



# TEDxComments



Lambda Functions - API



Next



# Table of contents



01 Lambda Functions

02 User Experience

03 Potential Flaws

04 Future Development



Back



Next

# Insert Comment - Lambda Functions



```
connect_to_db().then(() => {  
  talk.findOne({  
    _id: body.video_id  
  }).then((result) => {  
    let comment = {  
      comment_id: crypto.randomUUID(),  
      user_id: body.user_id,  
      timestamp: body.timestamp,  
      title: body.title,  
      body: body.body,  
      upvote: 0  
    };  
  
    if (body.type === "info") {  
      result.comments.info.push(comment);  
    } else if (body.type === "disc") {  
      result.comments.disc.push(comment);  
    } else if (body.type === "extra") {  
      result.comments.extra.push(comment);  
    }  
  
    return result.save();  
  })  
})
```

Per inserire un commento su un video, tramite *video\_id* si risale al video a cui il commento farà riferimento, e poi il commento verrà inserito nel vettore che combacia con la sua tipologia.



Back



Next

# Result

GET

https://r5juowrxdj.execute-api.us-east-1.amazonaws.com/default/TxC\_InsertComment

Params Authorization Headers (7) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON**

```
1 {}
2   ... "video_id": "138559",
3   ... "user_id": "utente0",
4   ... "timestamp": 260,
5   ... "title": "Nice video!!!",
6   ... "body": "Sounds gothic to me",
7   ... "type": "extra"
8 }
```

Body Cookies Headers (7) Test Results

Pretty

Raw

Preview

Visualize

JSON

≡

```
1 {
2   "message": "success"
3 }
```



```
▼ comments : Object
  ► info : Array (empty)
  ► disc : Array (empty)
  ▼ extra : Array (1)
    ▼ 0: Object
      comment_id : "9016246f-9591-4725-9c74-b953c7440614"
      user_id : "utente0"
      timestamp : 260
      title : "Nice video!!!"
      body : "Sounds gothic to me"
      upvote : 0
```

Back



Next

# Upvote Comment - Lambda Functions



```
connect_to_db().then(() => {  
  talk.findOne({ _id: body.video_id }).then((result) => { //trova il video a cui il commento appartiene  
    let comment = null; //variabile per salvare l'elemento che contiene il riferimento al commento  
  
    if (body.type === "info") { //sceglie in quale categoria di commenti da cercare  
      comment = result.comments.info.find((element)=>{return element.comment_id == body.comment_id});  
    }  
    else if (body.type === "disc") {  
      comment = result.comments.disc.find((element)=>{return element.comment_id == body.comment_id});  
    }  
    else if (body.type === "extra") {  
      comment = result.comments.extra.find((element)=>{return element.comment_id == body.comment_id});  
    }  
  
    comment.upvote++; //aggiorna il numero di upvote  
    return result.save(); //salva il risultato nel database  
  })  
})
```

Per permettere agli utenti di votare positivamente i commenti, cerchiamo il video a cui il commento appartiene tramite *video\_id*, poi selezioniamo tra i suoi commenti il commento giusto con *comment\_id*

Poi cerchiamo verifichiamo a che categoria appartiene il commento. Una volta individuato univocamente, si aggiorna il suo numero di *upvote*.

Back



Next



# Result



GET ▼ https://xphfyvpsj.execute-api.us-east-1.amazonaws.com/default/TxC\_UpvoteComment

Params Authorization Headers (7) **Body** ● Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▼

```
1 {  
2   ... "video_id": "138559",  
3   ... "comment_id": "9016246f-9591-4725-9c74-b953c7440614",  
4   ... "type": "extra"  
5 }
```

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize **JSON** ▼

```
1 {  
2   "message": "success"  
3 }
```

```
▼ comments : Object  
  ► info : Array (empty)  
  ► disc : Array (empty)  
  ▼ extra : Array (1)  
    ▼ 0: Object  
      comment_id : "9016246f-9591-4725-9c74-b953c7440614"  
      user_id : "utente0"  
      timestamp : 260  
      title : "Nice video!!!"  
      body : "Sounds gothic to me"  
      upvote : 1
```

Back



Next

# GetWatchNextById - Lambda Functions



```
connect_to_db().then(() => {
  talk.findOne({ _id: body.video_id }).then((result) => {
    callback(null, {
      statusCode: 200,
      body: JSON.stringify({ result: result })
    });
  }).catch((error) => {
    callback(null, {
      statusCode: 400,
      body: JSON.stringify({
        message: "could not get the talk"
      })
    });
  });
});
```



```
const talk_schema = new mongoose.Schema({
  _id: String,
  title: String,
  url: String,
  description: String,
  speakers: String,
  comments: {
    info: Array,
    disc: Array,
    extra: Array
  }
}, { collection: 'tedx_watch_next' });

module.exports = mongoose.model('talk', talk_schema);
```

Per permettere agli utenti di ottenere i video correlati di un talk, cerchiamo il video con l'id specificato tramite *video\_id*. Anzi che dalla collection i dati vengono letti dalla materialized view.

