficienza. È efficiente se sfrutta in maniera ottimale le risorse di calcolo a disposizione. Al fine di garantire questo requisito l'utilizzo delle risorse di calcolo e virtualizzazione offerte da Azure risulta ottimale. In quanto permette di usufruire di risorse che possono essere rilasciate dinamicamente in base al carico di lavoro richiesto.
- **Affidabilità:** un software si dice affidabile se opera come ci si aspetta che esso faccia ed è definita come la probabilità che il software si comporti come atteso per un certo intervallo di tempo. Anche in questo caso, affidarsi alle risorse Cloud permette di avere una maggiore affidabilità.
- **Robustezza:** un sistema è robusto se si comporta in maniera accettabile anche in circostanze non previste dalle specifiche. Un esempio può essere il caricamento di un dato non corretto.

Una volta individuati i requisiti, si è passati alla fase di progettazione dove vengono definite le linee principali della struttura del sistema in funzione delle caratteristiche richieste.

2.1 Architettura del sistema

L'architettura di massima del sistema è evidenziata in figura [2.1](#). Essa è suddivisa in cinque macro-componenti principali:

- Back-end Server
- Database
- Authentication Server
- Front-end Web App
- Microsoft Azure Services

Ogni macro-componente offre un servizio a sé stante. Questo permette di effettuare un deployment adattabile alle diverse necessità e di ridurre le dipendenze tra i diversi elementi del sistema.

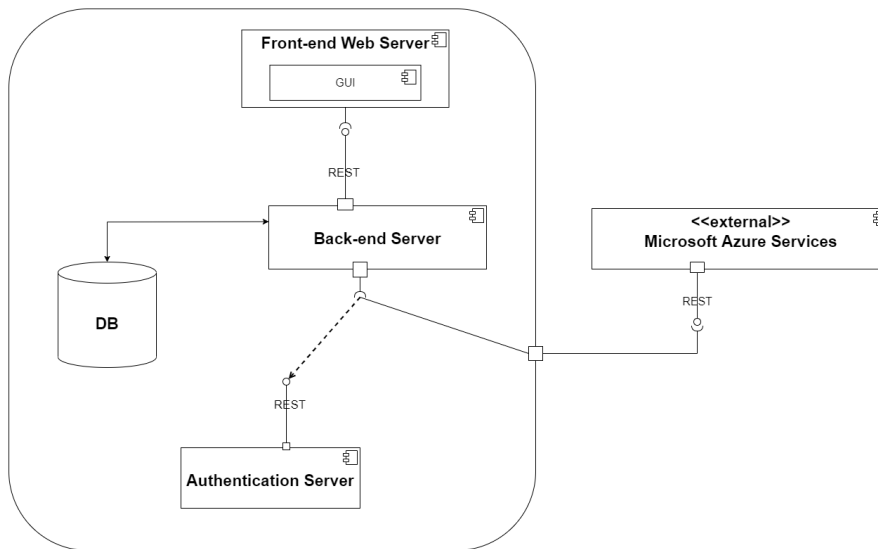


Figura 2.1: — *Architettura astratta del sistema*

2.2 Ambienti e tecnologie di sviluppo

Stabiliti i macro-componenti del sistema, per ognuno di essi si sono scelte le tecnologie più adatte per la loro implementazione. Quindi si è proseguito nello sviluppo del software. La figura 2.2 mostra una struttura più dettagliata del sistema.

Il back-end server è stato sviluppato attraverso il framework Spring Boot, in quanto estremamente flessibile e aperto a cambiamenti futuri. Il front-end, invece, è basato sul framework Flutter, facilmente integrabile con Spring Boot. Entrambi i framework sottintendono un pattern analogo a MVC (Model View Controller), anche se in Spring Boot, la view è spesso affidata a un frontend, come in questo caso. Per quanto riguarda l'Authentication Server si è scelto di utilizzare Keycloak, un prodotto software open source che consente il Single Sign-on con *Identity* e *Access Management* rivolto ad applicazioni e servizi moderni. Infine, il database utilizzato è PostgreSQL.

Gli ambienti di sviluppo scelti sono stati:

- IntelliJ Idea Ultimate: per lo sviluppo in Spring Boot e Flutter ed il test del sistema in locale.
- Visual Studio Code: per alcune operazioni di configurazione in quanto più *leggero* rispetto al software precedente.

Entrambi offrono supporto a VCS e alla gestione automatica delle dipendenze.

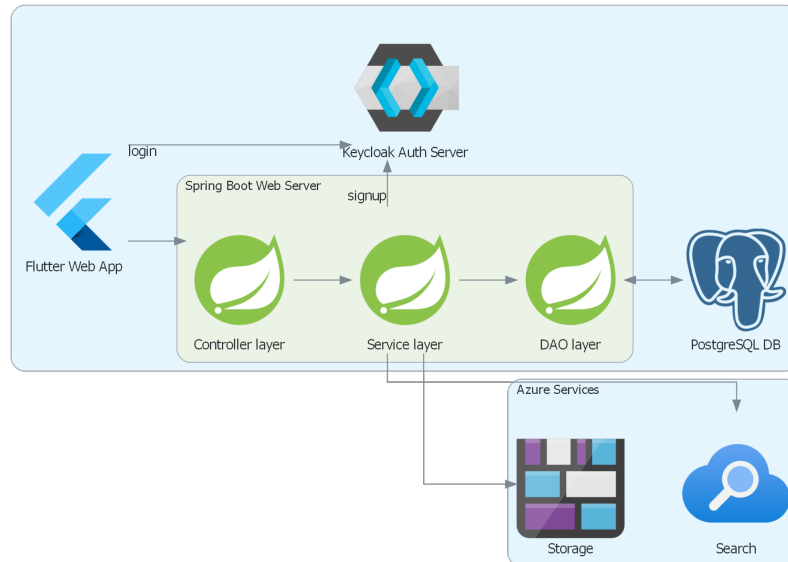


Figura 2.2: — *Architettura e componenti del sistema*

Le componenti principali sono ora elencate, con le relative funzionalità.

2.3 Schema di Database

Il database è stato strutturato per memorizzare le informazioni riguardanti gli utenti registrati, i documenti da loro caricati, condivisi e per gestirne i permessi di lettura. Inoltre, ad ogni documento sono associati i suoi metadata con i relativi tag. La tabella *Document*, oltre al proprietario del documento, memorizza il riferimento al file caricato su Cloud. I permessi di lettura sono stati gestiti attraverso una relazione Many-to-Many tra *Document* e *Users*. Infine, grazie all'utilizzo di un server esterno per l'autenticazione, non è stato necessario prevedere alcun campo per il login (come ad esempio un Hash della password degli utenti). Di seguito lo schema di database (fig. 2.3).

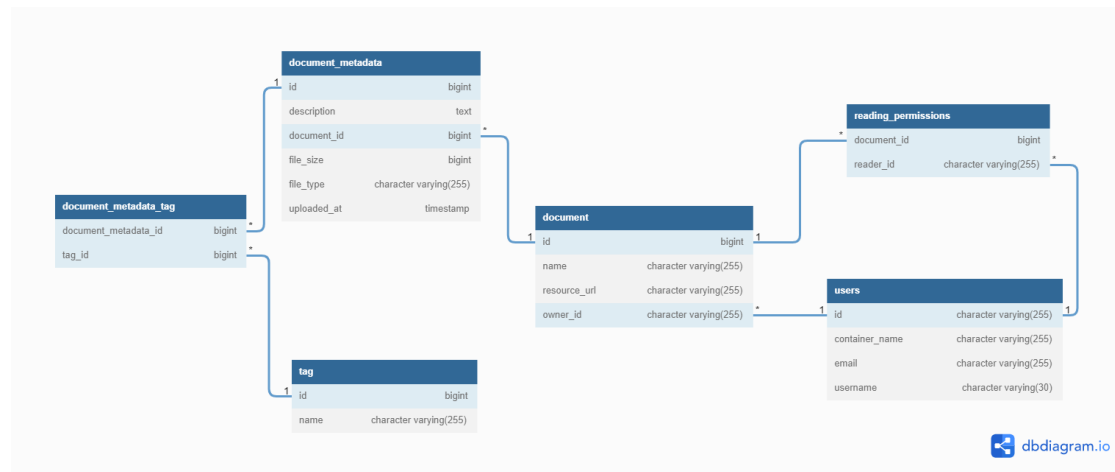


Figura 2.3: — Schema di Database

2.4 Microsoft Azure Services

La Web-App si basa principalmente su due servizi di Microsoft Azure: **Azure Blob Storage** ed **Azure Cognitive Search**. Rispettivamente per l'archiviazione e la ricerca dei documenti. Inoltre, per quanto riguarda la distribuzione del sistema si è utilizzato il servizio **Azure App Service** che consente di utilizzare le risorse di calcolo e di virtualizzazione di Azure per eseguire la Web-App ed offre una soluzione pronta all'uso per l'hosting del sito web. L'utilizzo di quest'ultimo servizio verrà approfondito nel prossimo capitolo (3).

