TEDx TOk

Progetto parte 5:
Aggiunte e revisioni

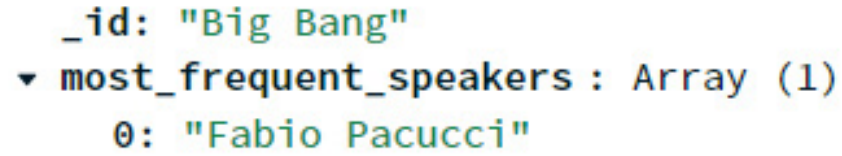
Mazzoleni Gabriele – 1079514
Masinari Gabriele - 1079692

Approfondimenti pt 2: PySpark

Per l'approfondimento della sezione del progetto riguardante i JOB ETL tramite PySpark, abbiamo deciso di implementare 2 nuovi job:
il primo è utilizzato per la creazione di un nuovo database MongoDB contenente i dati di login per l'applicazione TedTok (dati che verranno ripresi in seguito);
il secondo opera invece sui dati aggregati del dataset dei tag sviluppato in precedenza, calcolando per ogni tag lo speaker che ha tenuto il maggior numero di Talk.

TedTok_Log_In_Database

Get_Most_common_presenter_for_each_topic



TedTok_Log_In_Database

```python
##### FROM FILES
tedx_dataset_path = "s3://tedtok-2024-log-data/log.csv"

###### READ PARAMETERS
args = getResolvedOptions(sys.argv, ['JOB_NAME'])

##### START JOB CONTEXT AND JOB
sc = SparkContext()

glueContext = GlueContext(sc)
spark = glueContext.spark_session

job = Job(glueContext)
job.init(args['JOB_NAME'], args)

#### READ INPUT FILES TO CREATE AN INPUT DATASET
tedx_dataset = spark.read \
    .option("header","true") \
    .option("quote", "\"") \
    .option("escape", "\"") \
    .csv(tedx_dataset_path)

tedx_dataset.printSchema()
write_mongo_options = {
    "connectionName": "TEDx 2024 by GabTheBest",
    "database": "unibg_tedx_2024",
    "collection": "TedTok_Log_Ins",
    "ssl": "true",
    "ssl.domain_match": "false"}
from awsglue.dynamicframe import DynamicFrame
tedx_dataset_dynamic_frame = DynamicFrame.fromDF(tedx_dataset, glueContext, "nested")

glueContext.write_dynamic_frame.from_options(tedx_dataset_dynamic_frame, connection_type="mongodb", connection_options=write_mongo_options)
```

Il file log.csv contiene una breve lista di elementi con mail, username e password generici

Get_Most_common_presenter_for_each_topic

```python
tags_dataset_data= tags_dataset.join(tedx_dataset, tags_dataset.id == tedx_dataset.id, "left") \
    .select(tags_dataset["tag"], tedx_dataset["id"].alias("talk_id"),tedx_dataset["title"].alias("talk_title"),tedx_dataset["speakers"].alias("talk_speaker"))

# GROUP BY TAGS AND SPEAKERS
speaker_counts = tags_dataset_data.groupBy("tag", "talk_speaker").count()

#NOW, WE CAN FIND THE MOST FREQUENT SPEAKER FOR EACH TAG
window_spec = Window.partitionBy("tag").orderBy(col("count").desc())
#WE RANK ALL SPEAKERS AND THEN ONLY KEEP THE ONE IN FIRST POSITION
ranked_speakers = speaker_counts.withColumn("rank", row_number().over(window_spec)).filter(col("rank") == 1).drop("rank", "count")

# CREATE THE AGGREGATE MODEL, ONLY SHOWING TAG NAME AND MOST FREQUENT SPEAKER
tags_dataset_agg = ranked_speakers.groupBy(col("tag").alias("_id")) \
    .agg(collect_list(col("talk_speaker")).alias("most_frequent_speakers"))

tags_dataset_agg.printSchema()



# CREATE A NEW MONGODB COLLECTION
write_mongo_options = {
    "connectionName": "TEDx 2024 by GabTheDest",
    "database": "unibg_tedx_2024",
    "collection": "Topics_most_common_speakers",
    "ssl": "true",
    "ssl.domain_match": "false"}
from awsglue.dynamicframe import DynamicFrame
tags_dataset_dynamic_frame = DynamicFrame.fromDF(tags_dataset_agg, glueContext, "nested")

glueContext.write_dynamic_frame.from_options(tags_dataset_dynamic_frame, connection_type="mongodb", connection_options=write_mongo_options)
```

