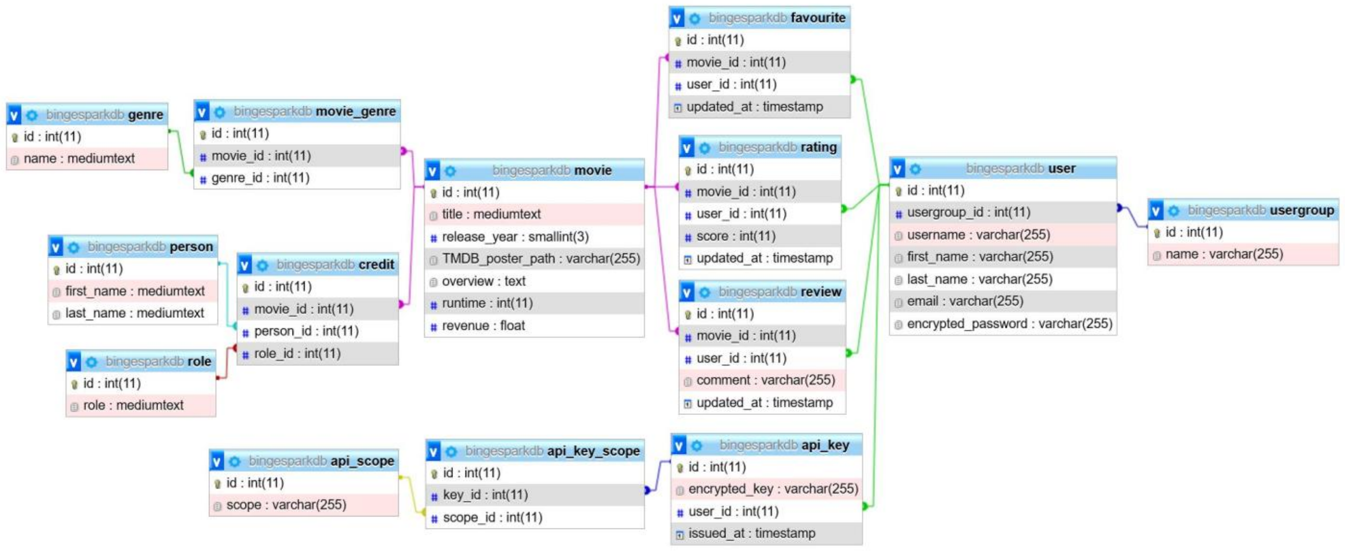
CSC7062 – Bingespark Assessement

Nicolas Deschatrettes - 40343717

1. Video demonstration

https://web.microsoftstream.com/video/4fb75bb6-1e6d-4908-acfb-f59eeffb81b7

2. Database design - ERD



Note that api_key.user_id is allowed to be null, as a key is not necessarily attached to a user: for instance the API key used by the website.

3. Website URI

https://ndeschatrettes01.webhosting6.eeecs.qub.ac.uk/bingespark

Some PHP compatibility issues may prevent the website from working properly…

4. Authentication credentials

website: **warning not implemented**

- admin : username: jadmin, password: CSC7062
- user : username: juser, password: CSC7062

rest API (generated for ease of use, not automatically generated by API)

- all access apikey: bingespark

5. RESTful API URIs

base URL: http://ndeschatrettes01.webhosting6.eeecs.qub.ac.uk/bingespark/api parameters marked (*) are optional

/movies

GET query: apikey, page*, pagesize*, sort*=(title,release_year,runtime,revenue,rating), order*=(asc, desc), average_rating_min*, average_rating_max*, release_year_min*, release_year_max*, runtime_min*, runtime_max*, revenue_min*, revenue_max*, person_id*, role_id*, genre_id*, release_year*
response (json): { page, total_pages, total_results, results = [{ id, title, release_year, TMDB_poster_path, overview, runtime, average_rating, favourites, genres [{ id, name}] }] }

POST query: apikey
body (json): title, release_year, overview, runtime, revenue
response (json): { id, title, release_year, TMDB_poster_path, overview, runtime, revenue }

/movies/{id}

GET query: apikey
response (json): { id, title, release_year, TMDB_poster_path, overview, runtime, average_rating, favourites, genres [{ id, name}] }

