图片包含 场记板, 文字

已生成极高可信度的说明

System Design Specification

iMovie Web Application

September 13th, 2017

**Team:** **36**

team members:

Jiahui CHEN 450554021

Zhe LIU 450182813

Xuan WEI 460175351

Meng YUAN 470241734

Jichen ZHAO 460225645

Table of Contents

[Data model 1](#_Toc493055710)

[Code structure 2](#_Toc493055711)

[Style guide 2](#_Toc493055712)

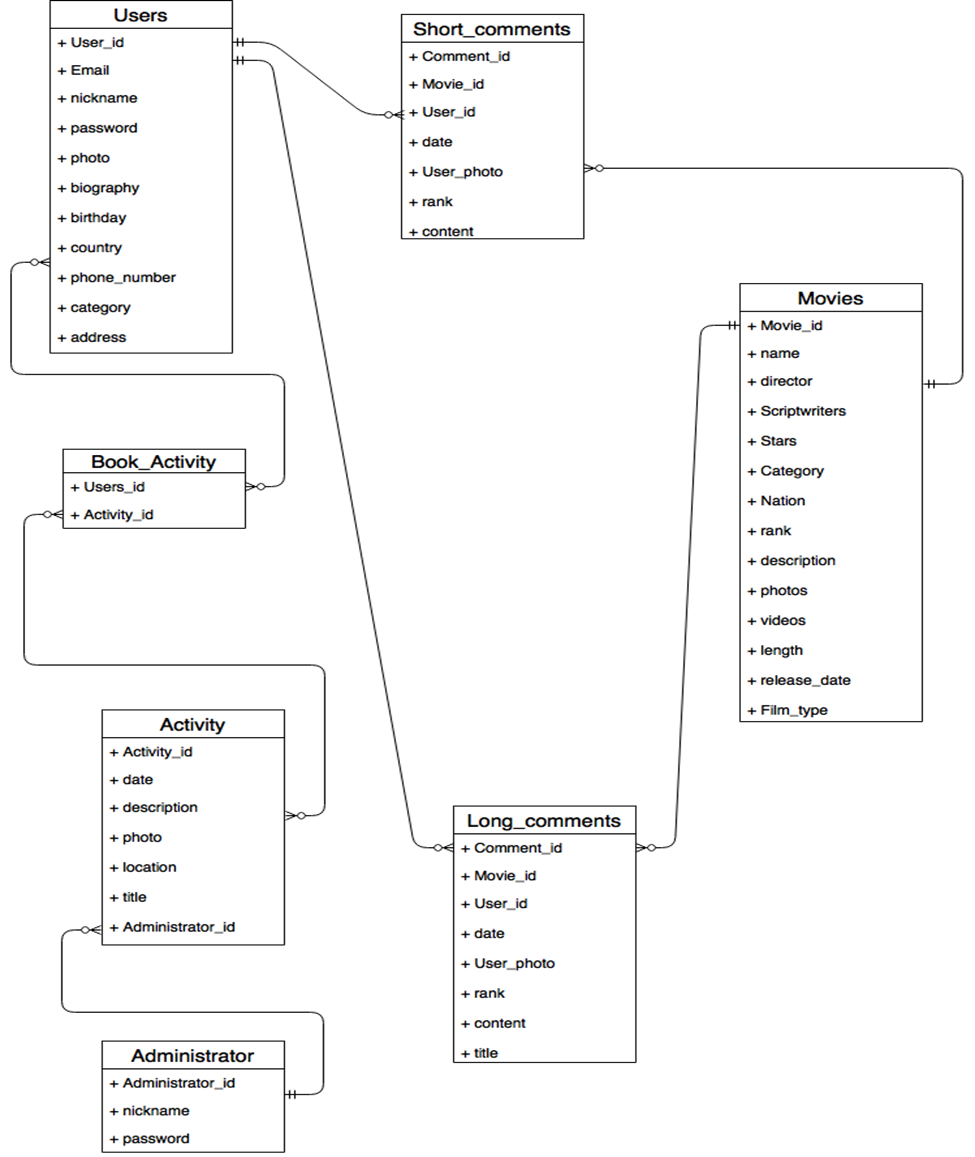
[File/folder layout 7](#_Toc493055713)

