django_saas Documentation

Release 0.3.6-dev

DjaoDjin inc.

Jan 31, 2019

Contents

djaodjin-saas is a Django application that implements the logic to support subscription-based Sofware-as-a-Service businesses.

Major Features:

- Separate billing profiles and authenticated users
- Double entry book keeping ledger
- Flexible security framework

Contents:

CHAPTER 1

Getting Started

1.1 Installation and configuration

First download and install the latest version of djaodjin-saas into your Python virtual environment.

```
$ pip install djaodjin-saas
```

Edit your project urls.py to add the djaojdin-saas urls

```
urlpatterns += [
    url(r'^', include('saas.urls')),
]
```

Edit your project settings.py to add saas into the `INSTALLED_APPS` and a SAAS configuration block

```
INSTALLED_APPS = (
    ...
    'saas'
)

SAAS = {
    'PROCESSOR': {
        'BACKEND': 'saas.backends.stripe_processor.StripeBackend',
        'PRIV_KEY': "_Your_Stripe_Private_Key_",
        'PUB_KEY': "_Your_Stripe_Public_Key_",
    }
}
```

Various *payment processor backends* are available.

The latest versions of django-restframework implement paginators disconnected from parameters in views (i.e. no more paginate_by). You will thus need to define `PAGE_SIZE` in your settings.py

```
REST_FRAMEWORK = {
    'PAGE_SIZE': 25,
```

```
    'DEFAULT_PAGINATION_CLASS':
        'rest_framework.pagination.PageNumberPagination',
}
```

There is no access policies by default on the djaodjin-saas URLs. It is thus your responsability to add the appropriate decorators to restrict which users can access which URL. A set of common decorators in Software-as-a-Service setups is provided as part of the *Flexible Security Framework*.

1.2 Setting up a Software-as-a-Service site

To setup a site with three plans (basic, premium and ultimate), we will create an `Organization` for the payment processor and an `Organization` for the provider / broker together with three `Plan` that belong to the provider (see *database schema*). We will also need to create an default `Agreement` for the terms of use of the site.

Example fixtures:

```
[{
    "fields": {
      "slug": "stripe",
      "full_name": "Stripe",
      "created_at": "2016-01-01T00:00:00-09:00",
      "processor": 1,
      "is_active": 1
    },
    "model": "saas.Organization", "pk": 1
},
{
    "fields": {
      "slug": "terms-of-use",
      "title": "Terms Of Use",
      "modified": "2016-01-01T00:00:00-09:00"
    },
    "model": "saas.agreement", "pk": 1
},
{
    "fields": {
      "slug": "cowork",
      "full_name": "ABC Corp.",
      "created_at": "2016-01-01T00:00:00-09:00",
      "email": "support@localhost.localdomain",
      "phone": "555-555-5555",
      "street_address": "1 ABC loop",
      "locality":  "San Francisco",
      "region": "CA",
      "postal_code": "94102",
      "country": "US",
      "processor": 1,
      "is_provider": 1,
      "is_active": 1
    },
    "model": "saas.Organization", "pk": 2
},
{
  "fields": {
    "slug": "basic",
    "title": "Basic",
```

```
    "created_at": "2016-01-01T00:00:00-09:00",
    "setup_amount": 0,
    "period_amount": 2000,
    "interval": 4,
    "description": "Basic Plan",
    "organization" : 2,
    "is_active": 1
  },
  "model" : "saas.Plan", "pk": 1
},
{
  "fields": {
    "slug": "premium",
    "title": "Premium",
    "created_at":"2016-01-01T00:00:00-09:00",
    "setup_amount": 0,
    "period_amount": 6900,
    "interval": 4,
    "description": "Premium Plan",
    "organization" : 2,
    "is_active": 1
  },
  "model" : "saas.Plan", "pk": 2
},
{
  "fields": {
    "slug": "ultimate",
    "title": "Ultimate",
    "created_at": "2016-01-01T00:00:00-09:00",
    "setup_amount": 0,
    "period_amount": 8900,
    "interval": 4,
    "description": "Ultimate Plan",
    "organization" : 2,
    "is_active": 1
  },
  "model" : "saas.Plan", "pk": 3
}]
```

1.3 Selling add-ons plans

Subscribers can be subscribed to any number of `Plan`. The cart API and *checkout pipeline* support orders for multiple plans in one payment. All you have to do is thus:

1. Create a new `Plan`
2. Modify the pricing page from a one-click to a shopping cart experience

1.4 Restricting features based on a plan

In decorators.py there is a `requires_paid_subscription` decorator which is part of the *Flexible Security Framework*.

What you would do to allow/deny access to certain features (i.e. URLs) based on the subscribed-to Plan is to decorate the view implementing the feature.

Example:

```
urls.py:

from saas.decorators import requires_paid_subscription
from .views import FeatureView

urlpatterns = [
\.\.\.
    url(r'^(?P<organization>[a-z])/(?P<subscribed_plan>[a-z])/feature/',
        requires_paid_subscription(FeatureView.as_view()), name='feature'),
 \.\.\.
]
```

CHAPTER 2

The Subscription Cycle

The `Subscription` model (see *models*) is the corner stone on which access to the service is authorized. It is also a fundamental block of the *Flexible Security Framework*.

class `saas.models.`**`Subscription`**(**args*, ***kwargs*)

`Subscription` represent a service contract (`Plan`) between two `Organization`, a subscriber and a provider, that is paid by the subscriber to the provider over the lifetime of the subscription.

When `auto_renew` is True, `extend_subscriptions` (usually called from a cron job) will invoice the organization for an additional period once the date reaches currenct end of subscription.

Implementation Note: Even through (organization, plan) should be unique at any point in time, it is a little much to implement with PostgreSQL that for each (organization, plan), there should not be overlapping timeframe [created_at, ends_at[.

The djaodjin-saas logic supports subscribe and unsubscribe to multiple plans at any one time but, by default, will generate credit card charges on a billing cycle. At the time a customer first subscribes to a `Plan`, a billing cycle is established for the `Organization`. From that time on, a customer is billed at the begining of each billing cycle for all services subscribed to.

Thus all subscriptions are charged as a single credit card charge through the payment processor (see *backends available*).

In normal business operations, service is available as soon as customer subscribes; service becomes unavailable at the end of a billing cycle.

Whenever potential fraud is detected, that is a customer's card is denied N number of times or a chargeback is created, a customer is locked out immediately.

2.1 Renewals

Plans can be configured to create one-time-only subcriptions, repeat subscriptions or auto-renewal subscriptions.

Plan.renewal_type	Description	Example
ONE_TIME	The provider wishes to upgrade subscriber to a different plan when the period ends.	A 30-day trial plan
REPEAT	The service is provided on request. It requires the customer to explicitely take action in the product.	Car rental with pre-negotiatied rates
AUTO_RENEW	The service is provided continuoulsly until canceled.	web hosting

When a `Subscription` for a `Plan` where `renewal_type == AUTO_RENEW` is created, `Subscription.auto_renew` is set to `True` to tell the *periodic renewal task* to automatically extends the subscription for one more period in the day before it ends.

The function `saas.renewals.extend_subscriptions(at_time)` iterates through all active subscription that ends in a day of `at_time` and which have `auto_renew == True`, and records a subscription order in the `Transaction` *ledger* for each of them.

Later, `saas.renewals.create_charges_for_balance` calls the *processor backend* to create an actual charge for the balance due by an `Organization`.

When a subscription is canceled, `auto_renew` is set to `False`. `ends_at` is set to the current date/time (cancel now) or left unchanged (cancel at end of period).

2.2 Expiration notices

Expiration notices (if any) are triggered 90, 60, 30, 15 and 1 day(s) before a subscription ends. This can be changed through the EXPIRE_NOTICE_DAYS in settings.py:

```
SAAS = {
    "EXPIRE_NOTICE_DAYS": [90, 60, 30, 15, 1]
}
```

Different types of expiration notices are sent based on the value of `Plan.renewal_type`, `Subscription.auto_renew`, and the subscriber payment method.

A subscriber payment method (i.e. `Organization`) can be either be absent, valid or expired. The payment method status is determined at the time a renewal `Charge` would be created.

There would be 18 (3 * 2 * 3) combinations of expiration notices if a few combinations could not happen.

- `Subscription.auto_renew` shall be false when `Plan.renewal_type` is

`ONE_TIME` because it does not make sense to have a subscription that renews if the plan states the subscription length is fixed to one period.

- `Subscription.auto_renew` shall also be set to false when `Plan.renewal_type` is `REPEAT`. Without adding this constraint, there is no direct means to detect subscriptions to a repeat plan that would be "canceled".

We assume here that cancelation of repeat plans is uninteresting (if either possible) and state that cancelation only makes sense with plans having an auto-renew behavior. Thus, instead of adding another state variable, we use `Plan.renewal == AUTO_RENEW && Subscription.auto_renew == false` to detect cancelations of auto-renewals.

