DeepL
Abbonati a DeepL Pro per tradurre file di maggiori dimensioni.
Per ulteriori informazioni, visita www.DeepL.com/pro.

Programmazione di sistemi distribuiti

ANNO 2024/25

Laboratorio 1

I due argomenti principali che vengono affrontati in questa attività di laboratorio sono:

- progettazione di schemi JSON;
- progettazione di API REST.

Per avere un'esperienza completa, un'implementazione delle API REST progettate dovrà anche essere sviluppata, completando un'implementazione già esistente.

Il contesto in cui si svolge questa attività di laboratorio è un servizio di *Film Manager*, dove gli utenti possono tenere traccia dei film che hanno visto e delle recensioni che hanno scritto per essi. Se avete frequentato il corso *Web Applications I* erogato dal Politecnico di Torino nell'A.A. 2023/2024, potreste già conoscere alcuni dei concetti principali alla base di *Film Manager*, e siete invitati a riutilizzare/estendere quanto fatto per i laboratori di quel corso per svolgere questa attività. Altrimenti, potete consultare la documentazione dei laboratori di Applicazioni Web I (https://github.com/polito- webapp1/lab-2024) e utilizzare, come punto di partenza per la vostra attività, la soluzione dell'ultimo laboratorio del corso di Applicazioni Web I (https://github.com/polito-webapp1/lab- 2024/tree/main/lab11-authentication).

Gli strumenti consigliati per lo sviluppo della soluzione sono:

- *Visual Studio Code* (https://code.visualstudio.com/) per la validazione dei file JSON rispetto agli schemi e per l'implementazione delle API REST;
- *Editor OpenAPI (Swagger)*, estensione di *Virtual Studio Code*, per la progettazione di API REST;
- *Swagger Editor* (https://editor.swagger.io/) per la generazione automatica di uno stub del server;
- *PostMan* (https://www.postman.com/) per testare il servizio web che implementa le API REST;
- *DB Browser for SQLite* (https://sqlitebrowser.org/) per la gestione del database.

Il linguaggio Javascript e il framework Express (https://www.npmjs.com/package/express) della piattaforma node.js sono consigliati per sviluppare l'implementazione del servizio web RESTful.

1. Progettazione di schemi JSON

La prima attività riguarda la progettazione di schemi JSON per tre strutture di dati fondamentali di *Film Manager*, ossia gli *utenti* che vogliono gestire le loro liste di film tramite questa applicazione, i *film* che hanno visto e/o recensito, gli *inviti a recensire* che un utente può rivolgere a un altro utente. Tutte le scelte progettuali per le quali non ci sono indicazioni specifiche sono lasciate agli studenti.

Una struttura di dati *utente* è composta dai seguenti campi:

- *id*: identificatore unico della struttura dati dell'utente nel servizio *Film Manager* (obbligatorio);
- *nome*: nome utente dell'utente;
- *email*: indirizzo email dell'utente, che deve essere utilizzato per l'autenticazione al servizio (obbligatorio, deve essere un indirizzo email valido);
- *password*: la password dell'utente, che deve essere utilizzata per l'autenticazione al sistema.
  (la password deve essere lunga almeno 6 caratteri e al massimo 20 caratteri).
  lungo).

La struttura dei dati di un *film* è composta dai seguenti campi:

- *id*: identificatore unico della struttura dati *del film* nel servizio *Film Manager* (obbligatorio);
- *titolo*: titolo testuale del film (obbligatorio);
- *proprietario:* l'id del proprietario della struttura dati del film, cioè l'utente che l'ha creata (obbligatorio);
- *private*: proprietà booleana, impostata su true se la struttura dati *del film* è contrassegnata come privata, su false se è pubblica (obbligatoria). Una struttura di dati *del film* è detta privata se solo
  il suo proprietario può accedervi, pubblico se ogni utente può accedervi;
- *watchDate*: la data in cui il film è stato visto dal proprietario, espressa nel formato YYYY-MM-DD (YYYY è l'anno, MM è il mese, DD è il giorno). Questo
  può essere inclusa nella struttura dei dati *del film* solo se *private* è vero;
- *rating*: un numero intero non negativo (massimo 10) che esprime la valutazione che il proprietario ha dato al film. Questa proprietà può essere inclusa nella struttura dei dati *del film*
  solo se *privato* è vero;
- *favorite*: una proprietà booleana, impostata su true se il film è tra i preferiti dell'utente, false altrimenti (valore predefinito: false). Questa proprietà può essere inclusa nel file
  struttura di dati *filmati* solo se *privato* è vero.

