# MEMORIA
## PRACTICA EXTRAORDINARIA

Para realizar esta práctica he seguido distintos pasos acorde a lo requerido en el enunciado.
En primer lugar creé un nuevo modelo llamado Vote:

```python
class Vote(models.Model):

    book = models.ForeignKey('Book', on_delete=models.SET_NULL, null=True, related_name='votes')
    user = models.ForeignKey(User, on_delete=models.SET_NULL, null=True)
    score = models.IntegerField(default=0, validators=[validar_gt_0, validar_lt_10])


    class Meta:
        ordering = ['book', 'user', 'score']

    def save(self, *args, **kwargs):

        if self.score < 0 or self.score >10:
            raise ValueError

        try:
            Vote.objects.get(book=self.book, user=self.user).delete()
        except:
            print("No existe voto")

        super(Vote, self).save(*args, **kwargs)
        self.book.save()

    def __str__(self):
        """
        String para representar el Objeto Modelo
        """
        return '%s-%s, %s' % (
                self.user.username,
                self.book.title,
                self.score
            )
```

El modelo cuenta con un usuario, un libro y un score (la puntuación de la votación). La mayor parte de la funcionalidad esta implementada en la función save(). Dentro de esta se controlan errores de valores del score (debido a los tests) y se controla que solo pueda haber un Vote por usuario, si existe un voto para un usuario y un libro, se elimina y se crea otro con la nueva puntuación. Y por último hago un save() de book para que se actualice el score de Book.

```python
score = models.DecimalField(
        default=None,
        decimal_places=2,
        max_digits=4,
        validators=[validar_gt_0, validar_lt_10],
        null=True
    )

def save(self, *args, **kwargs):
    self.slug = slugify(
            (self.title, self.date.strftime("%d-%b-%Y-%H-%M-%S"))
        )

    if len(self.votes.all()) > 0:
        num = len(self.votes.all())

        media = 0

        for vote in self.votes.all():
            media += vote.score
        self.score = media / num

    super(Book, self).save(*args, **kwargs)
```

Posteriormente, modifiqué el modelo de Book, para hacer que el score de Book sea la media de los votos que tenga:

Cambié el tipo de campo score (de integer a decimal) y en la función save() siempre que el libro tenga votos sumo todos los votos y los divido entre el número de votos que tiene dicho libro. Si el libro no tiene votos la puntuación es None.

Después creé el form de los votos y lo implemente en book_details.html

forms.py

```python
from django.forms import Form, CharField, IntegerField


class FilterForm(Form):
    search = CharField(required=False)

class VoteForm(Form):
    score = IntegerField(required=True, label="score", min_value=0, max_value=10)

```

views.py

```python
if request.method == 'POST':

    # Create a form instance and populate it with data from the request:
    form = CartAddBookForm(request.POST)
    vote_form = VoteForm(request.POST)

    # Check if the form is valid:
    if form.is_valid():
        # process the data in form.cleaned_data as required
        cart.add(book, quantity=form.cleaned_data['quantity'])

        # redirect to a new URL:
        return HttpResponseRedirect(reverse('cart_list'))

    if vote_form.is_valid():
        # process the data in form.cleaned_data as required
        Vote.objects.create(book=book, score=vote_form.cleaned_data['score'], user=request.user)

        # redirect to a new URL:
        return HttpResponseRedirect(reverse('index'))

# If this is a GET (or any other method) create the default form.
else:
    form = CartAddBookForm()
    vote_form = VoteForm()

context_dict['form'] = form
context_dict['vote_form'] = vote_form

return render(request, 'catalog/book_detail.html', context_dict)
```

book_details.html

```
<form action="" method="post">
  {% csrf_token %}
  <table>
  {{ vote_form }}
  </table>
  <input type="submit" value="Votar libro" />
</form>
```

A continuación actualicé el populate.py para que al ejecutarse se generen 100 votos con votaciones aleatorias en libros aleatorios de usuarios aleatorios (aleatorios entre los ya creados previamente en el populate). En este fichero no se calcula ningun score para book ya que cada vez que se llama a la funcion save() de book o vote, dicho score se actualiza automaticamente.

```
def vote(self):

    voteD = {}

    for i in range(100):

        voteD[i] = {
            'book': randrange(7)+101,
            'user': randrange(7)+100,
            'score': randrange(11)
        }


    for index, a in enumerate(voteD.values()):
        x = Vote.objects.get_or_create(
            score=a['score'],
            book=Book.objects.get(id=int(a['book'])),
            user=User.objects.get(id=int(a['user']))
        )[0]
        x.save()
```

