# מבוא

פרוייקט ב-django מכיל הרבה יישומים.

התחלת פרוייקט ב-django:

django-admin startproject mylib

הפקודה מוסיפה directory בשם project­\_name במבנה הבא. לדוגמה, הוספת פרוייקט בשם mylib

.

`-- mylib

|-- manage.py

`-- mylib

|-- \_\_init\_\_.py

|-- settings.py

|-- urls.py

`-- wsgi.py

הוספת יישום catalog

בשלב הראשון נוסיף את אתרי ויישומי האדמיניסטרציה.

כדי להוסיף יישום לפרוייקט, למשל catalog, נבצע:

cd mylib

python manage.py startapp catalog

התוצאה היא directory עם קבצים של הפרוייקט:

.

`-- mylib

|-- **catalog התיקיה החדש שנוצרה**

| |-- \_\_init\_\_.py

| |-- admin.py

| |-- apps.py

| |-- migrations

| | `-- \_\_init\_\_.py

| |-- models.py

| |-- tests.py

| `-- views.py

|-- manage.py

`-- mylib

|-- \_\_init\_\_.py

|-- \_\_pycache\_\_

| |-- \_\_init\_\_.cpython-37.pyc

| `-- settings.cpython-37.pyc

|-- settings.py

|-- urls.py

`-- wsgi.py

קבצים אלו ישמשו בהמשך. נפרט את מטרותיהם בהמשך.

# רישום הפרוייקט ביישום - registration

בקובץ proj/mylib/mylib/settings.py מוסיפים שורה ל-list בשם INSTALLED\_APPS:

INSTALLED\_APPS = [

'django.contrib.admin',

'django.contrib.auth',

'django.contrib.contenttypes',

'django.contrib.sessions',

'django.contrib.messages',

'django.contrib.staticfiles',

'**catalog.apps.CatalogConfig**'

]

השורה מצביעה ל-class בשם CatalogConfig שנמצא בקובץ apps. קובץ זה נמצא ב-directory בשם catalog.

ה-directory, הקובץ, וה-class נוצרו במסגרת ביצוע הפקודה startapp.

## קביעת מסד הנתונים - Database

את הפרמטרים למסד הנתונים קובעים ב-list בשם DATABASES, בקובץ proj/mylib/mylib/settings.py.

ברירת המחדל היא למסד הנתונים sqlite3. בשלב זה נשאיר את הבחירה הזו. כהערה, אפשר לקבוע כל מסד נתונים רלציוני.

## קביעת איזור הזמן - Time Zone

בקובץ proj/mylib/mylib/settings.py קובעים משתנה בשם TIME\_ZONE למחרוזת של איזור הזמן. לדוגמה:

TIME\_ZONE = 'Asia/Jerusalem' # איזור הזמן בארץ

איזור הזמן בחוף המזרחי בארה"ב הוא America/New York.

רשימה מלאה של אזורי הזמן באתר https://en.wikipedia.org/wiki/List\_of\_tz\_database\_time\_zones.

# מיפוי כתובות - URL Mapper

מיפוי הכתובות מתבצע בקובץ mylib/urls.py. הקובץ נוצר במהלך ביצוע startapp, עם התוכן הבא:

"""mylib URL Configuration

The `urlpatterns` list routes URLs to views. For more information please see:

https://docs.djangoproject.com/en/2.1/topics/http/urls/

Examples:

Function views

1. Add an import: from my\_app import views

2. Add a URL to urlpatterns: path('', views.home, name='home')

Class-based views

1. Add an import: from other\_app.views import Home

2. Add a URL to urlpatterns: path('', Home.as\_view(), name='home')

Including another URLconf

1. Import the include() function: from django.urls import include, path

2. Add a URL to urlpatterns: path('blog/', include('blog.urls'))

"""

from django.contrib import admin

from django.urls import path

urlpatterns = [

path('admin/', admin.site.urls),

]

כעת, מוסיפים את הקוד הבא לקובץ local\_lib/urls.py:

# Use include() to add paths from the catalog application

from django.conf.urls import include

from django.urls import path

urlpatterns += [

    path('catalog/', include('catalog.urls')),

]

קוד זה מפנה את כל הפניות ל-catalog לניפוי בקובץ catalog/urls.

בשלב הבא, נפנה את ה-URL הראשי (127.0.0.1:8000) ליישום ה-catalog, מאחר וזה היישום היחידי בינתיים. את זה עושים עם הוספת הקוד הבא לקובץ local\_lib/urls.py:

urlpatterns += [

    path('catalog/', include('catalog.urls')),

]

# קבצים סטטיים - Static Files

קבצים סטטיים - CSS, JavaScript, תמונות - לא מטופלים ב-django. יש צורך להבציע ל-django על המיקום שלהם. את זה עושים עם הקוד הבא, גם הוא ב-local\_lib/urls.py:

# Use static() to add url mapping to serve static files during development (only)

from django.conf import settings

from django.conf.urls.static import static

urlpatterns += static(settings.STATIC\_URL, document\_root=settings.STATIC\_ROOT)

לסיום, יש להוסיף את התוכן הבא לקובץ חדש שישמר בשם catalog/urls.py:

from django.urls import path

from catalog import views

urlpatterns = [

]

## הגירה - Migrate

השלב האחרון לפני השלמת שלד הפרוייקט הוא הכירה. זה מבוצע ב-2 הפקודות הבאות:

python manage.py makemigrations

python manage.py migrate

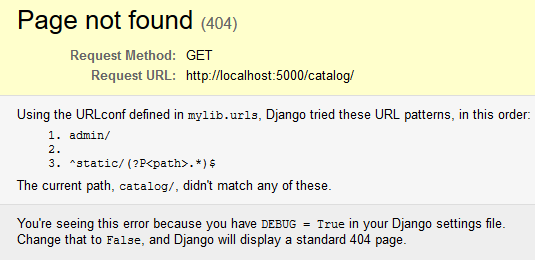
## הרצה

כעת אפשר להריץ את הפרוייקט, למרות שנקבל הודעת שגיאה. ההרצה היא בפקודה

python manage.py runserver [0.0.0.0:5000]

החלק בסוגריים המרובעים מיועד לעבודה בתוך Docker Container.

בשלב הזה נקבל את מסך השגיאה הבא:



השגירה אומרת לנו שה-URL המבוקש לא נמצא. זה הגיוני, היות ואין לנו עוד קבצי view.

לפני שנתחיל להציג נתונים, נחשוב על הנתונים שלנו, ועל הקשרים ביניהם.

ביישום שלנו של קטלוג הספריה, נכלול בשבל הראשון את הנתונים הבאים:

כותר, מחבר, ז'אנר, שפה, ספר.

לכל כותר יש מחבר אחד. בשלב הבא ננסה לשנות כל שלספר יכולים להיות מספר מחברים.

כל ספר שייך לז'נר אחד או יותר.

מחבר יכול לחבר ספר אחד או יותר.

ספר, שהוא עותק של עותר, תמיד מקושר לכותר יחיד.

כותר מקושר ל-0 או יותר ספרים (אפשרי שקיים ברשימה כותר שלא נמצא בספריה).

האיור הבא הוא ה-ERD של הנתונים האלו.



את הקשרים האלו מסכמים בקובץ catalog/models.py. הקוד הבא מדגים איך מגדירים שדה בטבלה.

השדה מוגדר בתוך class, שיורש מ-models.Model.

|  |  |
| --- | --- |
| from django.db import models  class MyModelName(models.Model):  """A typical class defining a model, derived from the Model class."""  # Fields  my\_field\_name = models.CharField(max\_length=20, help\_text='Enter field documentation')  ...  # Metadata  class Meta:  ordering = ['-my\_field\_name']  # Methods  def get\_absolute\_url(self):  """Returns the url to access a particular instance of MyModelName."""  return reverse('model-detail-view', args=[str(self.id)])    def \_\_str\_\_(self):  """String for representing the MyModelName object (in Admin site etc.)."""  return self.field\_name | דוגמה |

ב-class הזה מוגדר רק שדה אחד, שהוא string באורך 20.