PUT query: apikey
body (json): title, release_year, overview, runtime, revenue
response (json): [{ id, title, release_year, TMDB_poster_path, overview, runtime, revenue }]

DELETE query: apikey
response: null

/movies/{id}/genres

GET query: apikey, page*, pagesize*, sort*=(name), order*=(asc, desc)
response (json): { page, total_pages, total_results, results = [{ id, name }] }

POST query: apikey
body (json): genre_id
response (json): { page, total_pages, total_results, results = [{ id, name }] }

/movies/{id}/genres/{id}

DELETE query: apikey
response (json): null

/movies/{id}/credits

GET query: apikey, page*, pagesize*, sort*=(last_name, role), order*=(asc, desc)
response (json): { page, total_pages, total_results, results = [{ credit_id, person_id, first_name, last_name, role_id, role }] }

POST query: apikey
body (json): person_id, role_id
response (json): { credit_id, person_id, first_name, last_name, role_id, role }

/movies/{id}/credits/{id}

DELETE query: apikey
response (json): null

/movies/{id}/reviews

GET query: apikey, page*, pagesize*, sort*=(updated_at), order*=(asc,desc)
response (json): { page, total_pages, total_results, results = [{ id, user_id, comment, updated_at }] }

POST query: apikey
body (json): user_id, comment
response (json) { id, user_id, comment, updated_at }

/movies/{id}/reviews/{id}

DELETE query: apikey
response (json): null

/movies/{id}/ratings

GET query: apikey, page*, pagesize*, sort*=(updated_at), order*=(asc,desc)
response (json): { id, user_id, score }

POST query: apikey
body (json): { user_id, score }
response (json): { id, user_id, score }

/movies/{id}/ratings/{id}

DELETE response (json): null

/genres

GET query: apikey, page*, pagesize*, sort*=(name), order*=(asc, desc)
response (json): { page, total_pages, total_results, results = [{ id, name }] }

POST query: apikey
body (json): name
response (json): { id, name }

/genres/{id}

GET query: apikey
response (json): { id, name }

PUT query: apikey
body (json): name
response (json): { id, name }

DELETE query: apikey
response (json): null

/genres/{id}/movies

GET query: apikey, page*, pagesize*, sort*=(title,release_year,runtime,revenue,rating), order*=(asc, desc), average_rating_min*, average_rating_max*, release_year_min*, release_year_max*, runtime_min*, runtime_max*, revenue_min*, revenue_max*, person_id*, role_id*, genre_id*, release_year*
response (json): { page, total_pages, total_results, results = [{ id, title, release_year, TMDB_poster_path, overview, runtime, average_rating, favourites, genres [{ id, name}] }] }

/reviews

GET query: apikey, page*, pagesize*, sort*=(update_at), order*=(asc, desc)
response (json): { page, total_pages, total_results, results = [{ id, movie_id, user_id, comment, updated_at }] }

POST query: apikey
body (json): { movie_id, user_id, comment }
response (json): { id, movie_id, user_id, comment, updated_at }

/reviews/{id}

GET query: apikey
response (json): { id, movie_id, user_id, comment, updated_at }

PUT query: apikey
body (json): { movie_id, user_id, comment }
response (json): { id, movie_id, user_id, comment, updated_at }

DELETE query: apikey
response (json): null

/ratings

GET query: apikey, page*, pagesize*, sort*=(update_at), order*=(asc,desc)
response (json): { page, total_pages, total_results, results=[{ id, movie_id, user_id, score, updated_at }] }

POST query: apikey
body (json): { movie_id, user_id, score }
response (json): { id, movie_id, user_id, score, updated_at }

/ratings/{id}

GET query: apikey
response (json): { id, movie_id, user_id, score, updated_at }

PUT query: apikey
body (json): { movie_id, user_id, score }
response (json): { id, movie_id, user_id, score, updated_at }

DELETE query: apikey
response (json): null

/persons

GET query: apikey, page*, pagesize*, sort*=(last_name), order*=(asc, desc), first_name*, last_name*, movie_id*, role_id*, role*
response (json): {page, total_pages, total_results, results=[{id, first_name, last_name}] }