The signals triggered by `saas.renewals.trigger_expiration_notices` are such for the available combinations

Plan	Subscription	Organization payment method	ACTION
ONE_TIME	auto_renew=false	ABSENT	Upgrades notice
ONE_TIME	auto_renew=false	VALID	Upgrades notice
ONE_TIME	auto_renew=false	EXPIRED	Upgrades notice
REPEAT	auto_renew=false	ABSENT	Expiration notice
REPEAT	auto_renew=false	VALID	Expiration notice
REPEAT	auto_renew=false	EXPIRED	Expiration notice
AUTO_RENEW	auto_renew=false	ABSENT	None (canceled)
AUTO_RENEW	auto_renew=false	VALID	None (canceled)
AUTO_RENEW	auto_renew=false	EXPIRED	None (canceled)
AUTO_RENEW	auto_renew=true	ABSENT	Attach payment method notice
AUTO_RENEW	auto_renew=true	VALID	None
AUTO_RENEW	auto_renew=true	EXPIRED	Payment method will expire notice

CHAPTER 3

Placing on Order

A `Subscription` is created when a `Plan` is selected and paid for. As simple as it sounds, there are many variants to implement the previous sentence.

3.1 Basic Pipeline

In the most basic pipeline, a user becomes a subscriber in 2 steps:

1. Click a `Plan` on the /pricing/ page
2. Submit credit card information

3.2 Pipeline with Multiple Periods Paid in Advance

It is always better to receive more cash up-front so an intermediate step is often introduced to enable to pre-pay multiple periods in advance at a discount.

1. Click a `Plan` on the /pricing/ page
2. Pick the number of periods paid in advance
3. Submit credit card information

3.3 Pipeline with Multiple Products in Shopping Cart

A growing business often offers multiple products (i.e. `Plan`) that are cross-saled to new and existing customers. In that case, the /pricing/ page is replaced by a more complex catalog. Cart and checkout concepts appear.

1. Add multiple `Plan` to a user cart
2. Click a checkout button

3. Submit credit card information

3.4 Pipeline to Bulk Subscribe Third-parties

Software-as-a-Service that target businesses (B2B) and/or other kind of structured groups almost always require one entity to pay for subscriptions on behalf of users that belong to it. This can be implemented through managers (or custom roles) to the subscribed entity (see *Security*) or through the entity buying multiple subscriptions in bluk, on behalf of its users. The later case requires an extra step to subscribe those third parties.

1. Click a `Plan` on the /pricing/ page
2. Enter email address of users to subscribe
3. Submit credit card information

3.5 Full Pipeline

Of course, all of the above cases can be combined together, which leads to a full pipeline as such:

1. Add multiple `Plan` to a user cart
2. Click a checkout button
3. Pick the number of periods paid in advance
4. Enter email address of users to subscribe
5. Submit credit card information

3.5.1 Django Views

The two primary views to place an order are `CartView` and `BalanceView`. `CartView` is used to implement the checkout and place a new order while `BalanceView` is used to pay an `Organization` balance due, either because a charge wasn't sucessful and/or the provider implements a subscribe-pay-later policy.

1. `CartView` for items in the cart, create new subscriptions or pay in advance.
2. `BalanceView` for subscriptions with balance dues

class `saas.views.billing.`**`CartBaseView`**(***kwargs*)

The main pupose of `CartBaseView` is generate an list of invoicables from `CartItem` records associated to a `request.user`.

The invoicables list is generated from the following schema:

```
invoicables = [
        { "subscription": Subscription,
          "lines": [Transaction, ...],
          "options": [Transaction, ...],
        }, ...]
```

Each subscription is either an actual record in the database (paying more periods on a subscription) or `Subscription` instance that only exists in memory but will be committed on checkout.

The `Transaction` list keyed by "lines" contains in-memory instances for the invoice items that will be committed and charged when the order is finally placed.

The `Transaction` list keyed by "options" contains in-memory instances the user can choose from. Options usually include various number of periods that can be pre-paid now for a discount. ex:

> $189.00 Subscription to medium-plan until 2015/11/07 (1 month) $510.30 Subscription to medium-plan until 2016/01/07 (3 months, 10% off) $907.20 Subscription to medium-plan until 2016/04/07 (6 months, 20% off)

class `saas.views.billing.`**`BalanceView`**(***kwargs*)

Set of invoicables for all subscriptions which have a balance due.

While `CartView` generates the invoicables from the `CartItem` model, `BalanceView` generates the invoicables from `Subscription` for which the amount payable by the customer is positive.

The invoicables list is generated from the following schema:

```
invoicables = [
        { "subscription": Subscription,
          "name": "",
          "descr": "",
          "lines": [Transaction, ...],
          "options": [Transaction, ...],
        }, ...]
```

class `saas.views.billing.`**`CartView`**(***kwargs*)

`CartView` derives from `CartSeatsView` which itself derives from `CartPeriodsView`, all of which overrides the `get` method to redirect to the appropriate step in the order pipeline no matter the original entry point.

CHAPTER 4

Transaction ledger

Transactions are recorded in an append-only double-entry book keeping ledger using the following `Transaction` Model:

Name	Description
created_at	Date of creation
descr	Free-form text description (optional)
event_id	Tie-in to other models or third-party systems (optional)
dest_account	Target account (Funds, Income, Expenses, etc.)
dest_organization	Target `Organization`
dest_amount	Target amount in `dest_unit`
dest_unit	Currency unit of the target amount (defaults to 'usd')
orig_account	Source account (Funds, Income, Expenses, etc.)
orig_organization	Source `Organization`
orig_amount	Source amount in `orig_unit`
orig_unit	Currency unit of the source amount (defaults to 'usd')

A `Transaction` records the movement of an *amount* from an *source* to a *target*.

All transactions can be expored in ledger-cli format using the export command:

```
python manage.py ledger export
```

In a minimal cash flow accounting system, *orig_account* and *dest_account* are optional, or rather each `Organization` only has one account (Funds) because we only keep track of the actual transfer of funds.

In a more complex system, like here, we want to keep track of cash assets, revenue and expenses separately because those numbers are meaningful to understand the business. The balance sheet we want to generate at the end of each accounting period will dictate the number of accounts each `Organization` has as well as the movements recorded in the double-entry ledger.

In an online subscription business, there are two chain of events that trigger `Transaction` to be recorded: the *subscription pipeline* itself and the charge pipeline.

subscription pipeline:

- place a subscription order from a `Cart`
- period start
- period end

charge pipeline:

- charge sucessful
- refund or chargeback (optional)
- refund or chargeback expiration (cannot be disputed afterwards)

4.1 Accounts

The balance sheet we are working out of leads to 11 accounts, 9 directly derived from above then 2 more (Withdraw and Writeoff) to balance the books.

- **Backlog** Cash received by a *provider* that was received in advance of earning it.
- **Chargeback** Cash taken back out of a *provider* funds by the platform on a dispute.
- **Canceled** Receivables are written off
- **Expenses** Fees paid by *provider* to a *processor* to settle a credit card payment.
- **Funds** Cash amount currently held on the platform by a *provider*.
- **Income** Taxable income on a *provider* for service provided and invoiced.
- **Liability** Balance due by a *subscriber*.
- **Payable** Order of a subscription to a plan as recorded by a *subscriber*.
- **Offline** Record an offline payment to a *provider* (ex: paper check).
- **Receivable** Order of a subscription to a plan as recorded by a *provider*.
- **Refund** Cash willingly transfered out of a *provider* funds held on the platform.
- **Refunded** Cash transfered back to a *subscriber* credit card.
- **Withdraw** Cash that was taken out of the platform by a *provider*.
- **Writeoff** Payables that cannot and will not be collected by a *provider*

4.1.1 Place a subscription order from a `Cart`

`TransactionManager.`**`new_subscription_order`**(*subscription*, *nb_natural_periods*, *prorated_amount=0*, *created_at=None*, *descr=None*, *discount_percent=0*, *descr_suffix=None*)

Each time a subscriber places an order through the /billing/:organization/cart/ page, a `Transaction` is recorded as follow:

```
yyyy/mm/dd sub_***** description
    subscriber:Payable                      amount
    provider:Receivable
```

Example:

```
2014/09/10 subscribe to open-space plan
    xia:Payable                              $179.99
    cowork:Receivable
```

At first, `nb_periods`, the number of period paid in advance, is stored in the `Transaction.orig_amount`. The `Transaction` is created in `TransactionManager.new_subscription_order`, then only later saved when `TransactionManager.record_order` is called through `Organization.execute_order`. `record_order` will replace `orig_amount` by the correct amount in the expected currency.

4.1.2 Charge sucessful

`Charge.`**`payment_successful`**`()`

When a charge through the payment processor is sucessful, a unique `Transaction` records the charge through the processor. The amount of the charge is then redistributed to the providers (minus processor fee):

```
; Record the charge

yyyy/mm/dd cha_***** charge event
    processor:Funds                           charge_amount
    subscriber:Liability

; Compensate for atomicity of charge record (when necessary)

yyyy/mm/dd sub_***** invoiced-item event
    subscriber:Liability          min(invoiced_item_amount,
    subscriber:Payable                       balance_payable)

; Distribute processor fee and funds to the provider

yyyy/mm/dd cha_***** processor fee paid by provider
    provider:Expenses                         processor_fee
    processor:Backlog

yyyy/mm/dd sub_***** distribution to provider (backlog accounting)
    provider:Receivable                       plan_amount
    provider:Backlog

yyyy/mm/dd cha_***** distribution to provider
    provider:Funds                            distribute_amount
    processor:Funds
```

Example:

```
2014/09/10 Charge ch_ABC123 on credit card of xia
    stripe:Funds                             $179.99
    xia:Liability

2014/09/10 Keep a balanced ledger
    xia:Liability                            $179.99
    xia:Payable

2014/09/10 Charge ch_ABC123 processor fee for open-space
    cowork:Expenses                           $5.22
    stripe:Backlog
```

```
2014/09/10 Charge ch_ABC123 distribution for open-space
    cowork:Receivable                    $179.99
    cowork:Backlog

2014/09/10 Charge ch_ABC123 distribution for open-space
    cowork:Funds                         $174.77
    stripe:Funds
```

4.1.3 Refund and Chargeback

Refunds are initiated by the *provider* while chargebacks are initated by the *subscriber*. In either case, they represent a loss of income while the service was provided.