יש עוד הרבה סוגים של שדות והרבה אפשרויות לכל שדה.

ה-class הפנימי בשם Metadata, הדוגמה הזו, מגדיר את ברירת המחדל של אופן המיון. התו "-" מראה שהמיון הוא בסדר הפוך.

שם ה-class הוא שם היישות (אובייקט), כמו כותר או שפה.

## מודל הספריה

מודל הספריה שמור בקובץ catalog/models.py.

### שפה

נתחיל בהוספת אובייקט שפה. זהו אובייקט עם שדה אחד בלבד. הקוד הבא יוסיף גם את הטבלה למסד הנתונים, וגם ייצור את ה-class עם השדות שמייצגים אותו.

class Genre(models.Model):

    """Model representing a book genre."""

    name = models.CharField(max\_length=200, help\_text='Enter a book genre (e.g. Science Fiction)')

    def \_\_str\_\_(self):

        """String for representing the Model object."""

        return self.name

ה-class כאן מוסיף יישות בשם Genre (ז'אנר). יש לה שדה אחד בשם name, שהוא מחרוזת באורך 200 תווים.

הפונקציה \_\_str\_\_ מחזירה את תוכן השדה. בצורה כזו, מתבצעת המרה של רשומה ב-class ל-string.

### ספר

ספר במודל שלנו כולל מספר שדות: כותרת, מחבר (במקרה הזה לספר יכול להיות מחבר יחיד), תקציר, ז'אנר, ו-isbn.

כותרת - שדה string באורך 200.

מחבר - מפתח זר. המחברים עצמם מוגדרים ב-class נפרד. כאמור, במקרה הזה מוגדר מחבר יחיד לכל ספר.

תקציר - שדה טקסט (לא string) באורך של 1,000 תווים.

הבדלים בין string (varchar) ל-text:

|  |  |  |
| --- | --- | --- |
| תכונה | string | text |
| אורך המחרוזת | משתנה | אורך קבוע: 65535 |
| נפח זיכרון | 1 + c אם c ≤ 256 | 2 + c |
|  | 2 + c, אחרת |  |
| אינדקס | יכול לשמש באינדקס | לא יכול לשמש באינדקס |

isbn - שדה string בן 13 תווים.

ז'אנר - שדה של רבים לרבים. זה בכלל שכל ספר יכול להיות משוייך למספר ז'נארים, וכל ז'אנר יכול להתאים למספר ספרים. ב-django לא צריך להגדיר את השדה שאיתו מקשרים בין היישויות. זה מבוצע "מאחורי הקלעים".

בנוסף להגדרת השדות, במודך מוגדרים ההמרה ל-string עם הפונקציה \_\_str\_\_ שמחזירה את הכותרת, ופונקציה שמחזירה את ה-URL. ה-url יכול להיות למשל בספריה, או בחנות ספרים מקוונת.

זהו הקוד ב-Python למודל הספר

from django.urls import reverse # Used to generate URLs by reversing the URL patterns

class Book(models.Model):

    """Model representing a book (but not a specific copy of a book)."""

    title = models.CharField(max\_length=200)

# Foreign Key used because book can only have one author, but authors can have multiple books

# Author as a string rather than object because it hasn't been declared yet in the file

    author = models.ForeignKey('Author', on\_delete=models.SET\_NULL, null=True)

    summary = models.TextField(max\_length=1000, help\_text='Enter a brief description of the book')

    isbn = models.CharField('ISBN', max\_length=13, help\_text='13 Character <a href="https://www.isbn-international.org/content/what-isbn">ISBN number</a>')

    # ManyToManyField used because genre can contain many books. Books can cover many genres.

    # Genre class has already been defined so we can specify the object above.

genre = models.ManyToManyField(Genre, help\_text='Select a genre for this book')

    def \_\_str\_\_(self):

        """String for representing the Model object."""

        return self.title

    def get\_absolute\_url(self):

        """Returns the url to access a detail record for this book."""

        return reverse('book-detail', args=[str(self.id)])

### עותק ספר - BookInstance

עותק ספר הוא ייצוג של ספר פיסי שאפשר לרכוש, או להשאיל מהספירה, בניגוד לספר (או כותר) שהוא משהו דמיוני - וירטואלי.

עותק ספר עולל ID, קישור לספר, הערות לעותק, ו-סטטוס

ID - מספר שלם שמוכדר על ידי UUIDField (Universally Unique IDentifier). המספר הזה מופק על ידי קריאה ל-models.UUIDField. הערה: לא ברור לי מדוע לא צריך ID לסופר, ספר וכדומה.

ספר - מפתח זר ליישות Book, שהוגדרה ב-class בשם זה.

הערות - שדה string באורך 200 תווים.

תאריך החזרה - שדה תאריך: models.DateField (…).

סטטוס - במקרה שלנו (יותר לצורך ההדגמה) סטטוד מוגבל לרשימה סגורה. כלומר, רשימה של מספר מוגבל של ערכים. הערכים האלו מוגדרים בקוד - hard coded - ושינוי שלהם יצריך שינוי תוכנה. את הערכים עצמם מגדירים במשתנה מסוג list של strings. הקישור בין השדה לאפשרויות מבוצע באחד הפרמטרים של הפונקציה models.CharField.

בנוסף לשדות, מוגדרים ב-class מתודה \_\_str\_\_ שכבר פגשנו, ו-class פנימי בשם Meta, שמכיל משתנה בשם ordering. משתנה זה קובע לפי איזה שדה תמויין רשימה של היישות (BooInstance).

באיור הבא מוצג הקוד של ה-class.

|  |
| --- |
| import uuid # Required for unique book instances  class BookInstance(models.Model):  """Model representing a specific copy of a book (i.e. that can be borrowed from the library)."""  id = models.UUIDField(primary\_key=True, default=uuid.uuid4, help\_text='Unique ID for this particular book across whole library')  book = models.ForeignKey('Book', on\_delete=models.SET\_NULL, null=True)  imprint = models.CharField(max\_length=200)  due\_back = models.DateField(null=True, blank=True)  LOAN\_STATUS = (  ('m', 'Maintenance'),  ('o', 'On loan'),  ('a', 'Available'),  ('r', 'Reserved'),  )  status = models.CharField(  max\_length=1,  choices=LOAN\_STATUS,  blank=True,  default='m',  help\_text='Book availability',  )  class Meta:  ordering = ['due\_back']  def \_\_str\_\_(self):  """String for representing the Model object."""  return f'{self.id} ({self.book.title})' |

### מודל המחבר - Author

למחבר נגדיר שם פרטי, שם משפיה, תאריך לידה, ותאריך מוות.

שם פרטי - שדה string באורך 100.

שם משפחה - שדה string באורך 100.

תאריך לידה - שדה תאריך - DateField

תאריך מוות - שדה תאריך - DateField. אם הסופר עוד בחיים, השדה הזה יהיה NULL.

מיון רשימה סופרים, שנקבע ב-class הפנימי Meta, יהיה לפי שם משפחה ושם פרטי. זה יהיה גם ההמרה של רשומה ל-string.

