Python Basic Web Project: Phonebook

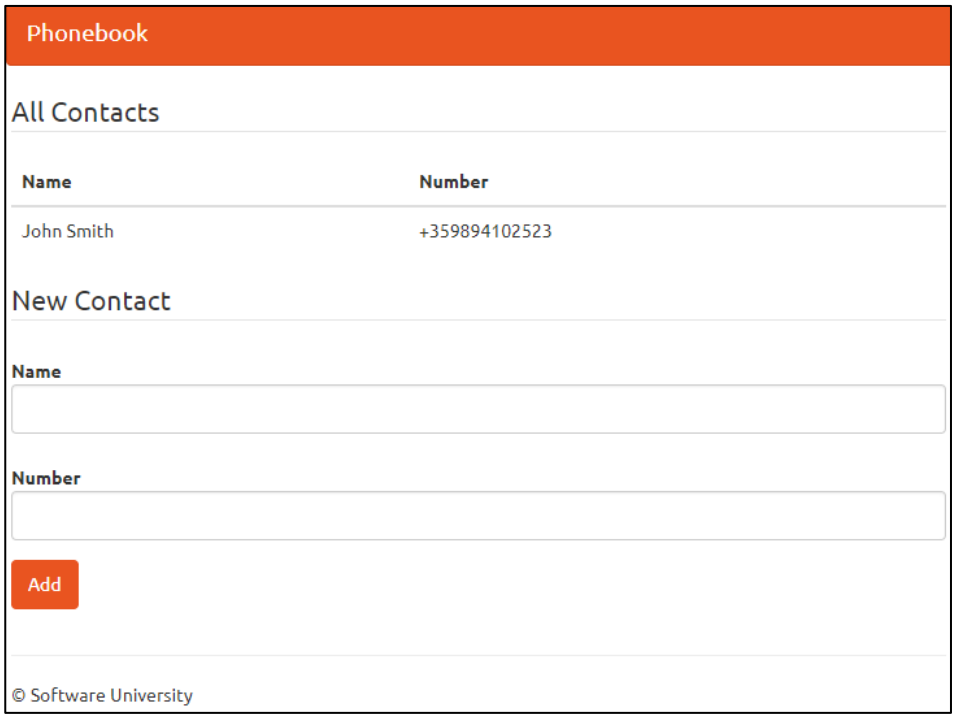
Problems for exercises for the "Programming Fundamentals" course @ SoftUni

1. Problem

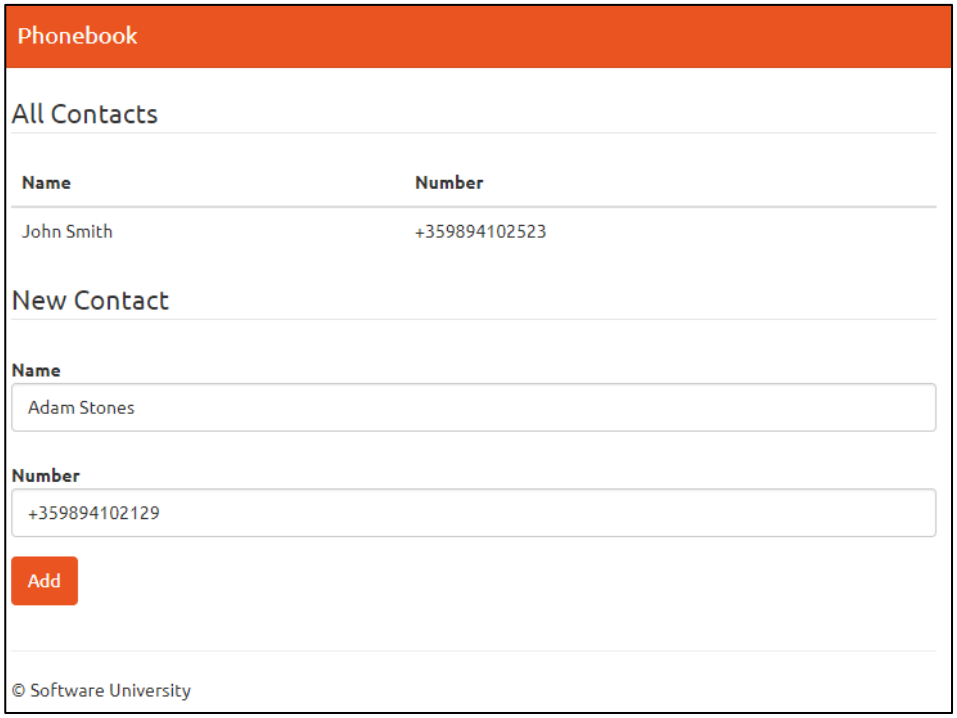
You have been tasked to create a simple **Phonebook** application. The application should hold **contacts**, which are the main app **model**.