Viene prima eseguito un group by e un count sugli speakers per i vari tag, per poi applicare l'ordinamento in base al valore del conteggio e limitare la visualizzazione al solo "primo classificato"

Approfondimenti pt 3: Lambda Function

Per l'approfondimento della sezione del progetto riguardante le Lambda Function, abbiamo deciso di ampliare i servizi forniti creando funzioni che, facendo riferimento al database di log-in visto al punto precedente, permettono il log-in, cioè la convalida dell'accesso con dati già presenti nel db e il sign-in, cioè l'aggiunta di nuovi dati per utenti all'interno del db. Queste due LF sono poi state riprese nell'applicazione Flutter TedxTok per consentire la gestione degli accessi.

TedTok_LogIn

```
module.exports.log_in = (event, context, callback) => {
    context.callbackWaitsForEmptyEventLoop = false;
    console.log('Received event:', JSON.stringify(event, null, 2));
    let body = {};
    if (event.body) {
        body = JSON.parse(event.body);
    }

    // Validazione dei dati di input
    if (!body.mail && !body.password) {
        return callback(null, {
            statusCode: 400,
            headers: { 'Content-Type': 'text/plain' },
            body: 'Please insert an email and password.'
        });
    }

    if (!body.mail) {
        return callback(null, {
            statusCode: 400,
            headers: { 'Content-Type': 'text/plain' },
            body: 'Missing email.'
        });
    }

    if (!body.password) {
        return callback(null, {
            statusCode: 400,
            headers: { 'Content-Type': 'text/plain' },
            body: 'Missing Password.'
        });
    }

    connect_to_db().then(() => {
        console.log('=> Accessing user data');
        appUser.findOne({ "Mail": body.mail, "Password": body.password })
            .then(user => {
                if (!user) {
                    console.log('=> User not found');
                    return callback(null, {
                        statusCode: 404,
                        headers: { 'Content-Type': 'text/plain' },
                        body: 'User not found.'
                    });
                }
                console.log('=> User found:', user);
                callback(null, {
                    statusCode: 200,
                    headers: { 'Content-Type': 'application/json' },
                    body: JSON.stringify({"Message": "Success", "Mail": user.Mail, "Username": user.Username, "Password": user.Password})
                });
            })
            .catch(err => {
                console.error('Database query failed:', err);
                callback(null, {
                    statusCode: 500,
                    headers: { 'Content-Type': 'text/plain' },
                    body: 'Could not fetch the user. ' + err.message
                });
            });
    }).catch(err => {
        console.error('Database connection failed:', err);
        callback(null, {
            statusCode: 500,
            headers: { 'Content-Type': 'text/plain' },
            body: 'Database connection failed. ' + err.message
        });
    });
};
```

La funzione richiede come parametri una mail e una password;
se sono già nel database vengono restituiti tutti i dati (quindi anche l'username) e un messaggio di successo, altrimenti viene restituito il messaggio che l'utente non è stato trovato.

esempio di INPUT

```
{
    "mail":"john.doe@example.com",
    "password":"password123"
}
```

esempio di OUTPUT

```
{
    "Message": "Success",
    "Mail": "john.doe@example.com",
    "Username": "JohnDoe",
    "Password": "password123"
}
```