זהו הקוד של היישות

|  |
| --- |
| class Author(models.Model):      """Model representing an author."""      first\_name = models.CharField(max\_length=100)      last\_name = models.CharField(max\_length=100)      date\_of\_birth = models.DateField(null=True, blank=True)      date\_of\_death = models.DateField('Died', null=True, blank=True)      class Meta:          ordering = ['last\_name', 'first\_name']        def get\_absolute\_url(self):          """Returns the url to access a particular author instance."""          return reverse('author-detail', args=[str(self.id)])      def \_\_str\_\_(self):          """String for representing the Model object."""          return f'{self.last\_name}, {self.first\_name}' |

## הכנת מסד הנתונים

ההכנה - יצירת הטבלאות והקשרים - מתבצעת בעזרת 2 הפקודות הבאות:

python manage.py makemigrations

python manage.py migrate

אתגר

* נניח שבספריה יש ספרים בשפות שונות, כמו ערבית, אנגלית, עברית, מאראטי).
* לאיזה אובייקט השפה צריכה להיות מקושרת? לספר (כותר) או לעותק?
* האם השפה צריכה להיות מיוצגת כל ידי מודל (class נפרד), שדה טקסט להזנה חופשית, או רשימה סגורה, שהיא hard coded?

נא לנמק.

תשובות

* שפה צריכה להיות מקושרת לספר (כותר), ולא לעותק. זה בגלל שעותק של ספר תמיד יהיה כתוב בשפה שבה נכתב הספר. במקרה של תרגום יש צורך בספר (כותר) אחר, וגם הכותרת של הספר תהיה שונה.
* ייצוג שפה
* שדה להזנה חופשית - אפשרי אבל רצוי שלא. אנחנו רוצים לכתוב נתון פעם אחת במסד הנתונים, ובמקרה הזה התוצאה הפוכה: לדוגמה, אם יש 1000 כותרים בעברית, נכתוב את המילה עברים 1000 פעמים במסד הנתונים.
* רשימה סגורה - אפשרי אבל רצוי שלא. לא נרצה לשנות את התוכנה בכל פעם שנוסף ספר בשפה חדשה.
* מודל - אפשר לרצוי. הערך נכתב פעם אחת ב-database, ואפשר לשנות ולהוסיף בלי לשנות תוכנה.

## אדמיניסטרציה: ניהול הנתונים - admin

אפשר לכתוב יישום שמנהל הוספת נתונים ליישויות, כמו הוספת מחברים, ספרים, עותקים, וכדומה, וגם מחיקה ושינוי שלהם. אבל ב-django יש כבר יישום כזה, והוא נקרא, שלא במפתיע, admin.

כדי ש-admin יכיר את היישויות שהגדרנו בקובץ models.py, "רושמים" אותם - פעולה שנקראת registration -בקובץ catalog/admin.py. את זה עושם בפקודות הבאות שמוסיפים לקובץ:

from catalog.models import Author, Genre, Book, BookInstance

admin.site.register(Book)

admin.site.register(Author)

admin.site.register(Genre)

admin.site.register(BookInstance)

### מי?

שאלה קבועה במערכות מידע, היא "מי?" למי מותר או למי ניתנת הזכות והאחריות להריץ את יישום ה-admin.

התשובה כאן היא superuser. קביעת superuser מבוצעת אינטרקטיבית על ידי הפקודה

python manage.py createsuperuser

אפשר להוסיף super user גם עם הקוד הבא:

echo "from django.contrib.auth.models import User; User.objects.create\_superuser('*first*', '*first\_last@example.com*', '*last*')" | python manage.py shell

כעת נשאר להריץ את השרת, ואת הדפדפן:

הרצת השרת היא על ידי

python manage.py runserver [ip\_address:port]

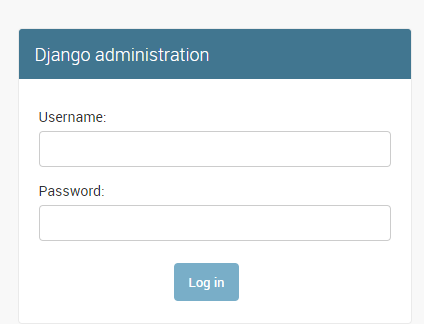
ובסביבת docker:

python manage.py runserver ip\_0.0.0.0:5000

בהרצת הדפדפן (בסביבת docker):

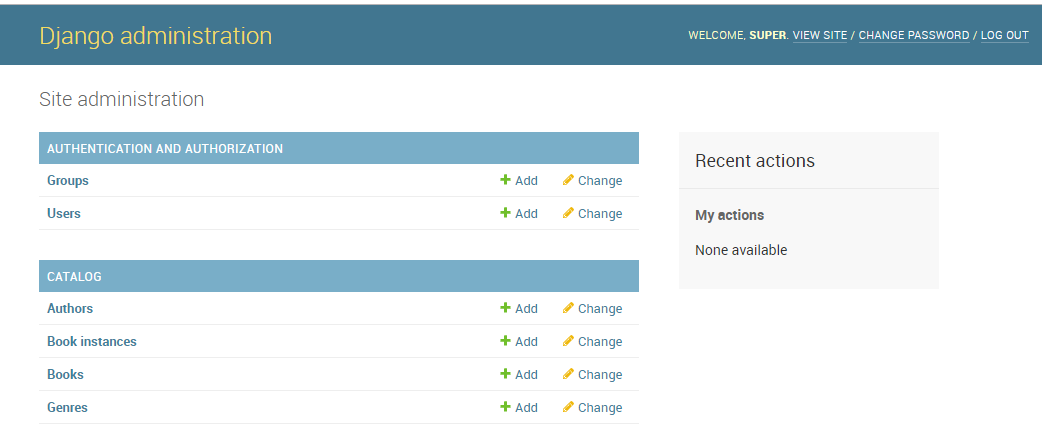
localhost:5000/admin

נקבל את המסך:



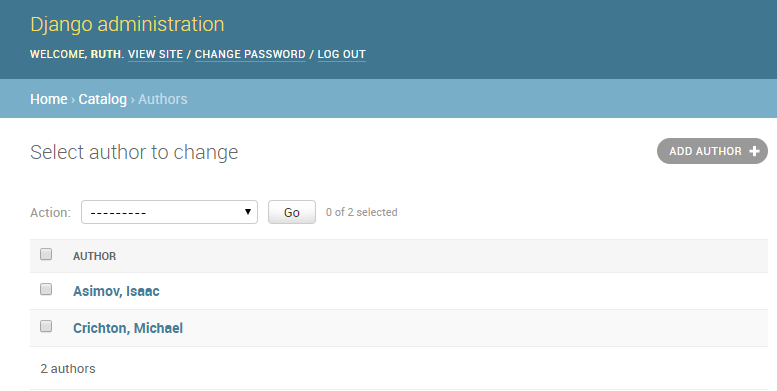
בדוגמה שלי הגדרתי שם משתמש user, וסיסמה user.

אחרי login נקבל את המסך הבא.



## שיפור הניהול - Admin

ברירת המחדל של הניהול מוצגת באיור הבא



התצוגה הזו סבירה כשיש מעט מחברים. גשיש הרבה פרטים - מעל עזרות - כדאי להוסיף עוד פרטים, ולהציג כטבלה עם אפשרות מיון.

את זה עושים על ידי תוספות בקובץ catalog/admin.py. לדוגמה, נשנה את תצוגת המחברים על ידי הקוד הבא

#admin.site.register(Author) # ביטול השורה על ידי הפיכתה להערה

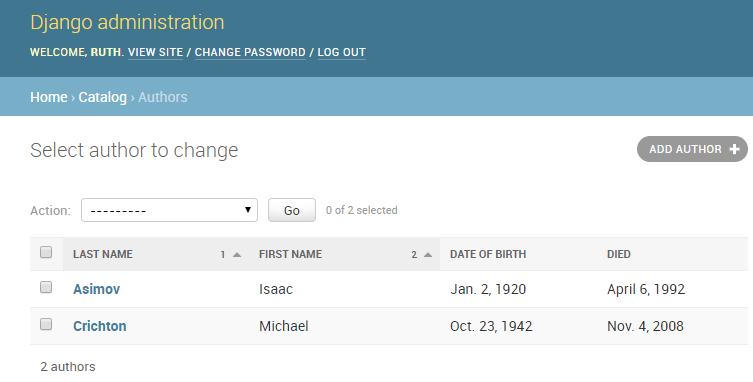
@admin.register(Author)

class AuthorAdmin(admin.ModelAdmin):

list\_display = ('last\_name', 'first\_name', 'date\_of\_birth', 'date\_of\_death')

מרחיבים את ההגדרה של Author עם דקורציה של class נוסף, בשם AuthorAdmin. ה-class הזה מרחיב את admin.ModelAdmin. על ידי הוספת תוכן למשתנה list\_display. התוכן הוא list של משתנים של Author. אם ננסה, למשל, להוסיף ערך שהוא לא שם של משתנה של ה-class נקבל run time error.