`ChargeItem.`**`create_refund_transactions`**(*refunded_amount*, *charge_available_amount*, *charge_fee_amount*, *corrected_available_amount*, *corrected_fee_amount*, *created_at=None*, *provider_unit=None*, *processor_unit=None*, *refund_type=None*)

Each `ChargeItem` can be partially refunded:

```
yyyy/mm/dd cha_*****_*** refund to subscriber
    provider:Refund                      refunded_amount
    subscriber:Refunded

yyyy/mm/dd cha_*****_*** refund of processor fee
    processor:Refund                     processor_fee
    processor:Funds

yyyy/mm/dd cha_*****_*** refund of processor fee
    processor:Refund                     distribute_amount
    provider:Funds
```

`Refund` is replaced by `Chargeback` if the refund was initiated by a chargeback event.

Example:

```
2014/09/10 Charge ch_ABC123 refund for subscribe to open-space plan
    cowork:Refund                        $179.99
    xia:Refunded

2014/09/10 Charge ch_ABC123 refund processor fee
    stripe:Refund                          $5.22
    stripe:Funds

2014/09/10 Charge ch_ABC123 cancel distribution
    stripe:Refund                        $174.77
    cowork:Funds
```

Stripe allows you to issue a refund at any time up to 90 days after the charge while for most transactions, subscribers have 120 days from the sale or when they discovered a problem with the product to dispute a charge.

The provider will incur an extra fee on the chargeback that we record as such:

```
yyyy/mm/dd chargeback fee
    processor:Funds                      chargeback_fee
    provider:Funds
```

4.1.4 Withdrawal

`Organization.`**`create_withdraw_transactions`**(*event_id*, *amount*, *unit*, *descr*, *created_at=None*, *dry_run=False*)

Withdraw funds from the site into the provider's bank account.

We record one straightforward `Transaction` for the withdrawal and an additional one in case there is a processor transfer fee:

```
yyyy/mm/dd po_***** withdrawal to provider bank account
    processor:Withdraw                          amount
    provider:Funds

; With StripeConnect there are no processor fees anymore
; for Payouts.
yyyy/mm/dd processor fee paid by provider
    processor:Funds                             processor_fee
    provider:Funds
```

Example:

```
2014/09/10 withdraw from cowork
    stripe:Withdraw                             $174.52
    cowork:Funds
```

`new_subscription_order` and `payment_successful` generates a seemingly complex set of `Transaction`. Now we see how the following events build on the previously recorded transactions to implement deferred revenue accounting.

The following events create "accounting" transactions. No actual funds is transfered between the organizations.

4.1.5 Period started

`TransactionManager.`**`create_period_started`**(*subscription*, *created_at=None*)

When a period starts and we have a payable balance for a subscription, we transfer it to a `Liability` account, recorded as follow:

```
yyyy/mm/dd sub_***** description
    subscriber:Liability                        period_amount
    subscriber:Payable
```

Example:

```
2014/09/10 past due for period 2014/09/10 to 2014/10/10
    xia:Liability                                $179.99
    xia:Payable
```

4.1.6 Period ended

`TransactionManager.`**`create_income_recognized`**(*subscription*, *amount=0*, *starts_at=None*, *ends_at=None*, *descr=None*, *event_id=None*, *dry_run=False*)

When a period ends and we either have a `Backlog` (payment was made before the period starts) or a `Receivable` (invoice is submitted after the period ends). Either way we must recognize income for that period since the subscription was serviced:

```
yyyy/mm/dd sub_***** When payment was made at begining of period
    provider:Backlog                  period_amount
    provider:Income

yyyy/mm/dd sub_***** When service is invoiced after period ends
    provider:Receivable               period_amount
    provider:Income
```

Example:

```
2014/09/10 recognized income for period 2014/09/10 to 2014/10/10
    cowork:Backlog                          $179.99
    cowork:Income
```

4.1.7 Write off

`Organization.`**`create_cancel_transactions`**(*at_time=None*, *user=None*)

Sometimes, a provider will give up and assume receivables cannot be recovered from a subscriber. At that point the receivables are written off.:

```
yyyy/mm/dd sub_***** balance ledger
    subscriber:Liability                      payable_amount
    subscriber:Payable

yyyy/mm/dd sub_***** write off liability
    provider:Writeoff                         liability_amount
    subscriber:Liability

yyyy/mm/dd sub_***** write off receivable
    subscriber:Canceled                       liability_amount
    provider:Receivable
```

Example:

```
2014/09/10 balance ledger
    xia:Liability                           $179.99
    xia:Payable

2014/09/10 write off liability
    cowork:Writeoff                         $179.99
    xia:Liability

2014/09/10 write off receivable
    xia:Canceled                            $179.99
    cowork:Receivable
```

4.1.8 Settled account

`TransactionManager.`**`new_subscription_statement`**(*subscription*, *created_at=None*, *descr_pat=None*, *balance_now=None*)

Since the ordering system is tightly coupled to the `Transaction` ledger, we create special "none" transaction that are referenced when a `Charge` is created for payment of a balance due by a subcriber:

```
yyyy/mm/dd sub_***** description
    subscriber:Settled                          amount
    provider:Settled
```

Example:

```
2014/09/10 balance due
    xia:Settled                               $179.99
    cowork:Settled
```

4.2 Charges

Charges are recorded in a table separate from the ledger. They undergo their own state diagram as follows.

`ChargeItem` records every line item for a `Charge`. The recorded relationships between `Charge`, `ChargeItem` and `Transaction.event_id` is critical to easily record refunds, chargeback disputes and reverted chargebacks in an append-only double-entry bookkeeping system.

CHAPTER 5

Flexible Role-based Access Control

Business logic sometimes dictates that a provider has minimal access the billing profile of a customer and sometimes a provider must be able to update the credit card associated to a customer while on a phone call with that customer.

In order to support multiple usage patterns and security constraints, authorization is not embeded into the djaodjin-saas logic but rather implemented as URL decorators. It is the responsability of the developper to associate decorators to URLs as dictated by the business requirements.

The security framework defines a generic `RoleDescription` whose purpose is to define a role a `User` has on an `Organization` (see *grant and request roles*).

A organization-agnostic manager role always exists and helps with bootstrapping the security policies. In most setups a second role, for example, a contributor role is implemented. Typically *manager* have full access to an Organization while *contributor* are restricted to read-only permissions.

5.1 Examples

Let's say you want to give POST access to contributors on */api/billing/charges/:charge/refund/*, you would write the following in your urls.py:

```
from urldecorators import url
from saas.api.charges import ChargeRefundAPIView

urlpatterns = [

    url(r'^billing/charges/(?P<charge>[a-zA-Z0-9_\-\+\.]+)/refund/',
        ChargeRefundAPIView.as_view(),
        name='saas_api_charge_refund',
        decorators=['saas.decorators.requires_provider_weak']),
]
```

The previous example uses django-urldecorators and a `saas.decorators.requires_provider_weak decorator`.

The `saas.urls` module has been split in "common" set of functionalities such that in many cases you can decorate each include() with an appropriate decorator instead of each URL one by one. (ex: testsite/urls.py)

A blog post on Django Rest Framework, AngularJS and permissions might also be a useful read.

5.2 Decorators Available

The access control logic is best configured in the site URLConf through extensions like django-urldecorators. This is not only more flexible but also make security audits a lot easier.

`saas.decorators.`**`requires_agreement`**(*function=None*, *agreement='terms-of-use'*, *redirect_field_name='next'*)

Decorator for views that checks that the user has signed a particular legal agreement, redirecting to the agreement signature or log-in page if necessary.

`saas.decorators.`**`requires_paid_subscription`**(*function=None*, *organization_kwarg_slug='organization'*, *plan_kwarg_slug='subscribed_plan'*, *redirect_field_name='next'*, *strength=1*, *roledescription=None*)

Decorator that checks a specified subscription is paid. It redirects to an appropriate page when this is not the case:

- Payment page when no charge is associated to the subscription,
- Update Credit Card page when `charge.status` is `failed`,
- Waiting page when `charge.status` is `in-progress`.

`saas.decorators.`**`requires_direct`**(*function=None*, *roledescription=None*, *redirect_field_name='next'*)

Decorator for views that checks that the authenticated `request.user` is a direct `roledescription` (ex: contributor) or manager for the `Organization` associated to the request.

Managers can issue all types of requests (GET, POST, etc.). while `roledescription` (ex: contributors) are restricted to GET requests.

`saas.decorators.`**`requires_provider`**(*function=None*, *roledescription=None*, *redirect_field_name='next'*)

Decorator for views that checks that the request authenticated `User` is a `roledescription` (ex: contributor) or manager for the `Organization` associated to the request itself or a `roledescription` (or manager) to a provider for the `Organization` associated to the request.

