Runbook to create a

Django Project

1. Creating Virtual Environment

Virtual environment will isolate your Python/Django setup on a per-project basis. This means that any changes you make to one project won't affect any others you're developing.

A. Create a project directory and change to it:

`mkdir django-project`

`cd django-project`

```
C:\Users\ravi>mkdir django-project

C:\Users\ravi>cd django-project

C:\Users\ravi\django-project>_
```

B. Run `pip install virtualenv` in your project directory.

```
C:\Users\ravi>cd django-project

C:\Users\ravi\django-project>pip install virtualenv
```

C. Create a virtual environment by running the following code:

`virtualenv django-project`

```
C:\Users\ravi\django-project>virtualenv django-project
Using base prefix 'c:\\programdata\\anaconda3'
New python executable in C:\Users\ravi\django-project\django-project\Scripts\python.exe
Installing setuptools, pip, wheel...
done.
```

D. Activate the created virtual environment:

`django-project\Scripts\activate.bat`

```
C:\Users\ravi\django-project>django-project\Scripts\activate.bat

(django-project) C:\Users\ravi\django-project>
```

2. Installing Django

Now install django in the newly created virtual environment `django-project`

pip install django

```
(django-project) C:\Users\ravi\django-project>pip install django
Collecting django
  Using cached https://files.pythonhosted.org/packages/a0/36/463632a2e9161a7e713488d719a280e8cb0c7e3a66ed32a32e801891caa
e/Django-2.2.7-py3-none-any.whl
Collecting sqlparse
  Using cached https://files.pythonhosted.org/packages/ef/53/900f7d2a54557c6a37886585a91336520e5539e3ae2423ff1102daf4f3a
7/sqlparse-0.3.0-py2.py3-none-any.whl
Collecting pytz
  Using cached https://files.pythonhosted.org/packages/e7/f9/f0b53f88060247251bf481fa6ea62cd0d25bf1b11a87888e53ce5b7c8ad
2/pytz-2019.3-py2.py3-none-any.whl
Installing collected packages: sqlparse, pytz, django
Successfully installed django-2.2.7 pytz-2019.3 sqlparse-0.3.0
```

The current version of Django is 2.2.7 as of now.

Note: The virtual environment folder `django-project` has nothing to do except for activating it. All the commands executed in `django-project` directory indicate project directory.

3. Creating a Django Project

Creating a django project involves running some scripts provided by Django that will create the skeleton of a Django project. Django needs to maintain a certain structure to be able to find important things.

Note: Remember to run everything in the `virtualenv`. If you don't see a prefix `(django-project)` in your console, you need to activate your `virtualenv`. Typing **`django-project\Scripts\activate.bat`** will do the job.

Run the following command in the project directory `django-project`:

`django-admin startproject mobis`

```
(django-project) C:\Users\ravi\django-project>django-admin startproject mobis

(django-project) C:\Users\ravi\django-project>
```

`django-admin.py` is a script that will create the directories and files for you. You should now have a directory structure which looks like this:

```
django-project
├── mobis
│   ├── mobis
│   │   ├── __init__.py
│   │   ├── settings.py
│   │   ├── urls.py
│   │   └── wsgi.py
│   └── manage.py
└── django-project
    └── wsgi.py
```

`manage.py` is a script that helps with management of the site. With it we will be able (amongst other things) to start a web server on our computer without installing anything else.

The `settings.py` file contains the configuration of your website.

`urls.py` file contains a list of patterns used by `urlresolver`.

4. Starting the Web Server

A. Change to directory mobis

`cd mobis`

```
(django-project) C:\Users\ravi\django-project>cd mobis

(django-project) C:\Users\ravi\django-project\mobis>_
```