הקצה של השינוי תיתן לנו את המסך הבא.



כותרת עמודה נקבעת לפי שם השדה, שבו underscore, "\_", מוחלף ברווח.

קיימים מקרים בהם התוצאה אינה של משתנה יחליד, אלא של רבים. למשל ז'אנרים בספר. במקרה הזה כותבים פונקציה שמכילה את כל הפרטים.

במקרה של Book (עם הז'אנרים), נוסיף פונקציה - מתודה של ה-class - שמחזירה את כל הז'אנרים:

def display\_genre(self):

"""Create a string for the Genre. This is required to display genre in Admin."""

return ', '.join(genre.name for genre in self.genre.all()[:3])

היות והטקסט שמחזירה הפונקציה יוצג כשדה בנוסף, נגדיר משתנה class - member - שיכיל את הכותרת:

display\_genre.short\_description = 'Genre'

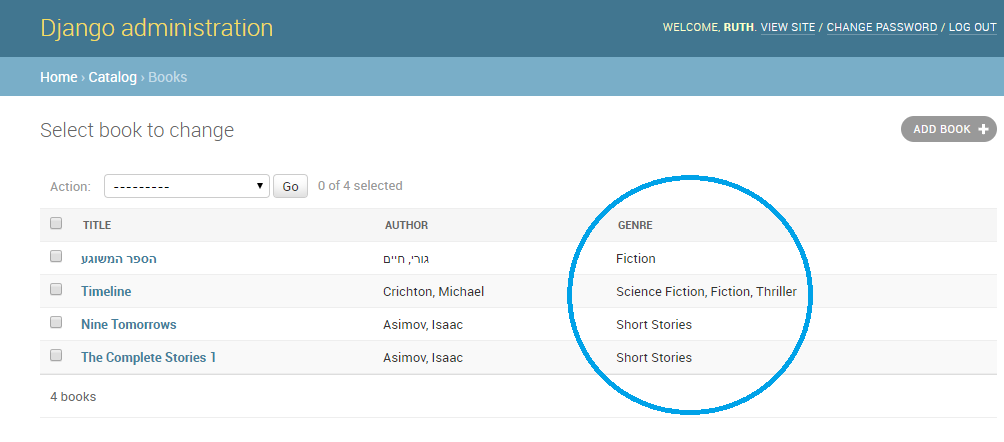
במקביל, נגדיר את השדה ביישום ה-admin של הספרים, על ידי הוספת איבר ל-list\_display (מודגש):

@admin.register(Book)

class BookAdmin(admin.ModelAdmin):

list\_display = ('title', 'author'**, 'display\_genre'**)

התוצאה של התוספת מוצעת באיור הבא, עם הדגשת העמודה החדשה



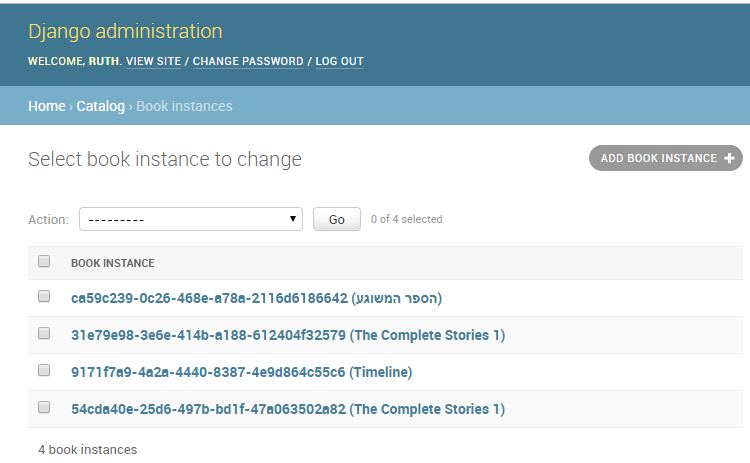
## סינון התצוגה - Filter

בהרבה מקרים, כשמוצגת רשימה ארוכה, רוצים לסנן אותה.

בסינון, מציגים רק את הרשומות שעומדות בקריטריון הסינון.

קריטריון הסינון ב-django נקבע על ידי המשתנה list\_filter. נממש ברשימת הספרים הפיסיים BookInstance, שבו נאפשר לסנן לפי תאריך ההחזרה והסטטוס.

לפני הוספת קריטריון הסינון נקבל את המסך הבא:



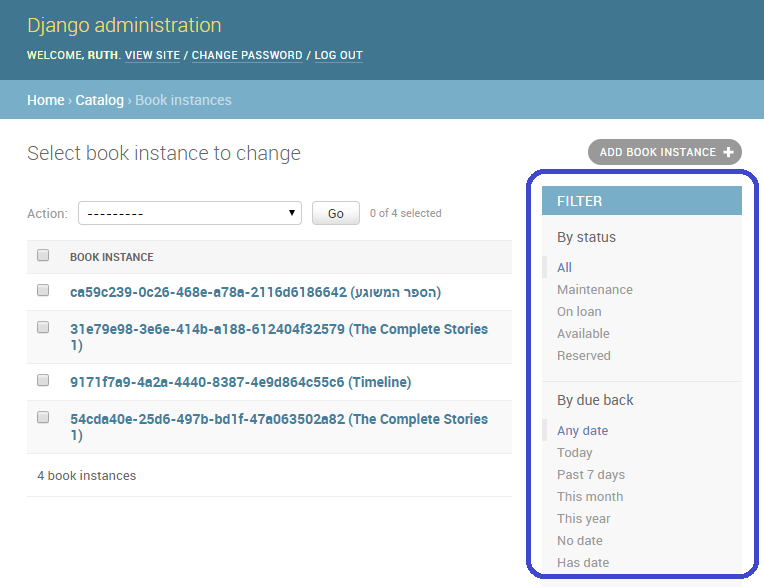
את הקריטריון הסינון נוסיף בקוד הבא:

@admin.register(BookInstance)

class BookInstanceAdmin (admin.ModelAdmin):

**list\_filter = ('status', 'due\_back')**

ונקבל



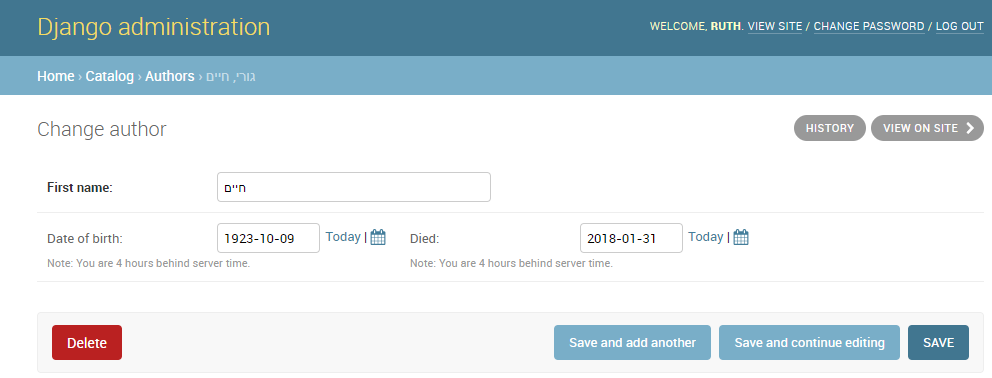
הערכים של status לקוחים מתוך האפשרויות של השדה.