[Dependencies 8](#_Toc493055714)

[Exclusions 9](#_Toc493055715)

[Reference 9](#_Toc493055716)

# Data model



**Figure1. Entity Relationship Diagram of the iMovie Web Application**

The Entity Relationship Diagram presents communication relationship between each entity. The User-Book\_activity relationship is many-to-many, which means each user can book any activities. However, each activity can only upload by one administrator as a one-to-many relationship between Administrator and Activity entity. Meanwhile, users could post many comments, and these comments can be linked to many movies. Nevertheless, Short\_Comments and Long\_Comments entities should involve movie\_id for identifying which movie the comment belong to.

# Code structure

## Style guide

**Python style**

* Please conform to the indentation style dictated in the .editorconfig file. We recommend using a text editor with [EditorConfig](http://editorconfig.org/) support to avoid indentation and whitespace issues. The Python files use 4 spaces for indentation and the HTML files use 2 spaces.
* Unless otherwise specified, follow [**PEP 8**](https://www.python.org/dev/peps/pep-0008).
* Use [flake8](https://pypi.python.org/pypi/flake8) to check for problems in this area. Note that our setup.cfg file contains some excluded files (deprecated modules we don’t care about cleaning up and some third-party code that Django vendors) as well as some excluded errors that we don’t consider as gross violations. Remember that [**PEP 8**](https://www.python.org/dev/peps/pep-0008) is only a guide, so respect the style of the surrounding code as a primary goal.
* An exception to [**PEP 8**](https://www.python.org/dev/peps/pep-0008) is our rules on line lengths. Don’t limit lines of code to 79 characters if it means the code looks significantly uglier or is harder to read. We allow up to 119 characters as this is the width of GitHub code review; anything longer requires horizontal scrolling which makes review more difficult. This check is included when you run flake8. Documentation, comments, and docstrings should be wrapped at 79 characters, even though [**PEP 8**](https://www.python.org/dev/peps/pep-0008) suggests 72.
* Use four spaces for indentation.
* Use four space hanging indentation rather than vertical alignment:
* **raise** **AttributeError**(  
   'Here is a multine error message '  
   'shortened for clarity.'  
  )
* Instead of:
* **raise** **AttributeError**('Here is a multine error message '  
   'shortened for clarity.')
* This makes better use of space and avoids having to realign strings if the length of the first line changes.
* Use single quotes for strings, or a double quote if the the string contains a single quote. Don’t waste time doing unrelated refactoring of existing code to conform to this style.
* Avoid use of “we” in comments, e.g. “Loop over” rather than “We loop over”.
* Use underscores, not camelCase, for variable, function and method names (i.e. poll.get\_unique\_voters(), not poll.getUniqueVoters()).
* Use InitialCaps for class names (or for factory functions that return classes).
* In docstrings, follow the style of existing docstrings and [**PEP 257**](https://www.python.org/dev/peps/pep-0257).
* In tests, use [assertRaisesMessage()](https://docs.djangoproject.com/en/dev/topics/testing/tools/#django.test.SimpleTestCase.assertRaisesMessage) instead of [assertRaises()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRaises) so you can check the exception message. Use [assertRaisesRegex()](https://docs.python.org/3/library/unittest.html#unittest.TestCase.assertRaisesRegex) only if you need regular expression matching.
* In test docstrings, state the expected behavior that each test demonstrates. Don’t include preambles such as “Tests that” or “Ensures that”.
* Reserve ticket references for obscure issues where the ticket has additional details that can’t be easily described in docstrings or comments. Include the ticket number at the end of a sentence like this:
* **def** test\_foo():  
   *"""*  
   *A test docstring looks like this (#123456).*  
   *"""*  
   ...