La struttura dei dati di una *recensione* è composta dai seguenti campi:

- *filmId*: identificativo univoco del film per il quale è stato emesso un invito a recensire (obbligatorio);
- *reviewerId*: identificativo univoco dell'utente che ha ricevuto l'invito alla revisione (obbligatorio);
- *completed*: proprietà booleana, impostata su true se la revisione è stata completata, false altrimenti (obbligatoria)*;*
- *reviewDate*: la data in cui la revisione è stata completata dall'utente invitato, espressa come stringa nel formato YYYY-MM-DD (YYYY è l'anno, MM è il mese), DD è il giorno). Questa proprietà può essere inclusa solo se *completed* è true e in tal caso è obbligatoria;
- *rating*: un numero intero non negativo (massimo 10) che esprime la valutazione della recensione completata dall'utente. Questa proprietà può essere inclusa solo se *completato* è vero, e in tal caso è obbligatorio;
- *recensione*: una descrizione testuale della recensione (non deve essere più lunga di 1000 caratteri). Questa proprietà può essere inclusa solo se *completed* è true, e in quel caso caso è obbligatorio

Lo standard JSON Schema che deve essere utilizzato per questa attività è la Bozza 7 (http://json- schema.org/draft-07/schema#).

Dopo aver completato la progettazione degli schemi, si suggerisce di scrivere alcuni file JSON come esempio e di convalidarli rispetto agli schemi in Visual Studio Code. In questo ambiente di sviluppo, gli errori di validazione vengono mostrati nell'editor e nella vista "Problema". È possibile accedere a questa vista in due modi alternativi:

- seguendo il percorso Visualizza → Problemi;
- premendo Ctrl+Maiusc +M.

2. Progettazione e implementazione di API REST

La seconda attività riguarda la progettazione di API REST per il servizio *Film Manager*. Specificare e documentare il progetto delle API REST mediante l'estensione "OpenAPI (Swagger) Editor" di Visual Studio Code. Per la progettazione, dovrete riutilizzare gli schemi sviluppati nella prima parte del compito, personalizzandoli per essere utilizzati nelle API REST. Quindi, il documento OpenAPI risultante può essere utilizzato come punto di partenza per sviluppare un'implementazione delle API REST progettate in modo semiautomatico: dopo aver importato il file

Il file OpenAPI nell'editor Swagger stand-alone (la versione online o quella installata localmente), può generare automaticamente uno stub del server, corrispondente al progetto, da riempire con l'implementazione delle funzionalità richieste. L'implementazione di alcune delle funzionalità necessarie è già disponibile nella soluzione dell'ultima attività di laboratorio del corso di Applicazioni Web I. Siete invitati a riutilizzarle nella vostra implementazione, mentre dovrete implementare da soli le altre funzionalità.
Più in dettaglio, il servizio deve essere progettato e implementato secondo le seguenti specifiche.

Il servizio *Film Manager* consente agli utenti di tenere traccia delle informazioni sui film che hanno visto, o per i quali vogliono inviare un invito a recensire o per i quali devono effettuare una recensione. Due concetti chiave sono il *proprietario* del film e il *recensore* del film. Il primo è l'utente che crea la struttura dati del film nel servizio, il secondo è un utente che si occupa di effettuare una recensione del film, dopo essere stato invitato dal proprietario a farla. Il servizio mantiene in un database le informazioni sugli utenti abilitati al suo utilizzo e sui loro film. Ogni utente è autenticato tramite una password personale che, come consiglia la prassi, non viene memorizzata nel database. Al contrario, viene calcolato un hash salato della password e memorizzato nel database. Il sale deve essere casuale e lungo almeno 16 byte.