Por último, antes de realizar los tests, modifiqué el views.py e index.html para que mostrase también los 5 libros más votados:

```
def index(request):
    """
    Función vista para la página inicio del sitio.
    """

    libros_mas_populares = Book.objects.order_by('-score')[:5]
    libros_mas_recientes = Book.objects.order_by('-date')[:5]

    context_dict = {}
    context_dict['libros_mas_populares'] = libros_mas_populares
    context_dict['libros_mas_recientes'] = libros_mas_recientes

    vote_count = {}
    for b in Book.objects.all():
        vote_count[b] = b.votes.count()

    list_sorted = []
    list_sorted = sorted(vote_count.items(), key=lambda x: x[1], reverse=True)[:5]

    libros_mas_votados = []
    for x, y in list_sorted:
        libros_mas_votados.append(x)

    context_dict['libros_mas_votados'] = libros_mas_votados

    return render(request, 'index.html', context_dict)
```

```
<h2>Libros con mas votos</h2>
{% for b1 in libros_mas_votados %}
        <li>
        <strong>
            <a href="{{ b1.get_absolute_url }}">{{ b1 }}</a>
        </strong>
        </li>
{% endfor %}
```

# Tests:

En cuanto a los tests he realizado 7:

**test01_vote:**
Este test prueba si el usuario y el libro del voto creado son los correctos.

**test02_vote:**
Este test prueba que no cree el voto si el score es superior a 10.

**test03_vote:**
Este test prueba que no cree el voto si el score es inferior a 0.

**test04_vote:**
Este test prueba que se crea correctamente el voto si el score esta entre 0 y 10.

**test05_vote:**
Este test prueba inicialmente que la media de dos votaciones de distintos usuarios sobre el mismo libro se realice correctamente y, posteriormente tras otra votación, que dicha media se actualiza correctamente.

**test06_vote:**
Este test prueba que un libro que no tiene votaciones, la media es None.

**test07_vote:**
Este test prueba que un usuario solo tiene un único voto, se crean dos votos a un mismo libro de un mismo usuario y únicamente la media tiene en cuenta el voto más reciente.

```
def test01_vote(self):
    user = self.create_check(self.userDict, User)
    book = self.create_check(self.bookDict, Book)
    voteDict = {    "book": book,
                    "user": user,
                    "score": randrange(11)
                    }
    v1 = self.create_check(voteDict, Vote)
    self.assertEqual(user.pk, v1.user.pk)
    self.assertEqual(book.pk, v1.book.pk)

def test02_vote(self):
    user = self.create_check(self.userDict, User)
    book = self.create_check(self.bookDict, Book)
    voteDict = {    "book": book,
                    "user": user,
                    "score": 11
                    }
    v1=None
    try:
        v1 = self.create_check(voteDict, Vote)
    except:
        self.assertEqual(v1, None)
```

```python
def test03_vote(self):
    user = self.create_check(self.userDict, User)
    book = self.create_check(self.bookDict, Book)
    voteDict = {    "book": book,
                    "user": user,
                    "score": -1
                    }
    v1=None
    try:
        v1 = self.create_check(voteDict, Vote)
    except:
        self.assertEqual(v1, None)

def test04_vote(self):
    user = self.create_check(self.userDict, User)
    book = self.create_check(self.bookDict, Book)
    voteDict = {    "book": book,
                    "user": user,
                    "score": randrange(11)
                    }
    v1 = self.create_check(voteDict, Vote)
    self.assertLessEqual(v1.score, 10)
    self.assertGreaterEqual(v1.score, 0)
```

```python
def test05_vote(self):
    user = self.create_check(self.userDict, User)
    book = self.create_check(self.bookDict, Book)
    voteDict = {    "book": book,
                    "user": user,
                    "score": randrange(11)
                    }
    v1 = self.create_check(voteDict, Vote)

    userDict2 = {
        "username": 'Andiamo',
        "password": 'troncomovil',
        "first_name": 'Pablo',
        "last_name": 'Marmol',
        "email": 'p.marmol@cantera.com',
    }
    user2 = self.create_check(userDict2, User)

    voteDict2 = {    "book": book,
                    "user": user2,
                    "score": randrange(11)
                    }
    v2 = self.create_check(voteDict2, Vote)

    self.assertEqual(book.score, (v1.score+v2.score)/2)

    userDict3 = {
        "username": 'Pepe',
        "password": 'troncomovil',
        "first_name": 'Pablo',
        "last_name": 'Marmol',
        "email": 'p.marmol@cantera.com',
    }
    user3 = self.create_check(userDict3, User)

    voteDict3 = {    "book": book,
                    "user": user3,
                    "score": randrange(11)
                    }
    v3 = self.create_check(voteDict3, Vote)

    self.assertEqual(book.score, (v1.score+v2.score+v3.score)/3)
```

```python
def test06_vote(self):
    user = self.create_check(self.userDict, User)
    book = self.create_check(self.bookDict, Book)

    self.assertEqual(book.score, None)

def test07_vote(self):
    user = self.create_check(self.userDict, User)
    book = self.create_check(self.bookDict, Book)
    voteDict = {    "book": book,
                    "user": user,
                    "score": randrange(11)
                    }
    v1 = self.create_check(voteDict, Vote)


    voteDict2 = {    "book": book,
                    "user": user,
                    "score": randrange(11)
                    }
    v2 = self.create_check(voteDict2, Vote)

    self.assertEqual(book.score, v2.score)
```