Managers can issue all types of requests (GET, POST, etc.). while `roledescription` (ex: contributors) are restricted to GET requests.

`saas.decorators.`**`requires_self_provider`**(*function=None*, *roledescription=None*, *redirect_field_name='next'*)

Decorator for views that checks that the request authenticated `User` is the user associated to the URL. Authenticated users that can also access the URL through this decorator are `roledescription` (ex: contributors) or managers for any `Organization` associated with the user served by the URL (the accessed user is a direct `roledescription` or manager of the organization) and transitively contributors (or managers) for any provider to one of these direct organizations.

Managers can issue all types of requests (GET, POST, etc.). while `roledescription` (ex: contributors) are restricted to GET requests.

Design Note

We used to decorate the saas views with the "appropriate" decorators, except in many projects appropriate had a different meaning. It turns out that the access control logic is better left to be configured in the site URLConf through extensions like django-urldecorators. This is not only more flexible but also make security audits a lot easier.

CHAPTER 6

Roles, Subscriptions and Opt-ins

There exists two types of relationships in djaodjin-saas:

- between users and organizations (ex: donny is a manager of cowork)
- between organizations themselves (ex: cowork is a subscriber of djaodjin)

A `Role` connects a `User` to an `Organization` through a `RoleDescription`. A `Subscription` connects an `Organization` to another `Organization` through a `Plan` (see *database schema*).

In both cases granting a new relationship will invite a `User` (or `Organization`) to the site or ask the `User` (or `Organization`) to opt-in into the relationship as necessary.

6.1 Connecting a User to an Organization

Grant of a role to an organization must be initiated by a user who has an existing relationship to the Organization and permissions to grant the new role. (see *Flexible Security Framework*).

The grant mechanism is also used to invite people to register to the site. Because users might have multiple e-mail addresses, already registered with one address yet invited on another, the first authenticated user that claims the `grant_key` will be associated with the `Role`.

Request for a role initiated by a user who has no pre-existing relationship to the organization will trigger a notification to all managers of the organization. A manager then has the opportunity to then accept the request and grant any `RoleDescription` to the requesting user.

As `grant_key` are random 40-characters long hexadecimal strings, it is virtually impossible for a random user to claim a grant unauthorized, yet gives an opportunity to already registered users to easily consolidate their accounts under a single sign-on.

In case an organization prefers to avoid the (minimal) risk of a grant key being intercepted, a manager can instructs a user to register an account on the site and request a role to the organization. Since in many cases the organization manager knows best which role to grant the requesting user, the `RoleDescription` is not part of the request but part of the accept.

We are always looking for feedback and new use cases. If you are building a SaaS product that requires much more strident identity checks on grants, please get in touch with us .

6.2 Connecting Two Organizations to each other

In ninety-nine percent of the cases, two organizations become connected together when one subscribes to a plan provided by the second through the orders.

There are two special cases. First, a provider can decide to directly grant a subscription to a plan for multiple reasons (ex: demo, invite-only plans, fixing a mistaken purchase from a customer). Second, even though a customer can subscribe and pay online, some plans require a setup and/or activation from the provider (ex: assign a desk at a co-working space).

In all cases where a provider subscribes an organization to one of its plan, there needs to be an opt-in from the new subscriber. This is because managers of the provider will then have access to profile information of the subscriber for support reasons (see *Flexible Security Framework*). The only exception is the broker organization (i.e. organization hosting the site) since the hosting service reliability team typically have direct access to the underlying database already.

We decided that charges happen at checkout even for `Plan` that require managers of the provider to accept the subscription (i.e. `Plan.optin_on_request = True`). If your business logic requires to charge after the subscription has been accepted by a provider's manager, please get in touch with us. We are always looking for feedback and new use cases.

CHAPTER 7

Periodic tasks

The renewals command is intended to be run as part of an automated script run at least once a day. It will

- recognize revenue for past periods (see *ledger*).
- extends active subscriptions
- create charges for new periods
- trigger expiration notices

Every functions part of the renewals script are explicitly written to be idempotent. Calling the scripts multiple times for the same timestamp (i.e. with the `--at-time` command line argument) will generate the appropriate `Transaction` and `Charge` only once.

Example cron setup:

```
$ cat /etc/cron.daily/renewals
#!/bin/sh

cd /var/*mysite* && python manage.py renewals
```

CHAPTER 8

HTML pages

URLs serving HTML pages are split in two sets: subscriber-facing and provider-facing pages.

If you want to present a completely different user interface to customers and managers of the provider backend, you will choose to have both sets of templates inherit from a different base template. If you are building an interface where managers see an augmented interface overlayed on top of regular subscriber interface, you might prefer all your templates to inherit from a common base template.

8.1 Subscriber facing pages

All the pages in this group are accessible to a subscriber, either to establish a relationship with the site by subscribing to a plan or to manage her billing profile, active subscriptions, etc.

You will want to edit these templates first since they directly impact the look and feel a customer will have of your site.

8.1.1 Create a subscription

GET /legal/

class saas.views.legal.**AgreementListView**(***kwargs*)

List all agreements and policies for a provider site. This typically include terms of service, security policies, etc.

Template:

To edit the layout of this page, create a local saas/legal/index.html (example).

Template context:

- agreement_list List of agreements published by the provider
- organization The provider of the product
- request The HTTP request object

GET /legal/:agreement/

class saas.views.legal.**AgreementDetailView**(***kwargs*)
Show a single agreement (or policy) document. The content of the agreement is read from saas/agreements/<slug>.md.

Template:

To edit the layout of this page, create a local saas/legal/agreement.html (example).

Template context:

- page The content of the agreement formatted as HTML.
- organization The provider of the product
- request The HTTP request object

GET /legal/:agreement/sign

POST /legal/:agreement/sign

class saas.views.legal.**AgreementSignView**(***kwargs*)
For a the request user to sign a legal agreement.

Template:

To edit the layout of this page, create a local saas/legal/sign.html (example).

Template context:

- page The content of the agreement formatted as HTML.
- organization The provider of the product
- request The HTTP request object

GET /pricing/

POST /pricing/

class saas.views.plans.**CartPlanListView**(***kwargs*)
GET displays the active plans available for subscription.

Template:

To edit the layout of this page, create a local saas/pricing.html (example).

Template context:

- plan_list List of plans a customer can subscribed to
- items_selected List of items currently in the request user cart.
- organization The provider of the product
- request The HTTP request object

POST adds the selected plan into the request user cart.

GET /redeem/

class saas.views.billing.**RedeemCouponView**(***kwargs*)
Stores a Coupon into the session for further use in the checkout pipeline.

GET /billing/cart/

GET /billing/:organization/cart/

POST /billing/:organization/cart/

CartView.**get**(*request*, **args*, ***kwargs*)
Prompt the user to enter her credit card and place an order.

On POST, the credit card will be charged and the organization subscribed to the plans ordered.

Template:

To edit the layout of this page, create a local saas/billing/cart.html (example). This template is responsible to create a token on Stripe that will then be posted back to the site.

Template context:

- STRIPE_PUB_KEY Public key to send to stripe.com
- invoicables List of items to be invoiced (with options)
- organization The provider of the product
- request The HTTP request object

GET /billing/:organization/cart-periods/

POST /billing/:organization/cart-periods/

class saas.views.billing.**CartPeriodsView**(***kwargs*)
Optional page to pay multiple periods in advance.

Template:

To edit the layout of this page, create a local saas/billing/cart-periods.html (example).

Template context:

- invoicables List of items to be invoiced (with options)
- organization The provider of the product
- request The HTTP request object

GET /billing/:organization/cart-seats/

POST /billing/:organization/cart-seats/

class saas.views.billing.**CartSeatsView**(***kwargs*)
Optional page to subcribe multiple organizations to a Plan while paying through through a third-party Organization (i.e. self.organization).

Template:

To edit the layout of this page, create a local saas/billing/cart-seats.html (example).

Template context:

- invoicables List of items to be invoiced (with options)
- organization The provider of the product
- request The HTTP request object

GET /billing/:organization/receipt/:charge

class saas.views.billing.**ChargeReceiptView**(***kwargs*)
Display a receipt for a Charge.

Template:

To edit the layout of this page, create a local saas/billing/receipt.html (example).

Template context:

- `charge` The charge object
- `organization` The provider of the product
- `request` The HTTP request object

This page will be accessible in the payment flow as well as through a subscriber profile interface. The template should take both usage under consideration.

`GET /app/new/`

class `saas.views.profile.`**`OrganizationCreateView`**(***kwargs*)

This page helps `User` create a new `Organization`. By default, the request user becomes a manager of the newly created entity.

`User` and `Organization` are separate concepts links together by manager and other custom `RoleDescription` relationships.

The complete `User`, `Organization` and relationship might be exposed right away to the person registering to the site. This is very usual in Enterprise software.

On the hand, a site might decide to keep the complexity hidden by enforcing a one-to-one manager relationship between a `User` (login) and an `Organization` (payment profile).

Template:

To edit the layout of this page, create a local `saas/app/new.html` (example).

Template context:

- `request` The HTTP request object

8.1.2 Manage subscriptions

These pages enable a subscriber to manage her profile on the site. She can update her personal information (email address, etc.), change her credit card on file, review the list of charges by a provider, pay a balance due, etc.