בשדה התאריך הערכים מוגדרים ב-class בשם DateFieldListFilter, שהוא חלק מ-django.

אפשר גם לסנן את השדות שמוצגים ב-form של כל פריט, ב-list שנשמר במשתנה files, לדוגמה באופן הבא:

fields = ['first\_name', ('date\_of\_birth', 'date\_of\_death')]

במקרה הזה יוצג רק השם הפרטי של המחבר. תאריכי הלידה והפטירה יוצגו בשורה אחת, בגלל שהם בתוך tuple ב-list. התוצאה תהיה:



## מקטעים במסך הניהול - Admin Sectioning

ניתן לחלק את מסך ה-admin לקטעים, למשל לפי נושאים. נראה דוגמה בספרים הפיסיים. נחלק את המסך לקטע ראשי, קטע זיהוי, וקטע זמינות. זה מושג על ידי השמה למשתנה fieldsets, כמו בדוגמה הבאה:

fieldsets = (

('Main', {

'fields': ('book',)

}),

('Identification', {

#'fields': ('imprint', 'id')

'fields': ('imprint', 'id')

}),

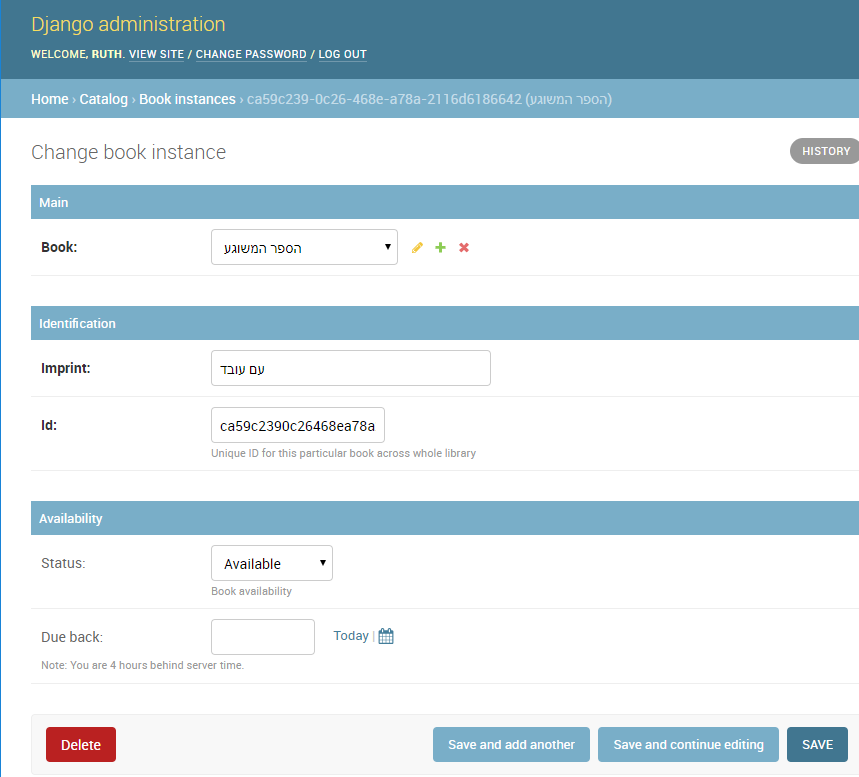
('Availability', {

'fields': ('status', 'due\_back')

}),

)

והתוצאה היא



## עריכה תוך כדי - inline editing

לפימים רוצים לערוך פריט אחד, ועל הדרך לראות (או לערוך) פרטים של פריט אחר. למשל, לערוך פרטים של ספר פיסי (BookInstance) תוך כדי עריכת פרטי הספר. זה נקרא inline editing בשפה של django.

את זה עושים עם 2 פעולות: הוספת class שמרחיב את TabularInline, והוספת שם ה-class ביישות שרוצים לערוך.

במקרה שלנו, נוסיף את class בשם BooksInstanceInline:

class BooksInstanceInline(admin.TabularInline):

model = BookInstance

זה ה-class שמסדר את העריכה של הספר הפיסי. כעת נקשור אותו ל-class שרוצים לערוך על הדרך. נוסיף שורה ל-BookAdmin, כך שנקבל את ה-class הבא.

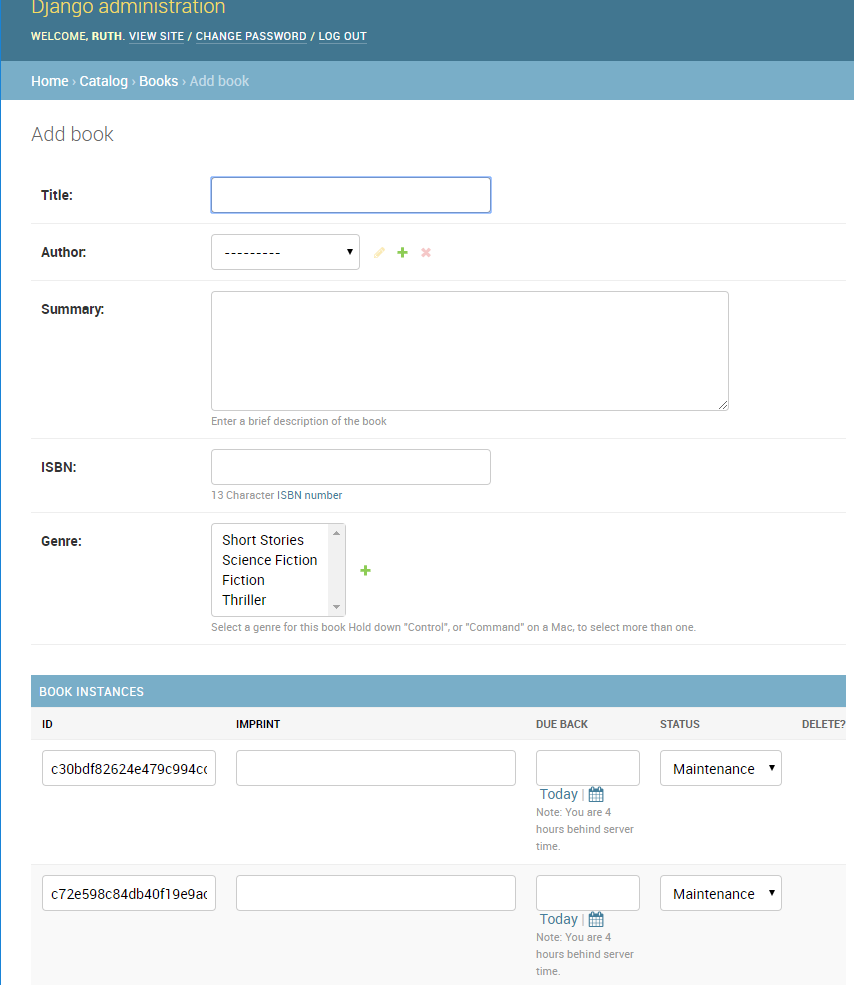
@admin.register(Book)

class BookAdmin(admin.ModelAdmin):

list\_display = ('title', 'author', 'display\_genre')

**inlines = [BooksInstanceInline]**

כעת, כשנערוך את פרטי הספר, או כשנוסיף ספר חדש, נוכל על הדרך גם להוסיף ספר פיסי. זו התוצאה המתקבלת בהוספת ספר:



רק צריך לשים לב לכך שהכפתורים בתחתית הדף.

# תצוגת דפים

עד עכשיו ראינו ניהול - admin - הגדרה של מודלים ועריכת הנתונים.

כעת נציג את תוכן האתר. הצגת הנתונים מבוצעת לפי התרשים הבא.



הדפדפן מפיק שאילתת HTTP, ומפנה אותה לאתר. האתר מפנה לקובץ url.py, שמפנה ל-view המתאים. ה-view לוקח את הפקודה, מקבל נתונים מהמודל, ונכין תגובת HTTP בהתאם לגלופה - template.