TedTok_SignIn

```
connect_to_db().then(() => {
    console.log('=> Accessing user data');
    appUser.findOne({ "Mail": body.mail})
        .then(user => {
            if (user) { //la mail è già nel db
                console.log('=> Mail is already in use');
                return callback(null, {
                    statusCode: 404,
                    headers: { 'Content-Type': 'text/plain' },
                    body: 'Mail is already in use.'
                });
            }
            console.log('->mail is free', user);
            //codice per inserire il nuovo utente nel db
            const newUser = new appUser({
                _id: new mongoose.Types.ObjectId().toString(), // Genera un nuovo ID unico
                Mail: body.mail,
                Username: body.username,
                Password: body.password
                //per un'applicazione a fini commerciali servirebbe eseguire hashing sulla password, ma per i nostri fini va bene così
            });

            newUser.save()
                .then(() => {
                    console.log('=> New user created');
                    return callback(null, {
                        statusCode: 201,
                        headers: { 'Content-Type': 'application/json' },
                        body: JSON.stringify({ message: 'User created successfully' })
                    });
                })
        })
        .catch(err => {
            console.error('Database query failed:', err);
            callback(null, {
                statusCode: 500,
                headers: { 'Content-Type': 'text/plain' },
                body: 'Could not fetch the user. ' + err.message
            });
        });
}).catch(err => {
    console.error('Database connection failed:', err);
    callback(null, {
        statusCode: 500,
        headers: { 'Content-Type': 'text/plain' },
        body: 'Database connection failed. ' + err.message
    });
});
```

La funzione richiede come parametri una mail, un username e una password (la verifica dell'inserimento non è inclusa nella slide per ragioni di spazio); se la mail è già nel database si comunica un messaggio di errore, altrimenti si inserisce un nuovo elemento nella collezione MongoDB e si restituisce il messaggio di operazione eseguita con successo.

Approfondimenti pt 4: applicazione Flutter

Per l'approfondimento della sezione del progetto riguardante l'utilizzo di Flutter, abbiamo implementato nuove funzionalità all'applicazione TedxTok, che avevamo previsto nel Trello creato all'inizio del progetto.

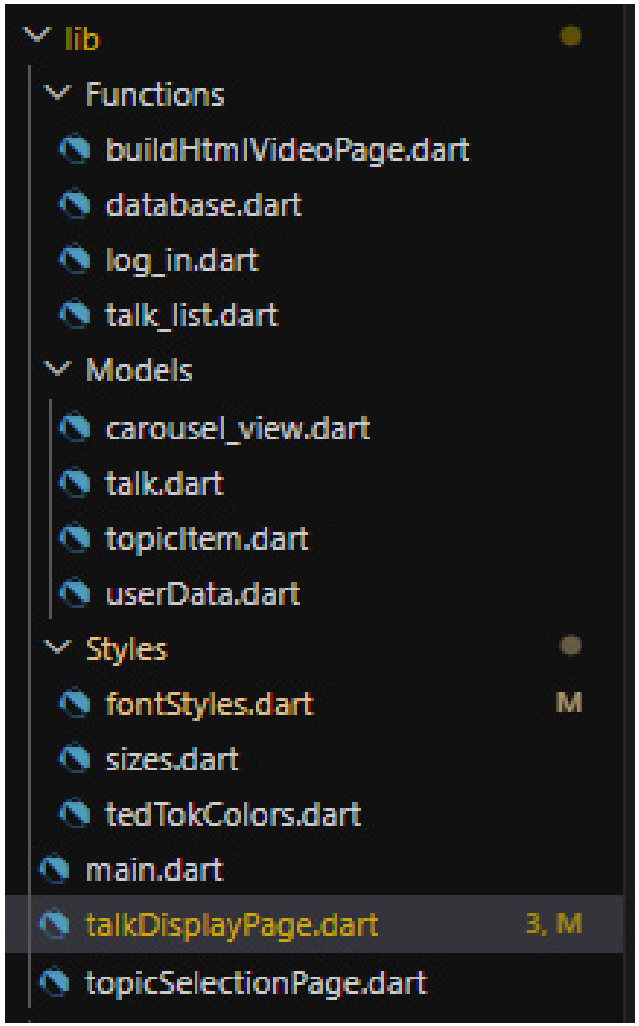
- Ritaglio dei video: la visualizzazione di ciascun video è limitata a un minuto da un timer, che al timeout causa lo scorrimento automatico al video successivo.
- Estensione della selezione dei tag: è stata effettuata la connessione al database per visualizzare tutti i tag; inoltre, è stato implementato il filtro di ricerca che permette di inserire una parola chiave restituendo i tag corrispondenti.
- Funzionalità di log-in, sign-in e log-out, attraverso le Lambda Function presentate al punto precedente.