The business requirements might require or prevent a manager or a custom role (ex: contributor) of a provider to access specific information about a subscriber. For example, you might allow your customer support team to update a subscriber credit card over the phone for convienience. You might also believe it is too much risk, deny the ability to do so by your customer support people and instruct them to hand out instructions to the subscriber on how to do so by herself. All scenarios can easily be implemented through a *Flexible Security Framework*.

`GET /billing/:organization/`

class `saas.views.billing.`**`BillingStatementView`**(***kwargs*)

This page shows a statement of `Subscription` orders, `Charge` created and payment refunded.

Template:

To edit the layout of this page, create a local `saas/billing/index.html` (example). You should insure the page will call back the */api/billing/:organization/payments/* API end point to fetch the set of transactions.

Template context:

- **`balance_price` A tuple of the balance amount due by the subscriber** and unit this balance is expressed in (ex: usd).
- `organization` The subscriber object
- `request` The HTTP request object

`GET /billing/:organization/balance/`

`POST /billing/:organization/balance/`

BalanceView.**get_queryset**()
GET displays the balance due by a subscriber.

Template:

To edit the layout of this page, create a local `saas/billing/balance.html` (example).

Template context:

- `STRIPE_PUB_KEY` Public key to send to stripe.com
- `invoicables` List of items to be invoiced (with options)
- `organization` The provider of the product
- `request` The HTTP request object

POST attempts to charge the card for the balance due.

GET /billing/:organization/card/

POST /billing/:organization/card/

class saas.views.billing.**CardUpdateView**(***kwargs*)
Page to update the Credit Card information associated to a subscriber.

Template:

To edit the layout of this page, create a local `saas/billing/card.html` (example).

Template context:

- `STRIPE_PUB_KEY` Public key to send to stripe.com
- `organization` The subscriber object
- `request` The HTTP request object

GET /profile/:organization/

POST /profile/:organization/

class saas.views.profile.**OrganizationProfileView**(***kwargs*)
Page to update contact information of an `Organization`.

Template:

To edit the layout of this page, create a local `saas/profile/index.html` (example).

Template context:

- `urls.organization.password_chage` URL to update user password.
- `organization` The organization object
- `request` The HTTP request object

GET /profile/:organization/subscriptions/

class saas.views.profile.**SubscriptionListView**(***kwargs*)
List of Plans this organization is subscribed to.

Template:

To edit the layout of this page, create a local `saas/profile/subscriptions.html` (example). You should insure the page will call back the */api/profile/:organization/subscriptions/* API end point to fetch the set of subscriptions for the organization.

Template context:

- `organization` The subscriber object
- `request` The HTTP request object

GET /profile/:organization/roles/managers/

GET /profile/:organization/roles/contributors/

class saas.views.profile.**RoleListView**(***kwargs*)
List all `RoleDescription` for an organization and the users under each role.

GET /users/roles/

GET /users/:user/roles/

class saas.views.users.**ProductListView**(***kwargs*)
List of organizations a `:user` has a role with.

Template:

To edit the layout of this page, create a local `saas/users/roles.html` (example). You should insure the page will call back the */api/users/:user/roles/* API end point to fetch the set of organization accessible by the user.

Template context:

- `user` The organization object users have permissions to.
- `request` The HTTP request object

8.2 Provider facing pages

Provider facing pages are only accessible to its managers. They are used to assess the performance of the business, set pricing strategy, and help with customer support.

8.2.1 Pricing strategy

GET /provider/billing/coupons/

GET /provider/billing/:organization/coupons/

class saas.views.billing.**CouponListView**(***kwargs*)
View to manage discounts (i.e. `Coupon`)

Template:

To edit the layout of this page, create a local `saas/billing/coupons.html` (example). You should insure the page will call back the */api/billing/:organization/coupons/* API end point to fetch the set of coupons for the provider organization.

Template context:

- `organization` The provider for the coupons
- `request` The HTTP request object

GET /provider/profile/plans/new/

GET /provider/profile/:organization/plans/new/

POST /provider/profile/:organization/plans/new/

class saas.views.plans.**PlanCreateView**(***kwargs*)
Create a new Plan for an Organization.

Template:

To edit the layout of this page, create a local saas/profile/plans/new.html (example).

Template context:

- organization The provider for the plans
- request The HTTP request object

GET /provider/profile/plans/:plan/

GET /provider/profile/:organization/plans/:plan/

class saas.views.plans.**PlanUpdateView**(***kwargs*)
Update information about a Plan.

Template:

To edit the layout of this page, create a local saas/profile/plans/edit.html (example).

Template context:

- plan The plan to update
- show_delete True if there never were subscriber to the plan
- organization The provider of the plan
- request The HTTP request object

8.2.2 Transfer subscriber payments to provider bank

GET /provider/billing/bank/

GET /provider/billing/:organization/bank/

class saas.views.billing.**BankUpdateView**(***kwargs*)

GET /provider/billing/transfers/

GET /provider/billing/:organization/transfers/

class saas.views.billing.**TransferListView**(***kwargs*)
List of payments made to a provider or funds transfered out of the platform to the provider bank account.

Template:

To edit the layout of this page, create a local saas/billing/transfers.html (example). You should insure the page will call back the */api/billing/:organization/transfers/* API end point to fetch the set of transactions.

Template context:

- organization The provider transactions refer to
- request The HTTP request object

GET /provider/billing/import/

GET /provider/billing/:organization/import/

class saas.views.billing.**ImportTransactionsView**(***kwargs*)
Insert transactions that were done offline for the purpose of computing accurate metrics.

Template:

To edit the layout of this page, create a local saas/billing/import.html (example).

Template context:

- organization The provider object
- request The HTTP request object

8.2.3 Manage subscribers and business performance

GET /provider/metrics/coupons/:coupon

GET /provider/metrics/:organization/coupons/:coupon

class saas.views.metrics.**CouponMetricsView**(***kwargs*)
Performance of Coupon based on CartItem.

Template:

To edit the layout of this page, create a local saas/metrics/coupons.html (example).

Template context:

- coupon The coupon the list of uses refers to
- organization The provider object
- request The HTTP request object

GET /provider/metrics/dashboard/

GET /provider/metrics/:organization/dashboard/

class saas.views.profile.**DashboardView**(***kwargs*)
High-level dashboard for a quick glance of the business in real-time.

Template:

To edit the layout of this page, create a local saas/metrics/dashboard.html (example).

Template context:

- organization The provider object
- request The HTTP request object

GET /provider/metrics/plans/

GET /provider/metrics/:organization/plans/

class saas.views.metrics.**PlansMetricsView**(***kwargs*)
Performance of Plans for a time period (as a count of subscribers per plan per month)

Template:

To edit the layout of this page, create a local saas/metrics/plans.html (example). The page will typically call back */api/metrics/:organization/plans/* to fetch the 12 month trailing performance in terms of subscribers of the plans of a provider.

Template context:

- organization The provider object

- `request` The HTTP request object

GET /provider/metrics/revenue/

GET /provider/metrics/:organization/revenue/

class saas.views.metrics.**RevenueMetricsView**(***kwargs*)
Reports cash flow and revenue in currency units.

Template:

To edit the layout of this page, create a local `saas/metrics/revenue.html` (example).

The page will typically call back */api/metrics/:organization/funds/* to fetch the 12 month trailing cash flow table, and/or */api/metrics/:organization/balances/* to fetch the 12 month trailing receivable/backlog/income revenue.

The example page also calls back */api/metrics/:organization/customers/* to fetch the distinct number of customers that generated the cash transactions.

Template context:

- `organization` The provider object
- `request` The HTTP request object

GET /provider/profile/subscribers/

GET /provider/profile/:organization/subscribers/

class saas.views.profile.**SubscriberListView**(***kwargs*)
List of organizations subscribed to a plan provided by the organization.

Template:

To edit the layout of this page, create a local `saas/profile/subscribers.html` (example).

This page will typically call back */api/metrics/:organization/active/* and/or */api/metrics/:organization/churned/* to fetch the set of active and/or churned subscribers for a provider plans.

Template context:

- `organization` The provider object
- `request` The HTTP request object

GET /provider/metrics/balances/:report/

class saas.views.metrics.**BalancesView**(***kwargs*)
Display a balance sheet named `:report`.

Template:

To edit the layout of this page, create a local `saas/metrics/balances.html` (example).

Template context:

- `organization` The provider object
- `request` The HTTP request object

CHAPTER 9

API Reference

The djaodjin-saas API is split in four sections: Billing, Subscription, Metrics and Search.

The Billing and Subscription APIs deal with the actual business logic of a Software-as-a-Service, that is the transfer of funds and access control respectively.

The Metrics and Search APIs aggregate the underlying data in various ways to keep on top of the performance of the business as well as provide rich interfaces.

9.1 Billing API

These API end points manage the transfer of funds between a subscriber and a provider through a processor.

GET /api/billing/:organization/bank/

class saas.api.backend.**RetrieveBankAPIView**(***kwargs*)

Pass through that calls the processor API to retrieve some details about the deposit account associated to a provider (if that information is available through the *payment processor backend* API).

This API does not trigger payment of a subscriber to a provider. Checkout of a subscription cart is done either through the *HTML page* or *API end point*.

