Django REST Framework

django
REST
framework

SoftUni Team

Technical Trainers

Software University

SoftUni

Software University

https://softuni.org

Table of Contents

Have a Question?

sli.do

#python-web

Django
REST

Django REST Framework

Definition and Usage

What is Django REST?

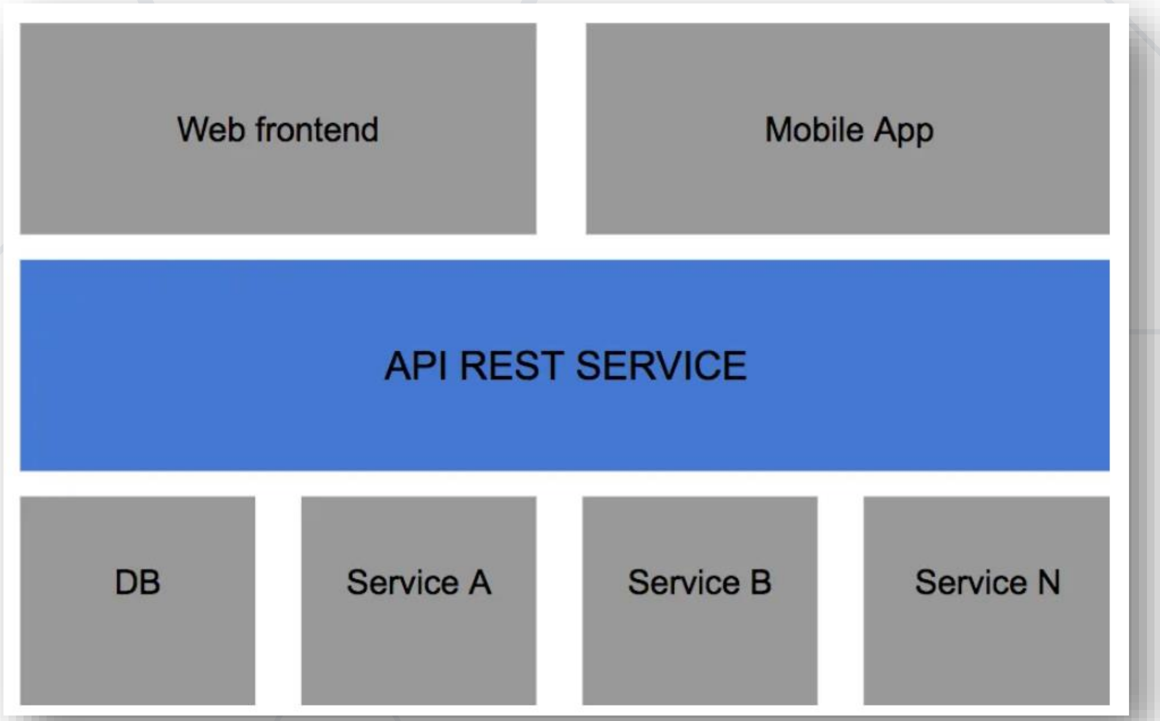
- Django REST framework is a **powerful** and **flexible** toolkit for building Web **APIs**
- API
  - is **A**pplication **P**rogramming **I**nterface
  - defines how other **components**/**systems** can use it
  - defines the kinds of **calls** or **requests** that can be made
  - can provide **extension** mechanisms for **extending** existing **functionalities**

Why use Django REST?

- The reasons you might want to use Django REST are
  - The Web browsable API is a huge **usability win**
  - **Authentication policies** including packages for **OAuth1a** and **OAuth2**
  - **Serialization** that supports both **ORM** and **non-ORM** data sources
  - **Used** and **trusted** by many **companies** including Mozilla, Red Hat, Heroku and Eventbrite

Django Rest and RESTful APIs

- Django Rest Framework lets you create **RESTful APIs** - a way to transfer information between an **interface** and a **database** in a simple way



RESTful Structure

- In a RESTful API, **endpoints** (URLs) define the **structure** of the API and how users **access** using the **HTTP** methods
  - GET, POST, PUT, DELETE

