Week 2, Backend for high load

Azamat Serek, PhD, Assist.Prof.

Backend

Many applications have a front end and a back end. The front end is the user interface—the part that users interact with. The back end runs behind the scenes on a server and is responsible for overall application functionality.

Whenever you submit form data, upload files, access databases, or run other complex operations, these commands are executed on the back end—and results are posted back to the front end.

So, you can think of back-end development as the process of building the heart of the application. During development, back-end engineers use different frameworks, open-source programming languages, libraries, and APIs to get their work done.

Front-End Development

upwork

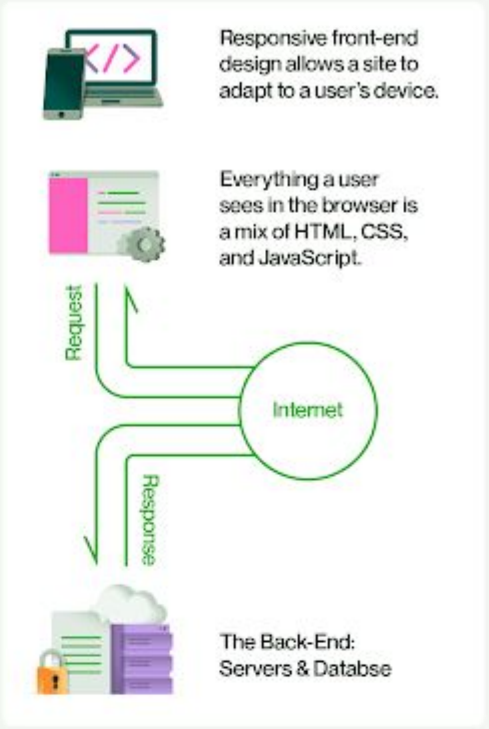
1 A site is loaded in a browser from the server.

2 Client-side scripts run in the browser and process requests without call-backs to the server.

3 When a call to the database is required JavaScript and AJAX send requests to the back end.

4 The back-end server-side scripts process the request, pull what they need from the database then send it back.

5 Server-side scripts process the data, then update the site— populating drop-down menus, loading products to a page, updating a user profile, and more.



Responsive front-end design allows a site to adapt to a user's device.

Everything a user sees in the browser is a mix of HTML, CSS, and JavaScript.

The Back-End: Servers & Databse

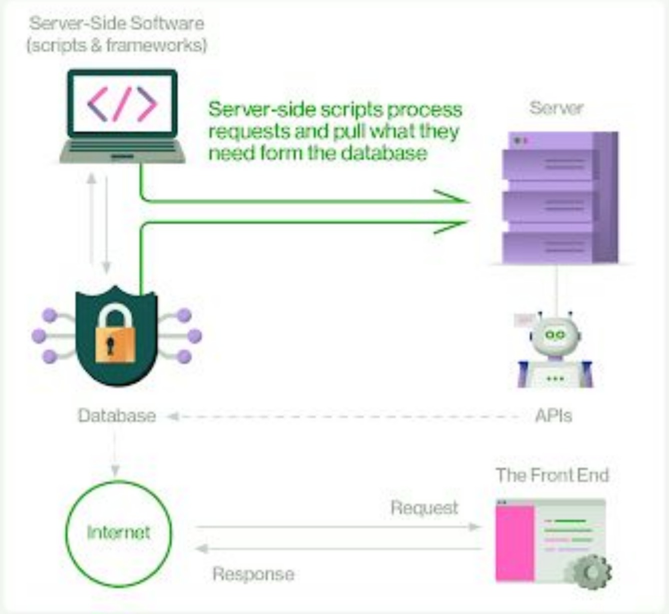
Back-End Development & Frameworks in Server-Side Software

upwork

Frameworks are libraries of server-side programming languages that construct the back-end structure of a site

The "stack" comprises the database, server-side framework, server, and operating system (OS).

APIs structure how data is exchanged between a database and any software accessing it.


Server-Side Software (scripts & frameworks)
Server-side scripts process requests and pull what they need form the database
Server
Database
APIs
The Front End
Internet
Request
Response

Languages and frameworks

As a back-end developer, you can choose from among many programming languages. Some of the most common are Python, Java, JavaScript, Ruby, SQL, and Golang.

To speed things up, you can also use frameworks like Node.js, Laravel, Express, Flask, Ruby on Rails, and Django to accelerate back-end development. These web frameworks come with built-in functions for handling things like database operations and file uploads, so you can focus on customizing the features for your specific project.

Servers and hosting

Servers are the hardware that store and retrieve data, process requests, and respond with the necessary information over a network. They’ve got RAM and storage drives for various computations and file storage.

When you deploy a website on a server, it gets assigned a specific URL and IP address. Users can then access your application using that URL. You can even host your back-end app and front-end web pages on separate servers and use an API to facilitate communication between them.