**Examples

```
GET /api/billing/cowork/bank/ HTTP/1.1
```

responds

```
{
  "bank_name": "Stripe Test Bank",
  "last4": "***-htrTZ",
  "balance_amount": 0,
  "balance_unit": "usd"
}
```

GET /api/billing/:organization/history/

class saas.api.transactions.**BillingsAPIView**(***kwargs*)

Queries a page (PAGE_SIZE records) of Transaction associated to {organization} while the organization acts as a subscriber.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

This API end point is typically used to display orders, payments and refunds of a subscriber (see subscribers pages)

**Examples

```
GET /api/billing/xia/history?start_at=2015-07-05T07:00:00.000Z&o=date&ot=desc␣
↪HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "balance": 11000,
    "unit": "usd",
    "results": [
        {
            "created_at": "2015-08-01T00:00:00Z",
            "description": "Charge for 4 periods",
            "amount": "($356.00)",
            "is_debit": true,
            "orig_account": "Liability",
            "orig_organization": "xia",
            "orig_amount": 112120,
            "orig_unit": "usd",
            "dest_account": "Funds",
            "dest_organization": "stripe",
            "dest_amount": 112120,
            "dest_unit": "usd"
        }
    ]
}
```

GET /api/billing/:organization/card/

class saas.api.backend.**RetrieveCardAPIView**(***kwargs*)

Pass through to the processor to retrieve some details about the payment method (ex: credit card) associated to a subscriber.

**Examples

```
GET /api/billing/cowork/card/ HTTP/1.1
```

responds

```
{
  "last4": "1234",
  "exp_date": "12/2015"
}
```

DELETE /api/billing/:organization/balance/cancel/

class saas.api.transactions.**CancelStatementBalanceAPIView**(***kwargs*)

Cancel the balance for a provider organization. This will create a transaction for this balance cancellation. A manager can use this endpoint to cancel balance dues that is known impossible to be recovered (e.g. an external bank or credit card company act).

The endpoint returns the transaction created to cancel the balance due.

**Examples

```
DELETE /api/billing/cowork/balance/ HTTP/1.1
```

GET /api/billing/:organization/coupons/

POST /api/billing/:organization/coupons/

class saas.api.coupons.**CouponListAPIView**(***kwargs*)

Queries a page (PAGE_SIZE records) of Coupon associated to a provider.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

**Examples

```
GET /api/billing/cowork/coupons?o=code&ot=asc&q=DIS HTTP/1.1
```

retrieves the list of Coupon for provider cowork where *code* matches 'DIS', ordered by *code* in ascending order.

```
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "code": "DIS100",
            "percent": 100,
            "created_at": "2014-01-01T09:00:00Z",
            "ends_at": null,
            "description": null
        },
        {
            "code": "DIS50",
            "percent": 50,
            "created_at": "2014-01-01T09:00:00Z",
            "ends_at": null,
            "description": null
        }
    ]
}
```

GET /api/billing/:organization/coupons/:coupon/

PUT /api/billing/:organization/coupons/:coupon/

DELETE /api/billing/:organization/coupons/:coupon/

class saas.api.coupons.**CouponDetailAPIView**(***kwargs*)

Retrieves a Coupon.

**Examples

```
GET /api/billing/cowork/coupons/DIS100 HTTP/1.1
```

```
 {
     "code": "DIS100",
     "percent": 100,
     "created_at": "2014-01-01T09:00:00Z",
     "ends_at": null,
     "description": null
}
```

GET /api/billing/:organization/receivables/

class saas.api.transactions.**ReceivablesListAPIView**(***kwargs*)

Queries a page (PAGE_SIZE records) of Transaction marked as receivables associated to {organization} while the organization acts as a provider.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

This API endpoint is typically used to find all sales for {organization} whether it was paid or not.

**Examples

```
GET /api/billing/cowork/receivables?start_at=2015-07-05T07:00:00.000Z&o=date&
↪ot=desc HTTP/1.1
```

```
{
    "count": 1,
    "total": "112120",
    "unit": "usd",
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2015-08-01T00:00:00Z",
            "description": "Charge <a href="/billing/cowork/receipt/1123">1123</a>
↪ distribution for demo562-open-plus",
            "amount": "112120",
            "is_debit": false,
            "orig_account": "Funds",
            "orig_organization": "stripe",
            "orig_amount": 112120,
            "orig_unit": "usd",
            "dest_account": "Funds",
            "dest_organization": "cowork",
            "dest_amount": 112120,
            "dest_unit": "usd"
        }
    ]
}
```

GET /api/billing/:organization/transfers/

class saas.api.transactions.**TransferListAPIView**(***kwargs*)

Queries a page (PAGE_SIZE records) of Transaction associated to {organization} while the organization acts as a provider.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

This API endpoint is typically used to find sales, payments, refunds and bank deposits for a provider. (see provider pages)

**Examples

```
GET /api/billing/cowork/transfers?start_at=2015-07-05T07:00:00.000Z&o=date&
↪ot=desc HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2015-08-01T00:00:00Z",
            "description": "Charge <a href="/billing/cowork/receipt/1123">1123</a>
↪ distribution for demo562-open-plus",
            "amount": "$1121.20",
            "is_debit": false,
            "orig_account": "Funds",
            "orig_organization": "stripe",
            "orig_amount": 112120,
            "orig_unit": "usd",
            "dest_account": "Funds",
            "dest_organization": "cowork",
            "dest_amount": 112120,
            "dest_unit": "usd"
        }
    ]
}
```

GET /api/billing/transactions/

class saas.api.transactions.**TransactionListAPIView**(***kwargs*)

Queries a page (PAGE_SIZE records) of Transaction from the *ledger*.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

**Examples

```
GET /api/billing/transactions?start_at=2015-07-05T07:00:00.000Z&o=date&ot=desc␣
↪HTTP/1.1
```

responds

```
{
    "ends_at": "2017-03-30T18:10:12.962859Z",
    "balance": 11000,
    "unit": "usd",
    "count": 1,
    "next": null,
    "previous": null,
```

```
    "results": [
        {
            "created_at": "2017-02-01T00:00:00Z",
            "description": "Charge for 4 periods",
            "amount": "($356.00)",
            "is_debit": true,
            "orig_account": "Liability",
            "orig_organization": "xia",
            "orig_amount": 112120,
            "orig_unit": "usd",
            "dest_account": "Funds",
            "dest_organization": "stripe",
            "dest_amount": 112120,
            "dest_unit": "usd"
        }
    ]
}
```

GET /api/billing/charges/

class saas.api.charges.**ChargeListAPIView**(***kwargs*)

Queries a page (PAGE_SIZE records) of Charge that were created on the processor.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

**Examples

```
GET /api/billing/charges?start_at=2015-07-05T07:00:00.000Z&o=date&ot=desc HTTP/1.1
```

Retrieve the list of charges that were created before 2015-07-05T07:00:00.000Z, sort them by date in descending order.

```
{
    "count": 1,
    "unit": "usd",
    "total": "112120",
    "next": null,
    "previous": null,
    "results": [{
        "created_at": "2016-01-01T00:00:00Z",
        "readable_amount": "$1121.20",
        "amount": 112120,
        "unit": "usd",
        "description": "Charge for subscription to cowork open-space",
        "last4": "1234",
        "exp_date"" "12/2016",
        "processor_key": "ch_XAb124EF",
        "state": "DONE"
    } ...]
```

GET /api/billing/charges/:charge/

class saas.api.charges.**ChargeResourceView**(***kwargs*)

Pass through to the processor and returns details about a Charge.

**Examples

```
GET /api/billing/charges/ch_XAb124EF/ HTTP/1.1
```

```
{
    "created_at": "2016-01-01T00:00:00Z",
    "readable_amount": "$1121.20",
    "amount": 112120,
    "unit": "usd",
    "description": "Charge for subscription to cowork open-space",
    "last4": "1234",
    "exp_date"" "12/2016",
    "processor_key": "ch_XAb124EF",
    "state": "DONE"
}
```

POST /api/billing/charges/:charge/email/

class saas.api.charges.**EmailChargeReceiptAPIView**(***kwargs*)

Email the charge receipt to the customer email address on file.

**Example

```
POST /api/billing/charges/ch_XAb124EF/email/  HTTP/1.1
```

responds

```
{
    "charge_id": "ch_XAb124EF",
    "email": "joe@localhost.localdomain"
}
```

The service sends a duplicate e-mail receipt for charge *ch_XAb124EF* to the e-mail address of the customer, i.e. *joe@localhost.localdomain*.

POST /api/billing/charges/:charge/refund/

class saas.api.charges.**ChargeRefundAPIView**(***kwargs*)

Partially or totally refund all or a subset of line items on a Charge.

**Example

```
POST /api/billing/charges/ch_XAb124EF/refund/ HTTP/1.1
```

```
{
    "lines": [
      {
          "num": 0,
          "refunded_amount": 4000,
      },
      {
          "num": 1,
          "refunded_amount": 82120,
      }
  ]
}
```

Refunds $40 and $821.20 from first and second line item on the receipt respectively. The API call responds with the Charge.

```
{
    "created_at": "2016-01-01T00:00:00Z",
    "readable_amount": "$1121.20",
    "amount": 112120,
    "unit": "usd",
    "description": "Charge for subscription to cowork open-space",
    "last4": "1234",
    "exp_date"" "12/2016",
    "processor_key": "ch_XAb124EF",
    "state": "DONE"
}
```

POST /api/cart/

class saas.api.billing.**CartItemAPIView**(***kwargs*)

Adds a Plan into the cart of the request.user.

