CRUD App using Vue.js and Django

DJANGO VUE JS

In this tutorial you will learn how to make a simple CRUD application using Vue.js 2.x and Django 2.0.2. Vue.js is progressive framework for building user interfaces while Django is a high level python Development framework that encourages rapid Develpoment.

Follow this steps to make a CRUD app using Vue.js and Django

Installing Django

Making the Django-project and app

Creating Models and Migrations

Installing Django-rest-framework

Creating Serializer, Viewset and Routers

Configuring Vue.js with Django

Installing Django

Make sure you have python 3, pip, virtualenv installed on you pc (Django 2.0 version have removed the support of python 2.x version).You can skip the steps till create-project if you already installed django installed on your system.

Create a project folder and run the command from terminal inside the folder

```
virtualenv venv -p $(which python3)
```

Virtualenv gives the virtual environment for python packages so that it won't harm your global packages version

Then activate this virtualenv.

```
source venv/bin/activate
```

You will see the (venv) written in the beginning of the command if it is activated. Now installing Django

```
pip install Django
```

Making the Django-project and app

Now we have installed the latest version of django in our virtualenv now make a project inside the same folder

```
django-admin startproject myproject
```

A myproject folder is created having a manage.py file and myproject folder containing settings.py which containing all the settings of your project. In django everything is break down into small apps so we would create a app. Come inside the myproject folder containing the manage.py and run the following command

```
python manage.py startapp article
```

This will create a folder article having files models.py admin.py tests.py views.py admin.py apps.py and migrations folder.

Models contain the database models, admin contains the configuration of the admin interface generated by the django and tests is for writing the tests for you app and views contain the controller function to interact with templates and models and migration folder contain the database migrations generated generated by our models. Before moving further mention the app name in the settings file in side the myproject directory /myproject/myproject/settings.py in the installed apps section so that our project can access the app.

```
INSTALLED_APPS = [
 ‘django.contrib.admin’,
 ‘django.contrib.auth’,
 ‘django.contrib.contenttypes’,
 ‘django.contrib.sessions’,
 ‘django.contrib.messages’,
 ‘django.contrib.staticfiles’,
 ‘article’
]
```

Create Models and Migrations

Models are the database models or you could say the database fields. Django orm is highly customized it could make database structure and database migrations from models.

Creating article model in the models.py inside the article app

```
# Create your models here.
class Article(models.Model):
    article_id = models.AutoField(primary_key=True)
    article_heading = models.CharField(max_length=250)
    article_body = models.TextField()
```

To get more info of the django models. Refer to this link https://docs.djangoproject.com/en/2.0/topics/db/models/

Now make migrations for the model. Go to the base myproject folder containing manage.py and run

```
python manage.py makemigrations
```

You will get the following outputs mentioning that your migration is created in the migrations folder

```
Migrations for ‘article’:
 article/migrations/0001_initial.py
 — Create model Article
```

Now migrate this file to create a database structure for this.

```
python manage.py migrate
```

After running you will see a lot of migrations going , there are of the user system (default given by the django)which you could use in your app.In the list you will also found the above migration.

Installing Django-rest-framework

Now we would install django rest framework. Django rest framework is a library built over the django to make rest api's. You make api's using custom function using django but you will miss some security exception or some status or base issues.Django rest framework has already accounted these issues.So there is no worry before using.

Get more info about django-rest-framework at http://www.django-rest-framework.org/

Installing django rest framework

```
pip install djangorestframework
```

Update the settings.py file for the rest framework

```
INSTALLED_APPS = [
 'django.contrib.admin',
 'django.contrib.auth',
 'django.contrib.contenttypes',
 'django.contrib.sessions',
 'django.contrib.messages',
 'django.contrib.staticfiles',
 'article',
 'rest_framework'
]
```

Finally yay!! you are almost done with 70% work for creating api's.Now just move to create viewsets and routers.

Creating Serializer, Viewset and Routers

Inside the article create a file **serializers.py** it contains serializers for you api. Serializers allow complex data such as querysets and model instances to be converted to native Python datatypes that can then be easily rendered into `JSON`, `XML` or other content types. Let's make serializers.

```
from rest_framework import serializers
from .models import Article

class ArticleSerializer(serializers.ModelSerializer):
    class Meta:
        model = Article
        fields = '__all__'
```

First of all we import the serializers class from the rest framework library and then import the model whose data we have to structure. Define a class for our serializers having the base class as a rest framework serializer. In the meta description mention the models and it's fields

Get more info about serializer here: http://www.django-rest-framework.org/api-guide/serializers/

Now Let's create Viewsets. Create a **viewsets.py** inside the same folder.