POST query: apikey
body (json): first_name, last_name
response (json): { id, fist_name, last_name }

bingespark/api/persons/{id}

GET query: apikey
response (json): { id, fist_name, last_name }

PUT query: apikey
body (json): first_name, last_name
response (json): { id, fist_name, last_name }

DELETE query: apikey
response (json): null

/persons/{id}/credits

GET query: apikey
response (json): { page, total_pages, total_results, results=[{id, movie_id, role_id, movie_title, role }] }

/persons/search

GET query: apikey, query, page*, pagesize*, sort*=(last_name), order*=(asc, desc), first_name*, last_name*, movie_id*, role_id*, role*
response (json): {page, total_pages, total_results, results=[{id, first_name, last_name}] }

/roles

GET query: apikey, page*, pagesize*, sort*=(role), order*=(asc,desc)
response (json): { page, total_pages, total_results, results=[{ id, role }] }

POST query: apikey
body (json): { role }
response (json): { id, role }

/roles/{id}

PUT query: apikey
body (json): { role }
response (json): { id, role }

DELETE query: apikey
response (json): null

/users

GET query: apikey, page*, pagesize*, sort*=(username, last_name, updated_at), order*=(asc, desc),
response (json): { page, total_pages, total_results, results={ [{ id, usergroup_id, username, first_name, last_name, email, encrypted_password, updated_at }] }

POST query: apikey
body (json): { usergroup_id, first_name, last_name, username, email, password }
response (json): { id, usergroups_id, first_name, last_name, username, encrypted_password, updated_at }

/users/{id}

GET query: apikey
response (json): { id, usergroup_id, username, first_name, last_name, email, encrypted_password }

PUT query: apikey
body (json): { usergroup_id, first_name, last_name, username, email, password* }
response (json): { id, usergroups_id, first_name, last_name, username, encrypted_password, updated_at }

DELETE query: apikey
response (json): null

users/{id}/favourites

GET query: apikey, page*, pagesize*, sort*=(username, last_name, updated_at), order*=(asc, desc),
response (json): { page, total_pages, total_results, results={ [{ id, movie_id, movie={ id, title, release_year} }] }

POST

/users/{id}/reviews

GET query: apikey, page*, pagesize*, sort*=(username, last_name, updated_at), order*=(asc, desc),
response (json): { page, total_pages, total_results, results={ [{ id, usergroup_id, username, first_name, last_name, email, encrypted_password, updated_at }] }

/users/{id}/ratings

GET query: apikey, page*, pagesize*, sort*=(username, last_name, updated_at), order*=(asc, desc),
response (json): { page, total_pages, total_results, results={ [{ id, usergroup_id, username, first_name, last_name, email, encrypted_password, updated_at }] }

/users/{id}/login

GET query: apikey
body (json) { username, password }
response (json): { id, first_name, last_name, username, email, encrypted_password, updated_at}
or response (json): null

/usergroups

GET query: apikey, page*, pagesize*, order*=(name), sort*=(asc, desc)
response: { page, total_pages, total_results, results = [{ id, name }] }

/usergroups/{id}

GET query: apikey
response: { id, name }

/usergroups/{id}/users

GET query: apikey, page*, pagesize*, order*=(), sort*=(asc, desc)
response: { page, total_page, total_results, results = [{ id, usergroup_id, username, first_name, last_name, email, encrypted_password }] }

/apikeys/

GET query: apikey, page*, pagesize*, sort*=(issued_at), order*=(asc, desc), user_id*
response (json): { page, total_pages, total_results, results = [{ id, encrypted_key, user_id, issued_at }] }

POST query: apikey
body (json): user_id
response (json): { id, encrypted_key, user_id, issued_at, apikey }

/apikeys/{id}

GET query: apikey
response (json): { id, encrypted_key, user_id, issued_at }

DELETE query: apikey
response (json): null

/apikeys/{id}/apiscopes

GET query: apikey, page*, pagesize*, sort*=(scope), order*=(asc, desc), user_id*
response (json): { page, total_pages, total_results, results = [{ id, key_id, user_id, scope_id, scope }] }

POST query: apikey
body (json): scope, scope_id
response (json): { id, key_id, user_id, scope_id, scope }

/apikeys/{id}/apiscopes/{id}

DELETE query: apikey
response: null

6. Code reference

Following extended research on software development, I have decided to apply a Model-View-Controller approach to designing my project here.