**Imports**

* Use [isort](https://github.com/timothycrosley/isort#readme) to automate import sorting using the guidelines below.
* Quick start:
* **$** pip install isort  
  **$** isort -rc .
* This runs isort recursively from your current directory, modifying any files that don’t conform to the guidelines. If you need to have imports out of order (to avoid a circular import, for example) use a comment like this:
* **import** **module** *# isort:skip*
* Put imports in these groups: future, standard library, third-party libraries, other Django components, local Django component, try/excepts. Sort lines in each group alphabetically by the full module name. Place all importmodule statements before from module import objects in each section. Use absolute imports for other Django components and relative imports for local components.
* On each line, alphabetize the items with the upper case items grouped before the lower case items.
* Break long lines using parentheses and indent continuation lines by 4 spaces. Include a trailing comma after the last import and put the closing parenthesis on its own line.
* Use a single blank line between the last import and any module level code, and use two blank lines above the first function or class.
* For example (comments are for explanatory purposes only):
* *# future*  
  **from** **\_\_future\_\_** **import** unicode\_literals  
    
  *# standard library*  
  **import** **json**  
  **from** **itertools** **import** chain  
    
  *# third-party*  
  **import** **bcrypt**  
    
  *# Django*  
  **from** **django.http** **import** Http404  
  **from** **django.http.response** **import** (  
   Http404, HttpResponse, HttpResponseNotAllowed, StreamingHttpResponse,  
   cookie,  
  )  
    
  *# local Django*  
  **from** **.models** **import** LogEntry  
    
  *# try/except*  
  **try**:  
   **import** **yaml**  
  **except** **ImportError**:  
   yaml = **None**  
    
  CONSTANT = 'foo'  
    
    
  **class** **Example**:  
   *# ...*
* Use convenience imports whenever available. For example, do this:
* **from** **django.views** **import** View
* instead of:
* **from** **django.views.generic.base** **import** View

**Template style**

* In Django template code, put one (and only one) space between the curly brackets and the tag contents.
* Do this:
* {{ foo }}
* Don’t do this:
* {{foo}}

**View style**

* In Django views, the first parameter in a view function should be called request.
* Do this:
* **def** my\_view(request, foo):  
   *# ...*
* Don’t do this:
* **def** my\_view(req, foo):  
   *# ...*

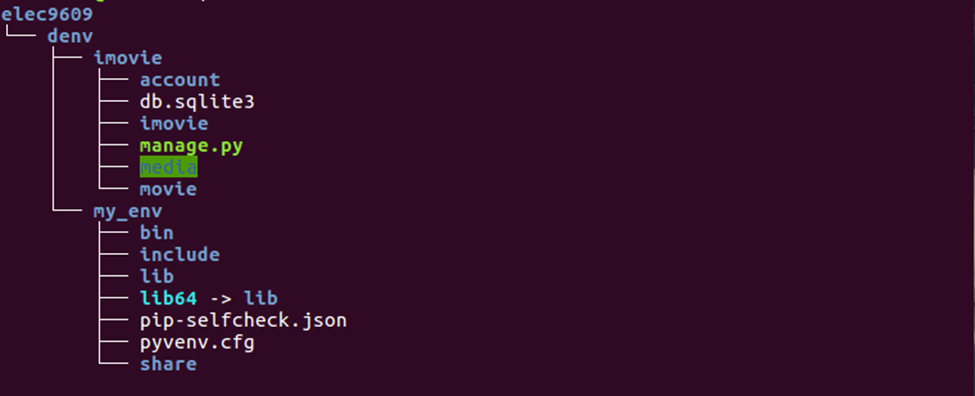
**Model style**

* Field names should be all lowercase, using underscores instead of camelCase.
* Do this:
* **class** **Person**(models.Model):  
   first\_name = models.CharField(max\_length=20)  
   last\_name = models.CharField(max\_length=40)
* Don’t do this:
* **class** **Person**(models.Model):  
   FirstName = models.CharField(max\_length=20)  
   Last\_Name = models.CharField(max\_length=40)
* The class Meta should appear *after* the fields are defined, with a single blank line separating the fields and the class definition.
* Do this:
* **class** **Person**(models.Model):  
   first\_name = models.CharField(max\_length=20)  
   last\_name = models.CharField(max\_length=40)  
    
