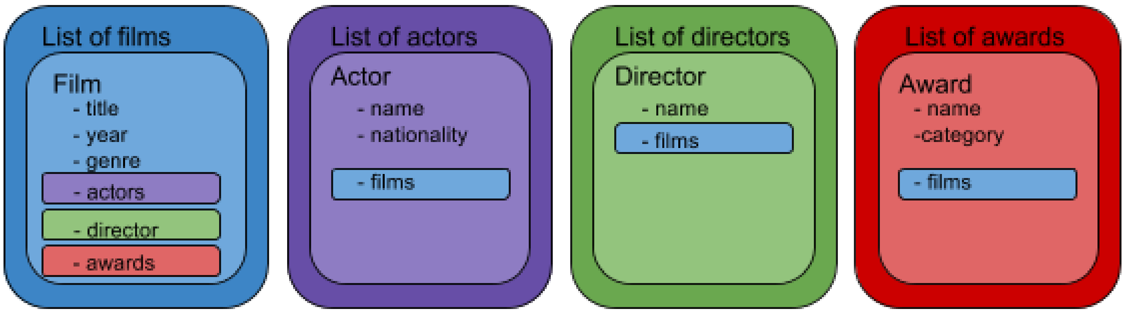
Web Programming Coursework 2 - Report

Leticia Farias Wanderley 150721453

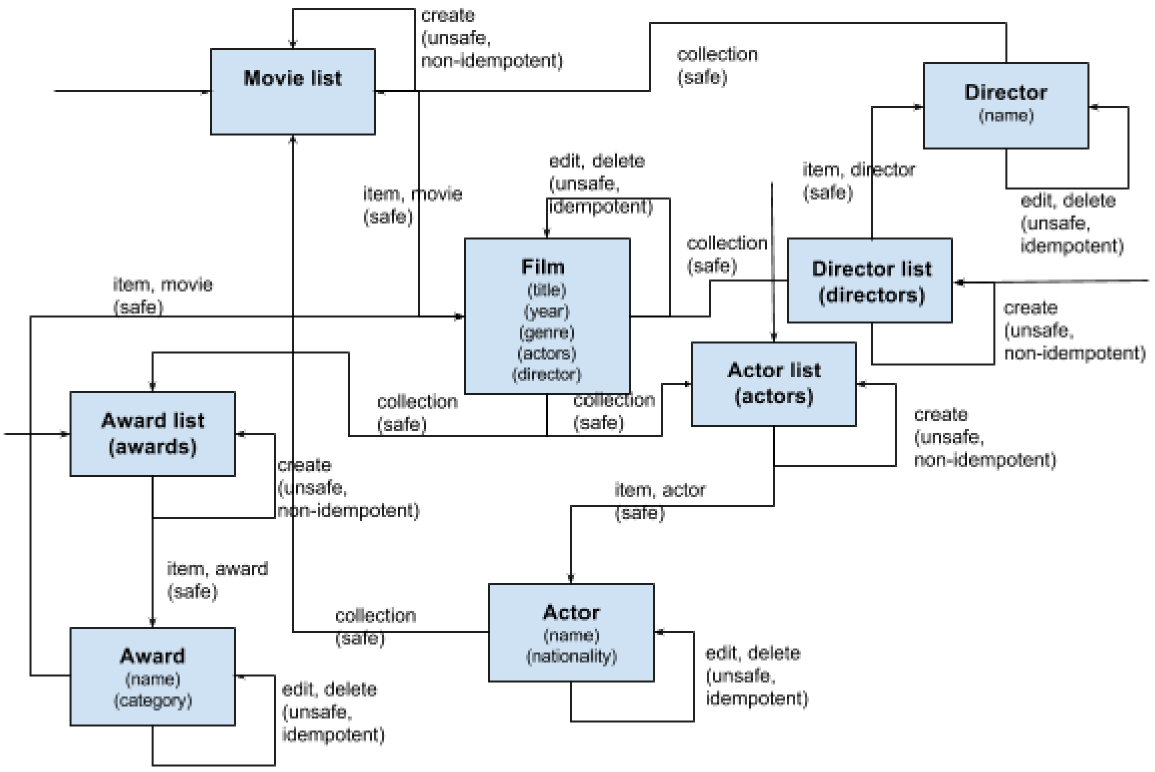
1. Domain

The domain chosen was “Film”, films as creative works objects represented as resources from the API. This choice implied in the creation of other resources, such as Actor and Director, to make the representation of films inside the application as close to reality as possible.

