How to use React in Django – the Hard Way

In the previous article, we've went through a basic tutorial on how to use React.js in Django. It mainly focuses on front-end and has little connection to Django view. I know it's not enough. Here comes the hard way.

In this article, we are going to dig into a common feature in web application - form submission. The Django conventional way is to build form in backend and render it in a main template. Add JS code if you need some interactions between user and GUI. The downside is that if form GUI is complex, front-end codes can be very hard to maintain. Because very likely form needs be rendered manually. Plus, we may also need to create and import an exclusive JS file for form interaction. Keep in mind that sometimes a page could have more than one form. What a headache! We need something that can put HTML and JS together in a single file. Then React comes to rescue. Thanks to its component-based design, we can maintain both template and interaction codes in a JS file per each form. It makes main template clean. You will know what I mean as you read conclusion at the end of this article.

Following steps in this article, we will architect a complete workflow that relies on both React in the front-end and Django in the back-end. As what I mentioned in the previous article, still I am going to use Django as a web framework. This article might require you have basic knowledge on React component, JSX syntax, Django form and Bootstrap. At the end of tutorial, we will create a form widget and be able to submit a POST request.

Dependencies:

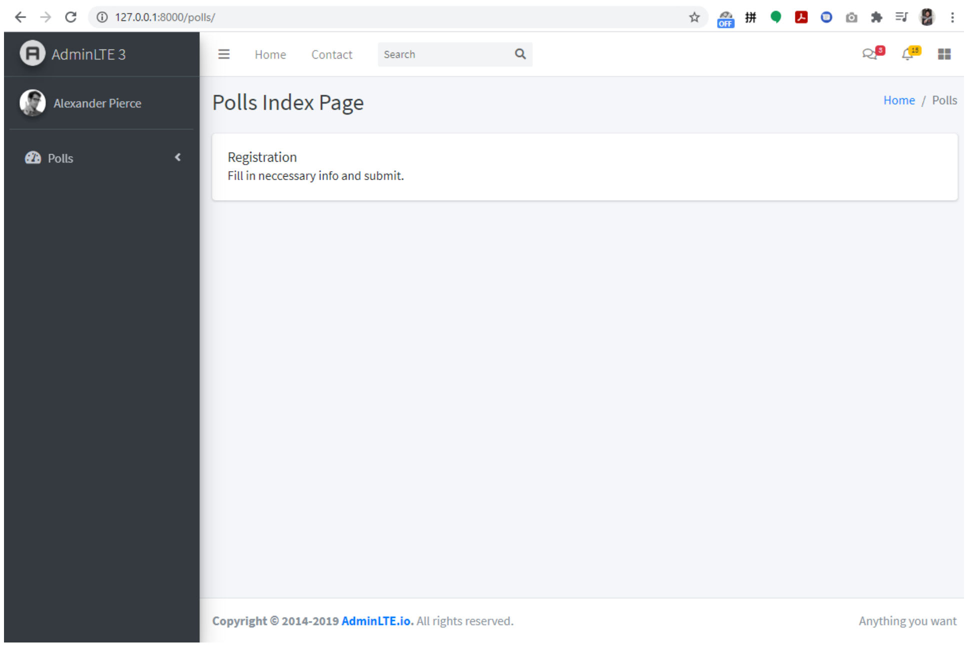
1. Python 3.7
2. Django 2.2.x
3. React 17
4. AdminLTE 3.0.5 (see my previous tutorial here for this theming library)

Step 0. Scratch project and goal

I've upload scratch project in release 1.0.0. Click the link below and download "ScratchProject".zip file.

https://github.com/slow999/DjangoAndReactComponentForm/releases/tag/1.0.0

Let's start project and see what we've got. In 127.0.0.1:8000/polls/ page, our goal is to add a form widget in white box.



