DJANGO VALIDATION

LOÏC BISTUER

- @loic84 on IRC and Twitter
- I work at the World Food Programme
- Django core developer since 2014
- Mostly contribute to Forms and ORM

MAIN CONCERNS WITH VALIDATION

MAIN CONCERNS WITH VALIDATION

1. Enforcement

MAIN CONCERNS WITH VALIDATION

1. Enforcement
2. User Experience

MAIN CONCERNS WITH VALIDATION

1. Enforcement
2. User Experience
3. Performance

MAIN CONCERNS WITH VALIDATION

1. Enforcement
2. User Experience
3. Performance
4. Convenience

MAIN CONCERNS WITH VALIDATION

1. Enforcement
2. User Experience
3. Performance
4. Convenience

MAIN CONCERNS WITH VALIDATION

1. Enforcement
2. User Experience
3. Performance
4. Convenience

MAIN CONCERNS WITH VALIDATION

1. Enforcement
2. User Experience
3. Performance
4. Convenience

WHERE TO VALIDATE DATA

WHERE TO VALIDATE DATA

1. Frontend

WHERE TO VALIDATE DATA

1. Frontend
    - JavaScript

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks

👍 Good for UX

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks

👍 Good for UX

👍 Works offline

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks

👍 Good for UX

👍 Works offline

👎 Need to keep in sync with server validation

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View

👎 Not designed for the task

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View

👎 Not designed for the task

👎 Easy to circumvent

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer

👍 Designed for the task

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer

👍🏾 Designed for the task

👎 Easy to circumvent

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer

👍🏾 Designed for the task

👎 Easy to circumvent

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
     - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model

👍 Designed for the task

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model

👍🏿 Designed for the task

👎 Doesn’t run by default

ENFORCE MODEL VALIDATION

```
class ValidateModelMixin(object):

    def save(self, *args, **kwargs):
        # Run model validation.
        self.full_clean()

        super(ValidateModelMixin, self).save(*args, **kwargs)
```

WHERE TO VALIDATE DATA

1. Frontend
    - JavaScript
    - HTML5 & Browser
        - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model

👍 Designed for the task

👎 Doesn't run by default

👍 Harder to circumvent if triggered from the `save()` method

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
     - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model

👍 Designed for the task

👎 Doesn't run by default

👍 Harder to circumvent if triggered from the `save()` method

👎 Not always accessible

WHERE TO VALIDATE DATA

1. Frontend
    - JavaScript
    - HTML5 & Browser
        - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model

👍 Designed for the task

👎 Doesn't run by default

👍 Harder to circumvent if triggered from the `save()` method

👎 Not always accessible

👎 Redundant

WHERE TO VALIDATE DATA

1. Frontend
    - JavaScript
    - HTML5 & Browser
        - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model

- 👍 Designed for the task
- 👎 Doesn't run by default
- 👍 Harder to circumvent if triggered from the `save()` method
- 👎 Not always accessible
- 👎 Redundant
- 👎 Breaks expectations

WHERE TO VALIDATE DATA

1. Frontend
    - JavaScript
    - HTML5 & Browser
    - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

👍 Designed for the task

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
   - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

👍 Designed for the task

👍 Always enforced

WHERE TO VALIDATE DATA

1. Frontend
    - JavaScript
    - HTML5 & Browser
        - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

- 👍 Designed for the task
- 👍 Always enforced
- 👍 Performance!

PERFORMANCE OF MODEL VALIDATION

```
def validate_title(title):
    if title == 'boom':
        raise ValidationError('Boom!', code='boom')

class Article(models.Model):
    title = models.CharField(
        max_length=42,
        validators=[validate_title],
    )
```

PERFORMANCE OF MODEL VALIDATION

```
def validate_title(title):
    if title == 'boom':
        raise ValidationError('Boom!', code='boom')

class Article(models.Model):
    title = models.CharField(
        max_length=42,
        validators=[validate_title],
    )

with transaction.atomic():
    for i in range(1000000):
        article = Article(title=str(i))
        article.full_clean()
        article.save()
```

PERFORMANCE OF MODEL VALIDATION

```
def validate_title(title):
    if title == 'boom':
        raise ValidationError('Boom!', code='boom')

class Article(models.Model):
    title = models.CharField(
        max_length=42,
        validators=[validate_title],
    )

ALTER TABLE article ADD CHECK(title <> 'boom');

Article.objects.bulk_create(
    Article(title=str(i))
        for i in range(1000000)
)
```

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
     - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

👍 Designed for the task

👍 Always enforced

👍 Performance!

👎 Backend specific

DATABASE CHECK CONSTRAINTS

```
class Article(models.Model):
    title = models.CharField(max_length=42)

    class Meta:
        indexes = [
            models.Constraint(
            [F('title') != Value('boom')],
            name='expression_check',
        ),
    ]
```

DATABASE CHECK CONSTRAINTS

```python
class Article(models.Model):
    title = models.CharField(max_length=42)

    class Meta:
        indexes = [
            models.Constraint(
                [F('title') != Value('boom')],
                name='expression_check',
            ),
        ]
```

```
ALTER TABLE article ADD CHECK(title <> 'boom');
```

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
     - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

👍 Designed for the task

👍 Always enforced

👍 Performance!

👎 Backend specific

👎 Harder to write

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
     - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

👍 Designed for the task

👍 Always enforced

👍 Performance!

👎 Backend specific

👎 Harder to write

👎 Harder to audit

WHERE TO VALIDATE DATA

1. Frontend
   - JavaScript
   - HTML5 & Browser
     - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

👍🏾 Designed for the task

👍🏾 Always enforced

👍🏾 Performance!

👎🏾 Backend specific

👎🏾 Harder to write

👎🏾 Harder to audit

👎🏾 Harder to maintain

WHERE TO VALIDATE DATA

1. Frontend
    - JavaScript
    - HTML5 & Browser
    - Native code / frameworks
2. Django View
3. Form / DRF Serializer
4. Model
5. Database

FIELD VALIDATION

FIELD DECLARATION

- Type validation
  - `class Field(object):`
  - `class CharField(Field):`
  - `class IntegerField(Field):`
  - `class DateField(Field):`
  - etc.

FIELD DECLARATION

- Presence validation

```
class Field(object):
    def __init__(self, required=True, ...):
      self.required = required

    ...

class Field(object):
    def __init__(self, blank=False, ...):
      self.blank = blank

    ...
```

FIELD DECLARATION

- Bounds validation

```
class CharField(Field):
    def __init__(self, max_length=None, min_length=None):
        ...
```

```
class IntegerField(Field):
    def __init__(self, max_value=None, min_value=None):
        ...
```

FIELD DECLARATION

- Choice validation

```
# django.forms.fields

class ChoiceField(forms.Field):
    def __init__(self, choices=()):
        ...

# django.db.models.fields

class Field(object):
    def __init__(self, choices=()):
        ...
```

FIELD DECLARATION

- Format validation

```
class RegexField(CharField):
    def __init__(self, regex, ...):
        ...
```

```
class DateField(Field):
    def __init__(self, input_formats, ...)
        ...
```

FIELD DECLARATION

- Format validation

```
class RegexField(CharField):
    def __init__(self, regex, ...):
        ...
```

```
class DateField(Field):
    def __init__(self, input_formats, ...)
        ...

class EmailField(CharField):
    default_validators = [validate_email]

class SlugField(CharField):
    default_validators = [validate_slug]
```

FIELD DECLARATION

- Uniqueness validation

```
class Field(object):
    def __init__(self,
                 unique=False,
                 unique_for_date=None,
                 unique_for_month=None,
                 unique_for_year=None, ...):
        ...
```

FIELD DECLARATION

- Field validators

```
class Field(object):
    default_validators = []

    def __init__(self, validators=(), ...):
        self.validators = list(itertools.chain(
            self.default_validators, validators))
```

FUNCTION-BASED VALIDATOR

```
def validate_even(value):
    if value % 2 != 0:
        raise ValidationError(
            _('%(value)s is not an even number'),
            params={'value': value},
        )

IntegerField(validators=[validate_even])
```

CLASS-BASED VALIDATOR

```
class MultipleOf(object):
    def __init__(self, base):
        self.base = base

    def __call__(self, value):
        if value % self.base != 0:
            raise ValidationError(
                _('Field must be a multiple of %(base)d.'),
                params={'base': self.base, 'value': value},
            )

IntegerField(validators=[MultipleOf(42)])
```

CLASS-BASED VALIDATOR

```
from django.utils.deconstruct import deconstructible

@deconstructible
class MultipleOf(object):
    def __init__(self, base):
        self.base = base

    def __call__(self, value):
        if value % self.base != 0:
            raise ValidationError(
                _('Field must be a multiple of %(base)d.'),
                params={'base': self.base, 'value': value},
            )

    def __eq__(self, other):
        return (
            isinstance(other, self.__class__) and
            self.base == other.base
        )
```

PARTIAL-BASED VALIDATOR

```
import functools

def multiple_of(base, value):
    if value % base != 0:
        raise ValidationError(
            _('Field must be a multiple of %(base)d.'),
            params={'base': base, 'value': value},
        )

multiple_of_42 = functools.partial(multiple_of, 42)

IntegerField(validators=[multiple_of_42])
```

FIELD DECLARATION

- Error messages

```
class Field(object):
    def __init__(self, error_messages=None, ...):
        ...
```

FIELD DECLARATION

- Error messages

```
class Field(object):
    default_error_messages = {
        'required': _('This field is required.'),
    }
```

```
def __init__(self, error_messages=None, ...):
   messages = {}
    for c in reversed(self.__class__.__mro__):
        messages.update(
            getattr(c, 'default_error_messages', {}))
    messages.update(error_messages or {})
    self.error_messages = messages

    ...
```

FIELD VALIDATION CYCLE

FIELD VALIDATION CYCLE

- `Field.clean()`

```
def clean(self, value):
    value = self.to_python(value)
    self.validate(value)
    self.run_validators(value)
    return value
```

FIELD VALIDATION CYCLE

- `Field.clean()`

```
def clean(self, value):
    value = self.to_python(value)
    self.validate(value)
    self.run_validators(value)
    return value
```

FIELD VALIDATION CYCLE

- `Field.clean()`

```
def clean(self, value):
    value = self.to_python(value)
    self.validate(value)
    self.run_validators(value)
    return value
```

FIELD VALIDATION CYCLE

- Field.validate()

```
def validate(self, value):
    if value in self.empty_values and self.required:
        raise ValidationError(
            self.error_messages['required'],
            code='required',
        )
```

FIELD VALIDATION CYCLE

- `Field.clean()`

```
def clean(self, value):
    value = self.to_python(value)
    self.validate(value)
    self.run_validators(value)
    return value
```

FIELD VALIDATION CYCLE

- `Field.run_validators()`

```
def run_validators(self, value):
    if value in self.empty_values:
        return
    errors = []
    for validator in self.validators:
        try:
            validator(value)
        except ValidationError as e:
            if hasattr(e, 'code') and e.code in self.error_messages:
                e.message = self.error_messages[e.code]
            errors.extend(e.error_list)

    if errors:
        raise ValidationError(errors)
```

FIELD VALIDATION CYCLE

- `Field.run_validators()`

```
def run_validators(self, value):
    if value in self.empty_values:
        return
    errors = []
    for validator in self.validators:
        try:
            validator(value)
        except ValidationError as e:
            if hasattr(e, 'code') and e.code in self.error_messages:
                e.message = self.error_messages[e.code]
            errors.extend(e.error_list)

    if errors:
        raise ValidationError(errors)
```

FIELD VALIDATION CYCLE

- `Field.clean()`

```
def clean(self, value):
    value = self.to_python(value)
    self.validate(value)
    self.run_validators(value)
    return value
```

FIELD VALIDATION CYCLE

- `Field.clean()`

```
class CharField(Field):
    def __init__(self, max_length, min_length, ...):
        ...
        self.max_length = max_length
        self.min_length = min_length
        self.validators.append(
            validators.MinLengthValidator(min_length))
        self.validators.append(
            validators.MaxLengthValidator(max_length))
```

VALIDATION ERROR

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):

    def __init__(self, message, code=None, params=None):
        ...
```

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):

    def __init__(self, message, code=None, params=None):
        ...

ValidationError("Foo")
```

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):

    def __init__(self, message, code=None, params=None):
        ...

ValidationError("Foo")

ValidationError(["Foo", "Bar"])
```

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):

    def __init__(self, message, code=None, params=None):
        ...

ValidationError("Foo")

ValidationError(["Foo", "Bar"])

ValidationError({
    "field1": ["Foo", "Bar"],
    "field2": "Baz",
})
```

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):

    def __init__(self, message, code=None, params=None):
        ...

ValidationError([
    "Bar",
     ValidationError("Foo"),
])
```

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):

    def __init__(self, message, code=None, params=None):
        ...

ValidationError({
    "field1": ValidationError([
        ValidationError("Foo"),
        ValidationError("Bar"),
    ]),
})
```

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):
    def __init__(self, message, code=None, params=None):
        ...

        if isinstance(message, dict):
            self.error_dict = {...}

        elif isinstance(message, list):
            self.error_list = [...]

        else:
            self.message = message
            self.code = code
            self.params = params
            self.error_list = [self]
```

ANATOMY OF VALIDATION ERROR

```
class ValidationError(Exception):
    def __init__(self, message, code=None, params=None):
        ...

        if isinstance(message, dict):
            self.error_dict = {...}

        elif isinstance(message, list):
            self.error_list = [...]

        else:
            self.message = message
            self.code = code
            self.params = params
            self.error_list = [self]
```

RAISING VALIDATION ERRORS

```
raise ValidationError("Invalid value: %s" % 42)
```

RAISING VALIDATION ERRORS

```
from django.utils.translation import ugettext as _

raise ValidationError(_("Invalid value: %s") % 42)
```

RAISING VALIDATION ERRORS

```
from django.utils.translation import ugettext as _

raise ValidationError(
    _("Invalid value: %s") % 42,
    code='invalid',
)
```

RAISING VALIDATION ERRORS

```
from django.utils.translation import ugettext as _

raise ValidationError(
    _("Invalid value: %s"),
    code='invalid',
    params=42,
)
```

RAISING VALIDATION ERRORS

```
from django.utils.translation import ugettext as _

raise ValidationError(
    _("Invalid value: %(value)s"),
    code='invalid',
    params={'value': 42},
)
```

RAISING VALIDATION ERRORS

```
from django.utils.translation import ugettext as _

raise ValidationError(
    message=_("%(model_name)s with this %(field_labels)s "
              "already exists."),
    code='unique_together',
    params={
        'model': self,
        'model_class': model_class,
        'model_name': verbose_name),
        'unique_check': unique_check,
    },
)
```

FORM VALIDATION

TRIGGERING FORM VALIDATION

- `Form.is_valid()`

```
def is_valid(self):
    return self.is_bound and not self.errors
```

TRIGGERING VALIDATION

- `Form.is_valid()`

```
def is_valid(self):
    return self.is_bound and not self.errors
```

- `Form.errors`

```
@property
def errors(self):
    if self._errors is None:
        self.full_clean()
    return self._errors
```

TRIGGERING VALIDATION

- Form.is_valid()

```
def is_valid(self):
    return self.is_bound and not self.errors
```

- Form.errors

```
@property
def errors(self):
    if self._errors is None:
        self.full_clean()
    return self._errors
```

- Form.full_clean()

FORM VALIDATION CYCLE

- `Form.full_clean()`

```
def full_clean(self):
    self._errors = ErrorDict()
    self.cleaned_data = {}

    self._clean_fields()
    self._clean_form()
    self._post_clean()
```

FORM VALIDATION CYCLE

- `Form.full_clean()`

```
def full_clean(self):
    self._errors = ErrorDict()
    self.cleaned_data = {}

    self._clean_fields()
    self._clean_form()
    self._post_clean()
```

FORM VALIDATION CYCLE

- `Form.full_clean()`

```
def full_clean(self):
    self._errors = ErrorDict()
    self.cleaned_data = {}
    self._clean_fields()
    self._clean_form()
    self._post_clean()
```

FORM VALIDATION CYCLE

- Form._clean_fields()

```
def _clean_fields(self):
    for name, field in self.fields.items():
        value = field.widget.value_from_datadict(
            self.data, self.files, self.add_prefix(name))
```

FORM VALIDATION CYCLE

- Form._clean_fields()

```
def _clean_fields(self):
    for name, field in self.fields.items():
        value = field.widget.value_from_datadict(
            self.data, self.files, self.add_prefix(name))
        try:
            value = field.clean(value)
            self.cleaned_data[name] = value
```

FORM VALIDATION CYCLE

- `Form._clean_fields()`

```
def _clean_fields(self):
    for name, field in self.fields.items():
        value = field.widget.value_from_datadict(
            self.data, self.files, self.add_prefix(name))

        try:
            value = field.clean(value)
            self.cleaned_data[name] = value

            if hasattr(self, 'clean_%s' % name):
                value = getattr(self, 'clean_%s' % name)()
                self.cleaned_data[name] = value
```

FORM VALIDATION CYCLE

- Form._clean_fields()

```
def _clean_fields(self):
    for name, field in self.fields.items():
        value = field.widget.value_from_datadict(
            self.data, self.files, self.add_prefix(name))

        try:
            value = field.clean(value)
            self.cleaned_data[name] = value

            if hasattr(self, 'clean_%s' % name):
                value = getattr(self, 'clean_%s' % name)()
                self.cleaned_data[name] = value

        except ValidationError as e:
            self.add_error(name, e)
```

FORM VALIDATION CYCLE

- `Form.full_clean()`

```
def full_clean(self):
    self._errors = ErrorDict()
    self.cleaned_data = {}

    self._clean_fields()
    self._clean_form()
    self._post_clean()
```

FORM VALIDATION CYCLE

- Form._clean_form()

```
def _clean_form(self):
    try:
        cleaned_data = self.clean()
    except ValidationError as e:
        self.add_error(None, e)
    else:
        if cleaned_data is not None:
            self.cleaned_data = cleaned_data
```

FORM VALIDATION CYCLE

- `Form.full_clean()`

```
def full_clean(self):
    self._errors = ErrorDict()
    self.cleaned_data = {}
    self._clean_fields()
    self._clean_form()
    self._post_clean()
```

- `Form._post_clean()`

```
def _post_clean(self):
    pass
```

FORM VALIDATION UTILS

- `Form.add_error()`

FORM VALIDATION UTILS

- Form.add_error()

```
def add_error(self, field, error):
    if not isinstance(error, ValidationError):
        error = ValidationError(error)
```

FORM VALIDATION UTILS

- `Form.add_error()`

```
def add_error(self, field, error):
    if not isinstance(error, ValidationError):
        error = ValidationError(error)
    if hasattr(error, 'error_dict'):
        error = error.error_dict
    else:
        error = {field or NON_FIELD_ERRORS: error.error_list}
```

FORM VALIDATION UTILS

- `Form.add_error()`

```
def add_error(self, field, error):
    if not isinstance(error, ValidationError):
        error = ValidationError(error)
    if hasattr(error, 'error_dict'):
        error = error.error_dict
    else:
        error = {field or NON_FIELD_ERRORS: error.error_list}

    for field, error_list in error.items():
        if field not in self.errors:
            self._errors[field] = self.error_class()
        self._errors[field].extend(error_list)
```

FORM VALIDATION UTILS

- `Form.add_error()`

```
def add_error(self, field, error):
    if not isinstance(error, ValidationError):
        error = ValidationError(error)
    if hasattr(error, 'error_dict'):
        error = error.error_dict
    else:
        error = {field or NON_FIELD_ERRORS: error.error_list}

    for field, error_list in error.items():
        if field not in self.errors:
            self._errors[field] = self.error_class()
        self._errors[field].extend(error_list)

        if field in self.cleaned_data:
            del self.cleaned_data[field]
```

FORM VALIDATION UTILS

- `Form.add_error()`

```
def add_error(self, field, error):
    if not isinstance(error, ValidationError):
        error = ValidationError(error)
    if hasattr(error, 'error_dict'):
        error = error.error_dict
    else:
        error = {field or NON_FIELD_ERRORS: error.error_list}

    for field, error_list in error.items():
        if field not in self.errors:
            self._errors[field] = self.error_class()
        self._errors[field].extend(error_list)

        if field in self.cleaned_data:
            del self.cleaned_data[field]
```

FORM VALIDATION UTILS

- Form.add_error()

```
def add_error(self, field, error):
    if not isinstance(error, ValidationError):
        error = ValidationError(error)
    if hasattr(error, 'error_dict'):
        error = error.error_dict
    else:
        error = {field or NON_FIELD_ERRORS: error.error_list}

    for field, error_list in error.items():
        if field not in self.errors:
            self._errors[field] = self.error_class()
        self._errors[field].extend(error_list)

        if field in self.cleaned_data:
            del self.cleaned_data[field]
```

FORM VALIDATION UTILS

- `Form.add_error()`

```
def add_error(self, field, error):
    if not isinstance(error, ValidationError):
        error = ValidationError(error)
    if hasattr(error, 'error_dict'):
        error = error.error_dict
    else:
        error = {field or NON_FIELD_ERRORS: error.error_list}

    for field, error_list in error.items():
        if field not in self.errors:
            self._errors[field] = self.error_class()
        self._errors[field].extend(error_list)

        if field in self.cleaned_data:
            del self.cleaned_data[field]
```

FORM VALIDATION UTILS

- Form.errors

```
class ErrorDict(dict):

    def as_ul(self):
        ...

    def as_text(self):
        ...

    def as_data(self):
        ...

    def as_json(self, escape_html=False):
        ...
```

FORM VALIDATION UTILS

- `Form.errors['field_name']`

```
class ErrorList(list):

    def as_ul(self):
        ...

    def as_text(self):
        ...
```

FORM VALIDATION UTILS

- `Form.errors['field_name']`

```
class ErrorList(UserList, list):

    def __contains__(self, item):
        return item in list(self)

    def __eq__(self, other):
        return list(self) == other

    def __ne__(self, other):
        return list(self) != other

    def __getitem__(self, i):
        error = self.data[i]
        if isinstance(error, ValidationError):
            return list(error)[0]
        return force_text(error)
```

FORM VALIDATION UTILS

- `Form.errors['field_name']`

```
class ErrorList(UserList, list):

    def as_data(self):
        return ValidationError(self.data).error_list

    def get_json_data(self):
        errors = []
        for error in self.as_data():
            errors.append({
                'message': list(error)[0],
                'code': error.code or '',
            })
        return errors

    def as_json(self):
        return json.dumps(self.get_json_data())
```

FORM VALIDATION UTILS

- Form.has_error()

```
def has_error(self, field, code=None):
    if code is None:
        return field in self.errors
    if field in self.errors:
        for error in self.errors.as_data()[field]:
            if error.code == code:
                return True

    return False
```

FORM VALIDATION UTILS

```
class ValidationError(Exception):

    def __str__(self):
        if hasattr(self, 'error_dict'):
            return repr(dict(self))
        return repr(list(self))
```

```
>>> str(ValidationError("Some error..."))
"[u'Some error...']"
```

FORM VALIDATION UTILS

```
class ValidationError(Exception):

    def __str__(self):
        if hasattr(self, 'error_dict'):
            return repr(dict(self))
        elif not hasattr(self, 'code'):
            return repr(list(self))
        else:
            return list(self)[0]

>>> str(ValidationError("Some error..."))
"Some error..."
```

MODEL VALIDATION

MODEL VALIDATION INSPIRED BY DJANGO'S FORM VALIDATION.

Django 1.2 release notes

TRIGGERING MODEL VALIDATION

- `Model.full_clean()`

```
try:
    article.full_clean()
except ValidationError:
    is_valid = False
else:
    is_valid = True
```

MODEL VALIDATION CYCLE

- `Model.full_clean()`

```
def full_clean(self, exclude=None, validate_unique=True):
    errors = {}

    ...

    if errors:
        raise ValidationError(errors)
```

MODEL VALIDATION CYCLE

- `Model.full_clean()`

```
def full_clean(self, exclude=None, validate_unique=True):
    ...

    try:
        self.clean_fields(exclude=exclude)
    except ValidationError as e:
        errors = e.update_error_dict(errors)
```

MODEL VALIDATION CYCLE

- Model.full_clean()

```
def full_clean(self, exclude=None, validate_unique=True):
    ...

    try:
        self.clean()
    except ValidationError as e:
        errors = e.update_error_dict(errors)
```

MODEL VALIDATION CYCLE

- `Model.full_clean()`

```
def full_clean(self, exclude=None, validate_unique=True):
    ...

    try:
        self.clean()
    except ValidationError as e:
        errors = e.update_error_dict(errors)
```

- `Model.clean()`

```
def clean(self):
    pass
```

MODEL VALIDATION CYCLE

- Model.full_clean()

```
def full_clean(self, exclude=None, validate_unique=True):
    ...

    if validate_unique:
        for name in errors.keys():
            if name not in exclude:
                exclude.append(name)

    try:
        self.validate_unique(exclude=exclude)
    except ValidationError as e:
        errors = e.update_error_dict(errors)
```

MODEL VALIDATION CYCLE

- `Field.unique_for_date`

```
ALTER TABLE article
ADD EXCLUDE USING GIST (
    title WITH =,
    daterange(pub_date, pub_date, '[]') WITH &&
);
```

MODEL VALIDATION CYCLE

- `Field.unique_for_month`

```
ALTER TABLE article
ADD EXCLUDE USING GIST (
    title WITH =,
    daterange(
        date_trunc('month', pub_date::timestamp)::date,
        (date_trunc('month', pub_date::timestamp)
             + '1month - 1day')::date,
        '[]'
    ) WITH &&
);
```

MODEL VALIDATION CYCLE

- `Field.unique_for_year`

```
ALTER TABLE article
ADD EXCLUDE USING GIST (
    title WITH =,
    daterange(
        date_trunc('year', pub_date::timestamp)::date,
        (date_trunc('year', pub_date::timestamp)
             + '1year - 1day')::date,
        '[]'
    ) WITH &&
);
```

MODEL VALIDATION CYCLE

- `Model.full_clean()`

```
def full_clean(self, exclude=None, validate_unique=True):
    ...

    if validate_unique:
        for name in errors.keys():
            if name not in exclude:
                exclude.append(name)

    try:
        self.validate_unique(exclude=exclude)
    except ValidationError as e:
        errors = e.update_error_dict(errors)
```

Model Form Validation

ModelForm VALIDATION CYCLE

- ModelForm._post_clean()

```
def _post_clean(self):
    self.instance = construct_instance(...)
```

ModelForm validation cycle

- ModelForm._post_clean()

```
def _post_clean(self):
    self.instance = construct_instance(...)

    exclude = self._get_validation_exclusions()
```

ModelForm validation cycle

- ModelForm._post_clean()

```
def _post_clean(self):
    self.instance = construct_instance(...)

    exclude = self._get_validation_exclusions()

    try:
        self.instance.full_clean(
            exclude=exclude, validate_unique=False)
    except ValidationError as e:
        self._update_errors(e)
```

ModelForm validation cycle

- ModelForm._post_clean()

```
def _post_clean(self):
    self.instance = construct_instance(...)

    exclude = self._get_validation_exclusions()

    try:
        self.instance.full_clean(
            exclude=exclude, validate_unique=False)
    except ValidationError as e:
        self._update_errors(e)
```

ModelForm validation cycle

- ModelForm._post_clean()

```
def _post_clean(self):
    self.instance = construct_instance(...)

    exclude = self._get_validation_exclusions()

    try:
        self.instance.full_clean(
            exclude=exclude, validate_unique=False)
    except ValidationError as e:
        self._update_errors(e)

    if self._validate_unique:
        self.validate_unique()
```

CLOSING WORDS

THANK YOU