Databases

Databases are a crucial part of back-end development since they store all sorts of information, from employee records and user info to product details and multimedia files.

Depending on your project, you might use SQL or NoSQL databases. SQL relational databases like Oracle Database and PostgreSQL store data in tables and rows, making them ideal for structured data. They also follow ACID principles, which guarantee better data safety, reliability, and integrity.

On the other hand, NoSQL databases like MongoDB and Firebase are primarily used for storing unstructured or semi-structured data. This makes them adaptable to changing data needs.

API

APIs (application programming interfaces) allow the back end to communicate with the front end

For example, users can enter info on the front end that gets sent to the back end via an API for processing or storage. The back end can also send requested info to the front end to be displayed to users. This is how Netflix fetches a list of movies and TV shows from its back-end database to show on the user interface.

Software applications typically require a CRUD (Create, Read, Update, and Delete) API. This means the API enables users to create new records on the databases, fetch existing data in JSON or other supported formats, update specific entries, or delete certain or all records.

Middleware

At the center of the front end and underlying APIs is the middleware. Middleware (server-side software) facilitates client-server connectivity, forming a middle layer between the app(s) and the network, server, database, operating system, and more.

Middleware can be organized into different layers of a site, whether it’s the presentation or business layer. This is where web APIs can come into play, bridging the gap between the business and presentation layers.

Middleware also lets cloud and on-premise applications communicate and provides services like data integration and error handling. Good middleware can maximize IT efficiency and power things like user engagement, business process management, content management, authentication, and more. Koa.js is an example of a server-side JavaScript framework.

1.Creating the API

Create a virtual environment:

```
virtualenv env
```

```
source env/bin/activate
```

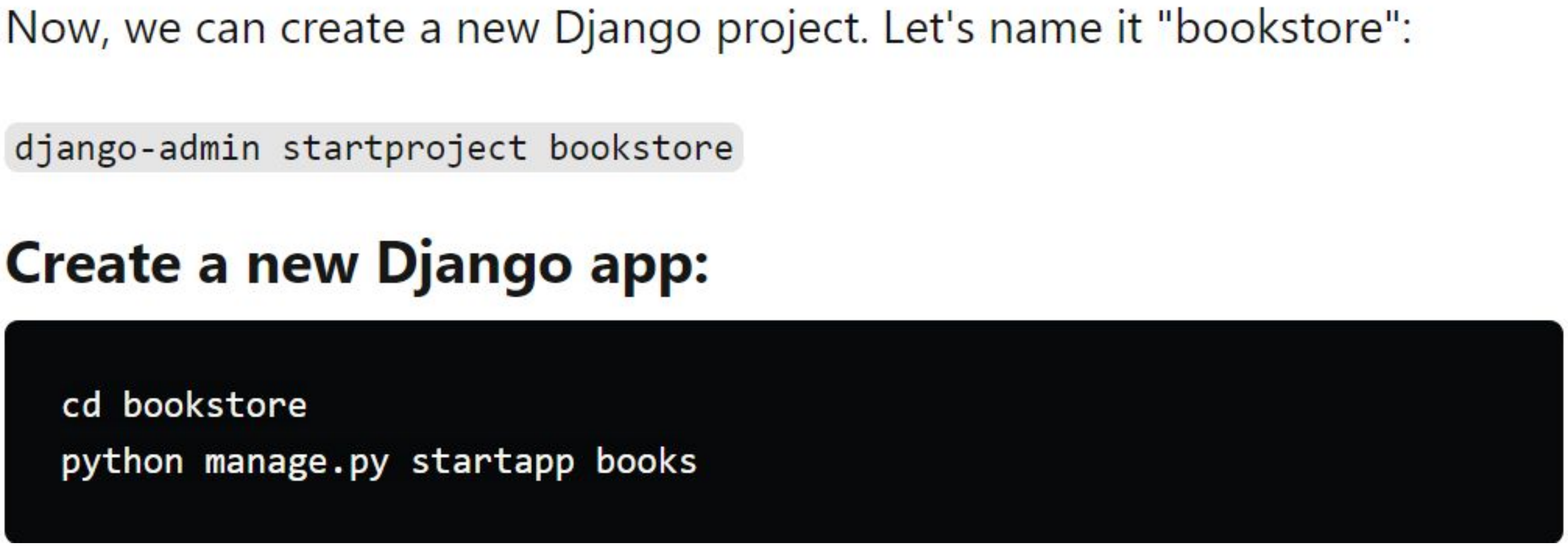
Set up a new Django project

Firstly, we need to install Django and Django REST framework. Make sure you have Python installed on your machine. You can install Django and Django REST framework using pip:

```
pip install django
```

```
pip install djangorestframework
```