Django REST framework allows you to combine the logic for a set of related views in a single class, called a `ViewSet`. In other frameworks you may also find conceptually similar implementations named something like 'Resources' or 'Controllers'.

```
from rest_framework import viewsets
from .models import Article
from .serializers import ArticleSerializer

class ArticleViewSet(viewsets.ModelViewSet):
    queryset = Article.objects.all()
    serializer_class = ArticleSerializer
```

first we import the viewsets class and then our model and serializers(which we created in previous step). Now define the viewset class ArticleViewSet i which we define the **queryset** the data we got in when we query and then the **serializer_class** to serialize that data.

Get more info about viewsets here: http://www.django-rest-framework.org/api-guide/viewsets/

Now creating a router one step behind creating rest api's in the django. Some Web frameworks such as Rails provide functionality for automatically determining how the URLs for an application should be mapped to the logic that deals with handling incoming requests. Router automatically create such request on it's own.Moreover create a common file for all the routers for various apps to handle api's easily.

Create a file **routers.py** inside the myproject folder where there is **settings.py** and **urls.py** file is present.

```
from rest_framework import routers
from article.viewsets import ArticleViewSet

router = routers.DefaultRouter()

router.register(r'article', ArticleViewSet)
```

First as we import the routers from the rest_framework and then import our viewset and then we define the function router where we later enter our viewset for various urls, now api would be somewhat like **/article**linked to ArticleViewSet.

Get more info about viewsets here: http://www.django-rest-framework.org/api-guide/routers/

Now we import this routers file inside the urls.py which contain all the url routes of our app.

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

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

We have imported our router file to include all urls built inside the routers file. We have added the api/ keyword just to seperate the api urls now they will called from **/api/article.**

Now you are completed with you api part

You got the following api's

GET: /api/article/

This will give all articles

POST: /api/article

This will help to add new article

DELETE: /api/article/{article_id}/

This will help to delete the article

GET: /api/article/{article_id}/

This will return particular article

PUT: /api/article/{article_id}/

This will help to update all the fields of a particular article

PATCH: /api/article/{article_id}/

This will help to make a patch inside the article

Configuring Vue.js with Django

Now let’s intergate these api’s inside our templates. Create a folder **templates** inside the article folder and inside the templates folder make a file **index.html**

```
<!DOCTYPE html>
<html lang=”en”>
 <head>
 <meta charset=”utf-8">
 <title>Vue-js | Django | Crud App</title>
 <meta name=”viewport” content=”width=device-width, initial-scale=1.0">
 <meta name=”description” content=”A simple crud app made with the vue js and
django”>

<meta name=”keywords” content=”vuejs, django, crudapp, restapi”>
 <! — bootstap →
 <link rel=”stylesheet”
href=”https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css"
integrity=”sha384-
Gn5384xqQ1aoWXA+058RXPxPg6fy4IWvTNh0E263XmFcJlSAwiGgFAW/dAiS6JXm”
crossorigin=”anonymous”>
 <! — boostrap css →

</head>

<body>

<! — bootrtap js files →
 <script src=”https://code.jquery.com/jquery-3.2.1.slim.min.js"
integrity=”sha384-
KJ3o2DKtIkvYIK3UENzmM7KCkRr/rE9/Qpg6aAZGJwFDMVNA/GpGFF93hXpG5KkN”
crossorigin=”anonymous”></script>
 <script
src=”https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.12.9/umd/popper.min.js"
integrity=”sha384-
ApNbgh9B+Y1QKtv3Rn7W3mgPxhU9K/ScQsAP7hUibX39j7fakFPskvXusvfa0b4Q”
crossorigin=”anonymous”></script>
 <script
src=”https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/js/bootstrap.min.js"
integrity=”sha384-
JZR6Spejh4U02d8jOt6vLEHfe/JQGiRRSQQxSfFWpi1MquVdAyjUar5+76PVCmYl”
crossorigin=”anonymous”></script>

 <! — vue.js files →
 <script src=”https://cdn.jsdelivr.net/npm/vue@2.5.13/dist/vue.js"></script>
 <script src=”https://cdn.jsdelivr.net/npm/vue-resource@1.3.5"></script>

</body>
</html>
```

Inside the file you will see i have the added the bootstrap css and js cdn links for design. Then I have added the vue.js links. vue.js is the main **vue js** library and then there is **vue-resource** library for calling rest api's.