B. Starting the web server

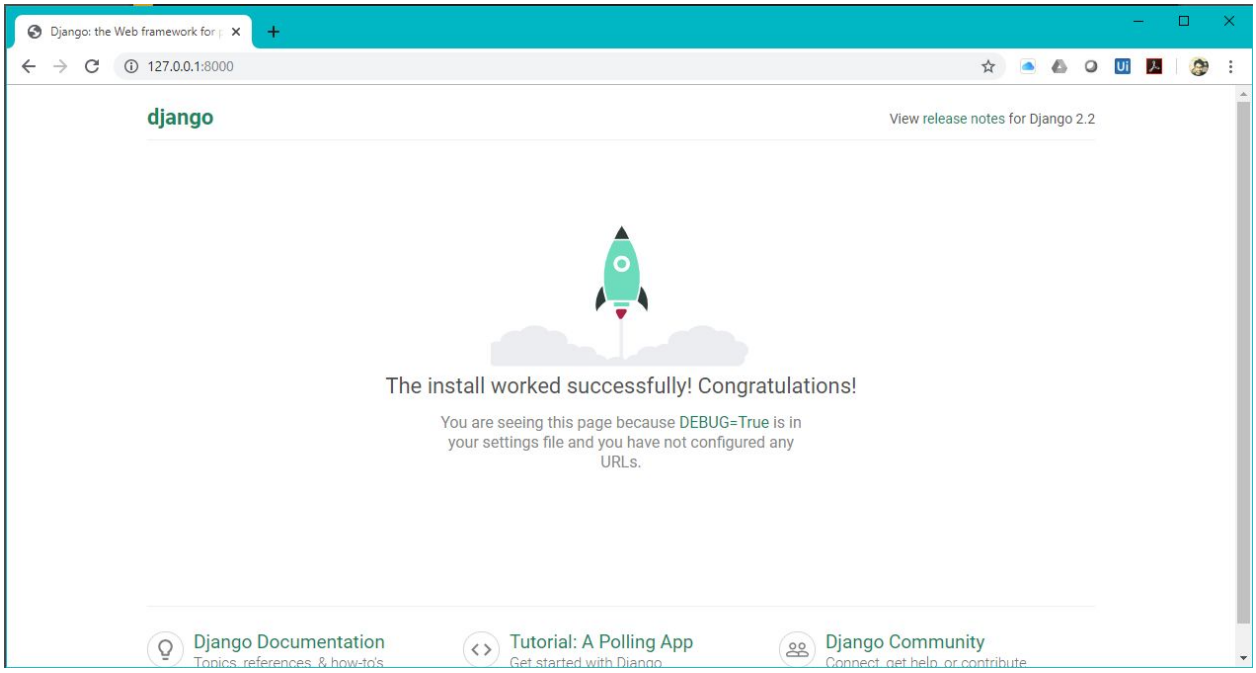
`python manage.py runserver`

```
(django-project) C:\Users\ravi\django-project\mobis>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...

System check identified no issues (0 silenced).

You have 17 unapplied migration(s). Your project may not work properly until you apply the migrations for app(s): admin,
 auth, contenttypes, sessions.
Run 'python manage.py migrate' to apply them.
November 13, 2019 - 11:19:17
Django version 2.2.7, using settings 'mobis.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

C. Open a web browser and enter `http://127.0.0.1:8000` to verify.



Note that a command window can only run one thing at a time, and the command window you opened earlier is running the web server.

As long as the web server is running and waiting for additional incoming requests, the terminal will accept new text but will not execute new commands.

To type additional commands while the web server is running, open a **new terminal window** and activate your `virtualenv`.

```
cd django-project
django-project\Scripts\activate.bat
```

```
Microsoft Windows [Version 10.0.18362.418]
(c) 2019 Microsoft Corporation. All rights reserved.

C:\Users\ravi>cd django-project\

C:\Users\ravi\django-project>django-project\Scripts\activate.bat

(django-project) C:\Users\ravi\django-project>cd mobis

(django-project) C:\Users\ravi\django-project\mobis>
```

To stop the web server, switch back to the window in which it's running and press CTRL+C - Control and C keys together.

5. Creating an Application

To create an application we need to run the following command in the new console (from `mobis` directory where `manage.py` file is):

```
cd mobis
python manage.py startapp bookshelf
```

```
(django-project) C:\Users\ravi\django-project\mobis>python manage.py startapp bookshelf

(django-project) C:\Users\ravi\django-project\mobis>
```

You will notice that a new `bookshelf` directory is created and it contains a number of files now. The directories and files in the project should look like this:

```
django-project
├── mobis
│   ├── mobis
│   │   ├──init__.py
│   │   ├── settings.py
│   │   ├── urls.py
│   │   └── wsgi.py
│   ├── bookshelf
│   │   ├── admin.py
│   │   ├── apps.py
│   │   ├── __init__.py
│   │   ├── migrations
│   │   │   └── __init__.py
│   │   ├── models.py
│   │   ├── tests.py
│   │   └── views.py
│   ├── db.sqlite3
│   └── manage.py
│
└── django-project
    └── ...
```

After creating an application, we also need to tell Django that it should use it. We do that in the file `django-project\mobis\mobis\settings.py` -- open it in your code editor.

Find `INSTALLED_APPS` and add a line containing **`'bookshelf.apps.BookshelfConfig',`** just above **`]`**. So the final product should look like this:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'bookshelf.apps.BookshelfConfig',
]
```

6. Creating a Model & Database

A model in Django is a special kind of object – it is saved in the database.

We will be using a SQLite database to store our data. This is the default Django database adapter

You can think of a model in the database as a spreadsheet with columns (fields) and rows (data).

A. Open `bookshelf\models.py` and paste the following code:

```
from django.db import models
# Create your models here.
class BookCategory(models.Model):
    category = models.CharField(unique=True, max_length=50)

    def __str__(self):
        return self.category
```

B. To create a database with the model we have created, run the following code in the command prompt in `\django-project\mobis`:
`python manage.py makemigrations bookshelf`
`python manage.py migrate`

```
(django-project) C:\Users\ravi\django-project\mobis>python manage.py makemigrations bookshelf
Migrations for 'bookshelf':
  bookshelf\migrations\0001_initial.py
    - Create model BookCategory
```

```
(django-project) C:\Users\ravi\django-project\mobis>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, bookshelf, contenttypes, sessions
Running migrations:
  Applying contenttypes.0001_initial... OK
  Applying auth.0001_initial... OK
  Applying admin.0001_initial... OK
  Applying admin.0002_logentry_remove_auto_add... OK
  Applying admin.0003_logentry_add_action_flag_choices... OK
  Applying contenttypes.0002_remove_content_type_name... OK
  Applying auth.0002_alter_permission_name_max_length... OK
  Applying auth.0003_alter_user_email_max_length... OK
  Applying auth.0004_alter_user_username_opts... OK
  Applying auth.0005_alter_user_last_login_null... OK
  Applying auth.0006_require_contenttypes_0002... OK
  Applying auth.0007_alter_validators_add_error_messages... OK
  Applying auth.0008_alter_user_username_max_length... OK
  Applying auth.0009_alter_user_last_name_max_length... OK
  Applying auth.0010_alter_group_name_max_length... OK
  Applying auth.0011_update_proxy_permissions... OK
  Applying bookshelf.0001_initial... OK
  Applying sessions.0001_initial... OK

(django-project) C:\Users\ravi\django-project\mobis>_
```

C. Let's add another model

Replace the `bookshelf\models.py` with the following code:

```python
from django.db import models

# Create your models here.
class BookCategory(models.Model):
    category = models.CharField(unique=True, max_length=50)

    def __str__(self):
        return self.category

class BookAuthor(models.Model):
    name = models.CharField(unique=True, max_length=100)

    def __str__(self):
        return self.name
```

D. To add this model to the database, **save the file** and make migrations again

`python manage.py makemigrations bookshelf`
`python manage.py migrate`

```
(django-project) C:\Users\ravi\django-project\mobis>python manage.py makemigrations bookshelf
Migrations for 'bookshelf':
  bookshelf\migrations\0002_bookauthor.py
    - Create model BookAuthor

(django-project) C:\Users\ravi\django-project\mobis>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, bookshelf, contenttypes, sessions
Running migrations:
  Applying bookshelf.0002_bookauthor... OK

(django-project) C:\Users\ravi\django-project\mobis>_
```

E. Add the final model `Book` and final `bookshelf\models.py` should like this:

```
from django.db import models

# Create your models here.
class BookCategory(models.Model):
    category = models.CharField(unique=True, max_length=50)

    def __str__(self):
        return self.category

class BookAuthor(models.Model):
    name = models.CharField(unique=True, max_length=100)

    def __str__(self):
        return self.name

class Book(models.Model):
    title = models.CharField(max_length=100)
    auth_name = models.ForeignKey(BookAuthor,
on_delete=models.PROTECT, to_field='name')
    book_cat = models.ForeignKey(BookCategory,
on_delete=models.PROTECT, to_field='category')

    def __str__(self):
        return self.title
```