Polls Index Page

Home / Polls

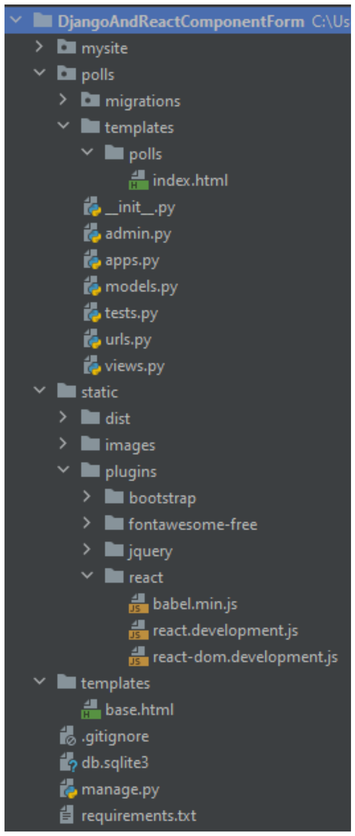
Registration

Fill in neccessary info and submit.

Copyright © 2014-2019 AdminLTE.io. All rights reserved.

Anything you want

The schema of scratch project is as below. We will mainly work in **polls** app and **index.html** template. React and JSX libraries are maintained in **static/plugins/react/** folder.



```
DjangoAndReactComponentForm C:\Us
  mysite
  polls
    migrations
    templates
      polls
        index.html
    __init__.py
    admin.py
    apps.py
    models.py
    tests.py
    urls.py
    views.py
  static
    dist
    images
    plugins
      bootstrap
      fontawesome-free
      jquery
      react
        babel.min.js
        react.development.js
        react-dom.development.js
  templates
    base.html
  .gitignore
  db.sqlite3
  manage.py
  requirements.txt
```

Step 1 Add CSRF script

One of Django's security mechanism is to provide protection against cross site request forgeries through CSRF middleware. It is required whenever a POST request is made. Since we use React to render form in template, the convention way of using {% csrf_token %} does not work anymore. Instead, we need to manually do it. The goal of this script is to produce a CSRF token <input> element that needs be added into form down the road. Code snippet is as below.

```
function getCookie(name) {
  let cookieValue = null;
  if (document.cookie && document.cookie !== '') {
    const cookies = document.cookie.split(';');
    for (let i = 0; i < cookies.length; i++) {
      const cookie = cookies[i].trim();
      // Does this cookie string begin with the name we want?
      if (cookie.substring(0, name.length + 1) === (name + '=')) {
        cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
        break;
      }
    }
  }
  return cookieValue;
}


const csrftoken = getCookie('csrftoken');

const CSRFToken = () => {
  return (
    <input type="hidden" name="csrfmiddlewaretoken" value={csrftoken} />
  );
};
// export default CSRFToken;
```

Let's import this script into **index.html** file in **extra_js** block.

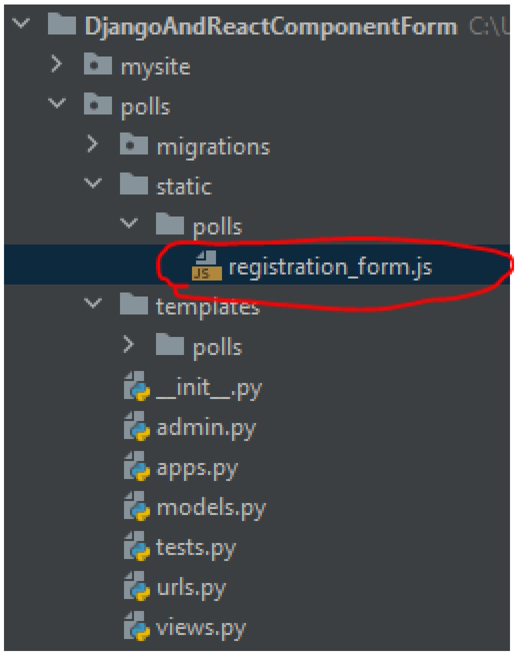
```
{% block extra_js %}
 <script src="{% static 'dist/js/csrftoken.js' %}" type="text/babel"></script>
{% endblock %}
```