2.4.1 Azure Blob Storage

Azure Blob Storage è la soluzione di Microsoft per l'archiviazione di oggetti. Blob Storage è ottimizzato per l'archiviazione di grandi moli di dati non strutturati, come ad esempio file di testo o dati binari. Come si legge dalla documentazione Microsoft, Blob Storage è progettato per:

- Invio di immagini o documenti direttamente da browser
- Archiviazione di file per l'accesso distribuito
- Archiviazione di dati per backup e ripristino, ripristino di emergenza e archiviazione.
- Archiviazione di dati a scopo di analisi da parte di un servizio locale o ospitato in Azure.

Gli oggetti sono accessibili tramite l'API REST di Azure, disponibili in diversi linguaggi tra cui Java, utilizzato nello sviluppo del back-end Spring.

Nelle specifiche di progetto viene richiesto di gestire documenti di *diversi formati* per il caricamento su Cloud. Blob Storage, quindi, appare il servizio più adatto per questo scopo. Inoltre, non essendo richiesta una gestione gerarchica dei documenti è possibile, attraverso Blob Storage, implementare una soluzione semplice e immediata.

Con il servizio Blob Storage entrano in gioco tre tipi di risorse:

- Account di archiviazione
- Container appartenente ad un account di archiviazione
- Oggetto Blob all'interno di un contenitore

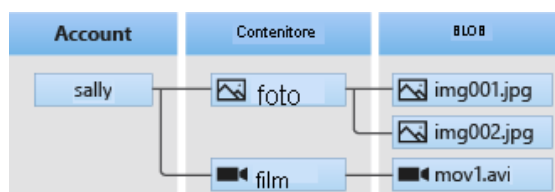


Figura 2.4: — Schema di Autenticazione

In questo progetto si è scelto di creare un unico account di archiviazione ed un container per ogni utente del sistema. Mentre, come si può facilmente intuire, un oggetto Blob corrisponde ad un documento caricato dall'utente. Così facendo si isolano i blob in base ai proprietari degli stessi.

2.4.2 Azure Cognitive Search

Azure Cognitive Search è un servizio Cloud di ricerca che supporta l'indicizzazione e query su indici definiti dall'utente che fanno riferimento a risorse memorizzate nel Cloud.

Cognitive Search è composto da tre tipi di risorse:

- Index: Definisce la struttura dei documenti ricercabili in formato Json. È formato da coppie del tipo nome_del_campo - tipo_di_dato.
- Datasource: Definiscono le fonti dati dalle quali recuperare i contenuti ricercabili.
- Indexer: È un sottoservizio (*crawler*) che si occupa di analizzare e recuperare i contenuti dai datasource.

È possibile integrare facilmente Cognitive Search con i dati memorizzati in Blob Storage. Per far ciò basta creare i datasource all'interno del servizio Cognitive Search ed associare ognuno di esse ai container del Blob Storage.

2.5 Spring Boot Web Server

Il framework Spring Boot Web MVC fornisce un'architettura Model-View-Controller (MVC) pronta all'uso che permette di separare i diversi aspetti dell'applicazione, come: Data logic, business logic e input logic. L'implementazione di ognuno di questi aspetti è stata suddivisa nei diversi layer riportati in figura 2.5.

- DAO layer: DAO sta per Data Access Object. Si interfaccia con una risorsa dati ed offre un'interfaccia generica per accedervi. Consente di modificare i meccanismi di accesso ai dati indipendentemente dal codice che li utilizza.
- Service layer: Incapsula la logica di business dell'applicazione, ne definisce le funzionalità offerte, indipendentemente dalla modalità di accesso.
- Controller layer: Espone le funzionalità dell'applicazione all'esterno sotto forma di interfaccia RESTful.

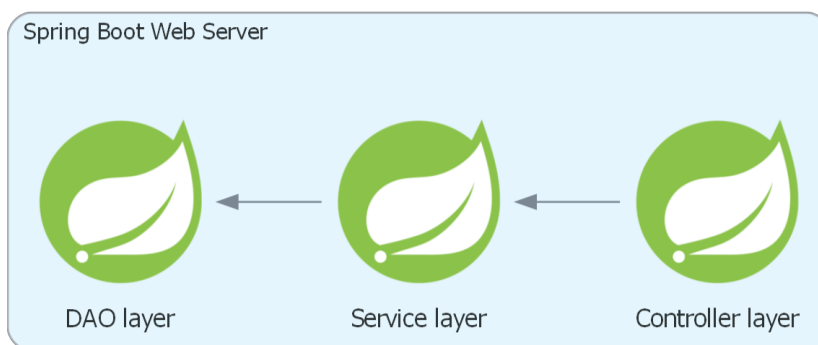


Figura 2.5: — Architettura Spring Boot Web Server

2.5.1 Controller layer

L'interfaccia del server Spring Boot è suddivisa in 3 componenti:

- FileController: fornisce un'interfaccia per caricare, eliminare i file, gestire i meta-data ed i permessi di lettura degli stessi.
- UserController: fornisce un'interfaccia per registrare un nuovo utente o eliminare l'account.
- SearchController: fornisce un'interfaccia per effettuare la ricerca dei documenti caricati.

Di seguito viene mostrata la documentazione delle API (fig: 2.6, 2.7, 2.8) accessibile pubblicamente dal seguente link: [API Docs](#). Quest'ultima è stata realizzata automaticamente utilizzando il tool *Swagger*, compatibile con Spring.

file-controller			^
PUT	/api/files/remove-readers	Remove readers from the specified file	▼
PUT	/api/files/remove-reader	Remove a reader from the specified file	▼
PUT	/api/files/add-readers	Add readers to the specified file	▼
PUT	/api/files/add-reader	Add a reader to the specified file	▼
POST	/api/files/upload	Upload a file as a blob in the user's container	▼
POST	/api/files/set_metadata/{doc_id}	Set metadata for the specified document	▼
GET	/api/files/share-suggestions	Get users that are readers of at least one document owned by logged user	▼
GET	/api/files/readersByDoc	Get readers of specified document	▼
GET	/api/files/get_metadata/{doc_id}	Get metadata for the specified document	▼
GET	/api/files/get_blob_metadata/{doc_id}	Get Blob metadata for the specified document	▼
GET	/api/files/download/{doc_id}	Download the specified file	▼
DELETE	/api/files/delete	Delete files	▼
DELETE	/api/files/delete/{doc_id}	Delete a file	▼

Figura 2.6: — File controller

user-controller			^
POST	/api/users/new	Create a new user	▼
GET	/api/users	Retrieve user by jwt	▼
GET	/api/users/byEmail/{email}	Retrieve user by email	▼
GET	/api/users/byEmail-contains/{email}	Retrieve user if email contains specified string	▼
DELETE	/api/users/delete	Delete user	▼
DELETE	/api/users/delete/{id}	Delete a user (only admin)	▼

Figura 2.7: — User controller

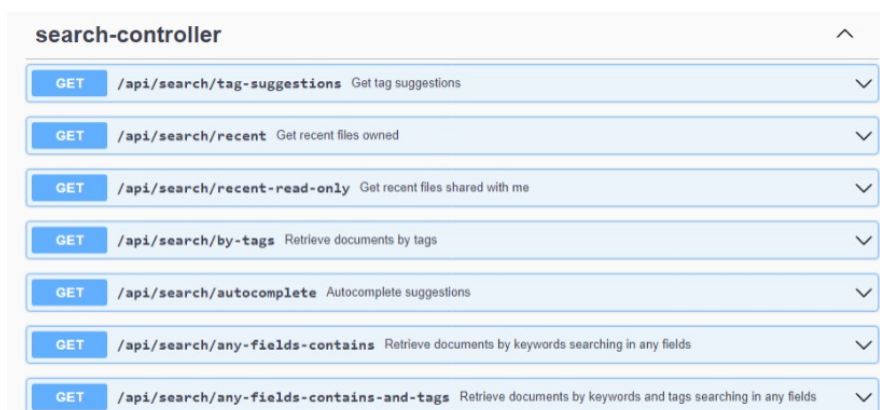


Figura 2.8: — Search controller

2.5.2 Service layer

Il service layer realizza la logica di business dell'applicazione. Esso riceve le richieste dal controller layer, recupera i dati necessari dal DAO layer e si interfaccia con i servizi esterni, in questo caso Keycloak e le risorse Cloud di Azure. Il service layer è stato suddiviso in più componenti sulla base delle diverse responsabilità in modo da ridurre le dipendenze tra le classi e mantenere un'alta coesione interna. I componenti principali sono:

- **AccountingService:** Si occupa della registrazione dell'utente nel sistema e della comunicazione con l'Authentication server.
- **FileService:** Gestisce tutte le operazioni riguardanti i file. Il caricamento, l'eliminazione, la gestione dei metadata e dei permessi di lettura. Si interfaccia con il servizio Azure Blob Storage.
- **SearchService:** Gestisce tutte le operazioni di ricerca dei file e si interfaccia con il servizio Azure Cognitive Search.