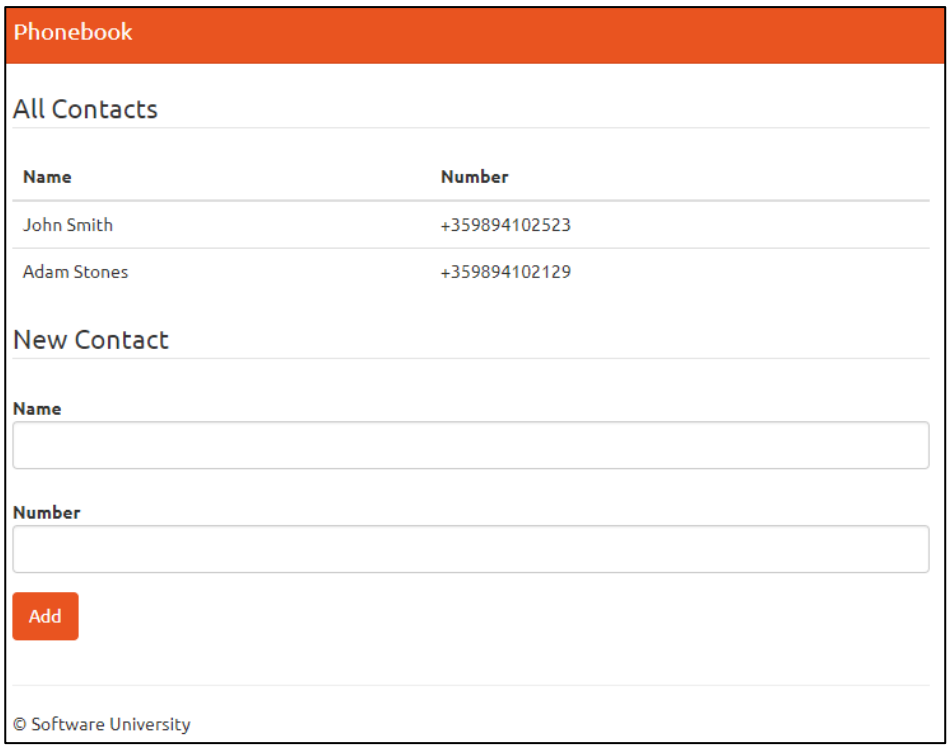
The functionality of the application should support:

- **Listing contacts**



- **Adding Contacts**





2. Overview

Requirements

- **Django** framework
- **SQLite** database

Data Model

The **`Contact`** model holds **2 properties**:

- **`name`** - Character field with a **max length** of **30 characters**
- **`number`** - Integer field

Project Skeletons

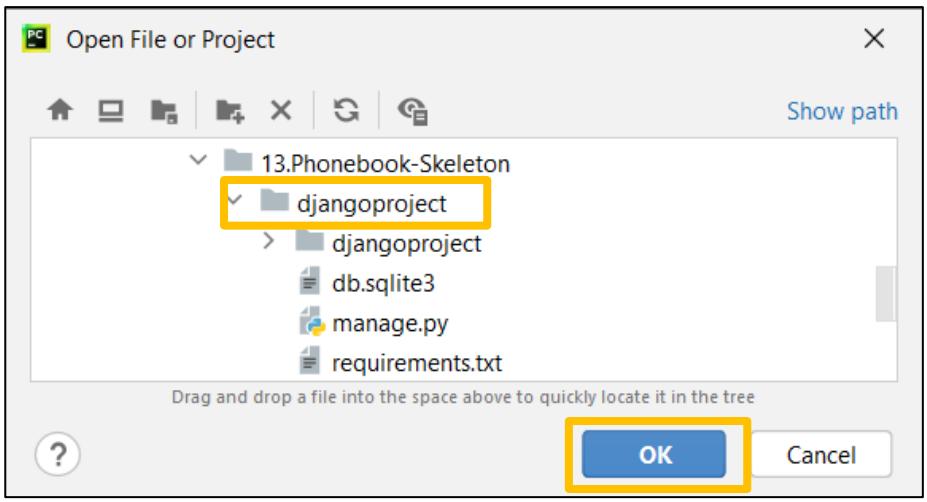
You will be given the applications' **`skeletons`**, which hold **most** of the logic. You'll need to write some code for the application to **function properly**.