Step 2 Add form division and React form script

In **index.html** file, add a division and assign id "registration_form". We will render React form in this division. Here is the code snippet.

```
<div class="col-lg-12">
 <div class="card">
  <div class="card-body">
   <h5 class="card-title">Registration</h5>
   <p class="card-text">
    Fill in neccessary info and submit.
   </p>
   <div id="registration_form"></div>
  </div>
 </div>
</div>
```

Create a **static** folder in **polls** app. In **static** folder we just created, create **polls** folder. Then create **registration_form.js** in **polls** folder we just created. This can be confusing. A pic means thousands of words. You would expect the schema as below.



In **registration_form.js** file, we will build React form as a component. I will paste code snippet in minutes. Something worth mentioning are

1. This form widget has two steps. GUI is controlled by “Go Back” and “Next’ buttons. The visibility of <input> elements is controlled by **d-none** class.
2. CSRF token input is declared within <form> tag through <CSRFToken/>.
3. Submit behavior is controlled by **handleSubmit** function.

Code snippet is as below.

```
'use strict';

const e = React.createElement;
```

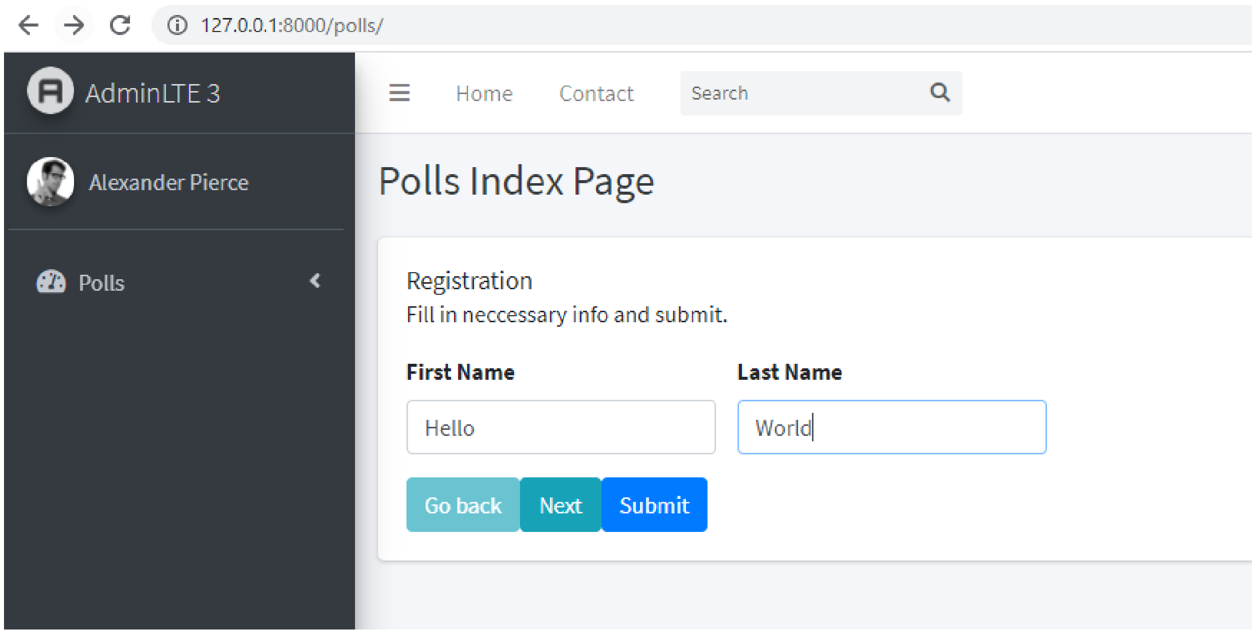
```
class MyForm extends React.Component {
 constructor(props) {
  super(props);
  this.state = {
   first_name: '',
   last_name: '',
   email: '',
   current_step: 1,
   total_steps: 2,
  };
  this.goToNextClick = this.goToNextClick.bind(this);
  this.goToPreviousClick = this.goToPreviousClick.bind(this);
  this.handleChange = this.handleChange.bind(this);
  this.handleSubmit = this.handleSubmit.bind(this);
 }

 handleChange(event) {
  this.setState({[event.target.name]: event.target.value});
 }

 goToNextClick() {
  event.preventDefault();
  this.setState({current_step: this.state.current_step + 1});
 }

 goToPreviousClick() {
  event.preventDefault();
  this.setState({current_step: this.state.current_step - 1});
 }

 handleSubmit(event) {
  console.log('submitting stuff');
 }

 render() {
  const current_step = this.state.current_step;
  const total_steps = this.state.total_steps;

  let visiableFields;
  visiableFields =
   <div className="row">
    <div className={"col-lg-3 col-md-3 first-name-wrapper " + (current_step === 1 ? null : 'd-none')}>
     <div className="form-group">
      <label htmlFor="id_first_name">First Name</label>
      <input type="text" name="first_name" className="form-control" maxLength="32" required
id="id_first_name"
          autoComplete="off" value={this.state.firstName} onChange={this.handleChange}/>
     </div>
    </div>
    <div className={"col-lg-3 col-md-3 last-name-wrapper " + (current_step === 1 ? null : 'd-none')}>
     <div className="form-group">