Oltre ai componenti precedentemente elencati, sono state utilizzate altre classi di servizio che si interfacciano con il DAO layer per ottenere i dati, manipolarli o aggiornarli e classi di configurazione per la creazione di Bean Spring, necessari per comunicare con le API di Azure. Un Bean è un oggetto Java la cui istanziazione, assemblamento e gestione è delegata completamente allo Spring IoC Container, questo lo rende facilmente accessibile da qualsiasi componente dell'applicazione. In particolare, sono state implementate le classi *AzureBlobStorageConfig* e *AzureSearchConfig*. La prima si occupa di istanziare il Bean *BlobServiceClient* che consente di connettersi ai servizi di Azure Blob Storage attraverso una chiave chiamata *connection string*. Mentre, la seconda ha il compito di istanziare i Bean necessari per raggiungere il servizio Azure Cognitive Search ed inoltre provvede a creare l'indice (fig. 2.9) che verrà utilizzato per effettuare la ricerca dei documenti. Quest'ultimo verrà creato esclusivamente al primo avvio dell'applicazione.

FileService

Il compito principale della classe FileService è quello di interfacciarsi con il servizio Azure Blob Storage e con il DAO layer per la gestione ed il caricamento su Cloud dei file e dei relativi metadata e tag. I metodi più importanti sono *createContainer* [2.1] che si occupa di creare il container associato all'utente in Blob Storage, *uploadToBlob* [2.2] il cui compito è di caricare il file su Cloud e *downloadDocument* [2.3].

Listing 2.1: createContainer

```
1 private BlobContainerClient createContainer(User u){
2     //Create a unique name for the container
3     String containerName = u.getId();
4     u.setContainerName(containerName);
5
6     // Create the container and return a container client object
7     BlobContainerClient blobContainerClient = blobServiceClient.
        createBlobContainer(containerName);
8
9     // Connect the container with Azure Cognitive Search
10    searchService.getOrCreateDataSourceConnection(containerName);
11    searchService.getOrCreateSearchIndexer(containerName);
12    return blobContainerClient;
13 }
```

Listing 2.2: uploadToBlob

```
1 private String uploadToBlob(User u, Document d, MultipartFile file
2     , Map<String, String> metadata) throws IOException {
3     BlockBlobClient blockBlobClient = getOrCreateContainerByOwner(
4         u).getBlobClient(d.getId().toString()).getBlockBlobClient()
5     ;
6     blockBlobClient.upload(new BufferedInputStream(file.
7         getInputStream()), file.getSize(), true);
8     blockBlobClient.setMetadata(metadata);
9     return blockBlobClient.getBlobUrl();
10 }
```

Listing 2.3: downloadDocument

```
1 public ByteArrayOutputStream downloadDocument(String userID, Long
2     docID)
3     throws ResourceNotFoundException, IOException,
4         UnauthorizedUserException {
5     User u;
6     Document d;
7     try {
8         u = userService.getById(userID);
9         d = documentService.getById(docID);
10     }
```

```
8      }catch (ResourceNotFoundException e){
9          logger.warning(e.toString());
10         throw e;
11     }
12     if(!canRead(u, d)){
13         logger.warning("User can't read");
14         throw new UnauthorizedUserException();
15     }
16     BlobContainerClient blobContainerClient =
17         getOrCreateContainerByOwner(d.getOwner());
18     // Get a reference to a blob
19     BlobClient blobClient = blobContainerClient.getBlobClient(d.
20         getId().toString());
21     ByteArrayOutputStream baos = new ByteArrayOutputStream();
22     // download file from azure blob storage with stream
23     logger.info("Trying to download file from blob");
24     blobClient.downloadStream(baos);
25     logger.info("File downloaded");
26     return baos;
27 }
```

SearchService

La ricerca tramite Cognitive Search è stata implementata nella classe `SearchService`. Questa utilizza i Bean della classe *AzureSearchConfig* precedentemente descritti per raggiungere il servizio Azure e definisce diversi metodi per la ricerca tramite parole chiave o tag. Prima di discutere delle funzionalità offerte da `SearchService`, è fondamentale spiegare come sono stati gestiti i metadata ed i tag all'interno del sistema. Ricordiamo dallo schema di database (fig. 2.3) che l'applicazione oltre a memorizzare l'url dei Blob caricati su Cloud, per ogni documento memorizza i relativi tag e metadata. Tuttavia, se lasciati esclusivamente nel DB locale, questi dati sarebbero stati irraggiungibili dal servizio Cognitive Search rendendo impossibile il suo utilizzo per la ricerca. Quindi, si è scelto di utilizzare le API di Azure per caricare oltre al documento in sé, anche i metadata ed i tag ad esso associati sul Blob Storage. Così facendo è stato possibile definire l'indice di ricerca mostrato in figura 2.9 utilizzando proprio questi dati.


Field name	Type	Retrievable	Filterable	Sortable	Facetable	Searchable	Analyzer
		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
 id	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standa... ▾
content	String	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	Standa... ▾
tags	StringCollection	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standa... ▾
fileType	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standa... ▾
description	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standa... ▾
filename	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standa... ▾
metadata_storage_size	int64	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
metadata_storage_last_modified	DateTimeOffset	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
metadata_storage_file_extension	String	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	Standa... ▾

Figura 2.9: — *Indice di ricerca*

I metodi previsti dalla classe `SearchService` sono:

- `searchAnyFieldsContains`: Data una stringa, trova i documenti che matchano con tale stringa ricercando sulla base di tutti i campi presenti nell'indice. Inoltre, grazie alla funzione Fuzzy di Cognitive Search è possibile trovare anche documenti che non hanno una corrispondenza perfetta con la stringa.
- `searchByAnyFieldsContainsAndTags`: Data una stringa e una lista di tag, trova i documenti che matchano con tale stringa ricercando sulla base di tutti i campi presenti nell'indice e li filtra sulla base dei tag in input.
- `searchByTags`: Trova i documenti filtrandoli sulla base dei tag in input.

Inoltre, per quanto riguarda i primi due metodi, si è fatto utilizzo di un parametro booleano per abilitare/disabilitare la ricerca all'interno del contenuto dei documenti. Questa scelta si è resa necessaria a causa della complessità computazionale di questo tipo di ricerca.

Infine, un aspetto critico della ricerca tramite Cognitive Search è stato garantire che tra i risultati della ricerca non comparissero documenti a cui l'utente non potesse accedere in quanto privo dei permessi di lettura. Per questo motivo, ogni metodo in `SearchService`, dato un utente, ottiene gli ID dei documenti a cui questo può accedere e usa queste informazioni per filtrare i documenti direttamente su Cognitive Search tramite le opzioni della query [2.4].

Listing 2.4: *Un esempio di ricerca*

```

1  ...
2
3  String ids = getIDAccessibleDocuments(user);
4  if(ids == null) return new LinkedList<>();
5
6  SearchOptions options = new SearchOptions()
```

```
7         .setFilter(String.format("search.in(id, %s)", ids))
8         .setQueryType(QueryType.FULL)
9         .setOrderBy("search.score()")
10        .setSearchFields(getSearchField(searchInContent).
11                           toArray(new String[0]));
12
13        SearchPagedFlux searchPagedFlux = searchAsyncClient.search(
14            inputFuzzy(text,1)+inputRegexContains(text),
15            options);
16    }
17    ...
18 }
```

2.6 Keycloak Authentication Server

Keycloak è un prodotto software open source che abilita il Single Sign-On (IdP) con Identity Management e Access Management per applicazioni e servizi moderni. Questo software è scritto in Java e supporta i protocolli standard per l'autenticazione SAML v2 e OpenID Connect (OIDC) / OAuth2. L'utilizzo di un IdP consente di delegare i meccanismi di autenticazione dell'applicazione e facilitarne lo sviluppo, consentendo agli sviluppatori di concentrarsi sulla logica di business del sistema senza doversi preoccupare degli aspetti di sicurezza dell'autenticazione. La figura 2.10 rappresenta lo schema di autenticazione utilizzato. Durante la fase di login, una volta ottenute le credenziali dall'utente, il front-end invia una richiesta all'Authentication Server per ottenere il token di accesso. Quindi, può richiedere le risorse protette al back-end Spring inoltrando a quest'ultimo l'Access Token, oltre ai dati necessari previsti dall'interfaccia REST. La fase di signup viene gestita dal back-end Spring il quale dopo aver ricevuto la richiesta dal front-end con le informazioni inserite dall'utente, lo memorizza nel suo DB e lo registra sull'Authentication Server.