Most files contain classes that are a part of the API or the website.

All data accessed by the website is intended to be through the API, which is very complete, and fully tested.

The API also encrypts passwords and api keys, and performs password and api key verifications.

The only time an api key will be visible in clear text is when returned by the API it is created (POST /apikeys).

Finally the API also returns HTTP responses (200, 400, 401 etc…) to all requests, with error messages.

The API code is in the api folder, and the website code is in the main folder and subfolders, except the api folder.

See below for the code reference.

```
bingespark                                project root directory
|  api                                    api root directory
|  └ controllers                          api controllers
|  |  └ apikeycontroller.php              These classes handle the requests and responses, and use the model
|  |  └ basecontroller.php                classes to access the database. They offer GET/POST/PUT/DELETE to
|  |  └ creditcontroller.php              CRUD mapping as well as additional endpoint functions. They also
|  |  └ favouritecontroller.php           handle security, by encrypting sensitive data (passwords, apikeys), and
|  |  └ genrecontroller.php               controlling the access to the data via apikeys.
|  |  └ moviecontroller.php
|  |  └ personcontroller.php
|  |  └ ratingcontroller.php
|  |  └ reviewcontroller.php
|  |  └ rolecontroller.php
|  |  └ usercontroller.php
|  |  └ usergroupcontroller.php
|  └ include                              generic utility files
|  |  └ bootstrap.php                     These files contain connection details and other utilities
|  |  └ config.php
|  |  └ utilities.php
|  └ models                               api models
|  |  └ apikeymodel.php                   These classes handle the data and provide a CRUD interface to the
|  |  └ apikeyscopemodel.php              database. They ensure data consistency by cleaning up the database
|  |  └ apiscopemodel.php                 during deletes, etc...
|  |  └ basemodel.php
|  |  └ creditmodel.php
|  |  └ favouritemodel.php
|  |  └ genremodel.php
|  |  └ mooviegenremodel.php
|  |  └ moviemodel.php
|  |  └ personmodel.php
|  |  └ ratingmodel.php
|  |  └ reviewmodel.php
|  |  └ rolemodel.php
|  |  └ usergroupmodel.php
|  └ apikeys.php                          These files are the endpoints of the API. HTTP requests are routed to
|  └ credits.php                          them using the .htaccess file. They instantiate the controllers and pass
|  └ favourites.php                       on the control over to them
|  └ genres.php
|  └ movies.php
|  └ persons.php
|  └ ratings.php
|  └ reviews.php
|  └ roles.php
|  └ usergroups.php
|  └ users.php
└ controllers                             website controllers
|  └ basecontroller.php                   The classes handle the
|  └ baselistcontroller.php
|  └ moviecontroller.php
|  └ movielistcontroller.php
└ data                                    original csv file, import script, and resulting sql database
|  └ bingespark.sql
|  └ import-TMDB.php
|  └ Movie-DataSet2_final.csv
└ images                                  Image files. The movie posters are not stored on the server.
|  └ bingesparklogo.png
|  └ noposter.jpg
└ models                                  website models
|  └ apikey.php                           These classes are used to access the data through the API. They consist
|  └ apikeylist.php                       of element classes and list of element classes. They allow for very
|  └ apiscope.php                         simple access to the data from a coding point of view, and the
|  └ apiscopelist.php                     basemodels also make it very easy to add models.
|  └ baselistmodel.php
|  └ basemodel.php
|  └ credit.php
|  └ creditlist.php
|  └ favourite.php
|  └ favouritelist.php
|  └ genre.php
|  └ genrelist.php
|  └ movie.php
|  └ movielist.php
|  └ person.php
|  └ personlist.php
|  └ rating.php
|  └ ratinglist.php
|  └ review.php
|  └ reviewlist.php
|  └ role.php
|  └ rolelist.php
|  └ user.php
|  └ userlist.php
└ server                                  This file is used to route the http requests to api endpoints
|  └ .htaccess
└ views                                   view classes
|  └ baseview.php                         These classes handle the presentation of the content to the website
|  └ movielistview.php                    user.
|  └ movieview.php
|  └ loginview.php
└ index.php                               main entry points for the website. These files instantiate models,
└ login.php                               controllers and views
└ movie.php
```