Also update the urls.py file

```
from django.contrib import admin
from django.urls import path, include
from .routers import router
from django.views.generic import TemplateView

urlpatterns = [
    path('admin/', admin.site.urls),
    path('api/', include(router.urls)),
    path('article', TemplateView.as_view(template_name='index.html')),
]
```

Below this we will create a vue js instance inside script tag

```
<script type="text/javascript">
 new Vue({
   el: '#starting',
   delimiters: ['${','}'],
   data: {
   articles: [],
   loading: false,
   currentArticle: {},
   message: null,
   newArticle: { 'article_heading': null, 'article_body': null },
 },
 mounted: function() {

},
 methods: {

}
 });
 </script>
```

Now understading this vue instance, the el is the id or the class of the divison(div in the body where the vue js instance gonna run, delimiters are the tags which we apply around the vue js variables to display data in html file, data contain all the data inside the vue js library, mounted is the function which runs before the mounting of the vue js instance, before that there is methods contain all the functions which gonna run in the vue instance.

First of all we gonna add following methods

getArticles → which give all the articles

getArticle → which give the particular article

addArticle → which will add a new article

updateArticle → which will update a article

deleteArticle → which will delete a article

```
mounted: function() {
 this.getArticles();
},
methods: {
 getArticles: function() {
  this.loading = true;
  this.$http.get('/api/article/')
      .then((response) => {
        this.articles = response.data;
        this.loading = false;
      })
      .catch((err) => {
       this.loading = false;
       console.log(err);
      })
 },
 getArticle: function(id) {
  this.loading = true;
  this.$http.get(`/api/article/${id}/`)
      .then((response) => {
        this.currentArticle = response.data;
        this.loading = false;
      })
      .catch((err) => {
        this.loading = false;
```

```
          console.log(err);
        })
    },
    addArticle: function() {
     this.loading = true;
     this.$http.post('/api/article/',this.newArticle)
        .then((response) => {
          this.loading = false;
          this.getArticles();
        })
        .catch((err) => {
          this.loading = false;
          console.log(err);
        })
    },
    updateArticle: function() {
     this.loading = true;
     this.$http.put(`/api/article/${this.currentArticle.article_id}/`,
  this.currentArticle)
        .then((response) => {
          this.loading = false;
          this.currentArticle = response.data;
          this.getArticles();
        })
        .catch((err) => {
          this.loading = false;
          console.log(err);
        })
    },
    deleteArticle: function(id) {
     this.loading = true;
     this.$http.delete(`/api/article/${id}/` )
        .then((response) => {
          this.loading = false;
          this.getArticles();
        })
        .catch((err) => {
          this.loading = false;
          console.log(err);
        })
    }
```

In the mounted function we have run the method getArticles to get all the articles when the page loads. There after in the getArticles we have loading variable this helps to show page loading when the api is loading.There after we have vue resource code to call the api and to handle it's response.Which is mostly same for the every function in the

```
  this.$http.request_type('api_url',payload)
      .then((response) => {
```

```
      // code if the api worked successfully
    })
    .catch((err) => {
      // code if the api show some error
    })
```

Now try to implement these functions in html file.

```
<body>
    <div id="starting">
      <div class="container">
        <div class="row">
          <h1>List of Articles
          <button class="btn btn-success">ADD ARTICLE</button>
          </h1>
          <table class="table">
            <thead>
              <tr>
                <th scope="col">#</th>
                <th scope="col">Heading</th>
                <th scope="col">Action</th>
              </tr>
            </thead>
            <tbody>
              <tr v-for="article in articles">
                <th scope="row">${article.article_id}</th>
                <td>${article.article_heading}</td>
                <td>
                  <button class="btn btn-info" v-
on:click="getArticle(article.article_id)">Edit</button>
                  <button class="btn btn-danger" v-
on:click="deleteArticle(article.article_id)">Delete</button>
                </td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
      <div class="loading" v-if="loading===true">Loading&#8230;</div>
    </div>
```

This is the base structure of our html body in this we have displayed the articles using the v-for tag which will loop through the articles array. Then we have dispalyed the vue.js data. You will notice vue.js data is enclosed in these tags **${}** these the delimeters which have set in the vue instance.Same if you see the bold div. You will see the id="starting" the same we have mentioned in the vue instance.Now work on more detailing.