The cart can later be checked out and paid by an Organization, either through the *HTML page* or *API end point*.

This end point is typically used when a user is presented with a list of add-ons that she can subscribes to in one checkout screen. The end-point works in both cases, authenticated or anonymous users. For authenticated users, the cart is stored in the database as CartItem objects. For anonymous users, the cart is stored in an HTTP Cookie.

The end-point accepts a single item or a list of items.

quantity is optional. When it is not specified, subsquent checkout screens will provide choices to pay multiple periods in advance When additional full_name, email and sync_on are specified, payment can be made by one Organization for another Organization to be subscribed (see *GroupBuy orders*).

**Examples

```
POST /api/cart/ HTTP/1.1
```

```
{
    "plan": "open-space",
    "option": 1
}
```

responds

```
{
    "plan": "open-space",
    "option": 1
}
```

option is optional. When it is not specified, subsquent checkout screens will provide choices to pay multiple periods in advance When additional full_name and sync_on are specified, payment can be made by one Organization for another Organization to be subscribed (see *GroupBuy orders*).

POST /api/cart/redeem/

DELETE /api/cart/:plan/upload/

class saas.api.billing.**CartItemUploadAPIView**(***kwargs*)

Add a Plan into the subscription cart of multiple users as per the content of an uploaded file.

This works bulk fashion of */cart/ endpoint*. The uploaded file must be a CSV containing the fields first_name, last_name and email. The CSV file must not contain a header line, only data.

**Examples

Content of `names.csv`:

```
POST /api/cart/:plan/upload/ HTTP/1.1

Content-Disposition: form-data; name="file"; filename="names.csv"
Content-Type: text/csv
```

responds

```
{
    "created" [
        {
            "first_name": "Joe",
            "last_name": "Smith",
            "email": "joesmith@example.com"
        },
        {
            "first_name": "Marie",
            "last_name": "Johnson",
            "email": "mariejohnson@example.com"
        }
    ],
    "updated": [],
    "failed": []
}
```

POST /api/billing/:organization/checkout

class saas.api.billing.**CheckoutAPIView**(***kwargs*)

Get a list indexed by plans of items that will be charged (*lines*) and options that could be charged instead.

In many subscription businesses, it is possible to buy multiple period in advance at a discount. The options reflects that.

**Examples

```
GET /api/billing/xia/checkout HTTP/1.1
```

responds

```
{"items":
[{
  "subscription":{
      "created_at":"2016-06-21T23:24:09.242925Z",
      "ends_at":"2016-10-21T23:24:09.229768Z",
      "description":null,
      "organization":{
          "slug":"xia",
          "full_name":"Xia",
          "printable_name":"Xia",
          "created_at":"2012-08-14T23:16:55Z",
          "email":"xia@localhost.localdomain"
      },
      "plan":{
          "slug":"basic",
          "title":"Basic",
          "description":"Basic Plan",
          "is_active":true,
```

```
            "setup_amount":0,
            "period_amount":2000,
            "interval":4,
            "app_url":"/app/"
        },
        "auto_renew":true
    },
    "lines":[{
        "created_at":"2016-06-21T23:42:13.863739Z",
        "description":"Subscription to basic until 2016/11/21 (1 month)",
        "amount":"$20.00",
        "is_debit":false,
        "orig_account":"Receivable",
        "orig_organization":"cowork",
        "orig_amount":2000,
        "orig_unit":"usd",
        "dest_account":"Payable",
        "dest_organization":"xia",
        "dest_amount":2000,
        "dest_unit":"usd"
    }],
    "options":[]
}]
}
```


9.2 Subscription API

These API end points manage the subscription logic, payments excluded.

POST /api/profile/:organization/plans/

class saas.api.plans.**PlanCreateAPIView**(***kwargs*)

Create a Plan for a provider.

**Examples

```
POST /api/profile/cowork/plans HTTP/1.1
```


```
{
    "title": "Open Space",
    "description": "A desk in our coworking space",
    "is_active": false,
    "period_amount": 12000,
    "interval": 1
}
```


responds


```
{
    "title": "Open Space",
    "description": "A desk in our coworking space",
    "is_active": false,
    "period_amount": 12000,
    "interval": 1
}
```

GET /api/profile/:organization/plans/:plan/

PUT /api/profile/:organization/plans/:plan/

DELETE /api/profile/:organization/plans/:plan/

class saas.api.plans.**PlanResourceView**(***kwargs*)

Retrieves a Plan.

The is_active boolean is used to activate a plan, enabling users to subscribe to it, or deactivate a plan, disabling users from subscribing to it.

**Examples

```
GET /api/profile/cowork/plans/open-space HTTP/1.1
```

```
{
    "title": "Open Space",
    "description": "A desk in our coworking space",
    "is_active": false,
    "period_amount": 12000,
    "interval": 1
}
```

GET /api/profile/:organization/plans/:plan/subscriptions/

POST /api/profile/:organization/plans/:plan/subscriptions/

class saas.api.subscriptions.**PlanSubscriptionsAPIView**(***kwargs*)

A GET request will list all Subscription to a specified :plan provided by :organization.

**Examples

```
GET /api/profile/:organization/plans/:plan/subscriptions/ HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
        "slug": "xia",
        "full_name": "Xia Lee",
        "created_at": "2016-01-14T23:16:55Z"
        }
    ]
}
```

GET /api/profile/:organization/

PUT /api/profile/:organization/

DELETE /api/profile/:organization/

class saas.api.organizations.**OrganizationDetailAPIView**(***kwargs*)

Retrieves profile information on an Organization.

**Examples

```
GET /api/profile/xia/ HTTP/1.1
```

responds

```
{
    "created_at": "2018-01-01T00:00:00Z",
    "email": "xia@locahost.localdomain",
    "full_name": "Xia Lee",
    "printable_name": "Xia Lee",
    "slug": "xia",
    "subscriptions": [
        {
            "created_at": "2018-01-01T00:00:00Z",
            "ends_at": "2019-01-01T00:00:00Z",
            "plan": "open-space",
            "auto_renew": true
        }
    ]
}
```

GET /api/users/:user/accessibles/

POST /api/users/:user/accessibles/

class saas.api.roles.**AccessibleByListAPIView**(***kwargs*)

Lists all relations where an Organization is accessible by a User. Typically the user was granted specific permissions through a Role.

see *Flexible Security Framework*.

**Examples

```
GET  /api/users/alice/accessibles/ HTTP/1.1
```

responds

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2018-01-01T00:00:00Z",
            "slug": "cowork",
            "printable_name": "ABC Corp.",
            "role_description": "manager",
            "request_key": null,
            "grant_key": null
        }
    ]
}
```

DELETE /api/users/:user/accessibles/:organization/

class saas.api.roles.**RoleDetailAPIView**(***kwargs*)

Dettach a user from one or all roles with regards to an organization, typically resulting in revoking permissions from this user to manage part of an organization profile.

**Examples

```
DELETE /api/profile/cowork/roles/managers/xia/ HTTP/1.1
```

GET /api/profile/:organization/roles/describe/

POST /api/profile/:organization/roles/describe/

class saas.api.roles.**RoleDescriptionListCreateView**(***kwargs*)

Lists roles by description"RoleDescription".

see *Flexible Security Framework*.

**Examples

```
GET /api/profile/cowork/roles/describe/ HTTP/1.1
```

responds

```
{
    "count": 2,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2018-01-01T00:00:00Z",
            "title": "Managers",
            "slug": "manager",
            "is_global": true,
            "roles": [
                {
                    "created_at": "2018-01-01T00:00:00Z",
                    "user": {
                        "slug": "donny",
                        "email": "donny@localhost.localdomain",
                        "full_name": "Donny Cooper",
                        "created_at": "2018-01-01T00:00:00Z"
                    },
                    "request_key": null,
                    "grant_key": null
                },
            ]
        },
        {
            "created_at": "2018-01-01T00:00:00Z",
            "name": "Contributors",
            "slug": "contributor",
            "is_global": false,
            "roles": []
        }
    ]
}
```

GET /api/profile/:organization/roles/describe/:role/

PUT /api/profile/:organization/roles/describe/:role/

DELETE /api/profile/:organization/roles/describe/:role/

class saas.api.roles.**RoleDescriptionDetailView**(***kwargs*)

Retrieves a `RoleDescription`.

see *Flexible Security Framework*.

**Examples

```
GET /api/profile/cowork/roles/describe/manager HTTP/1.1
```

responds

```
{
    "created_at": "2018-01-01T00:00:00Z",
    "name": "Managers",
    "slug": "manager",
    "is_global": true,
    "roles": [
        {
            "created_at": "2018-01-01T00:00:00Z",
            "user": {
                "slug": "donny",
                "email": "donny@localhost.localdomain",
                "full_name": "Donny Cooper",
                "created_at": "2018-01-01T00:00:00Z"
            },
            "request_key": null,
            "grant_key": null
        },
    ]
}
```

GET /api/profile/:organization/roles/:role/

POST /api/profile/:organization/roles/:role/

class saas.api.roles.**RoleListAPIView**(***kwargs*)

Lists all roles for an organization

**Examples

```
GET /api/profile/cowork/roles/ HTTP/1.1
```

responds

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2018-01-01T00:00:00Z",
            "role_description": {
                "name": "Manager",
                "slug": "manager",
                "organization": {
                    "slug": "cowork",
                    "full_name": "ABC Corp.",
                    "printable_name": "ABC Corp.",
                    "created_at": "2018-01-01T00:00:00Z",
                    "email": "support@localhost.localdomain"
                }
            },
            "user": {
                "slug": "alice",
                "email": "alice@localhost.localdomain",
                "full_name": "Alice Doe",
```

```
                "created_at": "2018-01-01T00:00:00Z"
            },
            "request_key": "1",
            "grant_key": null
        },
    ]
}
```

DELETE /api/profile/:organization/roles/:role/:user/

class saas.api.roles.**RoleDetailAPIView**(***kwargs*)

Dettach a user from one or all roles with regards to an organization, typically resulting in revoking permissions from this user to manage part of an organization profile.