   **class** **Meta**:  
   verbose\_name\_plural = 'people'
* Don’t do this:
* **class** **Person**(models.Model):  
   first\_name = models.CharField(max\_length=20)  
   last\_name = models.CharField(max\_length=40)  
   **class** **Meta**:  
   verbose\_name\_plural = 'people'
* Don’t do this, either:
* **class** **Person**(models.Model):  
   **class** **Meta**:  
   verbose\_name\_plural = 'people'  
    
   first\_name = models.CharField(max\_length=20)  
   last\_name = models.CharField(max\_length=40)
* The order of model inner classes and standard methods should be as follows (noting that these are not all required):
  + All database fields
  + Custom manager attributes
  + class Meta
  + def \_\_str\_\_()
  + def save()
  + def get\_absolute\_url()
  + Any custom methods
* If choices is defined for a given model field, define each choice as a tuple of tuples, with an all-uppercase name as a class attribute on the model. Example:
* **class** **MyModel**(models.Model):  
   DIRECTION\_UP = 'U'  
   DIRECTION\_DOWN = 'D'  
   DIRECTION\_CHOICES = (  
   (DIRECTION\_UP, 'Up'),  
   (DIRECTION\_DOWN, 'Down'),  
   )

(Django coding style, 2009)

## File/folder layout

The folder structure of this project shows below, including account application and movie application. Each application involves different kinds of file to support the project, such as urls.py, forms.py, admin.py, models.py and templates folder of html files. The media folder in django project in for saving media static files, such as images , videos and sounds.



**Figure2. Folder layout of the iMovie Web Application**

# Dependencies

|  |  |  |
| --- | --- | --- |
| Third-party software packages | Version | Description |
| Python | 3.5 | Basic development environment for Django |
| Django | 1.11 | Web framework |
| SQLite |  | SQLite is Python’s built-in database |
| Django-taggit | 0.17.1 | Django-taggit is a third party Django tagging application, can be used to tag user’s post which can achieve specific filter. |
| Markdown | 2.6.2 | Python markdown module can achieve creating custom template filters which can help develop modify variable in templates |
| Pillow | 2.9.0 | We use this library to manage our application’s images |
| Python-social-auth | 0.2.12 | This library can help us to add social authentication to our site and let user log in our website using their account of other service (Facebook, Twitter etc..) |
| Sorl-thumbnail | 12.3 | Image will be display in our detail page, but different images may have different dimensions, so Sorl-thumbnail can help us to create image thumbnails. |
| Java | 1.7+ | Java Runtime Environment is necessary for installing Solr. |
| Solr | 4.10.4 | Solr can conjunction with Django to build research engine for our application. Solr can provide us with advanced search features, such as full-text search, term search, hit highlighting, faceted search and dynamic clustering. |

**Table1.The third-party software of the iMovie Web Application**

# Exclusions

The purpose of our website is to recommend movies for users by users’ filter criteria or movies’ rating. To improve usability and conciseness, there are three functions that might benefit for stakeholders but will not include in our website. These exclusions are:

1. News of upcoming movies.
2. Trailers and ‘behind the screen’ videos.
3. Booking tickets.

First, news of upcoming movies function need an independence column of ‘news’ in this website, and updating these information needs lots of manpower of web administrator. Therefore, the design is only adding a movie that has been shown in cinemas, and ignoring the upcoming movies with relevant news.

The function of videos about trailers and the ‘behind screens’ for some movies is the second exclusion, since user might find the irrelevant contents when they do some search in search box. The expected prototype for search function is that users input keywords in search box, and the relevant movies will be shown as a result. Users could click one of these movies to view the information, ratings and comments of this movie. The function of trailers and ‘behind the screen’ videos might reduce user experience of searching function, which is a significant function of this website.

Finally, many websites of movie recommendation provide a method to book a ticket in cinema. Stakeholders could take advantage of cooperation with cinemas to earn some money by this way. However, users might not feel good if a website has some connections with their money, although a famous website might be fine to do that. Therefore, the design of iMovie website is to be a non-profit website with one purpose -- recommend you the movie you like, thus our team decide not to include the booking tickets function in this website.

# Reference

[1] Django coding style. (2009,July 05). Retrieved September 10, 2017, from Django documentation:https://docs.djangoproject.com/en/dev/internals/contributing/writing-code/coding-style/