First add Article pop-up.Make following changes in the index.html file

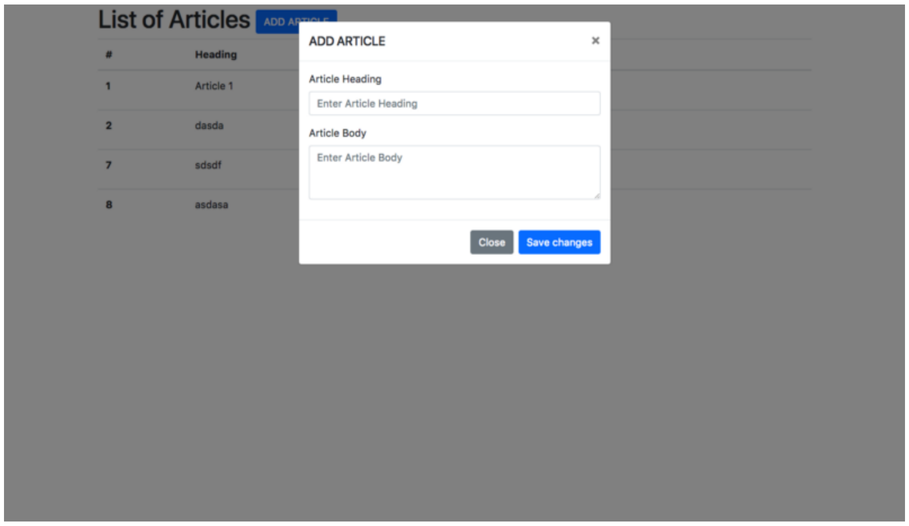
```
<button  type="button" class="btn btn-primary" data-toggle="modal" data-
target="#addArticleModal">ADD ARTICLE</button>
```

Now we gonna add the addArticle Modal below the table tag

```
<!-- Add Article Modal -->
          <div class="modal fade" id="addArticleModal" tabindex="-1"
role="dialog" aria-labelledby="exampleModalLongTitle" aria-hidden="true">
            <div class="modal-dialog" role="document">
              <div class="modal-content">
                <div class="modal-header">
                  <h5 class="modal-title" id="exampleModalLongTitle">ADD
ARTICLE</h5>
                  <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
                    <span aria-hidden="true">&times;</span>
                  </button>
                </div>
                <form v-on:submit.prevent="addArticle()">
                <div class="modal-body">
                    <div class="form-group">
                      <label for="article_heading">Article Heading</label>
                      <input
                        type="text"
                        class="form-control"
                        id="article_heading"
                        placeholder="Enter Article Heading"
                        v-model="newArticle.article_heading"
                        required="required" >
                    </div>
                    <div class="form-group">
                      <label for="article_body">Article Body</label>
                      <textarea
                        class="form-control"
                        id="article_body"
                        placeholder="Enter Article Body"
                        v-model="newArticle.article_body"
                        required="required"
                        rows="3"></textarea>
                    </div>
                </div>
                <div class="modal-footer">
```

```
                  <button type="button" class="btn btn-secondary" data-
dismiss="modal">Close</button>
                  <button type="submit" class="btn btn-primary">Save
changes</button>
                </div>
                </form>
              </div>
            </div>
           <div class="loading" v-if="loading===true">Loading&#8230;</div>
          </div>
          <!-- End of article modal -->
```

In this modal you will see v-on:submit.prevent function to submit form. Each input of the form is holded by the v-model attibute which is mapped with the data in the vue js instance. Moreover there is v-if clause which will run the api is loading. If will show a loader on the screen when then api request is loading. Some css classes are written for it. Please check in below github repo.


List of Articles

#	Heading
1	Article 1
2	dasda
7	sdsdf
8	asdasa

ADD ARTICLE

Article Heading

Enter Article Heading

Article Body

Enter Article Body

Add Article Window

Now the Edit Part and View

When you click on the Edit you will see the same form having the current click article info which you can edit.

```
<button class="btn btn-info" v-
on:click="getArticle(article.article_id)">Edit</button>
```

This function we have already defined and also mentioned for each article in the view. We will little update the vue.js function to view the edit Modal.