Endpoint	GET	POST	PUT	DELETE
/books/	Show all books	Add new book	Update all books	Delete all books
/books/<id>	Show <id>	N/A	Update <id>	Delete <id>

Requirements and Installation

Requirements

- To use Django REST Framework, we need
  - **Python** (3.6, 3.7, 3.8, 3.9, 3.10)
  - **Django** (2.2, 3.0, 3.1, 3.2, 4.0)
- Usage of the **officially supported** and **latest versions** of Python and Django are **highly recommended**
- Optional: coreapi, Markdown, Pygments, django-filter, django-guardian

Installation and Setup

- To install Django REST, we use the **pip** command

```
pip install djangorestdramework
```

- Next, we need to add it in our **INSTALLED_APPS** setting and include the **rest_framework.urls**

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

```
urlpatterns = [
    ...
    path('api/', include('rest_framework.urls'))
]
```

Creating Simple RESTful API

Books API

Creating a Model

- After **installing** the Django REST Framework and **setting it up**, we will create our Book model

```
from django.db import models

# Create your models here.
class Book(models.Model):
    title = models.CharField(max_length=20)
    pages = models.IntegerField(default=0)
    description = models.TextField(max_length=100, default="")
    author = models.CharField(max_length=20)
```

Create a Serializer

- Serializers allow **complex data** to be converted to native Python datatypes that can then be easily **rendered** into **JSON**, **XML**, etc.
- Serializers also provide **deserialization**, allowing parsed data to be converted back into complex types

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

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

Create the ListBooksView

- The ListBooksView will handle **GET** and **POST** requests on **`localhost:8000/api/books`**

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .models import Book
from .serializers import BookSerializer

class ListBooksView(APIView):
    def get(self, req):
        books = Book.objects.all()
        serializer = BookSerializer(books, many=True)
        return Response({"books": serializer.data})
```

Create the URL

- Now, we need to add the URL pattern

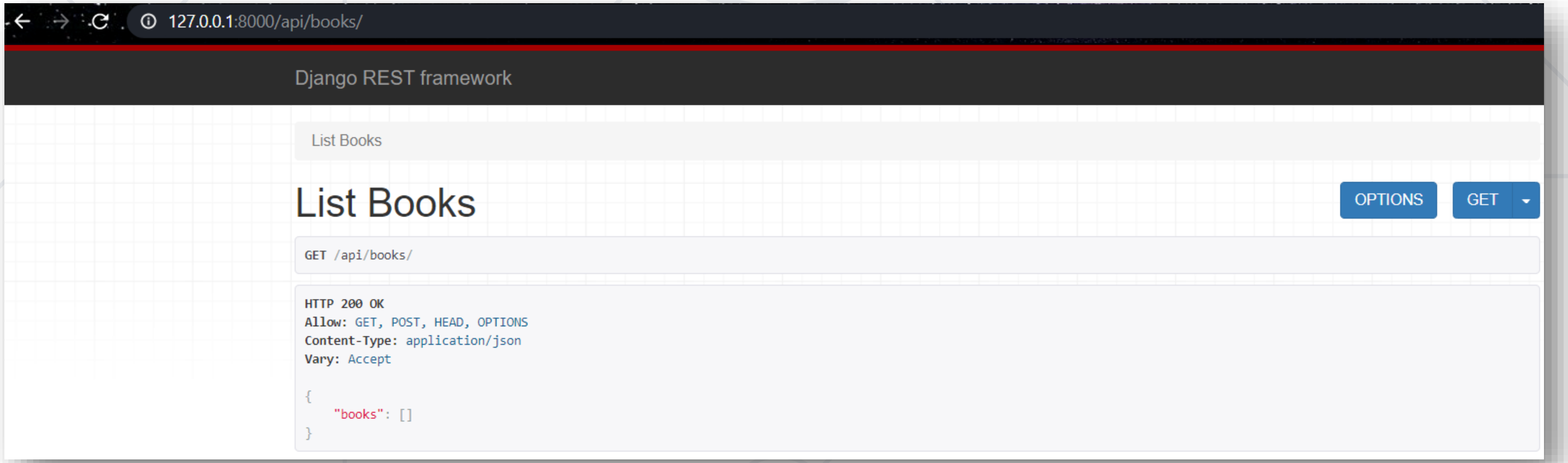
```
from django.urls import path
from books.views import ListBooksView

urlpatterns = [
    path('books/', ListBooksView.as_view(), name="books-all"),
]
```

Testing the API

- To run the API, we use the command for the standard Django project