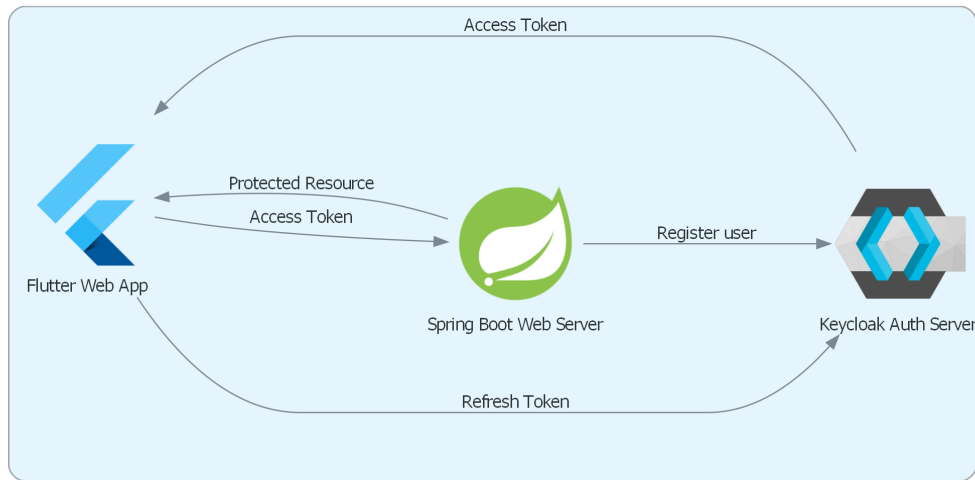


Figura 2.10: — *Schema di Autenticazione*

Per la creazione dell'Access Token si è scelto di utilizzare lo standard Json Web Token (JWT) che permette di scambiare informazioni tramite oggetti Json in modo sicuro. Inoltre, JWT è completamente integrato con Spring Security.

2.7 Flutter Web App

Flutter è un framework open-source creato da Google per la creazione di interfacce multi-piattaforma per Android, iOS, Linux, macOS, Windows e per il Web utilizzando un'unica *code base*. Questa caratteristica permette di ridurre notevolmente i costi ed i tempi di sviluppo nel caso in cui sia necessario far girare l'applicativo su dispositivi eterogenei. Inoltre, Flutter ha il vantaggio di essere estremamente facile da utilizzare anche per uno sviluppatore alle prime armi con framework per il front-end.

Le componenti del front-end sono suddivise in diversi package e sono:

- **api:** Definisce un'interfaccia per l'inoltro delle richieste al back-end Spring.
- **models:** Definisce i modelli dei dati. Mappa gli oggetti Json (che si ricevono dal back-end) in oggetti Dart.
- **supports:** Contiene classi di utilità
- **UI:** Definisce la User Interface dell'applicazione.

- controllers: Definisce oggetti condivisi che potranno essere utilizzati da diversi componenti per controllare il comportamento dell'applicazione. Ad esempio il MenuController è utilizzato per gestire il menù laterale della Web-App.
- managers: Contiene il RestManager ed il PersistentStorageManager. Il primo raccoglie le richieste da parte dell'ApiController, le incapsula in richieste HTTP e le inoltra al back-end. Il secondo è utilizzato per memorizzare temporaneamente i dati dell'utente loggato.

Distribuzione della Web-App

La fase finale del progetto ha riguardato la distribuzione della Web-App su Azure. Tra i diversi servizi di virtualizzazione offerti dal Cloud di Microsoft, come precedentemente anticipato, la scelta è ricaduta sul servizio Azure App Service. Quest'ultimo consente di utilizzare le risorse di calcolo e di virtualizzazione di Azure per eseguire la Web-App ed offre una soluzione pronta all'uso per l'hosting del sito web. Supporta sia Windows che Linux, fornisce sistemi di auto-scaling e dà la possibilità di effettuare il deployment attraverso repository GitHub, utilizzando Azure DevOps o attraverso l'utilizzo di repository Docker.

In questo progetto si è deciso di utilizzare Docker e quindi creare un'applicazione multi-container che incapsula tutti i componenti descritti nel precedente capitolo, con l'aggiunta di un server Nginx il quale funge da *reverse proxy*.

La figura (3.1) raffigura l'architettura completa del sistema.

3.1 Configurazione di Azure App Service

Per la configurazione di *App Service* si sono seguiti i seguenti step:

1. Creazione dell'App Service Plan.
2. Upload delle immagini custom di back-end e front-end su DockerHub.
3. Upload dei file di configurazione sulla VM Host di App Service tramite FTP.
4. Configurazione di variabili di ambiente sulla VM Host, come l'url del docker registry, la porta di default per accedere al sito web e il timeout per l'esecuzione dei container.

L'App Service Plan definisce l'insieme di risorse computazionali che verranno utilizzate per eseguire la Web-App. Ovvero, le caratteristiche della VM (RAM, SO, ecc...). In particolare si è scelto il Service Plan B1 con sistema operativo Linux (fig. 3.2), poiché compatibile con il budget messo a disposizione da Azure Students di 100€.

Instance	Cores	Ram	Storage	Pay as you go
B1	1	1.75 GB	10 GB	€0.095/hour

Figura 3.2: — App Service Plan - B1

Per il secondo step, si è scelto di utilizzare DockerHub come registry delle immagini Docker piuttosto che il servizio Container Registry di Azure in quanto a pagamento.

3.2 Docker

Docker è un software open-source che offre un insieme di prodotti platform as a service (PaaS) che utilizzano la virtualizzazione a livello dei sistemi operativi (containerizzazione). Questo approccio consente di eseguire processi in ambienti isolati, minimali e facilmente distribuibili chiamati *container*.

L'approccio basato su container consente di semplificare i processi di deployment di applicazioni software, proprio per questo motivo si è scelto di effettuare il deployment utilizzando Docker.

3.2.1 Configurazione di Docker

Per incapsulare i diversi componenti (Spring Boot Web Server, Flutter Server, Nginx, PostgreSQL DB, Keycloak Server) in container Docker si è utilizzato Docker-Compose, uno strumento che consente di definire un'applicazione Docker multi-container attraverso un file di configurazione YAML ed eseguirla attraverso un singolo comando.

Il file *docker-compose.yml* definisce i servizi che dovranno essere eseguiti come container docker. Ne specifica le caratteristiche, le variabili d'ambiente, i volumi ed eventuali comandi che dovranno essere eseguiti all'avvio. In figura [3.1], viene mostrato un esempio di servizio all'interno del docker compose file. In particolare il servizio riguardante il database Postgres. Viene definita l'immagine da utilizzare, ovvero quella di Postgres presente su DockerHub pubblico, le variabili d'ambiente e due volumi. Il primo viene utilizzato per importare lo schema di database all'avvio, mentre il secondo volume serve per garantire la persistenza dei dati. Così facendo, infatti, i dati del DB vengono mappati nel file system della macchina Host e nel caso in cui il container dovesse fermarsi i dati non verranno persi.

Infine, è stato necessario creare le immagini custom per il back-end Spring e per il front-end Flutter. A questo scopo sono stati definiti due DockerFile.

Listing 3.1: docker-compose.yaml - Esempio DB

```
1  ...
2
3  db:
4    image: postgres
5    container_name: db
6    restart: always
7    environment:
8      POSTGRES_DB: uptocloud db
9      POSTGRES_USER: postgres
10     POSTGRES_PASSWORD: uptocloud db
11     PGDATA: /var/lib/postgresql/data/pgdata
12   volumes:
13     - ${WEBAPP_STORAGE_HOME}/imports/db/uptocloud db.sql:/docker-
      entrypoint-initdb.d/uptocloud db.sql
14     - db_data:/var/lib/postgresql/data
15   ports:
16     - "5432:5432"
17   ...
```

3.3 Nginx

Prima di completare il deployment dell'applicazione si è reso necessario introdurre un ulteriore componente. Un Nginx Server, ovvero un reverse proxy il cui compito è quello di raccogliere le richieste dei client ed inoltrarle ai server appropriati, nascondendo la struttura interna del sistema. Seguendo lo schema in figura (3.3), Nginx è stato incapsulato all'interno del docker-compose insieme a tutti gli altri servizi. Così facendo, una volta distribuita la Web-App su App Service, è possibile esporre all'esterno solo il reverse proxy tramite la porta 80 così da isolare gli altri servizi e rendere l'applicazione più sicura. Inoltre, grazie al reverse proxy è possibile far comunicare indirettamente gli altri container tra di loro, evitando le restrizioni imposte da alcuni browser che potrebbero bloccare le richieste in caso di url diversi da quelli di origine (ad esempio richieste da front-end a back-end).