F. Save the file and make the migrations again:

`python manage.py makemigrations bookshelf`
`python manage.py migrate`

```
(django-project) C:\Users\ravi\django-project\mobis>python manage.py makemigrations
Migrations for 'bookshelf':
  bookshelf\migrations\0003_book.py
    - Create model Book

(django-project) C:\Users\ravi\django-project\mobis>python manage.py migrate
Operations to perform:
  Apply all migrations: admin, auth, bookshelf, contenttypes, sessions
Running migrations:
  Applying bookshelf.0003_book... OK

(django-project) C:\Users\ravi\django-project\mobis>_
```

7. Django Views

A *view* is a place where we put the "logic" of our application. It will request information from the `model` you created before and pass it to a `template.`

Views are placed in the `views.py` file. We will add our views to the `bookshelf\views.py` file.

Copy the following code and replace it with the one in `bookshelf\views.py` file:

```
from django.shortcuts import render
from bookshelf.models import Book, BookAuthor, BookCategory


def home(request):
    return render(request, 'bookshelf/home.html')
```

8. Django URLs

You can find all your **project URLs** in `mobis\urls.py`.

Add a line that will import URLs for our app bookshelf `bookshelf.urls` in `mobis\urls.py`.

A. Replace the existing code with the following code:

```
from django.contrib import admin
from django.urls import path, include

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include("bookshelf.urls"))
]
```

B. Save the file and close it.

C. Create an empty file in `bookshelf\urls.py` and paste the following code:

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.home, name='home'),
]
```

This is the **app URLs** file

D. Save the file

9. Django Templates

A template is a file that we can re-use to present different information in a consistent format.

First create a directory called `templates` inside your `bookshelf` directory.

Then create another directory called `bookshelf` inside your `templates` directory to save our templates:

```
bookshelf
└──templates
    └──bookshelf
```

A. Create a file `home.html` in `bookshelf\templates\bookshelf` and paste the following code:

```
<h3>Welcome to Book Shelf</h3>
```

B. Save the file.
C. Go to the command prompt where the server is running and restart the server.

```
python manage.py runserver
```

D. Reload the browser to verify the changes in application.

10. Extending our Application

Let's add more views to add more functionality to our application

A. Replace the `bookshelf\views.py` with the following code:

```
from django.shortcuts import render
from bookshelf.models import Book, BookAuthor, BookCategory

# Create your views here.
def home(request):
    return render(request, 'bookshelf/home.html')

def author_list(request):
    auth = BookAuthor.objects.all()
    return render(request, 'bookshelf/auth_list.html',
{'auth':auth})

def category_list(request):
    cat = BookCategory.objects.all()
    return render(request, 'bookshelf/cat_list.html', {'cat':cat})

def book_list(request):
    books = Book.objects.all()
    return render(request, 'bookshelf/book_list.html',
{'books':books})
```

B. Add the corresponding URLs to `bookshelf\urls.py` file. Replace the file with the following code:

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.home, name='home'),
    path("author/", views.author_list, name="author_list"),
    path("category/", views.category_list, name="category_list"),
    path("books/", views.book_list, name="book_list"),
]
```

C. Add corresponding templates in `bookshelf\templates\bookshelf\` folder:
    a. Create file `bookshelf\templates\bookshelf\auth_list.html` and save it:

```
<h2>Authors</h2>
<p><a href='/'>Home</a></p>
{% for author in auth %}
    <p>{{ author.name }}</p>
{% endfor %}
```

b. Create another file `bookshelf\templates\bookshelf\cat_list.html` and save it:

```
<h2>Book Categories</h2>
<p><a href='/'>Home</a></p>
{% for category in cat %}
    <p>{{ category.category }}</p>
{% endfor %}
```