```
python manage.py runserver
```



List Books

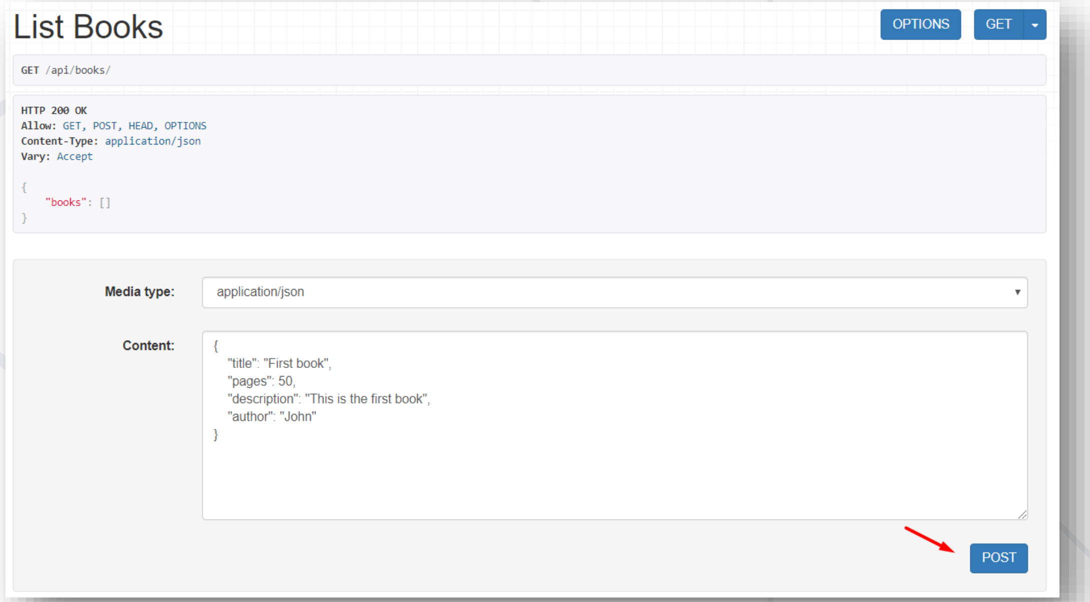
```
GET /api/books/
```

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

{
    "books": []
}
```

Implement POST in the View

```
class ListBooksView(APIView):
    ...
    def post(self, req):
        serializer = BookSerializer(data=req.data)
        if serializer.is_valid():
            serializer.save()
            return Response(serializer.data,
                      status=status.HTTP_201_CREATED)
        return Response(serializer.errors,
                  status=status.HTTP_400_BAD_REQUEST)
```



```
GET /api/books/
```

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

{
    "books": []
}
```

Media type: application/json

Content:

```
{
    "title": "First book",
    "pages": 50,
    "description": "This is the first book",
    "author": "John"
}
```

Create DetailBookView

- The View will handle **GET**, **POST** and **DELETE** methods

```
class DetailBookView(APIView):
    def get(self, req, id):
        book = Book.objects.get(pk=id)
        serializer = BookSerializer(book)
        return Response({"book": serializer.data})