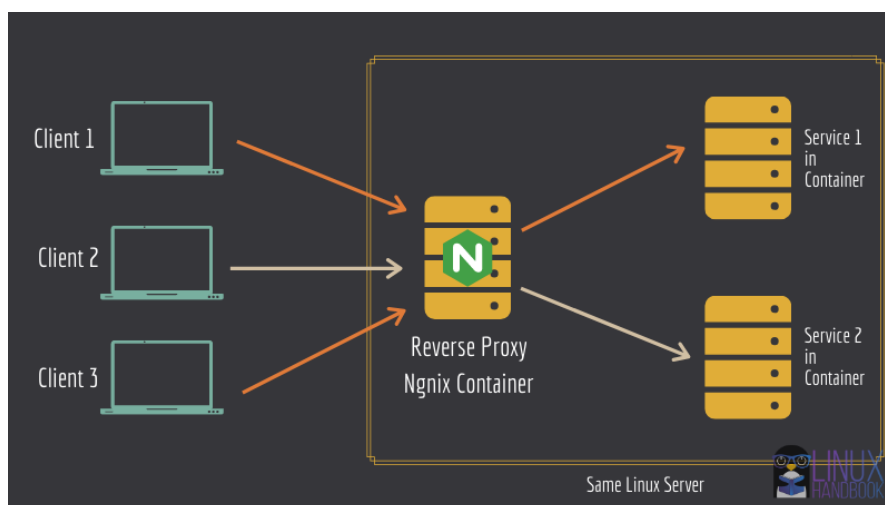


Figura 3.3: — *Deploy di più servizi tramite un container Nginx*

Per la configurazione di Nginx è stato necessario definire un file di configurazione, il quale è stato caricato successivamente nella VM Host di App Service tramite FileZilla, un software per lo scambio di dati tramite il protocollo FTP.

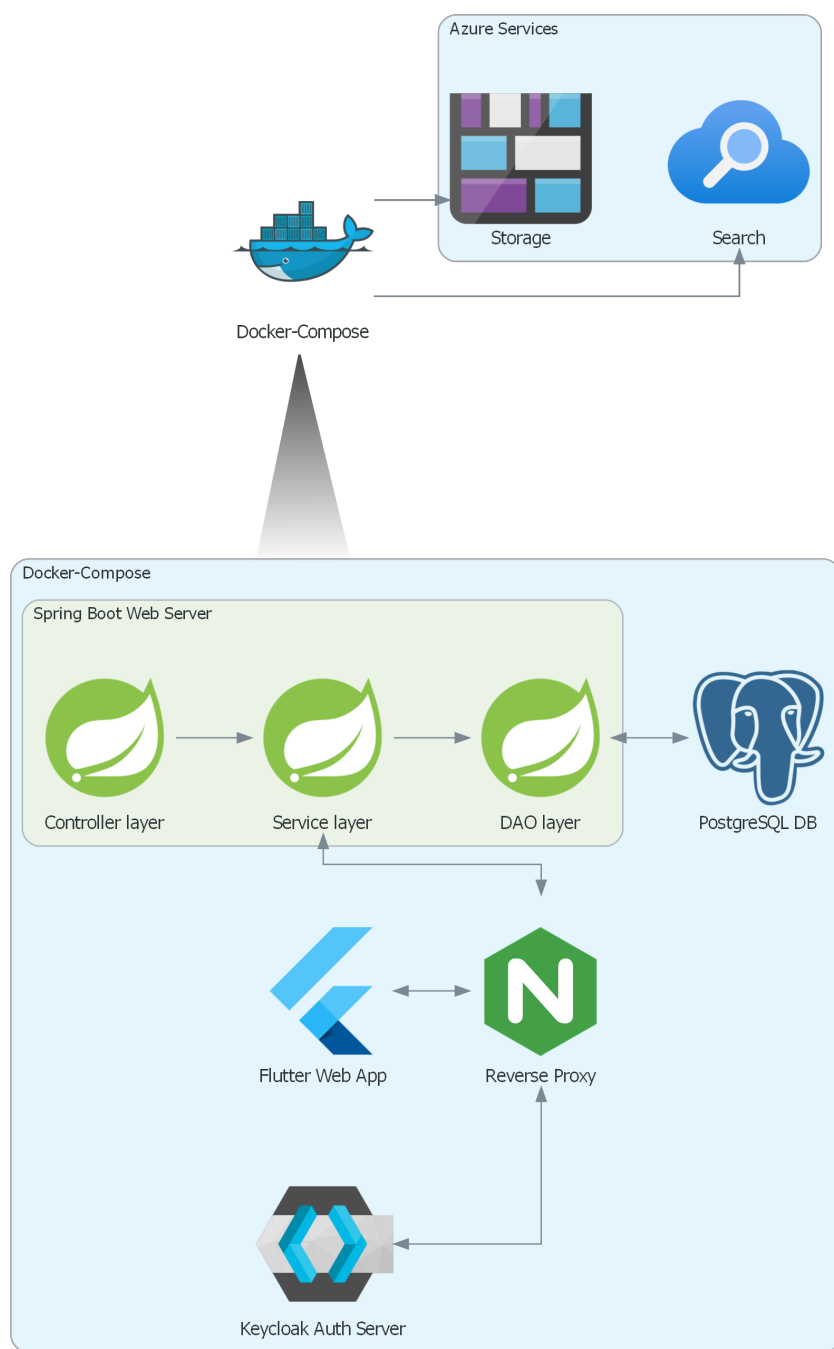


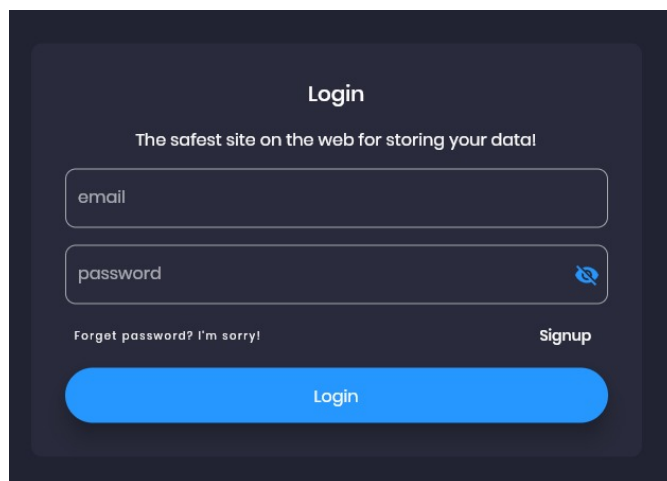
Figura 3.1: — Architettura completa del sistema

3.4 Demo della Web-App

In questo paragrafo verrà mostrata una demo della Web-App accessibile pubblicamente attraverso il seguente URL fornito dal servizio App Service: <https://uptocloud.azurewebsites.net>

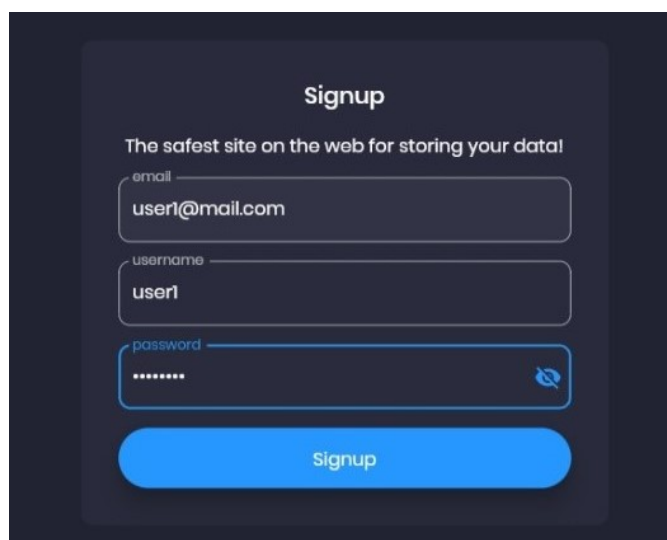
Login e Sign-up

Di seguito vengono riportati gli screenshot della pagina di login e di sign-up.