האתר צריך להראות מחברים, ספרים, עותקים ועוד. עדיף להראות נושא אחד בתוך דף ייעודי.

## דף המבוא

הגדרת ה-URL

נצטרך את הדפים וה-URLs הבאים:

* דף בית - catalog/
* דף ספרים = catalog/books/
* דף סופרים - catalog/authors/
* דף לספר ספציפי - catalog/book/<id>
* דף למחבר ספציפי - catalog/author/<id>
* הדפים האלו יהיו read only לקוראים
* יצירת דף הבית - Index

הדף יכלול כמו יישויות סטטיות, וסיכומים מה-database.

כשהדפדפן פונה ל-dsjango עם כתובת, הכוונה היא בעצם לקבל דף HTML. הקישור בין כתובת - URL למעשה - לבין הפונקציה שצריכה להחזיר את הדף, נמצא בקובץ local\_lib/urls.py.

בשלב הראשון, django מתחיל באיתור הכתובת ברשימת הכתובות שבאחריותו.

הרשימה הזו מוגדרת במשתנה urlpatterns, שמוגדר בקובץ local\_lib/urls.py. המשתנה urlpatterns מכיל list של class מסוג URLResolve. המטרה של ה-class היא להחזיר פונקציה מתוך URL, שתחזיר את ה-HTML המתאים. הפונקציה נקראת ה-view של ה-URL.

את התוכן של urlpatterns מגדירי עם קריאה לפונקציה path (django.urls.path). הפונקציה הזו מקבלת 4 ארגומנטים, מהם 2 חובה:

route - ה-URL

view - הפונקציה שמטפלת ב-URL. שיםן לב: ב-python פרוצדורה נשמרת כמשתנה.

kwargs - פרמטרים לפונקצית ה-view.

name - שם ייחודי מזהה. השם מתייחס לפרמטר שמופיע בתוך link, כמו במקרה הבא:

<a href="{% url '**index**' %}">Home</a>

בשלב הראשון נגדיר את ה-URL של דף הבית: הדף בלי תוספות, שנקבל בכתובת:

localhost:5000

נקבע ערך ל-urlpatterns באופן הבא:

urlpatterns = [

**path('', views.index, name='index'),**

]

הארגומנט הראשון מתייחס למה שמופיע אחרי כתובת האתר, כלומר אחרי localhost: מופיע כלום. הפונקציה המטפלת היא index בקובץ views.py.

## ה-View

ה-View שלנו מחזיר דף HTML, עם סיכומים שאותם הוא שולף מה-database, ומעביר כפרמטרים לדף התוצאה. במקרה שלנו, הסיכומים נשלפים על ידי הקריאה

*object*.all.count()

הארגומנט לדף מועבר בתוך dictionary לארגומנט בשם contect, בצורה הבאה

return render(request, 'index.html', context=context)

הארגומנט request מתקבל בעצמו כארגומנט ב-view. index.hhtml הוא ה-template, והארכגומנטים, כאמור, ב-context.

תצוגת ה-HTML מטופלת בקובץ index.html. הקובץ מרחיב את הגלופה הבסיסית, ומציג את הסיכומים שקיבל כפרמטרים.

<p>The library has the following record counts:</p>

<ul>

<li><strong>Books:</strong> {{ num\_books }}</li>

<li><strong>Copies:</strong> {{ num\_instances }}</li>

<li><strong>Copies available:</strong> {{ num\_instances\_available }}</li>

<li><strong>Authors:</strong> {{ num\_authors }}</li>

</ul>

וזו התוצאה בדפדפן.

## דף הספרים

פרוצדורת ה-view פשוטה מאוד, והיא נכתבת בקובץ catalog/view.py:

from django.views import generic

class BookListView(generic.ListView):

model = Book

התצוגה תתבצע בגלופה בשם local\_lib/templates/catalog/book\_list.html.

זוהי ברירת המחדל.

איך הגיעו לשם המוזר הזה? אז ככה.

ראשית, django מחפש template מהפורמט *app\_name*\_list.html. שנית, החיפוש הוא בתיקיה *app\_name*/templates.

גם שם המשתנה קבוע מראש, והוא נקרא objname\_list, או book\_list במקרה שלנו.

כל הפרמטרים האלו הם ברירות מחדל, וניתן לשנות אותם.

לדוגמה, נניח שנרצה להציג רק 7 ספרים המחילים את המילה love בכותרת.

נקבע את שם המשתנה ל-my\_book\_list, ואת הגלופה נשמור בשם books/my\_simple\_template\_name.html.

ככה תראה בפרוצדורה, כשהשינויים לברירות המחדל מודגשים:

class BookListView(generic.ListView):

model = Book

context\_object\_name = '**my\_book\_list**'

queryset = Book.objects.filter(title\_\_icontains='**love**')[:5] #

template\_name = '**books/my\_arbitrary\_template\_name\_list.html**'

אפשר להוסיף לרשימת הארגומנטים, במה שנקרא context data, רק שצריך לעשות את זה בצורה מסודרת.

שמירה של הארגומנטים הקיימים.

הוספה של הארגומנטים החדשיםהחזרת התוצאה.

class BookListView(generic.ListView):

model = Book

    def get\_context\_data(self, \*\*kwargs):

        # Call the base implementation first to get the context

        context = super(BookListView, self).get\_context\_data(\*\*kwargs)

        # Create any data and add it to the context

        context['some\_data'] = 'This is just some data'

        return context

## הגלופה של רשימת הספרים

נשמור את הגלופה בשם של ברירת המחדל: local\_lib/catalog/templates/catalog/book\_list.html.

נוסיף לקובץ את הקוד הבא.

{% extends "base\_generic.html" %}

{% block content %}

<h1>Book List</h1>

{% if book\_list %}

<ul>

{% for book in book\_list %}

<li>

<a href="{{ book.get\_absolute\_url }}">{{ book.title }}</a> ({{book.author}})

</li>

{% endfor %}

</ul>

{% else %}

<p>There are no books in the library.</p>

{% endif %}

{% endblock %}

שים לב שהלולאה היא על list בשם book\_list, שהיא ברירת המחדל.

כעת, נוסיף את הקישור. כבר ראינו שכדי להגיע ליישום צריך להוסיף catalog. נגיע לרשימת הספרים נוסיף books. המימוש הוא על ידי הוספת השורה המודגשת לקובץ catalog/urls.py:

urlpatterns = [

path('', views.index, name='index'),

**path('books/', views.BookListView.as\_view(), name='books'),**

]

נגיע לקישור books רק מתוך catalog, ככה שהקישור המלא הוא catalog/books.

אגב, הפונקציה as\_view() עושה את כל העבודה.

נוסיף גם לקישור בדף הבית (base\_generic.html), ככה שהקישור books יביא אותנו באמת לספרים. התוספת מודגשת:

<ul class="sidebar-nav">

<li><a href="{% url 'index' %}">Home</a></li>

<li><a href="**{% url 'books' %]**">All books</a></li>

<li><a href="">All authors</a></li>

</ul>

במקום לכתוב פונקציית view כמו שעשינו לדף הבית, אפשר להשתמש ב-class קיים, בשם generic.ListView.

איך עושים את זה?

מימוש ה-view הוא ב-class בשם BooksListView. זה מה שקבענו בקובץ urls.py. נקבע שה-class הזה ירחיב את generic.ListView, ומכאן זה ממש פשוט. ה-class הזה, אגב, ממומש בקובץ catalog/views.py:

from django.views import generic

class BookListView(generic.ListView):

model = Book

## פרטי הספר

מיפוי url מבוצע בקובץ urls.py, על ידי הוספת השורה המודגשת.

urlpatterns = [

path('', views.index, name='index'),

    path('books/', views.BookListView.as\_view(), name='books'),

**path('book/<int:pk>', views.BookDetailView.as\_view(), name='book-detail'),**

]

הערה: הצרוף int:pk מסמל, לדעתי, integer primary key.