    def post(self, req, id):
        book = Book.objects.get(pk=id)
        serializer = BookSerializer(book, data=req.data)
        if serializer.is_valid():
            serializer.save()
        # TODO: Return response
    # TODO: Implement the DELETE
```

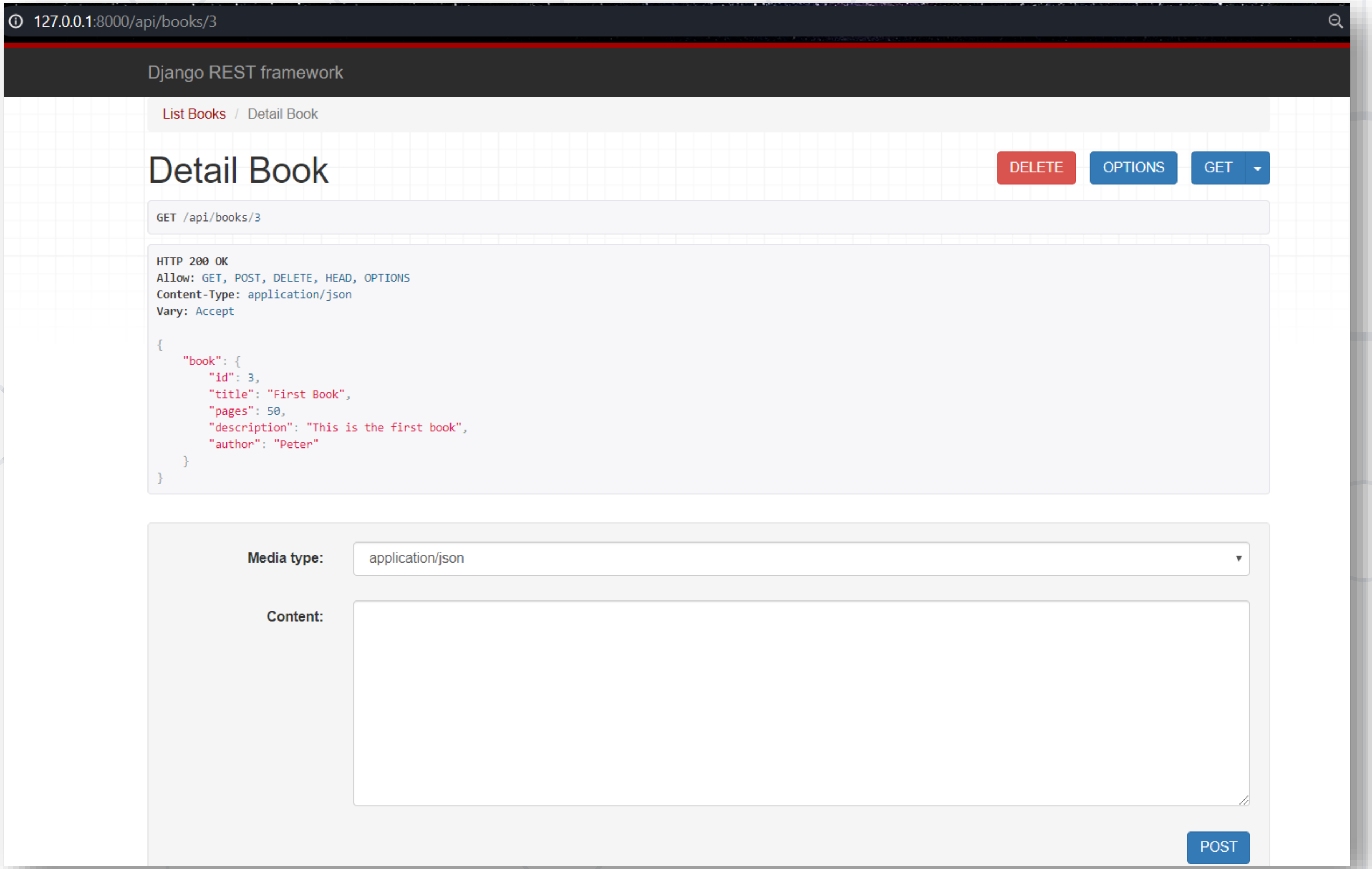
Create the URL

- On "./books/{id}" we will be able to Update and Delete a book

```
from django.urls import path
from books.views import ListBooksView, DetailBookView

urlpatterns = [
    path('books/', ListBooksView.as_view(), name="books-all"),
    path('books/<int:id>', DetailBookView.as_view(), name="books-detail")
]
```

Create the URL (2)



Detail Book

```
GET /api/books/3
```

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

{
    "book": {
        "id": 3,
        "title": "First Book",
        "pages": 50,
        "description": "This is the first book",
        "author": "Peter"
    }
}
```