The screenshot shows a dark-themed login form. At the top, the word "Login" is centered. Below it, the text "The safest site on the web for storing your data!" is displayed. There are two input fields: "email" and "password". The "password" field has a small icon of an eye with a slash, indicating a toggle for visibility. Below the "password" field, there is a link "Forgot password? I'm sorry!" on the left and a "Signup" link on the right. At the bottom, there is a large blue button labeled "Login".

Figura 3.4: Login



The screenshot shows a dark-themed signup form. At the top, the word "Signup" is centered. Below it, the text "The safest site on the web for storing your data!" is displayed. There are three input fields: "email", "username", and "password". The "email" field contains the text "user1@mail.com". The "username" field contains the text "user1". The "password" field has a small icon of an eye with a slash, indicating a toggle for visibility. At the bottom, there is a large blue button labeled "Signup".

Figura 3.5: Signup

Feedback

Per aumentare l'usabilità, l'applicazione prevede l'utilizzo di pop-up in modo da dare un feedback all'utente in caso di richiesta andata a buon fine o meno, oppure Pop-up Dialog per confermare alcune scelte irreversibili come l'eliminazione di un account o di un file.

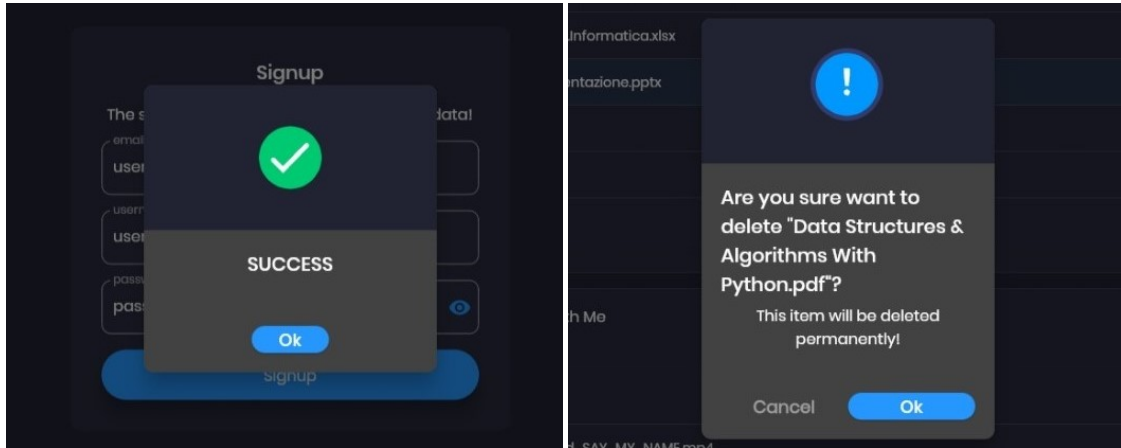


Figura 3.6: Feedback in caso di richiesta andata a buon fine

Figura 3.7: Pop-up Dialog per confermare la scelta di rimuovere un file

The image shows a 'Signup' form with the title 'The safest site on the web for storing your data!'. It has three input fields: 'email' with the value 'user1@mailcom', 'username' with the value 'user1@31', and 'password' with three dots. Red error messages are displayed below each field: '* Enter a valid email' under the email field, 'Username may only contain letters, numbers and _' under the username field, and 'Password should be at least 6 characters' under the password field. A blue 'Signup' button is at the bottom.

Figura 3.8: Feedback in caso di formato errato

HomePage

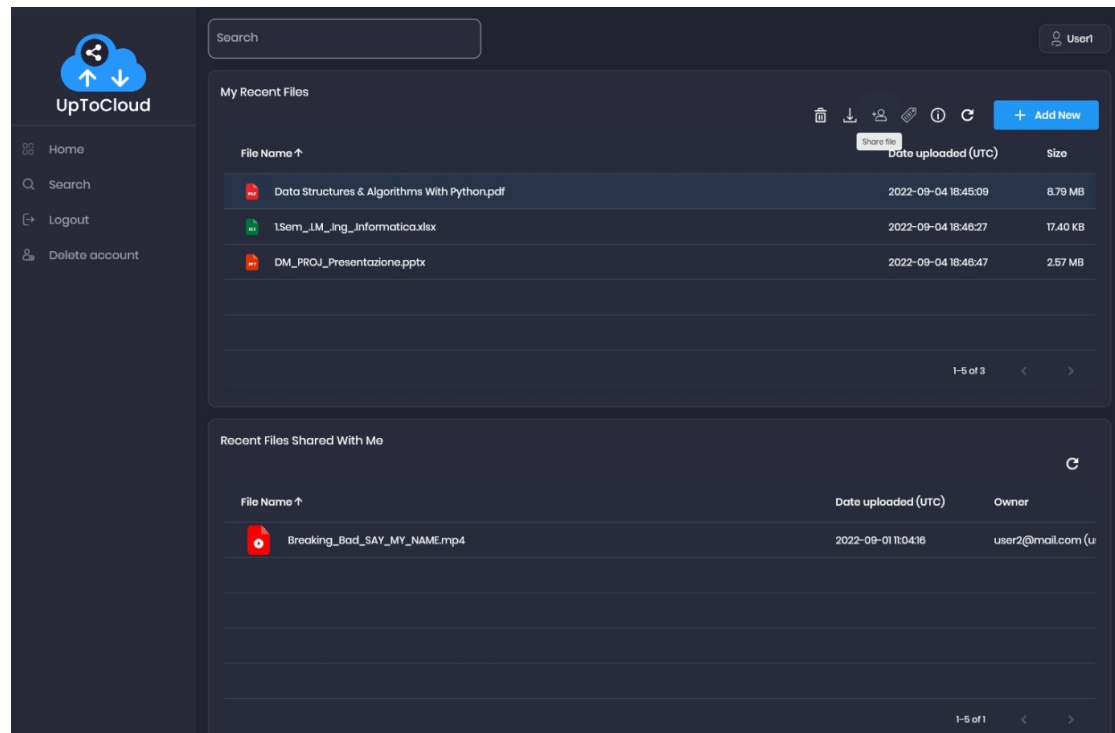


Figura 3.9: *HomePage*

L'HomePage presenta due tabelle. La prima mostra i file recenti caricati dall'utente loggato. Mentre la seconda mostra i file recenti che gli altri utenti hanno condiviso con l'utente con loggato. Ogni file della tabella è caratterizzato dal titolo, dalla data di caricamento e dalla dimensione. Inoltre, per i file condivisi viene mostrata la e-mail e lo username dell'utente *owner*.

Una volta selezionato un file è possibile, mediante dei bottoni situati nella parte superiore della tabella, eseguire diverse operazioni:

- Eliminare il file.
- Scaricare il file.
- Modificare i metadata del file (titolo, descrizione, tag).
- Condividere il file ad altri utenti.
- Visualizzare maggiori informazioni sul file
- Aggiornare la lista dei file
- Caricare un nuovo file

Alcune delle operazioni appena descritte sono mostrate nei seguenti screenshot.