La maggior parte delle funzioni del servizio è accessibile solo agli utenti autenticati. Le uniche operazioni che possono essere utilizzate da chiunque senza autenticazione sono:

- l'operazione di autenticazione stessa (login);
- l'operazione per recuperare l'elenco di tutti gli elementi *del film* che sono contrassegnati come pubblici;
- l'operazione di recupero di un singolo elemento *filmico* pubblico;
- l'operazione per recuperare l'elenco delle recensioni di un *film* pubblico;
- l'operazione di recupero di una singola recensione di un *film* pubblico.

Per l'autenticazione, l'utente invia l'e-mail e la password al servizio, che verifica se queste credenziali sono corrette. Le e-mail utilizzate per l'autenticazione devono essere uniche nel servizio. Il meccanismo di autenticazione utilizzato dal servizio è un'autenticazione basata sulla sessione, in cui i dati della sessione sono salvati sul lato server e il client riceve una sessione identificata in un cookie. La gestione di questo meccanismo di autenticazione basato sulla sessione può essere implementata utilizzando il middleware Passport (http://www.passportjs.org/) e l'opzione *express-session* (https://www.npmjs.com/package/express-session).
Affinché questo meccanismo di autenticazione e la comunicazione autenticata siano sufficientemente sicuri, sarebbe opportuno

necessario che le API REST siano disponibili solo su HTTPS. Tuttavia, nell'implementazione prodotta per questo laboratorio, che non è destinata alla produzione, esporremo le API su HTTP, per facilitare il debug.

A differenza delle attività di laboratorio del corso "Applicazioni Web I", qui non si deve progettare un'API esplicita per il logout. Il logout deve invece essere gestito nel modo seguente:

- Quando un utente accede, la sessione autenticata deve durare al massimo 5 minuti. Dopo questo tempo, l'utente viene automaticamente disconnesso.
- Quando arriva una richiesta di accesso per un nuovo utente, il servizio Film Manager controlla se la richiesta è stata inviata da un utente già autenticato. In tal caso, l'utente precedentemente autenticato viene disconnesso e quindi il nuovo utente effettua il login. In caso contrario, il nuovo utente accede direttamente.

Un utente autenticato ha accesso alle operazioni CRUD per gli elementi *del film*:

- L'utente può creare un nuovo film. Se la creazione del filmato va a buon fine, il servizio gli assegna un identificativo unico. Il creatore del filmato ne diventa il proprietario. Se il filmato

  è contrassegnato come pubblico, l'invito a recensire non viene automaticamente assegnato al suo creatore o a qualsiasi altro utente. L'invito a recensire un film agli utenti è spiegato più avanti in questo documento.
- L'utente può recuperare un singolo filmato esistente se è soddisfatta una delle seguenti condizioni:
  1) il filmato è contrassegnato come pubblico;
  2) l'utente è il proprietario del film.

  L'utente può anche recuperare l'elenco di tutti i film che ha creato e l'elenco di tutti i film che è stato invitato a recensire.
- L'utente può aggiornare un filmato esistente, se ne è il proprietario. Tuttavia, questa operazione non consente di cambiare la visibilità da pubblica a privata (e viceversa).

  versa).
- L'utente può eliminare un filmato esistente, se ne è il proprietario.

Una caratteristica centrale del servizio *Film Manager* è l'emissione di inviti a recensire film pubblici agli utenti che hanno accesso al servizio. Le principali operazioni relative a questa funzione sono:

- Il proprietario di un filmato pubblico può inviare un invito a recensire a un utente (il recensore può essere il proprietario stesso).
- Il proprietario di un film può rimuovere un invito a recensire, se la recensione non è ancora stata completata dal recensore.

- Un recensore invitato per un film può contrassegnare la recensione come completata, aggiornando anche la data della recensione, la valutazione e la descrizione testuale della recensione.