      <label htmlFor="id_last_name">Last Name</label>
```

```
          <input type="text" name="last_name" className="form-control" maxLength="32" required
id="id_last_name"
                autoComplete="off" value={this.state.lastName} onChange={this.handleChange}/>
        </div>
      </div>
      <div className={"col-lg-3 col-md-3 email-wrapper " + (current_step === 2 ? null : 'd-none')}>
        <div className="form-group">
          <label htmlFor="id_email">Email</label>
          <input type="text" name="email" className="form-control" maxLength="128" required id="id_email"
                autoComplete="off" value={this.state.email} onChange={this.handleChange}/>
        </div>
      </div>
    </div>

    return (
      <form className="form" method="post" onSubmit={this.handleSubmit}>
        <CSRFToken/>
        {visiableFields}
        <button className="btn btn-info" disabled={current_step === 1} name="go-back"
onClick={this.goToPreviousClick}>Go back</button>
        <button className="btn btn-info" disabled={current_step === total_steps} name="go-to-next"
onClick={this.goToNextClick}>Next</button>
        <button type="submit" className="btn btn-primary" name="action" value="submit">Submit</button>
      </form>
    );
  }
}

ReactDOM.render(<MyForm/>, document.getElementById('registration_form'));
```

Finally, import this script into **index.html** in **extra_js** block. We would expect following result.

```
{% block extra_js %}
  <script src="{% static 'dist/js/csrftoken.js' %}" type="text/babel"></script>
  <script src="{% static 'polls/registration_form.js' %}" type="text/babel"></script>
{% endblock %}
```

Start project and check result. Form widget is live! At this point, submit button will raise exception. Let's fix that in next step.



Polls Index Page

Registration
Fill in neccessary info and submit.

First Name: Hello
Last Name: World

Step 3 Add Django form

The usage of Django form is to validate form inputs. We call it back-end validation. In **polls** app, create a **forms.py** file. And add following codes. Please be aware that we use form.EmailField to validate the pattern of email value.

```python
from django import forms


class RegistrationForm(forms.Form):
    first_name = forms.CharField(
        widget=forms.TextInput(attrs={'class': 'form-control'}),
        max_length=32,
        label='First Name'
    )
    last_name = forms.CharField(
        widget=forms.TextInput(attrs={'class': 'form-control'}),
        max_length=32,
        label='Last Name'
    )
    email = forms.EmailField(
        widget=forms.TextInput(attrs={'class': 'form-control'}),
        max_length=254,
        label='Email'
    )
```

Step 4 Modify view function

In **polls/views.py** file, our goal is to do the followings.