The application's templates will be given to you fully implemented. You only need to include them in your business logic.

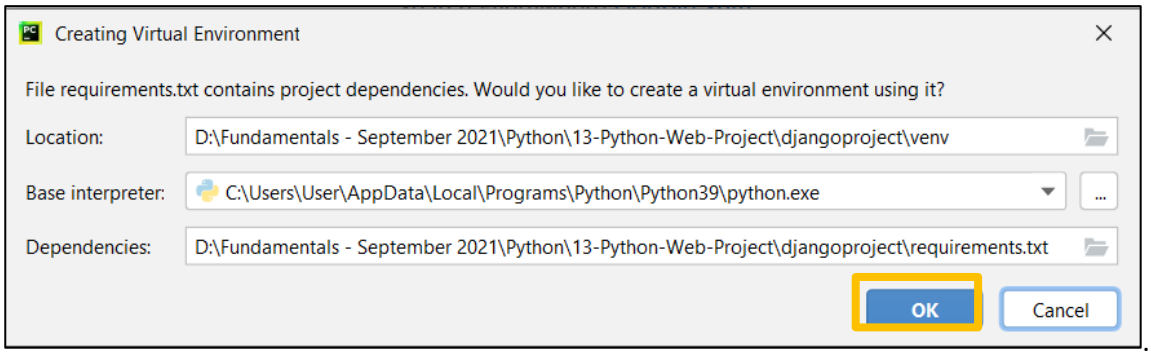
Everything that has been given to you inside the skeleton is **correctly implemented**, and if you write your code **correctly**, the application should work just fine. You are free to change anything in the skeleton on your account.

3. Setting Up PyCharm Configuration

Start **PyCharm** and **import** the skeleton. From the **File** menu click **Open**, **choose** the directory you've downloaded your skeleton and **click OK.**



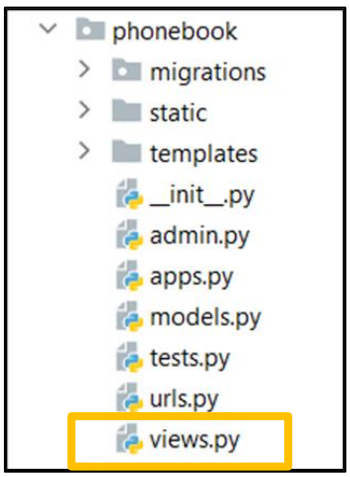
In the project skeleton, you have a **`requirements.txt`** file. It is used for specifying **what python packages** are **required** to run the project you are looking at. PyCharm will ask you to **create a virtual environment** and **install** the packages using the **`requirements.txt`** file immediately after opening the project. Click **OK**:



You can start working on your code!

4. Landing Page View

In the **phonebook app**, you can see the files that **define our app**:



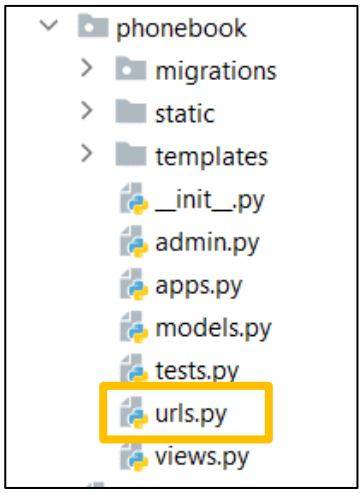
Open the **`views.py`** file and **create a view** for our **landing page**. A view is a Python **function** that takes a **Web request** and returns a **Web response**. In this case, our view function will return a **Web response** - the **HTML contents of our Web page**. For the example, we will name the function **`landing_page`**:

```python
from django.shortcuts import render, redirect


def landing_page(request):
    return render(request, 'phonebook/index.html')