TedxTok: struttura dell'app

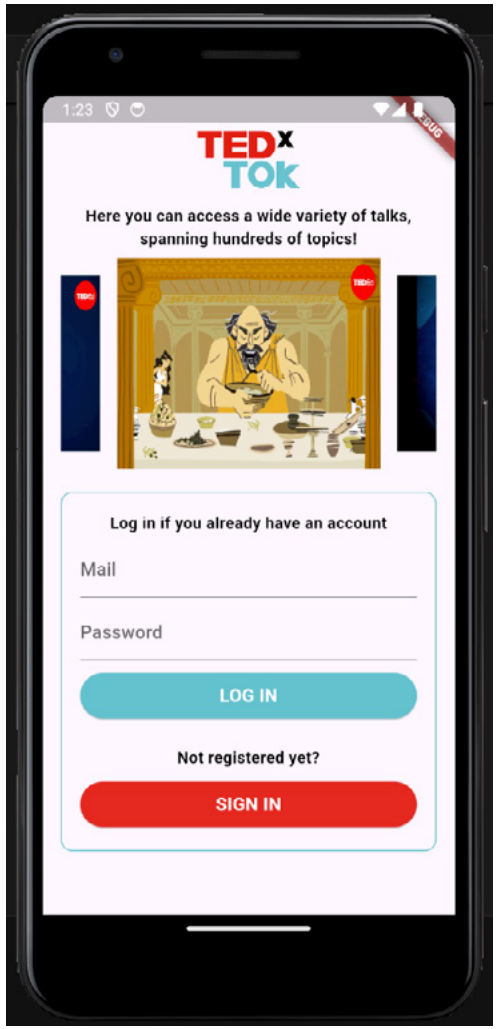
L'applicazione Flutter è strutturata in questo modo:

- Functions: cartella contenente le funzioni che richiamano le API delle Lambda Function e/o eseguono altre elaborazioni utili;
- Models: cartella contenente i modelli/le classi utilizzate nell'applicazione;
- Styles: cartella contenente file con categorie standardizzate di colori, dimensioni e stili di testo, a cui fanno riferimento le varie pagine dell'app.

I tre file rimanenti rappresentano le tre pagine principali di cui è composta l'app, che sono illustrate nelle prossime slide.

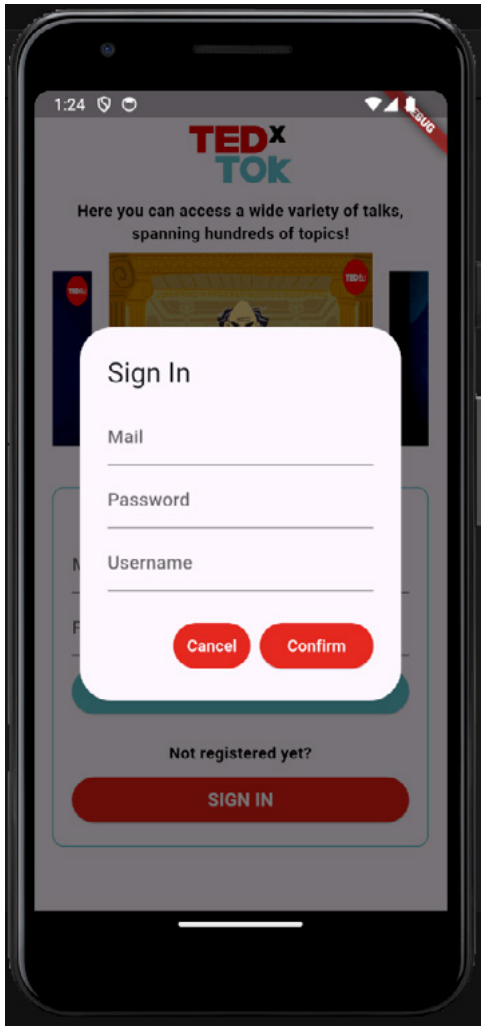


TedxTok: main.dart



Le immagini presenti sono racchiuse all'interno di un widget di tipo carousel, che consente di far scorrere un numero limitato di immagini da noi scelte.