Gli inviti alla revisione di un film possono essere inviati a più utenti contemporaneamente. Ogni utente a cui è stato inviato l'invito a recensire può contrassegnare la recensione di quel film come completata. *Suggerimento*: potrebbe essere necessario creare una nuova tabella nel database, per rappresentare questo tipo di relazione tra film e utenti, e definire alcune chiavi esterne in modo da collegare questa tabella con le tabelle che memorizzano i record relativi a utenti e film.

Infine, il servizio deve offrire un'operazione aggiuntiva per emettere automaticamente inviti a rivedere i film per i quali non è stato ancora emesso alcun invito, in modo tale che gli inviti agli utenti siano bilanciati. Mentre la progettazione di questa funzione è obbligatoria, la sua implementazione è facoltativa.

Ecco alcune raccomandazioni per la progettazione e lo sviluppo delle API REST:

- Quando si recupera un elenco di film, si raccomanda un meccanismo di paginazione, per limitare le dimensioni dei messaggi che il servizio invia.
- Quando gli utenti inviano un elemento *filmato* JSON al servizio (ad esempio, per la creazione del filmato nel database o per l'aggiornamento di un filmato esistente), questo pezzo di input di

  devono essere validati rispetto allo schema JSON corrispondente. *Suggerimento*: un middleware express.js suggerito per la validazione delle richieste rispetto agli schemi JSON è *express-json-validator-middleware* (https://www.npmjs.com/package/express- json-validator-middleware), basato sul modulo *ajv* (https://www.npmjs.com/ package/ajv).
- Il servizio deve essere conforme a HATEOAS (Hypermedia As The Engine of Application State). I collegamenti ipertestuali devono essere inclusi nelle risposte per consentire l'uso di link.

  La navigazione e le funzionalità devono essere autodescrittive. Ad esempio, quando il servizio *Film Manager* invia un *film* o un oggetto *utente* in formato JSON, dovrebbe includere un link che rimanda all'URI in cui la risorsa può essere recuperata con un'operazione GET. Inoltre, un client dovrebbe essere in grado di eseguire tutte le operazioni senza dover costruire gli URI.

Linee guida per lo sviluppo della soluzione

Come far funzionare lo stub del server

Dopo aver generato automaticamente lo stub del server utilizzando l'editor Swagger, è necessario eseguire alcune operazioni *di instradamento*. L'instradamento si riferisce alla determinazione del modo in cui un'applicazione risponde a una richiesta del client a un particolare endpoint, che è un URI (o percorso) e uno specifico metodo di richiesta HTTP (GET, POST e così via). In particolare, è necessario definire alcuni *metodi di rotta*: un metodo di rotta deriva da uno dei metodi HTTP ed è collegato a un'istanza della classe express.

Purtroppo, le rotte non sono gestite automaticamente nel meccanismo di generazione del codice fornito da Swagger Editor. Pertanto, prima di testare il server stub, è necessario introdurre i metodi di rotta necessari.

Supponiamo di dover mappare un metodo GET al percorso "/api/films/public"; inoltre, nel metodo della rotta corrispondente, la funzione di callback che deve essere invocata è *getPublicFilms*, che si trova nel file "controllers/Films.js" generato automaticamente da Swagger Editor.
Per gestire l'instradamento di questo metodo GET, il file da modificare è "index.js". In particolare, dopo aver creato l'oggetto Express *app*, è necessario eseguire due operazioni:

1) importando il modulo rappresentato da "controllers/Films.js" tramite la funzione *require* e ottenendo così un oggetto (*filmController*) che dà accesso alle funzioni esportate di "controllers/Films.js";
2) creare il metodo di instradamento per l'operazione GET e mappare il percorso "/api/films/public" al callback "filmController.getPublicFilms".

Queste due operazioni sono rappresentate nella seguente schermata, dove compaiono nelle righe 18-19. Confrontando questa schermata con il file "index.js", si può notare che queste due righe sono le uniche a essere state modificate.