1. Add a POST request case
2. Validate inputs by using RegistrationForm that we built in previous step
3. Produce message

We would expect codes as below. Notice that we will print inputs in console.

```
from django.shortcuts import render
from django.contrib import messages
from polls.forms import RegistrationForm


def index(request):
  context_dict = {'form': None}
  form = RegistrationForm()

  if request.method == 'GET':
    context_dict['form'] = form
  elif request.method == 'POST':
    form = RegistrationForm(request.POST)
    context_dict['form'] = form
    if form.is_valid():
      cleaned_data = form.cleaned_data
      print(cleaned_data)
      messages.success(request, 'Your data has been submitted')
    else:
      messages.error(request, 'Something is wrong in form.')

  return render(request, 'polls/index.html', context_dict)
```

Now we need to render message in template. In polls/templates/polls/**index.html** file, add following codes before main content division. This piece of codes covers following cases.

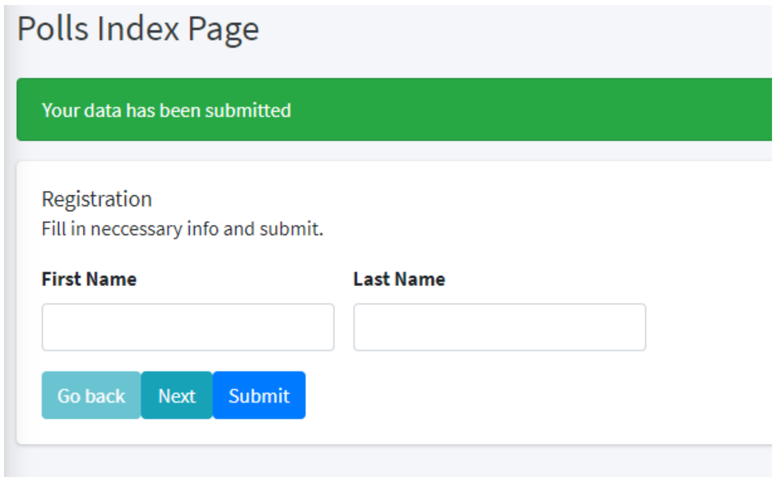
1. If submission is successful, it displays success message in a green box.
2. If form input is invalid, it displays error message in a red box.

```
<div class="content">
 <div class="container-fluid">
  {% if messages %}
   {% for message in messages %}
    {% if message.tags == 'error' %}
     <div class="alert alert-danger" role="alert">
    {% else %}
     <div class="alert alert-{{ message.tags }}" role="alert">
    {% endif %}
     {{ message }}
    </div>
   {% endfor %}
  {% endif %}

  {% if form.errors %}
  <div class="alert alert-danger" role="alert">
   {{ form.non_field_errors }}
   {% for field in form %}
    {% for error in field.errors %}
     {{ error }}
    {% endfor %}
   {% endfor %}
  </div>
  {% endif %}
 </div>
</div>
```

At this point, we have completed everything. Let's check out final product! Fill in data and submit. We should expect similar inputs printed in console as below.

```
{'first_name': 'hello', 'last_name': 'world', 'email': '1234@hello.com'}
```

Front-end should have success message as below.