הסוגרים המשולשים סוגרים על הארגומנט שיועבר ל-View. במקרה הזה, ה-":" מסמן את שם הארגומנט - pk - מפתח ראשי לטבלת הספרים.

הערות:

השם pk מיועד ל-view הגנרי. ב-view עצמאי אפשר להשתמש בכל שם שרוצים.

אפשר להוסיך סינון נוסף ל-url, בעזרת הפונקציה re‑path.

צריך לשים לב לכך ש-pk מופיע גם ב-regula expression. הטבלה הבאה מסכמת חלק מהפקודות ב-regula expression:

| Symbol | Meaning |
| --- | --- |
| ^ | Match the beginning of the text |
| $ | Match the end of the text |
| \d | Match a digit (0, 1, 2, ... 9) |
| \w | Match a word character, e.g. any upper- or lower-case character in the alphabet, digit or the underscore character (\_) |
| + | Match one or more of the preceding character. For example, to match one or more digits you would use \d+. To match one or more "a" characters, you could use a+ |
| \* | Match zero or more of the preceding character. For example, to match nothing or a word you could use \w\* |
| ( ) | Capture the part of the pattern inside the brackets. Any captured values will be passed to the view as unnamed parameters (if multiple patterns are captured, the associated parameters will be supplied in the order that the captures were declared). |
| (?P<*name*>...) | Capture the pattern (indicated by ...) as a named variable (in this case "name"). The captured values are passed to the view with the name specified. Your view must therefore declare an argument with the same name! |
| [  ] | Match against one character in the set. For example, [abc] will match on 'a' or 'b' or 'c'. [-\w] will match on the '-' character or any word character. |

בהרחבה ל-pattern

| **Pattern** | **Description** |
| --- | --- |
| **r'^book/(?P<pk>\d+)$'** | This is the RE used in our URL mapper. It matches a string that has book/ at the start of the line (**^book/**), then has one or more digits (\d+), and then ends (with no non-digit characters before the end of line marker).  It also captures all the digits **(?P<pk>\d+)** and passes them to the view in a parameter named 'pk'. **The captured values are always passed as a string!**  For example, this would match book/1234 , and send a variable pk='1234' to the view. |
| **r'^book/(\d+)$'** | This matches the same URLs as the preceding case. The captured information would be sent as an unnamed argument to the view. |
| **r'^book/(?P<stub>[-\w]+)$'** | This matches a string that has book/ at the start of the line (**^book/**), then has one or more characters that are *either* a '-' or a word character (**[-\w]+**), and then ends. It also captures this set of characters and passes them to the view in a parameter named 'stub'.  This is a fairly typical pattern for a "stub". Stubs are URL-friendly word-based primary keys for data. You might use a stub if you wanted your book URL to be more informative. For example /catalog/book/the-secret-garden rather than /catalog/book/33. |

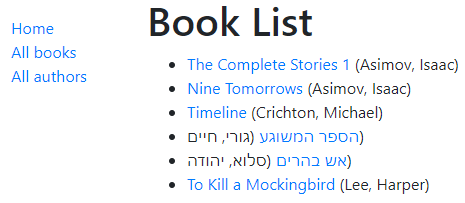
לפעמים רוצים להשתמש באותו view למספר אובייקטים. במקרה הזה, מעבירים פרמטר ל-view שמוגדר בתורת dictionary. כמו בדוגמה הבאה

path('url/', views.my\_reused\_view, **{'my\_template\_name': 'some\_path'}**, name='aurl'),

path('anotherurl/', views.my\_reused\_view, **{'my\_template\_name': 'another\_path'}**, name='anotherurl'),

החלק שבסוגרים המשולשים נקרה captured pattern.

התוצאה של רשימת הספרים היא המסך הבא.



ל-link בכל שורה נוסף ה-ID של הספר. לדוגמה, זו השורה של הספר האחרון ברשימה:

<a href="/catalog/book/6">To Kill a Mockingbird</a>

הסיבה לכך שאין link על שם המחבר היא שלא יצרנו תצוגה כזו.

### מימוש ה-view לספר

המימוש מצריך עם פונקציית תצוגה וגם גלופה.

### פונקציית התצוגה

פונקציית התצוגה מוגדרת בקובץ view.py. כמו שראינו קודם, שם ה-class שמממש את ה-view של הספר הוא BookDetailView, שמוגדר בקובץ view.py. ההגדרה מרחיבה את generic.DetailView:

class BookDetailView(generic.DetailView):

    model = Book

אם הספר לא נמצא, מוצג דף 404.

אפשר לממש עצמאית על ידי שימוש בפונקציה get\_object\_or\_404, בצורה הבאה:

from django.shortcuts import get\_object\_or\_404

def book\_detail\_view(request, primary\_key):

  book = get\_object\_or\_404(Book, pk=primary\_key)

return render(request, 'catalog/book\_detail.html',   
context={'book': book})

כמובן שיש צורך להגדיר את book\_detail\_view בקובץ urls.py.

### הגלופה

הגלופה - קובץ html עם קוד jinja - צפויה להימצא בקובץ בשם

catalog/templates/catalog/*object-name*\_detail.html.

הנה דוגמה לקובץ

{% extends "base\_generic.html" %}

{% block content %}

<h1>Title: {{ book.title }}</h1>

<p><strong>Author:</strong> <a href="">{{ book.author }}</a></p> <!-- author detail link not yet defined -->

<p><strong>Summary:</strong> {{ book.summary }}</p>

<p><strong>ISBN:</strong> {{ book.isbn }}</p>

<p><strong>Language:</strong> {{ book.language }}</p>

<p><strong>Genre:</strong> {% for genre in book.genre.all %} {{ genre }}{% if not forloop.last %}, {% endif %}{% endfor %}</p>

<div style="margin-left:20px;margin-top:20px">

<h4>Copies</h4>

{% for copy in book.bookinstance\_set.all %}

<hr>

<p class="{% if copy.status == 'a' %}text-success{% elif copy.status == 'm' %}text-danger{% else %}text-warning{% endif %}">{{ copy.get\_status\_display }}</p>

{% if copy.status != 'a' %}

<p><strong>Due to be returned:</strong> {{copy.due\_back}}</p>

{% endif %}

<p><strong>Imprint:</strong> {{copy.imprint}}</p>

<p class="text-muted"><strong>Id:</strong> {{copy.id}}</p>

{% endfor %}

</div>

{% endblock %}

למעשה, החלפנו את החלק התחתון בטבלה, כך שזה התוכן:

{% extends "base\_generic.html" %}

{% block content %}

<h1>Title: {{ book.title }}</h1>

<p><strong>Author:</strong> <a href="">{{ book.author }}</a></p> <!-- author detail link not yet defined -->

<p><strong>Summary:</strong> {{ book.summary }}</p>

<p><strong>ISBN:</strong> {{ book.isbn }}</p>

<p><strong>Language:</strong> {{ book.language }}</p>

<p><strong>Genre:</strong> {% for genre in book.genre.all %} {{ genre }}{% if not forloop.last %}, {% endif %}{% endfor %}</p>

<div style="margin-left:20px;margin-top:20px">

<h4>Copies</h4>

<table border="1" width="80%">

<tr>

<th><b>status</b></th>

<th><b>Imprint</b></th>

<th><b>Due to be returned</b></th>

</tr>

{% for copy in book.bookinstance\_set.all %}

<tr>

<td><p class="{% if copy.status == 'a' %}text-success{% elif copy.status == 'm' %}text-danger{% else %}text-warning{% endif %}">{{ copy.get\_status\_display }}</p></td>