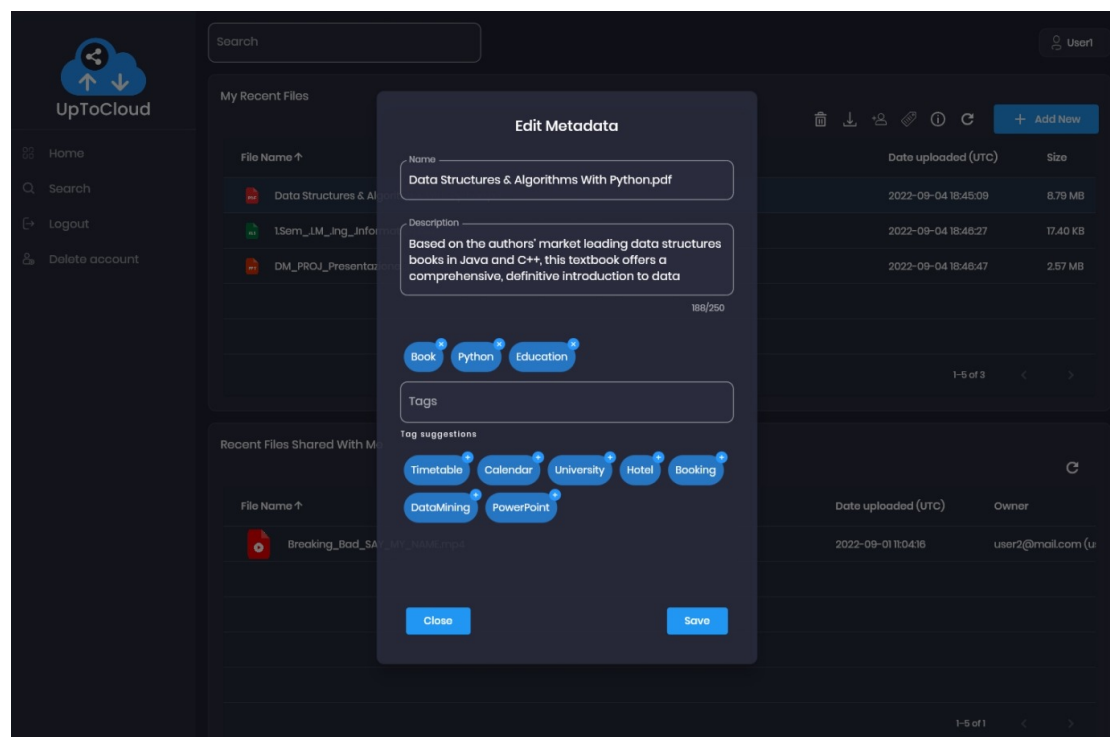


Figura 3.10: Modifica dei metadata

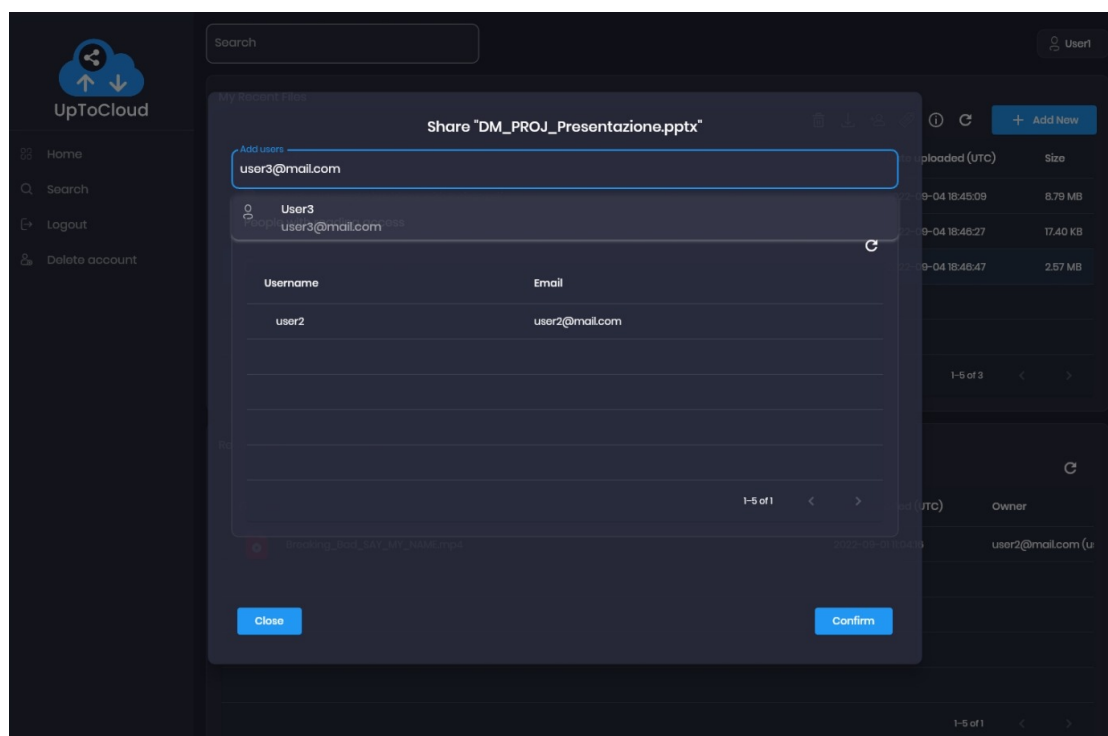


Figura 3.11: Condivisione dei file

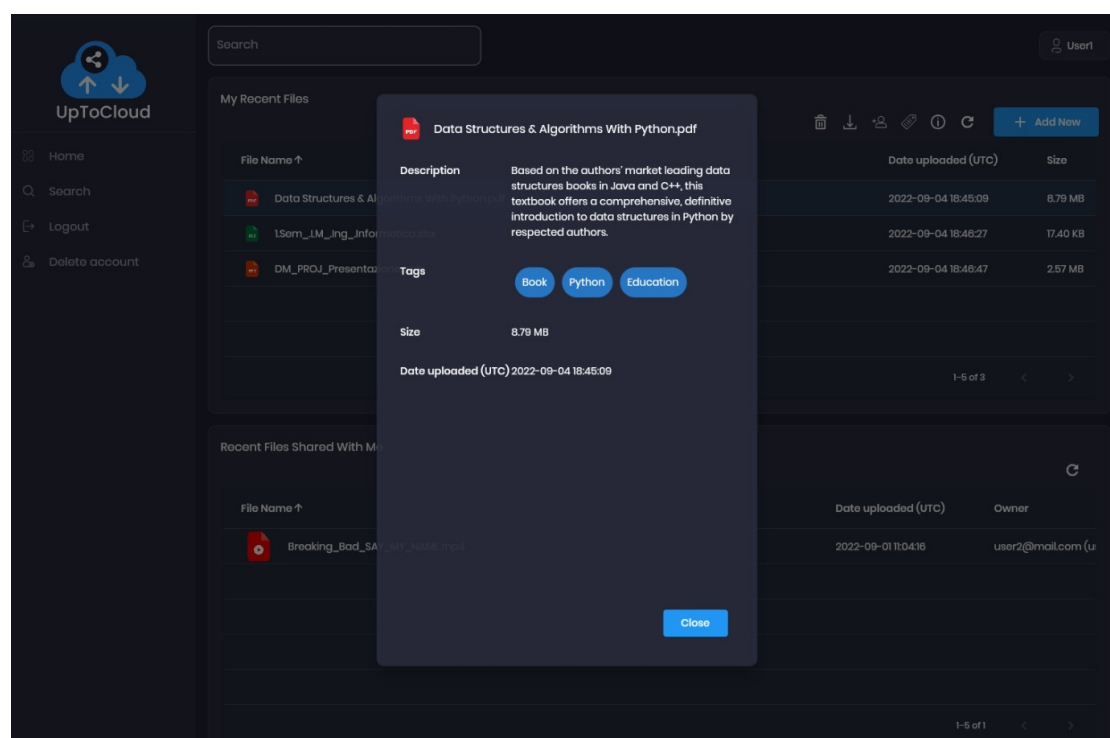


Figura 3.12: Visualizza maggiori informazioni

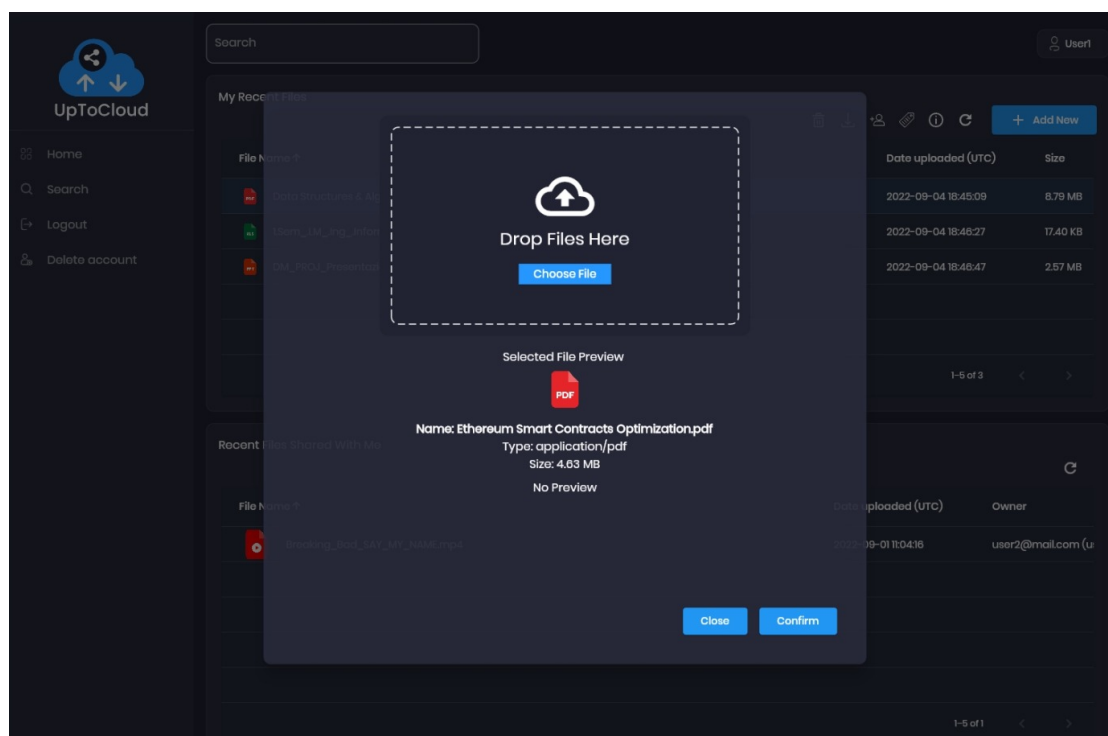


Figura 3.13: Carica un nuovo file

Inoltre, è stato previsto un menù laterale che consente di passare dalla HomePage alla pagina di ricerca, eliminare l'account o effettuare logout.