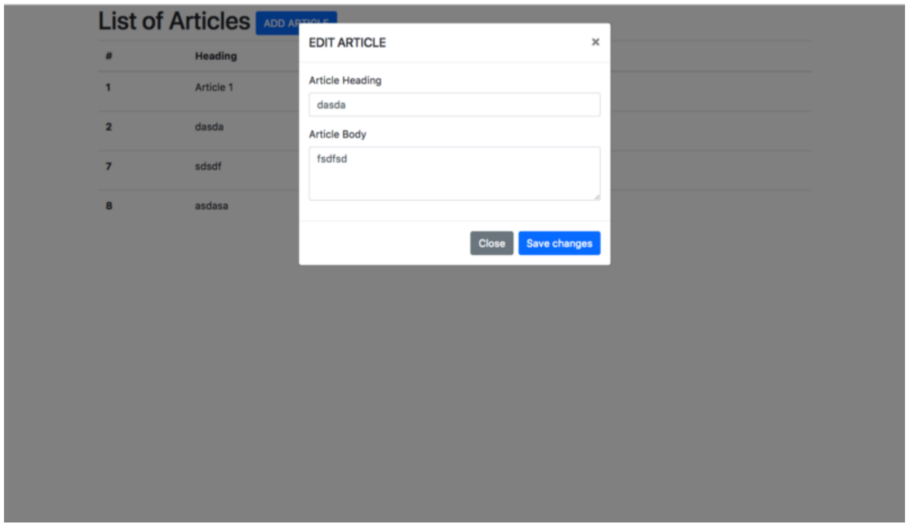
```
getArticle: function(id) {
          this.loading = true;
          this.$http.get(`/api/article/${id}/`)
              .then((response) => {
                this.currentArticle = response.data;
                $("#editArticleModal").modal('show');
                this.loading = false;
              })
              .catch((err) => {
                this.loading = false;
                console.log(err);
              })
        },
```

Now we gonna add the edit modal code below the add modal code

```
<!-- Edit Article Modal -->
          <div class="modal fade" id="editArticleModal" tabindex="-1"
role="dialog" aria-labelledby="exampleModalLongTitle" aria-hidden="true">
            <div class="modal-dialog" role="document">
              <div class="modal-content">
                <div class="modal-header">
                  <h5 class="modal-title" id="exampleModalLongTitle">EDIT
ARTICLE</h5>
                  <button type="button" class="close" data-dismiss="modal" aria-
label="Close">
                    <span aria-hidden="true">&times;</span>
                  </button>
                </div>
                <form v-on:submit.prevent="updateArticle()">
                <div class="modal-body">
                    <div class="form-group">
                      <label for="article_heading">Article Heading</label>
                      <input
                        type="text"
                        class="form-control"
                        id="article_heading"
                        placeholder="Enter Article Heading"
```

```
                          v-model="currentArticle.article_heading"
                          required="required" >
                      </div>
                      <div class="form-group">
                        <label for="article_body">Article Body</label>
                        <textarea
                          class="form-control"
                          id="article_body"
                          placeholder="Enter Article Body"
                          v-model="currentArticle.article_body"
                          required="required"
                          rows="3"></textarea>
                      </div>
                  </div>
                  <div class="modal-footer">
                    <button type="button" class="btn btn-secondary m-progress"
data-dismiss="modal">Close</button>
                    <button type="submit" class="btn btn-primary">Save
changes</button>
                  </div>
                  </form>
                </div>
              </div>
              <div class="loading" v-if="loading===true">Loading&#8230;</div>
            </div>
            <!-- End of edit article modal -->
```

It is almost similar only the variables are changed.


List of Articles

#	Heading
1	Article 1
2	dasda
7	sdsdf
8	asdasa

EDIT ARTICLE

Article Heading: dasda

Article Body: fsdfsd

EDIT ARTICLE

Now Delete Part

We have already done the delete part if you will observe

```
<button class=”btn btn-danger” v-
on:click=”deleteArticle(article.article_id)”>Delete</button>
```

Yayyy!! You have completed the tutorial.Now you can make crud app using django and vue.js. Just do one thing. hit

```
python manage.py runserver
```

and then browser the

```
http://127.0.0.1:8000/article
```

you will see your crud app working, add article, edit article, delete article.

Please refer to **https://github.com/ShubhamBansal1997/crud-app-vuejs-django** to see the code and this webapp is active here **https://vue-django-crud.herokuapp.com**.If you found any error please do mention in the comments. Please also these related articles

Token Based Authentication for Django Rest Framework
Django is of the popular web development framework based on python having a large…
medium.com

SearchFilter using Django and Vue.js
The post is in continuation with my previous

post where I explained to make CRUD api's…
medium.com

For more assistance and want to hire me. Drop me a mail at shubhambansal17@hotmail.com