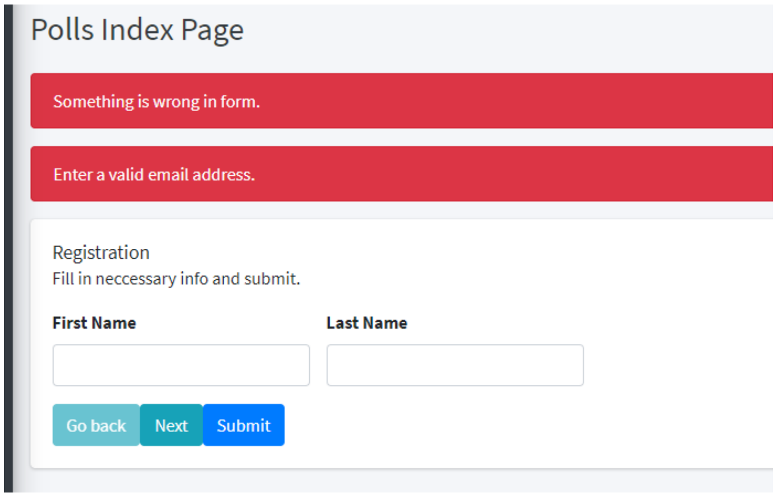

Polls Index Page

Your data has been submitted

Registration
Fill in neccessary info and submit.

First Name **Last Name**

Error messages are as below.

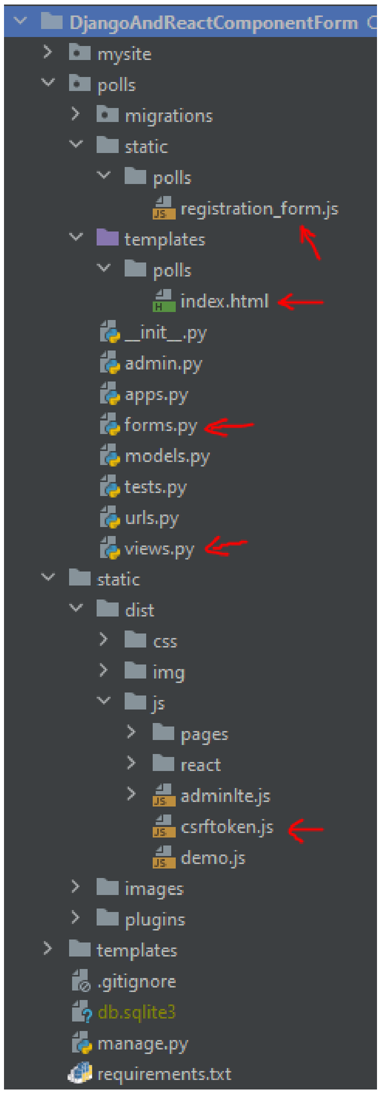

Polls Index Page

Something is wrong in form.

Enter a valid email address.

Registration
Fill in neccessary info and submit.

Final schema is as below.



```
DjangoAndReactComponentForm
  mysite
  polls
    migrations
    static
      polls
        registration_form.js
    templates
      polls
        index.html
    __init__.py
    admin.py
    apps.py
    forms.py
    models.py
    tests.py
    urls.py
    views.py
  static
    dist
      css
      img
      js
        pages
        react
        adminlte.js
        csrftoken.js
        demo.js
    images
    plugins
  templates
  .gitignore
  db.sqlite3
  manage.py
  requirements.txt
```

Conclusion

In this tutorial, we have taken full usage of Django and React. React is responsible for rendering a complex GUI. **index.html** file is main template. Rather than dumping form template and js codes into main template, we build a **registration_form.js** to encapsulates form component. All we need to do is to import it and tell React in which division we would like to render form into. In addition, this React form component can be reused in other pages as well. Django is responsible for handling HTTP request and provide necessary validation. The whole piece is running on Django web framework that provides security mechanism through CSRF token. Now enjoy fancy GUI without a worry on security! Hope this architect can give you a hint on your current project.

If you are interested, I've uploaded this project on my GitHub. Project name is DjangoAndReactComponentForm.

https://github.com/slow999/DjangoAndReactComponentForm

Thanks for reading. I would like to know your thoughts on React + Django combination. Please leave your comments below. Stay tuned.