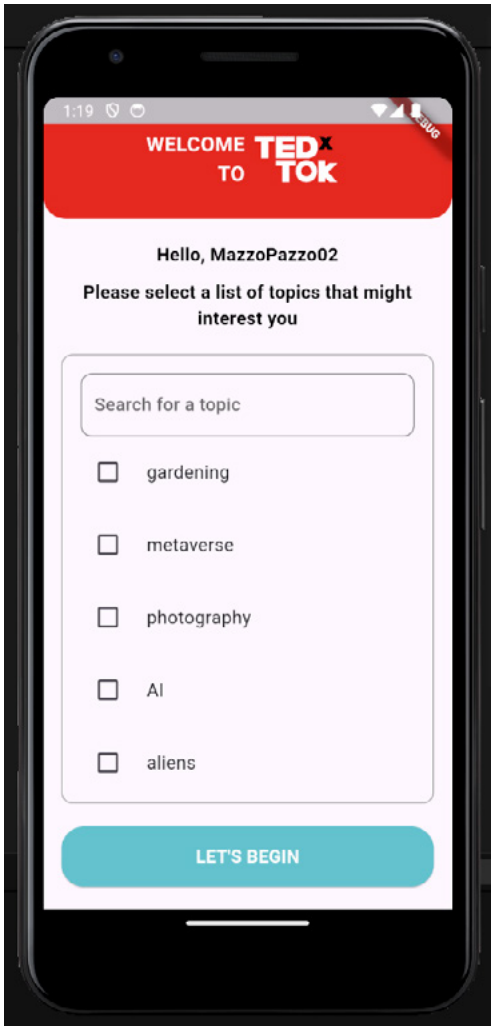
In caso di inserimento di dati errati nel log-in compare un messaggio di errore e annulla quanto inserito.



Per l'inserimento dei dati del sign-in, abbiamo utilizzato un widget di tipo modal.

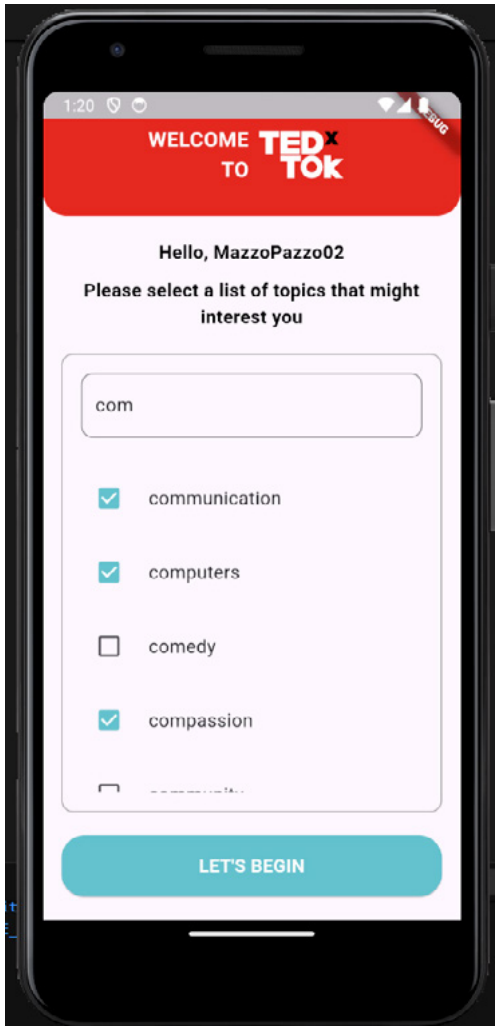
In questo modo è possibile creare il proprio account e accedervi direttamente. In caso di errore, il comportamento è il medesimo della schermata di log-in.