```

Next, let's design our **URL pattern** that will retrieve the created view in the browser. To **call the view**, we need to map it to a URL - and for this, we need a **URLconf**. In the **phonebook app directory,** open a file called **`urls.py`**:



We want to see the page as we open the localhost, so in this case, the **route** will be an **empty string** (**`""`**), and the view will be the **`landing_page`**. Open the **`urls.py`** file and **add** the following code:

```python
from django.urls import path
from djangoproject.phonebook.views import landing_page

urlpatterns = [
    path('', landing_page, name='landing-page'),
]

```

One more step, we need to point it to the **root URLconf**. To do that, in **`djangoproject/urls.py`** include the **`djangoproject.phonebook.urls`** module, so the file looks like this:

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

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

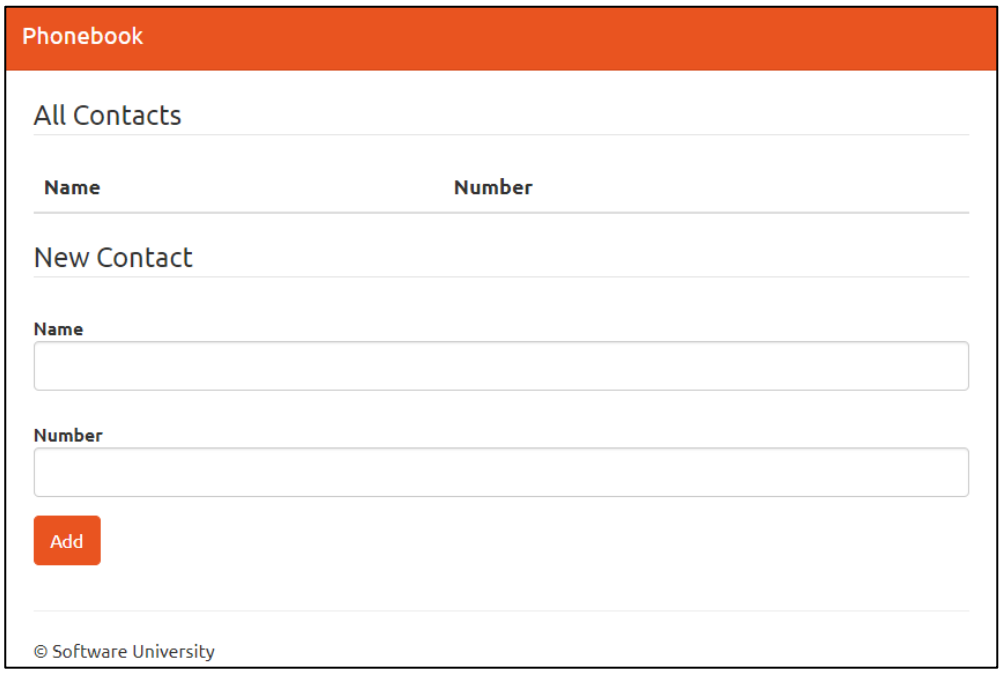
You can now **run** the created project so far. Open the **Terminal** and **write the command**:

```
py manage.py runserver
```

You'll see the following output on the command line:

```
System check identified no issues (0 silenced).
September 28, 2021 - 18:39:50
Django version 3.2.7, using settings 'djangoproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
```

Now that the server's running. Visit http://127.0.0.1:8000/ with your Web browser. You can see the **landing page**:



5. Contact Model

It's time to create our first **model**. Open the **`models.py`** file. The file should look like this:

```python
from django.db import models

# Create your models here.