Pagina di ricerca

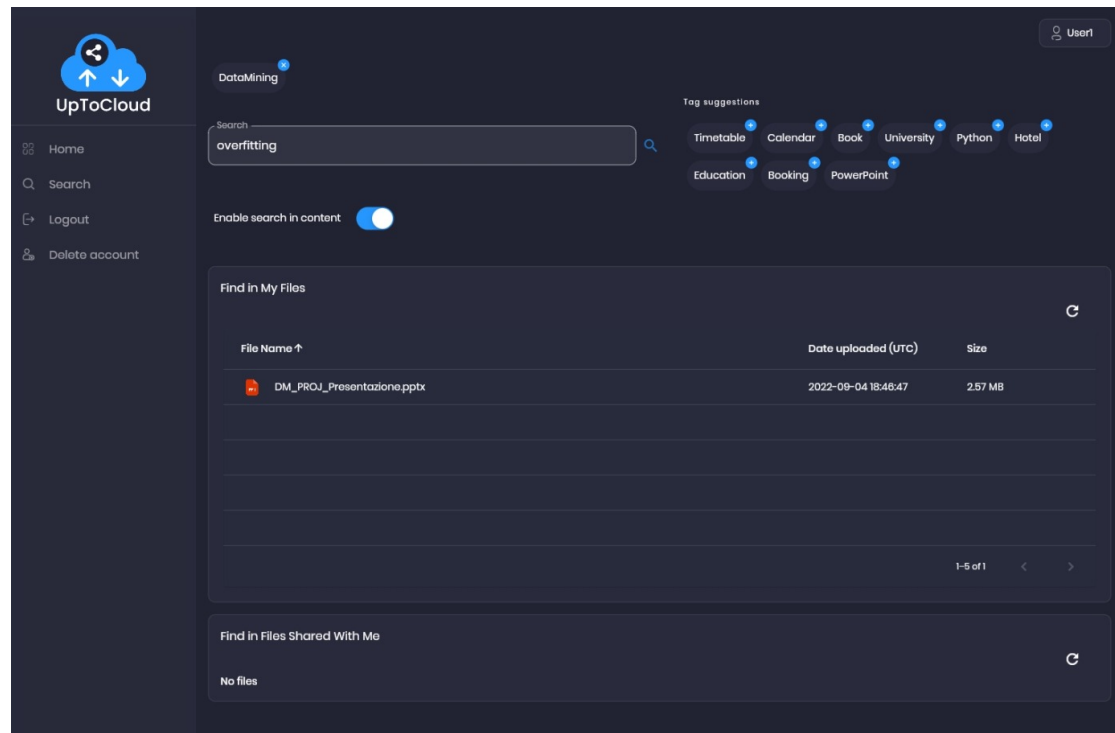


Figura 3.14: Pagina di ricerca

La pagina di ricerca prevede due tabelle (fig. 3.14) dove vengono i risultati, i quali sono suddivisi in modo analogo alla HomePage. Nella parte superiore è presente un campo di ricerca, affiancato da tag di ricerca suggeriti dal sistema sulla base dell'input. Abilitando la ricerca nel contenuto dei file è possibile trovare documenti mediante parole chiave che sono presenti all'interno degli stessi, anche avendo a che fare con file di tipo pdf, pptx, ecc...

Inoltre, come si può vedere in figura 3.15, il campo di ricerca ha la funzionalità di autocompletamento dell'input per facilitare la ricerca.

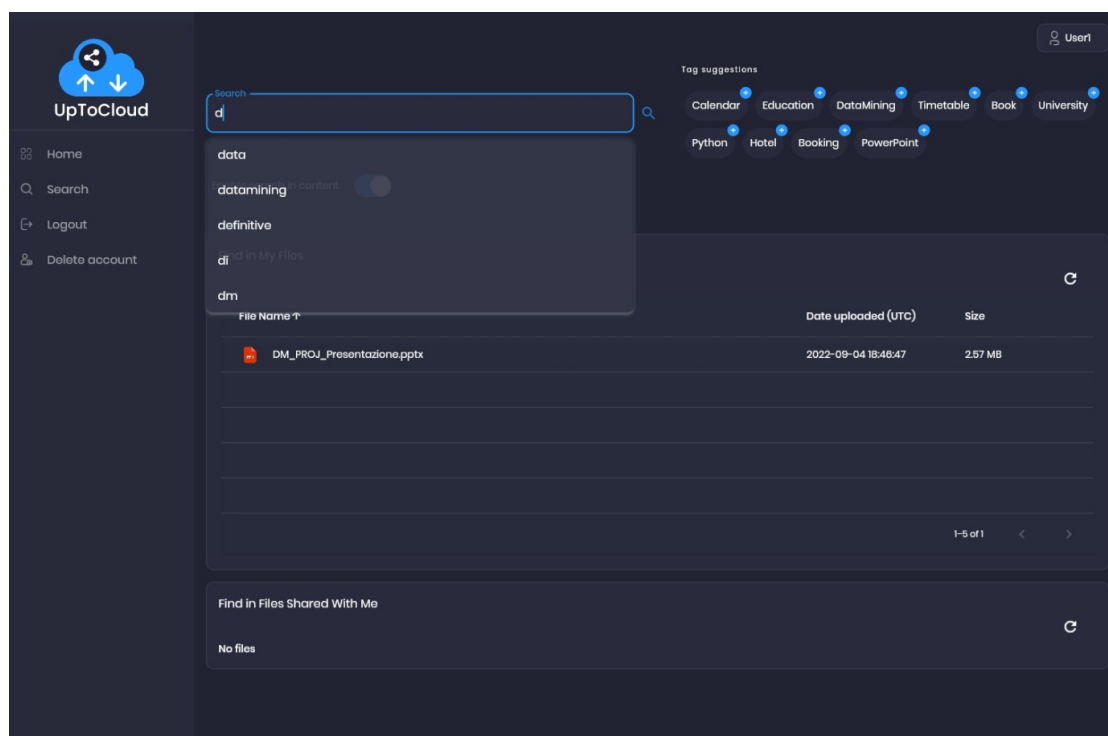


Figura 3.15: Autocompletamento

In conclusione, vengono affrontati alcuni punti critici del sistema ed eventuali sviluppi futuri.

4.1 Limiti e possibili sviluppi futuri

L'utilizzo delle risorse di calcolo e di virtualizzazione di Azure permettono di scalare facilmente l'applicazione, utilizzando configurazione della VM più performanti o utilizzando istanze multiple per distribuire il carico di lavoro. Inoltre, è possibile configurare delle politiche di scaling automatico, in modo che l'aggiunta o la rimozione di risorse sia modulato in base alle richieste. L'implementazione della ricerca tramite Cognitive Search permette di ampliare le funzionalità di ricerca in modo semplice attraverso strumenti potenti e pronti all'uso. Ad esempio, si potrebbe pensare di configurare Cognitive Search per la ricerca su immagini. Inoltre, si potrebbe pensare di implementare la gestione gerarchica dei documenti permettendo all'utente di creare directory e subdirectory.

Passando ai punti critici del sistema, l'utilizzo di database all'interno di container Docker in produzione potrebbe non essere la soluzione migliore. Nonostante l'utilizzo appropriato dei volumi, infatti, la gestione di un database containerizzato risulta più difficile e più rischiosa. Una soluzione migliore potrebbe essere quella di utilizzare database gestiti da un Cloud provider (in questo caso Azure), che consentirebbe di ridurre al minimo la complessità di gestione, come ad esempio l'esecuzione di **backup** regolari ed il **ridimensionamento**. In ogni caso, una volta configurato il database, questo problema è facilmente risolvibile grazie all'utilizzo del pattern Data Access Object (DAO). Infatti, avendo a disposizione il DAO layer che fornisce un'interfaccia astratta dei dati è necessario modificare l'endpoint della fonte dati nel file di configurazione *application.properties* del back-end Spring.