Dopo aver installato i moduli node.js necessari, è possibile testare lo stub del server con Postman. A questo punto, si può procedere a popolare lo stub con il codice necessario a fornire le funzionalità precedentemente descritte in questo documento.

```
'use strict';

var path = require('path');
var http = require('http');
var oas3Tools = require('oas3-tools');
var serverPort = 3001;

/*** Swagger configuration ***/

var options = {
    routing: {
        controllers: path.join(__dirname, './controllers')
    },
};

var expressAppConfig = oas3Tools.expressAppConfig(path.join(__dirname, 'api/openapi.yaml'), options);
var app = expressAppConfig.getApp();

var filmController = require(path.join(__dirname, 'controllers/FilmsController'));
app.get('/api/films/public', filmController.getPublicFilms);

// Initialize the Swagger middleware

http.createServer(app).listen(serverPort, function() {
    console.log('Your server is listening on port %d (http://localhost:%d)', serverPort, serverPort);
    console.log('Swagger-ui is available on http://localhost:%d/docs', serverPort);
});
```

Come gestire un errore che si verifica dopo la generazione dello stub del server

Quando lo stub del server viene generato automaticamente con l'editor Swagger, viene utilizzato oas3-tools (https://www.npmjs.com/package/oas3-tools) per la configurazione dell'applicazione Express. Purtroppo, l'ultima versione di questo modulo presenta un bug, che rende la definizione dei metodi delle rotte invisibile all'applicazione Express. Poiché questo bug non è ancora stato risolto, proponiamo un facile workaround: utilizzare una versione precedente (la 2.0.2) di oas3-tools.

Per farlo, è sufficiente modificare la dipendenza relativa a oas3-tools nel file package.json nel modo seguente:

```
"dependencies": {
  "connect": "^3.2.0",
  "js-yaml": "^3.3.0",
  "oas3-tools": "2.0.2"
}
```

Fare attenzione a non scrivere "^2.0.2", altrimenti la dipendenza viene risolta con la versione più
versione recente.

Come gestire l'autenticazione degli utenti con Passport e express-session

L'autenticazione dell'utente con Passport e express-session deve essere gestita lato server nel modo seguente:

1. quando un utente tenta di autenticarsi al servizio, deve essere utilizzata una strategia locale di Passport per verificare se l'e-mail dell'utente esiste e se la password specificata è corretta;
2. se l'autenticazione ha successo, il servizio crea una sessione autenticata con il modulo *express-session*. Nella creazione di questa sessione, occorre specificare il seguente valore per i tre parametri richiesti:
    a. si deve impostare il parametro *secret* a una stringa (possibilmente casuale) che viene usata per firmare il cookie dell'id di sessione (chiamato *connect-sid*);
    b. si deve impostare il parametro *resave* su false;
    c. si deve impostare il parametro *saveUninitialized* su false;
3. il server include il cookie *connect-sid* nella risposta HTTP e lo rimanda al client (questa operazione è gestita automaticamente da Passport ed è trasparente per l'utente);
4. Quando un utente richiede un'operazione che richiede l'autenticazione, un middleware di autenticazione deve verificare se l'utente richiedente è stato precedentemente autenticato. In questo middleware, si può usare *req.isAuthenticated()* per verificare l'autenticazione dell'utente.

Come gestire il calcolo dell'hash

Per il calcolo dell'hash, si può usare il modulo bcrypt di node.js. Si può usare il seguente sito web per generare l'hash di una password, secondo bcrypt: https://www.browserling.com/tools/bcrypt. Se l'hash generato inizia con $2y$, sostituire questa parte con $2b$, poiché il modulo bcrypt di node non supporta gli hash di $2y$.

Gestione del database

Siete liberi di creare il vostro database per la vostra soluzione personale, utilizzando *DB Browser per SQLite*. Tuttavia, vi forniamo un database, già popolato con alcuni record, che potete utilizzare o estendere per la vostra implementazione del server. Se utilizzate questo database,

con i dati preinstallati è possibile utilizzare le seguenti credenziali per autenticarsi al sistema Servizio di *Film Manager*:

- indirizzo e-mail: "user.dsp@polito.it";
- password: "password".