c. Create another file `bookshelf\templates\bookshelf\book_list.html` and save it:

```
<h2>Books</h2>
<p><a href='/'>Home</a></p>
<table>
    <tr>
        <th>Book Title</th>
        <th>Author</th>
        <th>Category</th>
    </tr>
    {% for book in books %}
    <tr>
        <td>{{ book.title }}</td>
        <td>{{ book.auth_name }}</td>
        <td>{{ book.book_cat }}</td>
    </tr>
    {% endfor %}
</table>
```

d. Also, modify the `bookshelf\templates\bookshelf\home.html` file and save it:

```
<h3>Welcome to Book Shelf</h3>

<a href='author/'>Author List</a><br />
<a href='category/'>Category List</a><br />
<a href='books/'>Book List</a><br />
```

11. Django Forms

Django provides a Form class which is used to create HTML forms. It describes a form and how it works and appears.

Forms have their own file: `forms.py`.

A. Create a file `forms.py` in the `bookshelf` directory:

```
bookshelf
    └── forms.py
```

B. Add the following code into the newly created `bookshelf\forms.py` file:

```
from django import forms
from .models import BookCategory, BookAuthor, Book

class CategoryForm(forms.ModelForm):

    class Meta:
        model = BookCategory
        fields = ('category',)

class AuthorForm(forms.ModelForm):

    class Meta:
        model = BookAuthor
        fields = ('name',)

class BookForm(forms.ModelForm):

    class Meta:
        model = Book
        labels = {
            'title': 'Book Title',
            'auth_name': 'Name of the Author',
            'book_cat': 'Category of the Book'
        }
        fields = ('title', 'auth_name', 'book_cat')
```

C. Save the file

Add more Features to the Application

Let's add more views to update our database.

A. Replace `bookshelf\views.py` with the following code and save it:

```
from django.shortcuts import render, redirect
from bookshelf.models import Book, BookAuthor, BookCategory
from .forms import CategoryForm, AuthorForm, BookForm

# Create your views here.
def home(request):
    return render(request, 'bookshelf/home.html')

def author_list(request):
    auth = BookAuthor.objects.all()
    return render(request, 'bookshelf/auth_list.html',
{'auth':auth})

def category_list(request):
    cat = BookCategory.objects.all()
    return render(request, 'bookshelf/cat_list.html', {'cat':cat})

def book_list(request):
    books = Book.objects.all()
    return render(request, 'bookshelf/book_list.html',
{'books':books})

def add_category(request):
    if request.method=='POST':
        form=CategoryForm(request.POST)
        if form.is_valid():
            cat = form.save()
            cat.save()
        return redirect('category_list')
    else:
        form=CategoryForm()
    return render(request, 'bookshelf/cat_add.html', {'form':form})

def add_author(request):
    if request.method=='POST':
        form=AuthorForm(request.POST)
        if form.is_valid():
            author = form.save()
            author.save()
        return redirect('author_list')
    else:
        form=AuthorForm()
    return render(request, 'bookshelf/auth_add.html', {'form':form})
```

```
def add_book(request):
    if request.method=='POST':
        form=BookForm(request.POST)
        if form.is_valid():
            title = form.save()
            title.save()
        return redirect('book_list')
    else:
        form=BookForm()
    return render(request, 'bookshelf/book_add.html', {'form':form})
```

B. Add corresponding URLs in the `bookshelf\urls.py` file. Replace the file with the following code and save it:

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.home, name='home'),
    path("author/", views.author_list, name="author_list"),
    path("category/", views.category_list, name="category_list"),
    path("books/", views.book_list, name="book_list"),
    path("add_category/", views.add_category, name='add_category'),
    path("add_author/", views.add_author, name='add_author'),
    path("add_book/", views.add_book, name='add_book'),
]
```

C. Add corresponding templates in `bookshelf\templates\bookshelf\` folder:
    a. Create file `bookshelf\templates\bookshelf\cat_add.html` and save it:

```
<h2>New Category</h2>
    <form method="POST">{%  csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Add
Category</button>
    </form>
```

b. Create file `bookshelf\templates\bookshelf\auth_add.html` and save it:

```
<h2>New Author</h2>
    <form method="POST">{%  csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Add
Author</button>
    </form>
```

c. Create another file `bookshelf\templates\bookshelf\book_add.html` and save it:

```
<h2>New Book</h2>
    <form method="POST">{%  csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn btn-default">Add
Book</button>
    </form>
```

d. Finally modify the file `bookshelf\templates\bookshelf\home.html` and save it:

```
<h3>Welcome to Book Shelf</h3>

<a href='author/'>Author List</a><br />
<a href='category/'>Category List</a><br />
<a href='books/'>Book List</a><br />
<a href='add_category/'>Add Category</a><br />
<a href='add_author/'>Add Author</a><br />
<a href='add_book/'>Add Book</a><br />
```

D. Restart your server and reload the browser to populate the values

Add Edit Feature to the Books

Let's create a view that enables user to update the `Book` model.

A. Replace `bookshelf\views.py` with:

```
from django.shortcuts import render, redirect, get_object_or_404
from bookshelf.models import Book, BookAuthor, BookCategory
from .forms import CategoryForm, AuthorForm, BookForm

# Create your views here.
def home(request):
    return render(request, 'bookshelf/home.html')

def author_list(request):
    auth = BookAuthor.objects.all()
    return render(request, 'bookshelf/auth_list.html', {'auth':auth})

def category_list(request):
    cat = BookCategory.objects.all()
    return render(request, 'bookshelf/cat_list.html', {'cat':cat})

def book_list(request):
    books = Book.objects.all()
    return render(request, 'bookshelf/book_list.html', {'books':books})

def add_category(request):
    if request.method=='POST':
        form=CategoryForm(request.POST)
        if form.is_valid():
            cat = form.save()
            cat.save()
        return redirect('category_list')
    else:
        form=CategoryForm()
    return render(request, 'bookshelf/cat_add.html', {'form':form})

def add_author(request):
    if request.method=='POST':
        form=AuthorForm(request.POST)
        if form.is_valid():
            author = form.save()
            author.save()
        return redirect('author_list')
    else:
        form=AuthorForm()
    return render(request, 'bookshelf/auth_add.html', {'form':form})
```

```
def add_book(request):
    if request.method=='POST':
        form=BookForm(request.POST)
        if form.is_valid():
            title = form.save()
            title.save()
        return redirect('book_list')
    else:
        form=BookForm()
    return render(request, 'bookshelf/book_add.html', {'form':form})

def edit_book(request, pk):
    book = get_object_or_404(Book, pk=pk)
    if request.method=='POST':
        form=BookForm(request.POST, instance=book)
        if form.is_valid():
            book = form.save()
            book.save()
            return redirect('book_list')
    else:
        form=BookForm(instance=book)
    return render(request, 'bookshelf/book_edit.html', {'form':form})
```

B. Save the file

C. Add it's corresponding URL in `bookshelf\urls.py` file. Replace the file with:

```
from django.urls import path
from . import views

urlpatterns = [
    path("", views.home, name='home'),
    path("author/", views.author_list, name="author_list"),
    path("category/", views.category_list, name="category_list"),
    path("books/", views.book_list, name="book_list"),
    path("add_category/", views.add_category, name='add_category'),
    path("add_author/", views.add_author, name='add_author'),
    path("add_book/", views.add_book, name='add_book'),
    path("books/edit_book/<int:pk>", views.edit_book, name='edit_book'),
]
```

And save the file.

D. And it's associated template is `bookshelf\templates\bookshelf\book_edit.html`. Create a new file and copy the following code into it:

```
<h2>Edit Book</h2>
    <form method="POST" >{%  csrf_token %}
        {{ form.as_p }}
        <button type="submit" class="save btn
btn-default">Update</button>
    </form>
```

And save it.

E. Modify the `bookshelf\templates\bookshelf\book_list.html`:

```
<h2>Books</h2>
<p><a href='/'>Home</a></p>
<table>
    <tr>
        <th>Book Title</th>
        <th>Book Author</th>
        <th>Book Category</th>
    </tr>
    {% for book in books %}
    <tr>
        <td>{{ book.title }}</td>
        <td>{{ book.auth_name }}</td>
        <td>{{ book.book_cat }}</td>
        <td><a href='edit_book/{{ book.pk }}'>Edit</a></td>
    </tr>
    {% endfor %}
</table>
```

F. Reload the browser