2. Semantic descriptors



3. API State Diagram


Movie list
create (unsafe, non-idempotent)
collection (safe)
Director (name)
item, director (safe)
edit, delete (unsafe, idempotent)
item, movie (safe)
edit, delete (unsafe, idempotent)
Film (title) (year) (genre) (actors) (director)
collection (safe)
Director list (directors)
create (unsafe, non-idempotent)
item, movie (safe)
Actor list (actors)
create (unsafe, non-idempotent)
Award list (awards)
collection (safe)
collection (safe)
create (unsafe, non-idempotent)
item, actor (safe)
item, award (safe)
collection (safe)
Actor (name) (nationality)
edit, delete (unsafe, idempotent)
Award (name) (category)
edit, delete (unsafe, idempotent)

The names in the API state diagram were reconciled based on IANA registered link relations and Schema.org’s Movie object (https://schema.org/Movie).

4. Implementation

A Django model was defined for each non-list semantic descriptor in the API design, Film, Actor, Director and Award. These classes relate with each other in the following way: Film has a many-to-many list of Actors and a many-to-one attribute of the class Director, each object of the class Award has a many-to-one attribute of the class Film. These means that Film has a cast of actors, a director and a set of awards, an Actor can be part of several film's casts, a Director can direct several films and an Award is related to a single film, similarly to the real world.

Django rest framework provides tools to represent these models in a web language or media type, in this case JSON. For each of the models a serializer was created, the serializer is a parser to and from the Model. When the client asks for an object from the Database the serializer returns a representation of that object in JSON and when the client tries to add a new object on the database from the application side the serializer validates the JSON data received and creates the new object.

Each semantic descriptor also has a defined view, basically there's one view for a collection of objects and one for a single object. The client can add objects to the system when it is in a collection state and it can edit or delete objects when it is in a item, or single object, state. Since most resources are nested inside Film, this resource has a link for its specific collections of nested resources, e.g., a film x has a link for its actors list, this list only shows actors in the cast of film x and when the client adds a new actor in that state the new actor is automatically related to film x. The same happens when creating a new director or award from a film's collection of directors or awards states. Of course, the client can create actors, awards & directors without connecting them with a film, for this to happen it only needs to make a POST request for those new resources in their respective collection states.

A resource can be edited or deleted in its specific view, i.e., in its single state. However, there are a few of its attributes that can not be changed, for example it's not possible to change the list of films of which an actor is part of by making a PUT request for that actor. These constraints helps to keep the system safe from changes that don't make sense for the domain. There are also ways of filtering resources inside the system, actors and directors resources have links that lead to their films, the attribute "films" is a link that filters the film collection based on which actor or director the client is requesting from. The award resources have a link to a single film, the film that won that award.

5. Extras

Although Django rest framework gives a lot of facilities to build an API, there are a few things not covered by its toolkit. For example, when a client makes a request for an resource that doesn't exist it causes the serializer to raise an exception and this exception is carried untreated to the client in an Internal Server Error (HTTP 500).

In order to improve the response given to the client a checking is made in the GET request before sending it to the serializer. If the resource requested actually exists the transaction goes on as usual. However, if it doesn't exist the view class responds the request with a more detailed explanation about what happened, in this case a Resource Not Found message with HTTP status code 404.

6. Appendix

URL of the deployed application: http://restapicw-lfw.apps.devcloud.eecs.qmul.ac.uk/

URI	Method	Collection	Operation	Business Operation
/film?actor=:actor&director=:director	GET	films	retrieve	get_films
/film/{id}	GET	films	retrieve	get_film
/film	POST	films	create	create_film

/film/{id}	PUT	films	edit	edit_film
/film/{id}	DELETE	films	delete	delete_film
/film/{id}/actor	GET	actors	retrieve	get_actors
/film/{id}/award	GET	awards	retrieve	get_awards
/film{id}/director	GET	directors	retrieve	get_directors
/film/{id}/actor	POST	actors	create	create_actor
/film/{id}/award	POST	awards	create	create_actor
/film{id}/director	POST	directors	create	create_actors
/actor	GET	actors	retrieve	get_actors
/actor/{id}	GET	actors	retrieve	get_actor
/actor	POST	actors	create	create_actor
/actor/{id}	PUT	actors	edit	edit_actor
/actor/{id}	DELETE	actors	delete	delete_actor
/award	GET	awards	retrieve	get_awards
/award/{id}	GET	awards	retrieve	get_award
/award	POST	awards	create	create_award
/award/{id}	PUT	awards	edit	edit_award
/award/{id}	DELETE	awards	delete	delete_award
/director	GET	directors	retrieve	get_directors
/director/{id}	GET	directors	retrieve	get_director
/director	POST	directors	create	create_director
/director/{id}	PUT	directors	edit	edit_director
/director/{id}	DELETE	directors	delete	delete_director