```

We need to define our **`contact`** model. Create a **new python class** called **`Contact`** with its **attributes**:

```python
class Contact(models.Model):
    name = models.CharField(max_length=30)
    number = models.IntegerField()

```

Our **`Contact`** model is ready. The only thing left to do is to **migrate the created model** to the database. For this project, we are using **SQLite**. Run the Django command-line utilities to **create the database tables** automatically:

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

You can see the migrations in the automatically created .py file in the **`phonebook/migrations`** directory:

```
class Migration(migrations.Migration):

    initial = True

    dependencies = [
    ]

    operations = [
        migrations.CreateModel(
            name='Contact',
            fields=[
                ('id', models.BigAutoField(auto_created=True, primary_key=True, serialize=False, verbose_name='ID')),
                ('name', models.CharField(max_length=30)),
                ('number', models.IntegerField()),
            ],
        ),
    ]
```

6. Creating New Contact

We have reached the point where we can develop our business logic for **creating contact** with a **name** and a **number**. Open the **`views.py`** file and create a second view. For the example, we will name it **`create_contact`**:

```
from django.shortcuts import render, redirect
from djangoproject.phonebook.models import Contact

def create_contact(request):
    name = request.POST['name']
    number = request.POST['number']
    contact = Contact(
        name=name,
        number=number
    )
    contact.save()
    return redirect('landing-page')

```

This function will **save** contacts and store them in the database. In the end, it redirects the user back to the landing page as a response. Although we **do not have an HTML template** for this view, we need to create an **URL path**. Open the **phonebook/`urls.py`** file and **add a second path**:

```
from djangoproject.phonebook.views import landing_page, create_contact

urlpatterns = [
    path('', landing_page, name='landing-page'),
    path('new-contact', create_contact, name='new-contact')
]
```

Thanks to the form, we can process **POST requests**:

```
<!--  adding contact form  -->
    <form class="form-horizontal" method="POST">
        {% csrf_token %}
        <fieldset>
            <legend>New Contact</legend>
            <div class="form-group">
                <label for="name" class="col-lg-2 control-label">Name</label>
                <div class="col-lg-10">
                    <input type="text" autofocus="autofocus" name="name" title="Name" class="form-control"
                        id="name"/>
                </div>
            </div>
            <div class="form-group">
                <label for="number" class="col-lg-2 control-label">Number</label>
                <div class="col-lg-10">
                    <input type="number" autofocus="autofocus" name="number" title="Number" class="form-control"
                        id="number"/>
                </div>
            </div>
            <div class="form-group">
                <div class="col-lg-10 col-lg-offset-2">
                    <button type="submit" class="btn btn-primary">Add</button>
                </div>
            </div>
        </fieldset>
    </form>
<!--  end adding contact form  -->
```

In the **`templates/phonebook/index.html`** file, we see the **`<form> … </form>`** that allows a visitor to enter text, select options, manipulate objects or controls, and so on, and then send that information back to the server. Each form must define two things:

- The **HTTP method** is used to **send the data** using the **`method`** attribute
- The **destination of the data** on the server using the **`action`** attribute

In our template, we have **already implemented the method**. The only thing left to do is to **add an action parameter**.

```
<!--  adding contact form  -->
    <form class="form-horizontal" action="{% url 'new-contact' %}" method="POST">
        {% csrf_token %}
```

To connect the **form** with our **`create_contact`** view. We use the **name** of the created in the **`urls.py` path: `"new-contact"`**.

So far, we have implemented the business logic for creating a contact and sending the user data to the database. Each time a user clicks on the **`"Add"`** button, a name with a number will be saved in the database.

7. Adding the Created Contact

Finally, we need to **add the contact to our list** and give a **response** to the user.

In the **`views.py`** file, in our **`landing_page`** view, we need to implement the following code:

```python
def landing_page(request):
    context = {
        'contacts': Contact.objects.all(),
    }
    return render(request, 'phonebook/index.html', context)

```

The **`render()`** function takes the **request object** as its first argument, a **template name** as its second argument, and a dictionary with the **context** we want to return as its optional third argument. It returns an **HttpResponse object** of the **given template rendered** with the **given context**. In our case, the **context** needs to be **all created contacts** (objects).

The key name **`"contacts"`** is used as a **variable name** in the **template `index.html`**:

```html
<tr>{% for contact in contacts %}
<tr>
    <td> {{ contact.name }} </td>
    <td>{{ contact.number }}</td>
    {% endfor %}
</tr>
```

The code above implements logic for **iterating over the collection of contacts** and taking **each name** and **number**. Thus, they can be **visualized in the contacts table** on our localhost server.

With that, we finished our **Python Phonebook**. Feel free to **build on your project even further**. ☺