Live Demo

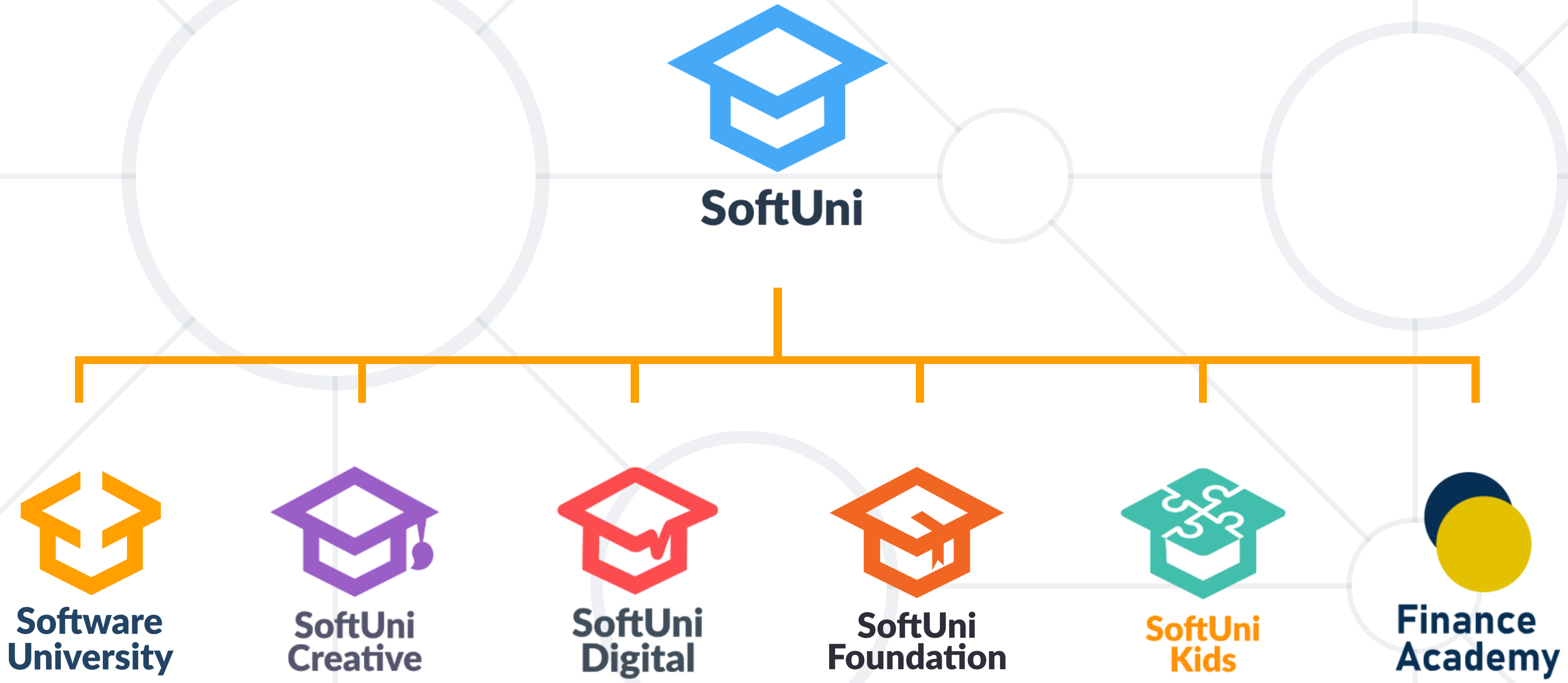
Exercise in Class (Lab)

Summary

- The Django REST framework is a **powerful** and **flexible** toolkit for building Web **APIs**
- The Django REST framework is **used** and **trusted** by many **companies**
  - Mozilla
  - Red Hat
  - Heroku
  - Eventbrite

Questions?

Software University



SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

POKERSTARS
POKER | CASINO | SPORTS
a Flutter International brand

INDEAVR
Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

createX

Postbank
Решения за твоето утре

BOSCH

DXC TECHNOLOGY

SmartIT

Educational Partners

CODE IT UP

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, softuni.org
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License