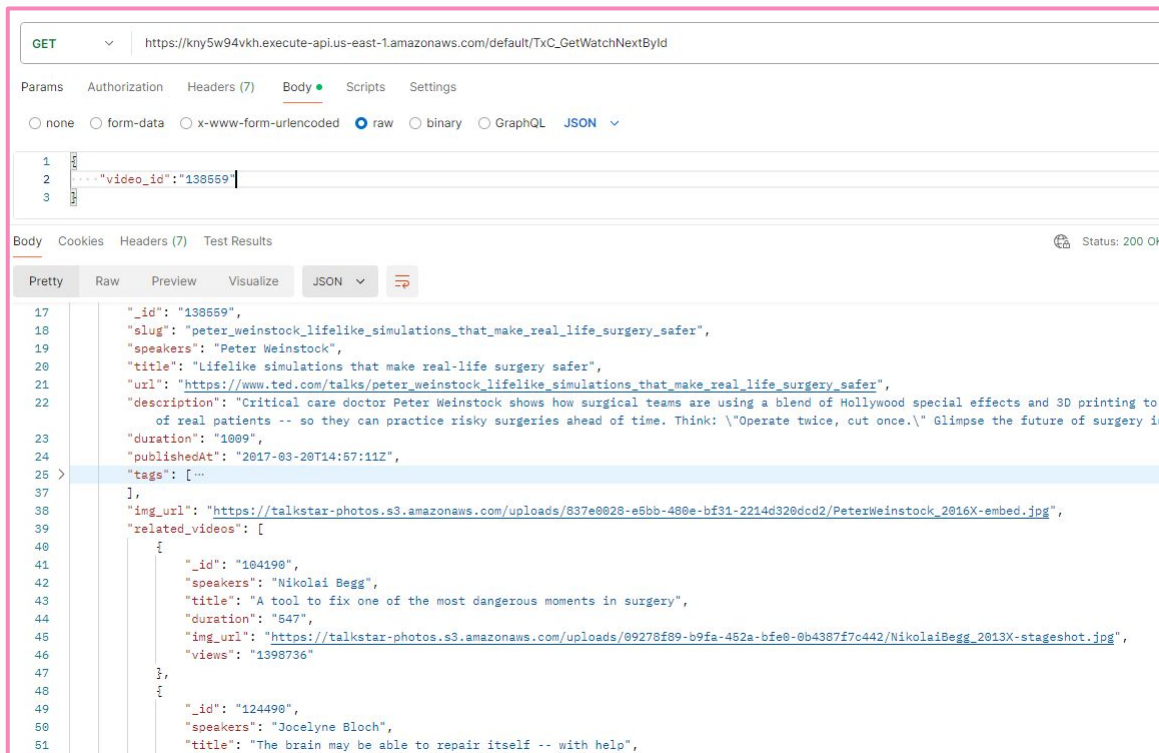


Back



Next

# Result



GET [https://kny5w94vkh.execute-api.us-east-1.amazonaws.com/default/TxC\\_GetWatchNextById](https://kny5w94vkh.execute-api.us-east-1.amazonaws.com/default/TxC_GetWatchNextById)

Params Authorization Headers (7) **Body** Scripts Settings

☐ none ☐ form-data ☐ x-www-form-urlencoded ☒ raw ☐ binary ☐ GraphQL **JSON** ▾

```
1 {
2   "video_id": "138559"
3 }
```

Body Cookies Headers (7) Test Results Status: 200 OK

Pretty Raw Preview Visualize **JSON** ▾

```
17 {
18   "_id": "138559",
19   "slug": "peter_weinstock_lifelike_simulations_that_make_real_life_surgery_safer",
20   "speakers": "Peter Weinstock",
21   "title": "Lifelike simulations that make real-life surgery safer",
22   "url": "https://www.ted.com/talks/peter_weinstock_lifelike_simulations_that_make_real_life_surgery_safer",
23   "description": "Critical care doctor Peter Weinstock shows how surgical teams are using a blend of Hollywood special effects and 3D printing to
24     of real patients -- so they can practice risky surgeries ahead of time. Think: \"Operate twice, cut once.\" Glimpse the future of surgery i
25   "duration": "1009",
26   "publishedAt": "2017-03-20T14:57:11Z",
27   "tags": [ ...
28 ],
29   "img_url": "https://talkstar-photos.s3.amazonaws.com/uploads/837e0628-e5bb-480e-bf31-2214d320dcd2/PeterWeinstock_2016X-embed.jpg",
30   "related_videos": [
31     {
32       "_id": "104190",
33       "speakers": "Nikolai Begg",
34       "title": "A tool to fix one of the most dangerous moments in surgery",
35       "duration": "547",
36       "img_url": "https://talkstar-photos.s3.amazonaws.com/uploads/09278f89-b9fa-452a-bfe0-0b4387f7c442/NikolaiBegg_2013X-stageshot.jpg",
37       "views": "1390736"
38     },
39     {
40       "_id": "124490",
41       "speakers": "Jocelyne Bloch",
42       "title": "The brain may be able to repair itself -- with help",
43     }
44   ]
45 }
```



Back



Next



# UX - User Experience



02:57 - Analizzando questo...

02:46 - Come...

Watch Next

- 
- 
-

Back



Next



# Potential Flaws



01

02

03



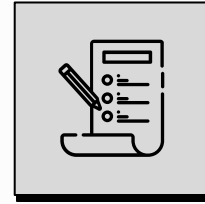
## Collisioni ID

Utilizzando il metodo *crypto* per creare i *comment\_id* è possibile (per quanto estremamente improbabile) che potrebbero esserci delle collisioni



## Autenticazione

Con l'implementazione attuale, le API sono pubbliche e chiunque può creare e votare i commenti senza autenticarsi



## Update

Nel caso di cambiamenti alla struttura dei dati, aggiornare i dati potrebbe essere complicato



Back



Next



# Future development



Inserimento di layer	Autenticazione	Controlli
Disaccoppiare maggiormente il db dall'handler	Configurare token per regolare l'accesso alle API Gateway	Migliorare l'usabilità delle API e la sicurezza inserendo controlli per limitare gli upvote



Back



Next



# Thanks!

Colpani Filippo - 1078874  
Foglieni Luca - 1081399



## TEDxComments

**Credits:** This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**

Back

