Django\_template

Here is the Django template fragment for User Registration:  
  
```html  
<!-- registration.html -->  
<h2>Register</h2>  
  
<form method="post">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Submit</button>  
</form>  
  
<script src="{% static 'js/script.js' %}"></script>  
<style>  
 /\* Add some basic styling \*/  
 body {  
 font-family: Arial, sans-serif;  
 }  
 label {  
 display: block;  
 margin-bottom: 10px;  
 }  
 input[type="text"], input[type="email"] {  
 width: 100%;  
 height: 40px;  
 padding: 10px;  
 border: 1px solid #ccc;  
 }  
 button[type="submit"] {  
 background-color: #4CAF50;  
 color: #fff;  
 padding: 10px 20px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
 button[type="submit"]:hover {  
 background-color: #3e8e41;  
 }  
</style>  
```  
  
And here is the corresponding JavaScript file (`script.js`) for validation purposes:  
  
```javascript  
// script.js  
function validateForm() {  
 // Get all input fields  
 const inputs = document.querySelectorAll('input');  
  
 // Loop through each input field  
 inputs.forEach((input) => {  
 // Check if the input field has an error message  
 if (input.nextElementSibling && input.nextElementSibling.className === 'error') {  
 // Remove the error message  
 input.nextElementSibling.remove();  
 }  
  
 // Add event listener for change events  
 input.addEventListener('change', () => {  
 // Validate the input field  
 validateInput(input);  
 });  
 });  
  
 function validateInput(input) {  
 // Get the value of the input field  
 const value = input.value.trim();  
  
 // Check if the input field is required  
 if (!input.required) {  
 return true;  
 }  
  
 // Check if the input field is empty  
 if (!value.length) {  
 // Show an error message  
 const errorMessage = document.createElement('div');  
 errorMessage.textContent = 'This field is required';  
 errorMessage.className = 'error';  
 input.parentNode.appendChild(errorMessage);  
  
 // Return false to prevent form submission  
 return false;  
 }  
  
 // Input field is valid, remove any existing error messages  
 const errorMessages = input.parentNode.querySelectorAll('.error');  
 Array.prototype.forEach.call(errorMessages, (errorMessage) => {  
 errorMessage.remove();  
 });  
  
 // Return true to allow form submission  
 return true;  
 }  
}  
  
document.addEventListener('DOMContentLoaded', () => {  
 // Call the validateForm function when the page loads  
 validateForm();  
});  
```  
  
Note: The above code assumes that you have a `forms.py` file where you define a form class using Django's built-in forms module. You can customize the form fields and their validation logic as per your requirements.  
  
Please let me know if this meets your expectations! If there's anything I missed or need further clarification on, please don't hesitate to ask!

Here is the Django template fragment for User Registration:  
  
```html  
<!-- registration.html -->  
<h2>Register</h2>  
  
<form method="post">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Submit</button>  
</form>  
  
<script src="{% static 'js/script.js' %}"></script>  
<style>  
 /\* Add some basic styling \*/  
 body {  
 font-family: Arial, sans-serif;  
 }  
 label {  
 display: block;  
 margin-bottom: 10px;  
 }  
 input[type="text"], input[type="email"] {  
 width: 100%;  
 height: 40px;  
 padding: 10px;  
 border: 1px solid #ccc;  
 }  
 button[type="submit"] {  
 background-color: #4CAF50;  
 color: #fff;  
 padding: 10px 20px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
 button[type="submit"]:hover {  
 background-color: #3e8e41;  
 }  
</style>  
```  
  
And here is the corresponding JavaScript file (`script.js`) for validation purposes:  
  
```javascript  
// script.js  
function validateForm() {  
 // Get all input fields  
 const inputs = document.querySelectorAll('input');  
  
 // Loop through each input field  
 inputs.forEach((input) => {  
 // Check if the input field has an error message  
 if (input.nextElementSibling && input.nextElementSibling.className === 'error') {  
 // Remove the error message  
 input.nextElementSibling.remove();  
 }  
  
 // Add event listener for change events  
 input.addEventListener('change', () => {  
 // Validate the input field  
 validateInput(input);  
 });  
 });  
  
 function validateInput(input) {  
 // Get the value of the input field  
 const value = input.value.trim();  
  
 // Check if the input field is required  
 if (!input.required) {  
 return true;  
 }  
  
 // Check if the input field is empty  
 if (!value.length) {  
 // Show an error message  
 const errorMessage = document.createElement('div');  
 errorMessage.textContent = 'This field is required';  
 errorMessage.className = 'error';  
 input.parentNode.appendChild(errorMessage);  
  
 // Return false to prevent form submission  
 return false;  
 }  
  
 // Input field is valid, remove any existing error messages  
 const errorMessages = input.parentNode.querySelectorAll('.error');  
 Array.prototype.forEach.call(errorMessages, (errorMessage) => {  
 errorMessage.remove();  
 });  
  
 // Return true to allow form submission  
 return true;  
 }  
}  
  
document.addEventListener('DOMContentLoaded', () => {  
 // Call the validateForm function when the page loads  
 validateForm();  
});  
```  
  