TedxTok: topicSelectionPage.dart



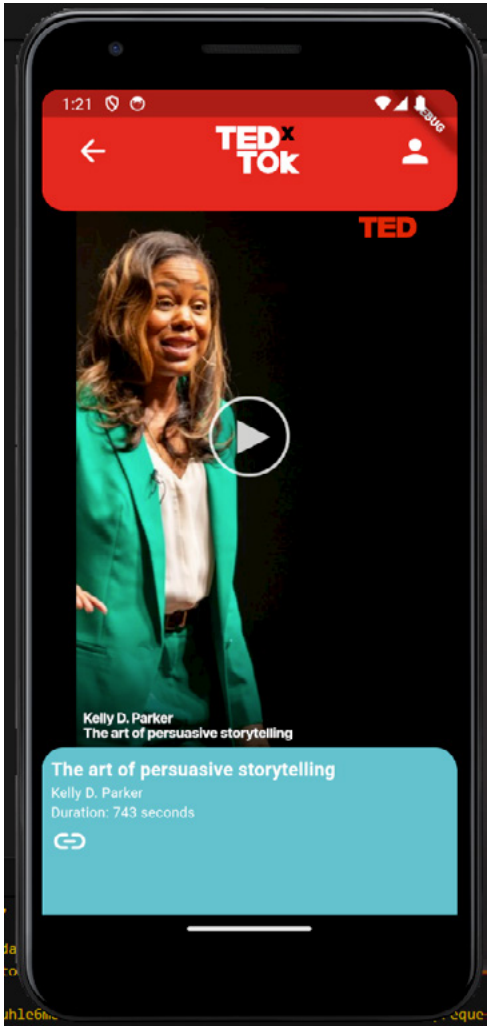
La pagnia di selezione dei tag ora mostra l'username dell'utente che ha effettuato il log-in.

I topic selezionabili sono tutti quelli presenti nel dataset MongoDB "TedTok_tags"



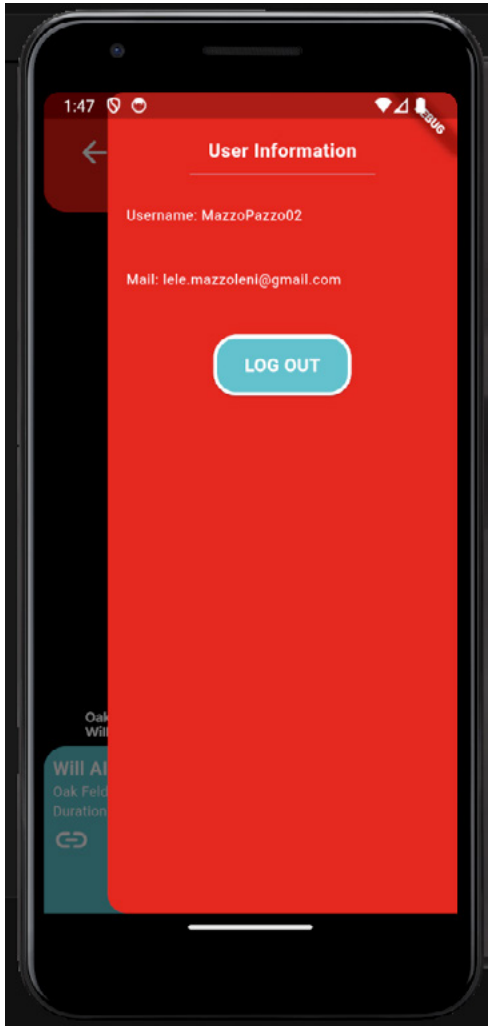
L'utente ha la possibilità di inserire una parola chiave per cercare i topic che più gli interessano, anche solo digitando le prime lettere.

TedxTok: talkDisplayPage.dart



Il collegamento alla pagina Tedx è stato posizionato in basso a sinistra assieme ai dati del talk;
in alto a destra è posizionata l'icona del profilo utente.

Il video player va avviato dall'utente come in precedenza, ma si interrompe dopo 1 minuto, con lo scorrimento al talk successivo.



Nel profilo utente è possibile visualizzare i propri dati di log-in attraverso un widget drawer ed effettuare il log-out ritornando alla pagina di log-in.

Per chiudere il drawer e tornare alla visione dei talk è sufficiente scorrere verso destra sullo schermo.

TEDx TOk

Trello board

GitHub