Creating project



Now, we can create a new Django project. Let's name it "bookstore":

```
django-admin startproject bookstore
```

Create a new Django app:

```
cd bookstore
python manage.py startapp books
```

Define the Book model

In books/models.py, define a Book model:

```python
from django.db import models

class Book(models.Model):
    title = models.CharField(max_length=200)
    author = models.CharField(max_length=100)
    publication_date = models.DateField()
    isbn = models.CharField(max_length=20)
    summary = models.TextField()
```

Don't forget to add 'books' to installed apps in bookstore/settings.py:

```
INSTALLED_APPS = [
    ...
    'rest_framework',
    'books',
]
```

Run migrations to create the corresponding database table:

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

Create a serializer for the Book model

In books/serializers.py, define a BookSerializer:

```python
from rest_framework import serializers
from .models import Book

class BookSerializer(serializers.ModelSerializer):
    class Meta:
        model = Book
        fields = '__all__'
```

Set up a view for handling REST API requests

In books/views.py, define a BookViewSet:

```
from rest_framework import viewsets
from .models import Book
from .serializers import BookSerializer

class BookViewSet(viewsets.ModelViewSet):
    queryset = Book.objects.all()
    serializer_class = BookSerializer
```

Configure the URL routing

In bookstore/urls.py, add a route for the books application:

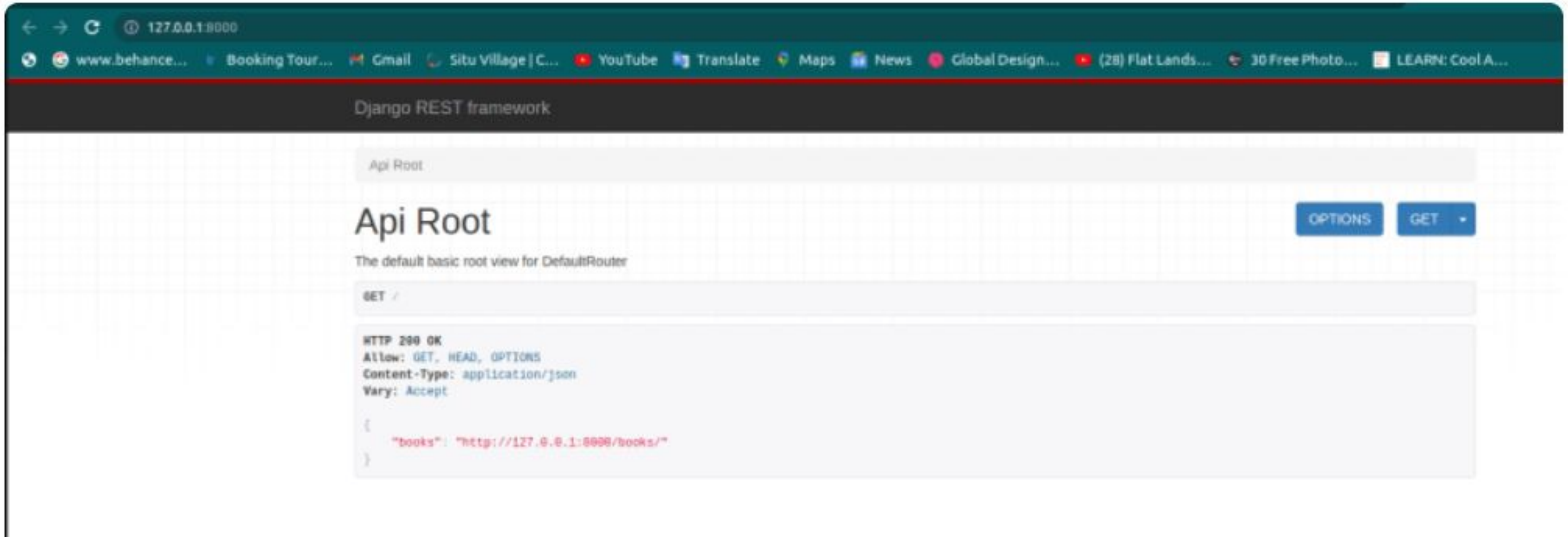
```python
from django.urls import include, path
from rest_framework.routers import DefaultRouter
from books.views import BookViewSet

router = DefaultRouter()
router.register(r'books', BookViewSet)

urlpatterns = [
    path('', include(router.urls)),
]
```

Test the API locally

Start the Django development server:

```
python manage.py runserver
```

You should now be able to access the API at localhost as shown:



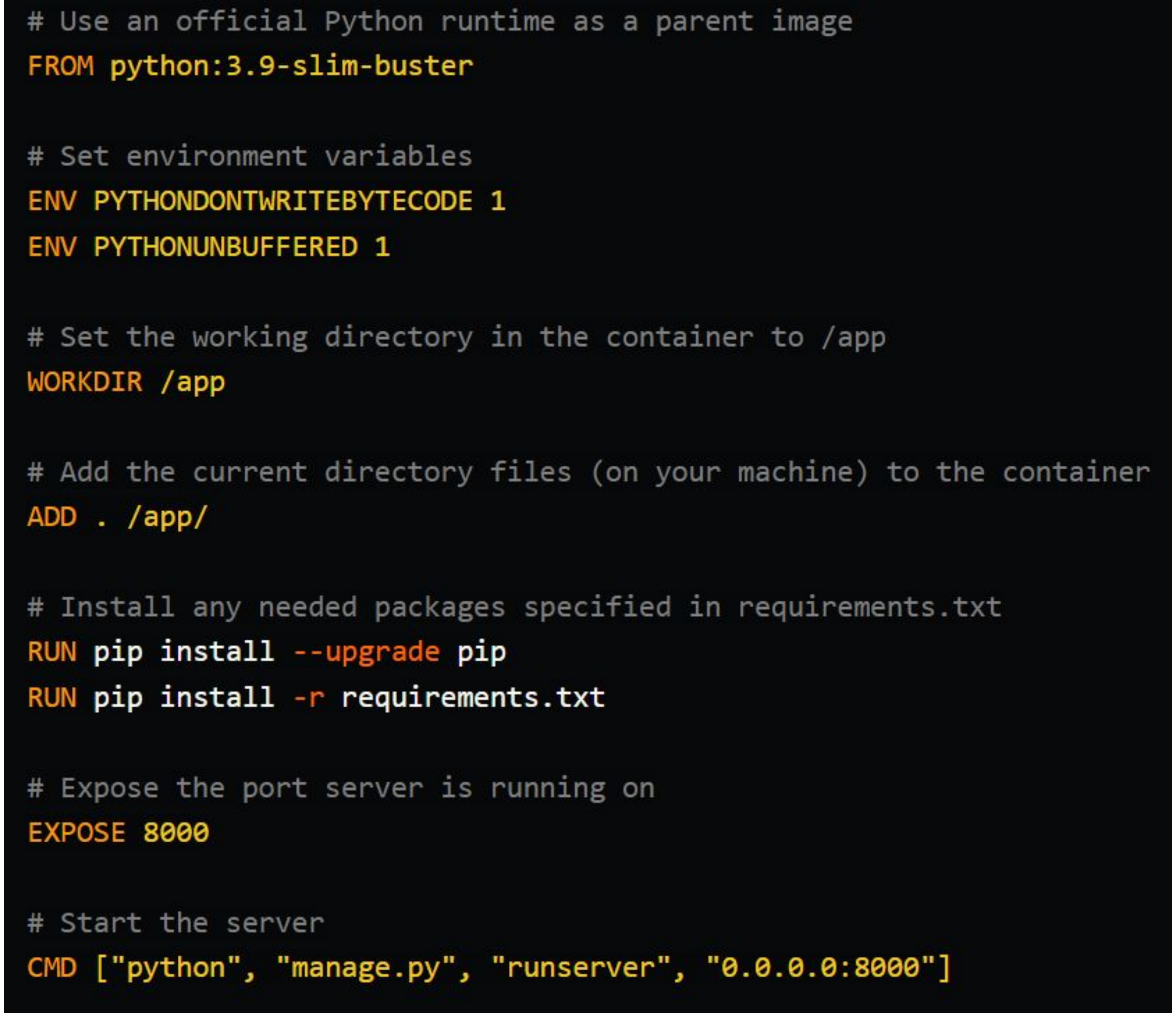
Api Root

The default basic root view for DefaultRouter

```
GET /
```

```
HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

{
    "books": "http://127.0.0.1:8000/books/"
}
```

Dockerfile



```
# Use an official Python runtime as a parent image
FROM python:3.9-slim-buster

# Set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# Set the working directory in the container to /app
WORKDIR /app

# Add the current directory files (on your machine) to the container
ADD . /app/

# Install any needed packages specified in requirements.txt
RUN pip install --upgrade pip
RUN pip install -r requirements.txt

# Expose the port server is running on
EXPOSE 8000

# Start the server
CMD ["python", "manage.py", "runserver", "0.0.0.0:8000"]
```

Create a requirements.txt file

You also need to create a requirements.txt file that lists the Python dependencies of your project so that Docker can install them. You can generate a requirements.txt using :

```
pip freeze > requirements.txt
```

Build the Docker image

Now you can build the Docker image from the Dockerfile. Make sure Docker Desktop is running, and then run the following command in the terminal from the directory that contains the Dockerfile:

```
docker build -t bookstore .
```

You can use this command to look at your new Django Image:

```
docker images
```

References

1) https://www.upwork.com/resources/beginners-guide-back-end-development