Note: The above code assumes that you have a `forms.py` file where you define a form class using Django's built-in forms module. You can customize the form fields and their validation logic as per your requirements.  
  
Please let me know if this meets your expectations! If there's anything I missed or need further clarification on, please don't hesitate to ask!

Here is the Django template fragment for User Registration:  
  
```html  
<!-- registration.html -->  
<h2>Register</h2>  
  
<form method="post">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Submit</button>  
</form>  
  
<script src="{% static 'js/script.js' %}"></script>  
<style>  
 /\* Add some basic styling \*/  
 body {  
 font-family: Arial, sans-serif;  
 }  
 label {  
 display: block;  
 margin-bottom: 10px;  
 }  
 input[type="text"], input[type="email"] {  
 width: 100%;  
 height: 40px;  
 padding: 10px;  
 border: 1px solid #ccc;  
 }  
 button[type="submit"] {  
 background-color: #4CAF50;  
 color: #fff;  
 padding: 10px 20px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
 button[type="submit"]:hover {  
 background-color: #3e8e41;  
 }  
</style>  
```  
  
And here is the corresponding JavaScript file (`script.js`) for validation purposes:  
  
```javascript  
// script.js  
function validateForm() {  
 // Get all input fields  
 const inputs = document.querySelectorAll('input');  
  
 // Loop through each input field  
 inputs.forEach((input) => {  
 // Check if the input field has an error message  
 if (input.nextElementSibling && input.nextElementSibling.className === 'error') {  
 // Remove the error message  
 input.nextElementSibling.remove();  
 }  
  
 // Add event listener for change events  
 input.addEventListener('change', () => {  
 // Validate the input field  
 validateInput(input);  
 });  
 });  
  
 function validateInput(input) {  
 // Get the value of the input field  
 const value = input.value.trim();  
  
 // Check if the input field is required  
 if (!input.required) {  
 return true;  
 }  
  
 // Check if the input field is empty  
 if (!value.length) {  
 // Show an error message  
 const errorMessage = document.createElement('div');  
 errorMessage.textContent = 'This field is required';  
 errorMessage.className = 'error';  
 input.parentNode.appendChild(errorMessage);  
  
 // Return false to prevent form submission  
 return false;  
 }  
  
 // Input field is valid, remove any existing error messages  
 const errorMessages = input.parentNode.querySelectorAll('.error');  
 Array.prototype.forEach.call(errorMessages, (errorMessage) => {  
 errorMessage.remove();  
 });  
  
 // Return true to allow form submission  
 return true;  
 }  
}  
  
document.addEventListener('DOMContentLoaded', () => {  
 // Call the validateForm function when the page loads  
 validateForm();  
});  
```  
  
Note: The above code assumes that you have a `forms.py` file where you define a form class using Django's built-in forms module. You can customize the form fields and their validation logic as per your requirements.  
  
Please let me know if this meets your expectations! If there's anything I missed or need further clarification on, please don't hesitate to ask!

Here is the Django template fragment for User Registration:  
  
```html  
<!-- registration.html -->  
<h2>Register</h2>  
  
<form method="post">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Submit</button>  
</form>  
  
<script src="{% static 'js/script.js' %}"></script>  
<style>  
 /\* Add some basic styling \*/  
 body {  
 font-family: Arial, sans-serif;  
 }  
 label {  
 display: block;  
 margin-bottom: 10px;  
 }  
 input[type="text"], input[type="email"] {  
 width: 100%;  
 height: 40px;  
 padding: 10px;  
 border: 1px solid #ccc;  
 }  
 button[type="submit"] {  
 background-color: #4CAF50;  
 color: #fff;  
 padding: 10px 20px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
 button[type="submit"]:hover {  
 background-color: #3e8e41;  
 }  
</style>  
```  
  
And here is the corresponding JavaScript file (`script.js`) for validation purposes:  
  
```javascript  
// script.js  
function validateForm() {  
 // Get all input fields  
 const inputs = document.querySelectorAll('input');  
  
 // Loop through each input field  
 inputs.forEach((input) => {  
 // Check if the input field has an error message  
 if (input.nextElementSibling && input.nextElementSibling.className === 'error') {  
 // Remove the error message  
 input.nextElementSibling.remove();  
 }  
  
 // Add event listener for change events  
 input.addEventListener('change', () => {  
 // Validate the input field  
 validateInput(input);  
 });  
 });  
  
 function validateInput(input) {  
 // Get the value of the input field  
 const value = input.value.trim();  
  
 // Check if the input field is required  
 if (!input.required) {  
 return true;  
 }  
  
 // Check if the input field is empty  
 if (!value.length) {  
 // Show an error message  
 const errorMessage = document.createElement('div');  
 errorMessage.textContent = 'This field is required';  
 errorMessage.className = 'error';  
 input.parentNode.appendChild(errorMessage);  
  
 // Return false to prevent form submission  
 return false;  
 }  
  
 // Input field is valid, remove any existing error messages  
 const errorMessages = input.parentNode.querySelectorAll('.error');  
 Array.prototype.forEach.call(errorMessages, (errorMessage) => {  
 errorMessage.remove();  
 });  
  
 // Return true to allow form submission  
 return true;  
 }  
}  
  
document.addEventListener('DOMContentLoaded', () => {  
 // Call the validateForm function when the page loads  
 validateForm();  
});  
```  
  
Note: The above code assumes that you have a `forms.py` file where you define a form class using Django's built-in forms module. You can customize the form fields and their validation logic as per your requirements.  
  
Please let me know if this meets your expectations! If there's anything I missed or need further clarification on, please don't hesitate to ask!

Here is the Django template fragment for User Registration:  
  
```html  
<!-- registration.html -->  
<h2>Register</h2>  
  
<form method="post">  
 {% csrf\_token %}  
 {{ form.as\_p }}  
 <button type="submit">Submit</button>  
</form>  
  
<script src="{% static 'js/script.js' %}"></script>  
<style>  
 /\* Add some basic styling \*/  
 body {  
 font-family: Arial, sans-serif;  
 }  
 label {  
 display: block;  
 margin-bottom: 10px;  
 }  
 input[type="text"], input[type="email"] {  
 width: 100%;  
 height: 40px;  
 padding: 10px;  
 border: 1px solid #ccc;  
 }  
 button[type="submit"] {  
 background-color: #4CAF50;  
 color: #fff;  
 padding: 10px 20px;  
 border: none;  
 border-radius: 5px;  
 cursor: pointer;  
 }  
 button[type="submit"]:hover {  
 background-color: #3e8e41;  
 }  
</style>  
```  
  
And here is the corresponding JavaScript file (`script.js`) for validation purposes:  
  
```javascript  
// script.js  
function validateForm() {  
 // Get all input fields  
 const inputs = document.querySelectorAll('input');  
  
 // Loop through each input field  
 inputs.forEach((input) => {  
 // Check if the input field has an error message  
 if (input.nextElementSibling && input.nextElementSibling.className === 'error') {  
 // Remove the error message  
 input.nextElementSibling.remove();  
 }  
  
 // Add event listener for change events  
 input.addEventListener('change', () => {  
 // Validate the input field  
 validateInput(input);  
 });  
 });  
  
 function validateInput(input) {  
 // Get the value of the input field  
 const value = input.value.trim();  
  
 // Check if the input field is required  
 if (!input.required) {  
 return true;  
 }  
  
 // Check if the input field is empty  
 if (!value.length) {  
 // Show an error message  
 const errorMessage = document.createElement('div');  
 errorMessage.textContent = 'This field is required';  
 errorMessage.className = 'error';  
 input.parentNode.appendChild(errorMessage);  
  
 // Return false to prevent form submission  
 return false;  
 }  
  
 // Input field is valid, remove any existing error messages  
 const errorMessages = input.parentNode.querySelectorAll('.error');  
 Array.prototype.forEach.call(errorMessages, (errorMessage) => {  
 errorMessage.remove();  
 });  
  
 // Return true to allow form submission  
 return true;  
 }  
}  
  
document.addEventListener('DOMContentLoaded', () => {  
 // Call the validateForm function when the page loads  
 validateForm();  
});  
```  
  
Note: The above code assumes that you have a `forms.py` file where you define a form class using Django's built-in forms module. You can customize the form fields and their validation logic as per your requirements.  
  
Please let me know if this meets your expectations! If there's anything I missed or need further clarification on, please don't hesitate to ask!