<td> {{copy.imprint}} </td>

<td> {% if copy.status != 'a' %}

<p>{{copy.due\_back}}</p>

{% endif %}

</td>

</tr>

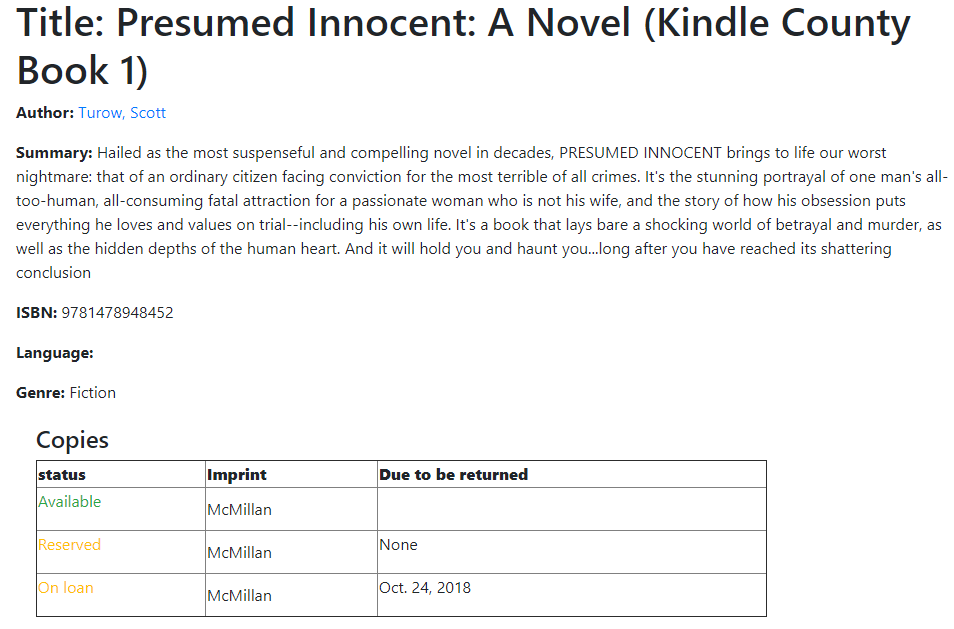
{% endfor %}

</table>

</div>

{% endblock %}

התוצאה של הדף הזה מוצגת באיור הבא.



### חלוקה לדפים - Pagination

כשיש מעט רשומות, התצוגה של רשימת הספרים יכולה להספיק. כשיש עשרות אלפים, צריך להגביל את הרשימה. אחת האפשרויות היא עם חלוקה לדפים.

נדגים את החלוקה לדפים ברשימת הספרים.

החלוקה הזו מבוצעת גם בתצוגה - view.py - וגם בגלופה, בקובץ base\_generic.html.

התצוגה

כדי להוסף חלוקה לדפים בתצוגה, מספיק להוסיף את השורה המודגשת, כאמור בקובץ view.py.

class BookListView(generic.ListView):

model = Book

**paginate\_by = 4**

### הגלופה

כדי להציג את הדפים על המסך, צריך גם לתמוך בחלוקה לדפים ברמת ה-HTML. את זה עושים על ידי הוספת הקוד הבא

**{% block pagination %}**

**{% if is\_paginated %}**

**<div class="pagination">**

**<span class="page-links">**

**{% if page\_obj.has\_previous %}**

**<a href="{{ request.path }}?page={{ page\_obj.previous\_page\_number }}">previous</a>**

**{% endif %}**

**<span class="page-current">**

**<p>Page {{ page\_obj.number }} of {{ page\_obj.paginator.num\_pages }}.</p>**

**</span>**

**{% if page\_obj.has\_next %}**

**<a href="{{ request.path }}?page={{ page\_obj.next\_page\_number }}">next</a>**

**{% endif %}**

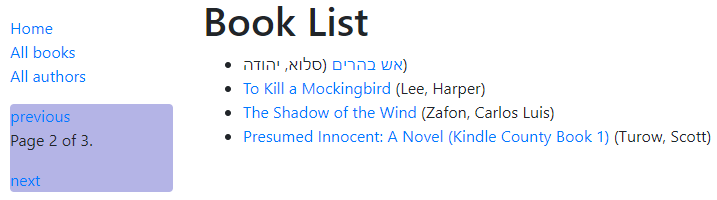
**</span>**

**</div>**

**{% endif %}**

**{% endblock %}**

כל נראת התוצאה:



הערה: הרקע הכהה לבלוק רק כאן לצורך ההדגשה.

תרגיל: להוסיף רשימת סופרים ודף פרטי הסופר.

פתרון:

1. הוספת link לתפריט. בצד שמאל של כל דף יש קישור לדף הבית, לרשימת הספרים ולרשימת הסופרים. הקישורים האלו מוגדרים בגלופת הבסיס catalog/template/base\_generic.html. שם נוסיף את הקישור (מודגש), בדומה לקישור לרשימת הספרים.

<ul class="sidebar-nav">

<li><a href="{% url 'index' %}">Home</a></li>

<li><a href="{% url 'books' %}">All books</a></li>

<li><a href="**{% url 'authors' %}**">All authors</a></li>

</ul>

ניסיון להריץ כעת את היישום יתקל בהודעת שגיאה, משום שעוד לא הוספנו את הגלופה לרשימת הסופרים.

המפתח ל-url הוא השם בין ה-quotes: authors. השם הזה נמצא בקובץ catalog/urls.py, כארגומנט בשם name לפונקציה path/

1. הוספת ה-path לקובץ catalog/urls.py

from django.urls import path

from catalog import views

urlpatterns = [

path('', views.index, name='index'),

path('books/', views.BookListView.as\_view(), name='books'),

path('book/<int:pk>', views.BookDetailView.as\_view(), name='book-detail'),

**path('authors/', views.AuthorsListView.as\_view(), name='authors'),**

**path('author/<int:pk>', views.**AuthosDetailView**.as\_view(), name='author-detail')**,

]

את השורה השניה הוספנו על הדרך. נפרט בהמשך.

בנוסף לשם, הוספנו גם הפניה לשני class-ים חדשים. אלו ה-class-ים שמטפלים בנתוני התצוגה.

1. הוספת class-ים בקובץ catalog/view.py. שימו לב להתאמות הצבעים:

class AuthorsListView(generic.ListView):

model = Author

class AuthorDetailView(generic.DetailView):

model = Author

הוספנו path, הוספנו class, וכעת נשאר ליצור את הגלופה. חפרנו קצת על זה, קודם. שם הגלופה צריך להיות שם האובייקט, עם \_list.html אחריו.

1. נוסיף את הגלופה templates/catalog/author\_list.html, על בסיס book\_list.html:

{% extends "base\_generic.html" %}

{% block content %}

<h1>Authors List</h1>

{% if author\_list %}

<ul>

{% for author in author\_list %}

<li>

<a href="{{ author.get\_absolute\_url }}">{{ author.last\_name }} {{ author.first\_name }}</a>

</li>

{% endfor %}

</ul>

{% else %}

<p>There are no authors in the library.</p>

{% endif %}

{% endblock %}

בלולאת ה-for, נוסף link לכל סופר, כשכותרת היא שם הסופר. ה-link הוא לפרטי הסופר.

ניצור את הקובץ החסר, למשל עם touch, ונקבל את רשימת הסופרים



כעת נרצה שה-link של הסופר, שמוביל לדף של פרטי הסופר, יכיל את הפרטים של הסופר ואת והספרים שלו.

1. 5. נוסיף את הגלופה לפרטי הספר, בקובץ catalog/template/catalog/author\_detail.html:

{% extends "base\_generic.html" %}

{% block content %}

<h2>{{ author.full\_name }}</h2>

<p>{{author.date\_of\_birth }} - {{author.date\_of\_death }}

<hr>

{% for book in author.book\_set.all %}

<a href="{{ book.get\_absolute\_url }}">{{ book.title }}</a><br>

{% endfor %}

{% endblock %}

התוצאה היא:

