Re2o et les cotisations

Rézo Rennes et Metz Fédérés

Klafyvel

10/07/2021

Sommaire

Contrainte principale :
Ne pas faire de la merde avec les sous des adhérents.

Les principales structures de données

L'objet Facture

- Représente une facture;

```
class Facture(BaseInvoice):
    user = models.ForeignKey("users.User",...)
    paiement = models.ForeignKey("Paiement", ...)
    banque = models.ForeignKey("Banque", ...)
    cheque = models.CharField(...)
    valid = models.BooleanField(...)
    control = models.BooleanField(...)
```

L'objet Facture

- Représente une facture;
- Visible par les adhérents;

```
class Facture(BaseInvoice):
    user = models.ForeignKey("users.User",...)
    paiement = models.ForeignKey("Paiement", ...)
    banque = models.ForeignKey("Banque", ...)
    cheque = models.CharField(...)
    valid = models.BooleanField(...)
    control = models.BooleanField(...)
```

L'objet Facture

- Représente une facture;
- Visible par les adhérents;
- Est censé être controlée par les trésoriers.

```
class Facture(BaseInvoice):
    user = models.ForeignKey("users.User",...)
    paiement = models.ForeignKey("Paiement", ...)
    banque = models.ForeignKey("Banque", ...)
    cheque = models.CharField(...)
    valid = models.BooleanField(...)
    control = models.BooleanField(...)
```

L'objet Vente

- Représente une vente de `number` articles(s) au prix unitaire `prix`;
- Stocke une information sur la durée d'adhésion / cotisation.

```
class Vente(RevMixin, AclMixin, models.Model):
    facture = models.ForeignKey("BaseInvoice", ...)
    number = models.IntegerField(...)
    name = models.CharField(...)
    prix = models.DecimalField(...)
    duration_connection = models.PositiveIntegerFiel
    duration_days_connection = models.PositiveIntege
    duration_membership = models.PositiveIntegerFiel
    duration_days_membership = models.PositiveIntege
```

L'objet Article

- Une recette pour créer des `Ventes`;
- On ne lie pas directement les ventes aux articles, on peut supprimer les articles sans contrainte.

```
class Article(RevMixin, AclMixin, models.Model):
    name = models.CharField(...)
    prix = models.DecimalField(...)
    duration_membership = models.PositiveIntegerFiel
    duration_days_membership = models.PositiveIntege
    duration_connection = models.PositiveIntegerFiel
    duration_days_connection = models.PositiveIntege
    need_membership = models.BooleanField(...)
    type_user = models.CharField(...)
    available_for_everyone = models.BooleanField(...
```

L’objet Cotisation

- Information sur les adhésions / cotisations des utilisateurs;
- C’est ce qui est utilisé par l’objet `User` pour savoir si un utilisateur est adhérent.

```
class Cotisation(RevMixin, AclMixin, models.Model):
    vente = models.OneToOneField("Vente",...)
    date_start_con = models.DateTimeField(...)
    date_end_con = models.DateTimeField(...)
    date_start_memb = models.DateTimeField(...)
    date_end_memb = models.DateTimeField(...)
```

L’objet Paiement

- Enregistre un moyen de paiement;
- On peut y brancher des méthodes de paiement personnalisées.

```
class Paiement(RevMixin, AclMixin, models.Model):
    moyen = models.CharField(...)
    available_for_everyone = models.BooleanField(...)
    is_balance = models.BooleanField(...)
```

La vue new_facture

Vue assez classique de formulaire Django.

```python
@login_required
@can_create(Facture)
@can_edit(User)
def new_facture(request, user, userid):
    """De la doc..."""
    invoice = Facture(user=user)
    article_list = Article.objects.filter(...)
    invoice_form = FactureForm(...)
    article_formset = formset_factory(SelectArticleForm)(...)
    if invoice_form.is_valid() and article_formset.is_valid():
        # On verra après

    # Gestion du solde
    p = Paiement.objects.filter(is_balance=True)
    if len(p) and p[0].can_use_payment(request.user):
        balance = user.solde
    else:
        balance = None

    return form(...)
```

À l'intérieur de ce if

```
new_invoice_instance = invoice_form.save(commit=False)
articles = article_formset
# Check if at least one article has been selected
if any(art.cleaned_data for art in articles):
    # Encore un peu de patience, slide suivante
else :
    messages.error(request, _("You need to choose at least one article."))
```

Roulement de tambours...

```python
# Building a purchase for each article sold
purchases = []
total_price = 0
for art_item in articles:
    if art_item.cleaned_data:
        article = art_item.cleaned_data["article"]
        quantity = art_item.cleaned_data["quantity"]
        total_price += article.prix * quantity
        new_purchase = Vente(...)
        purchases.append(new_purchase)
p = find_payment_method(new_invoice_instance.paiement)
if hasattr(p, "check_price"):
    price_ok, msg = p.check_price(total_price, user)
    invoice_form.add_error(None, msg)
else:
    price_ok = True
```

```python
if price_ok:
    new_invoice_instance.save()
    # Saving purchases so the invoice can find them.
    # will modify them after being validated to put t
    for p in purchases:
        p.facture = new_invoice_instance
        p.save()
    return new_invoice_instance.paiement.end_payment(
        new_invoice_instance, request
    )
```

Le retour du modèle Paiement

Pour finir un paiement,

- Si une méthode de paiement personnalisée existe et que la transisition n'est pas explicitement interdite, on branche;
- Sinon on valide la facture, on affiche un message et on redirige.

```python
def end_payment(self, invoice, request,
        use_payment_method=True):
    payment_method = find_payment_method(self)
    if payment_method is not None \
        and use_payment_method:
        return payment_method.end_payment(invoice,
            request)

    # So make this invoice valid, trigger send mail
    invoice.valid = True
    invoice.save(request=request)

    # Du code pas intéressant
    return redirect(...)
```

Un exemple simple : FreePayment

```python
class FreePayment(PaymentMethodMixin, models.Model):
    """..."""

    class Meta:
        verbose_name = _("Free payment")

    payment = models.OneToOneField(
        Paiement,
        on_delete=models.CASCADE,
        related_name="payment_method_free",
        editable=False,
    )
```

```python
    def end_payment(self, invoice, request):
        """Ends the payment normally.
        """
        return invoice.paiement.end_payment(invoice,
            request,
            use_payment_method=False
        )

    def check_price(self, price, user, *args, **kwarg
        """Checks that the price meets the requiremen
        to be paid with user balance.
        """
        return (
            price == 0,
            _("You can't pay this invoice for free.")
        )
```

Un exemple intéressant : ComnPay

Déroulé

Déroulé

- Le end_payment construit une requête spéciale pour que l'utilisateur l'envoie vers comnpay. La facture est invalide;

Déroulé

- Le end_payment construit une requête spéciale pour que l'utilisateur l'envoie vers comnpay. La facture est invalide;
- L'utilisateur fait son affaire avec ComnPay;

Déroulé

- Le end_payment construit une requête spéciale pour que l'utilisateur l'envoie vers comnpay. La facture est invalide;
- L'utilisateur fait son affaire avec ComnPay;
- ComnPay notifie re2o que le paiement a été effectué. La facture est validée;

Déroulé

- Le end_payment construit une requête spéciale pour que l'utilisateur l'envoie vers comnpay. La facture est invalide;
- L'utilisateur fait son affaire avec ComnPay;
- ComnPay notifie re2o que le paiement a été effectué. La facture est validée;
- L'utilisateur est redirigé vers re2o.

Un petit tour dans le code.

Conclusion

Conclusion

- Le système de paiement est modulable;
- Mais du coup il est complexe;
- Pour intégrer HelloAsso il faudrait s'inspirer du système ComnPay.