**Examples

```
DELETE /api/profile/cowork/roles/managers/xia/ HTTP/1.1
```

GET /api/profile/:organization/subscribers/

class saas.api.organizations.**SubscribersAPIView**(***kwargs*)

List all `Organization` which have or had a subscription to a plan provided by `:organization`.

The value passed in the `q` parameter will be matched against:

- Organization.slug
- Organization.full_name
- Organization.email
- Organization.phone
- Organization.street_address
- Organization.locality
- Organization.region
- Organization.postal_code
- Organization.country

The result queryset can be ordered by:

- Organization.created_at
- Organization.full_name

**Examples

```
GET /api/profile/:organization/subscribers/?o=created_at&ot=desc HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
        "slug": "xia",
        "full_name": "Xia Lee",
        "created_at": "2016-01-14T23:16:55Z"
        }
```

```
    ]
}
```

GET /api/profile/:organization/subscriptions/

POST /api/profile/:organization/subscriptions/

class saas.api.subscriptions.**SubscriptionListAPIView**(***kwargs*)

GET queries all Subscription of an Organization. The queryset can be further refined to match a search filter (q) and sorted on a specific field. The returned queryset is always paginated.

**Examples

```
GET /api/profile/:organization/subscriptions/?o=created_at&ot=desc HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2016-01-14T23:16:55Z",
            "ends_at": "2017-01-14T23:16:55Z",
            "description": null,
            "organization": {
                "slug": "xia",
                "printable_name": "Xia Lee"
            },
            "plan": {
                "slug": "open-space",
                "title": "Open Space",
                "description": "open space desk, High speed internet
                                - Ethernet or WiFi, Unlimited printing,
                                Unlimited scanning, Unlimited fax service
                                (send and receive)",
                "is_active": true,
                "setup_amount": 0,
                "period_amount": 17999,
                "interval": 4,
                "app_url": "http://localhost:8020/app"
            },
            "auto_renew": true
        }
    ]
}
```

DELETE /api/profile/:organization/subscriptions/<subscribed_plan>/

class saas.api.subscriptions.**SubscriptionDetailAPIView**(***kwargs*)

Retrieves a Subscription.

**Examples

```
GET /api/profile/cowork/plans/open-space/subscriptions/xia/ HTTP/1.1
```

```
{
    ... XXX ...
}
```

9.3 Metrics API

GET /api/metrics/registered/

class saas.api.users.**RegisteredAPIView**(***kwargs*)

Lists all User which have no associated role or a role to an Organization which has no Subscription, active or inactive.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

**Examples

```
GET  /api/metrics/registered?o=created_at&ot=desc HTTP/1.1
```

responds

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "slug": "alice",
            "email": "alice@djaodjin.com",
            "full_name": "Alice Cooper",
            "created_at": "2014-01-01T00:00:00Z"
        }
    ]
}
```

GET /api/metrics/:organization/active/

class saas.api.subscriptions.**ActiveSubscriptionAPIView**(***kwargs*)

Lists all Subscription to a plan whose provider is {organization} and which are currently in progress.

Optionnaly when an ends_at query parameter is specified, returns a queryset of Subscription that were active at ends_at. When a start_at query parameter is specified, only considers Subscription that were created after start_at.

The queryset can be filtered for at least one field to match a search term (q).

Query results can be ordered by natural fields (o) in either ascending or descending order (ot).

**Examples

```
GET /api/metrics/cowork/active?o=created_at&ot=desc HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2016-01-14T23:16:55Z",
            "ends_at": "2017-01-14T23:16:55Z",
            "description": null,
            "organization": {
```

```
                "slug": "xia",
                "printable_name": "Xia Lee"
            },
            "plan": {
                "slug": "open-space",
                "title": "Open Space",
                "description": "open space desk, High speed internet
                            - Ethernet or WiFi, Unlimited printing,
                            Unlimited scanning, Unlimited fax service
                            (send and receive)",
                "is_active": true,
                "setup_amount": 0,
                "period_amount": 17999,
                "interval": 4,
                "app_url": "http://localhost:8020/app"
            },
            "auto_renew": true
        }
    ]
}
```


GET /api/metrics/:organization/balances/

class saas.api.metrics.**BalancesAPIView**(***kwargs*)

Generate a table of revenue (rows) per months (columns).

**Examples

```
GET /api/metrics/cowork/balances HTTP/1.1
```


```
{
    "title": "Balances",
    "scale": 0.01,
    "unit": "usd",
    "table": [
        {
            "key": "Income",
            "values": [
                ["2014-09-01T00:00:00Z", 0],
                ["2014-10-01T00:00:00Z", 1532624],
                ["2014-11-01T00:00:00Z", 2348340],
                ["2014-12-01T00:00:00Z", 3244770],
                ["2015-01-01T00:00:00Z", 5494221],
                ["2015-02-01T00:00:00Z", 7214221],
                ["2015-03-01T00:00:00Z", 8444221],
                ["2015-04-01T00:00:00Z", 9784221],
                ["2015-05-01T00:00:00Z", 12784221],
                ["2015-06-01T00:00:00Z", 14562341],
                ["2015-07-01T00:00:00Z", 16567341],
                ["2015-08-01T00:00:00Z", 17893214],
                ["2015-08-06T02:24:50.485Z", 221340]
            ],
        },
        {
            "key": "Backlog",
            "values": [
                ["2014-09-01T00:00:00Z", 1712624],
                ["2014-10-01T00:00:00Z", 3698340],
```

```
                ["2014-11-01T00:00:00Z", 7214770],
                ["2014-12-01T00:00:00Z", 10494221],
                ["2015-01-01T00:00:00Z", 14281970],
                ["2015-02-01T00:00:00Z", 18762845],
                ["2015-03-01T00:00:00Z", 24258765],
                ["2015-04-01T00:00:00Z", 31937741],
                ["2015-05-01T00:00:00Z", 43002401],
                ["2015-06-01T00:00:00Z", 53331444],
                ["2015-07-01T00:00:00Z", 64775621],
                ["2015-08-01T00:00:00Z", 75050033],
                ["2015-08-06T02:24:50.485Z", 89156321]
            ],
        },
        {
            "key": "Receivable",
            "values": [
                ["2014-09-01T00:00:00Z", 0],
                ["2014-10-01T00:00:00Z", 0],
                ["2014-11-01T00:00:00Z", 0],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 0],
                ["2015-02-01T00:00:00Z", 0],
                ["2015-03-01T00:00:00Z", 0],
                ["2015-04-01T00:00:00Z", 0],
                ["2015-05-01T00:00:00Z", 0],
                ["2015-06-01T00:00:00Z", 0],
                ["2015-07-01T00:00:00Z", 0],
                ["2015-08-01T00:00:00Z", 0],
                ["2015-08-06T02:24:50.485Z", 0]
            ],
        }
    ]
}
```

GET /api/metrics/balances/:report/

class saas.api.balances.**BrokerBalancesAPIView**(***kwargs*)

Queries a balance sheet named `{report}` for the broker.

To add lines in the report see */api/metrics/balances/{report}/lines/*.

**Examples

```
GET /api/metrics/balances/taxes/ HTTP/1.1
```

responds

```
{
    "scale": 0.01,
    "unit": "usd",
    "title": "Balances: taxes",
    "table": [
        {
            "key": "Sales",
            "selector": "Receivable",
            "values": [
                ["2015-05-01T00:00:00Z", 0],
                ["2015-08-01T00:00:00Z", 0],
                ["2015-11-01T00:00:00Z", 0],
```

```
                ["2016-02-01T00:00:00Z", 0],
                ["2016-05-01T00:00:00Z", 0],
                ["2016-05-16T21:08:15.637Z", 0]
            ]
        }
    ]
}
```

GET /api/metrics/lines/:report/

POST /api/metrics/lines/:report/

class saas.api.balances.**BalanceLineListAPIView**(***kwargs*)

Queries the list of rows reported on a balance sheet named *{report}*.

**Examples

```
GET  /api/metrics/balances/taxes/lines/ HTTP/1.1
```

responds

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "title": "Sales",
            "selector": "Receivable",
            "rank": 1
        }
    ]
}
```

GET /api/metrics/:organization/churned/

class saas.api.subscriptions.**ChurnedSubscriptionAPIView**(***kwargs*)

Lists all `Subscription` to a plan whose provider is `:organization` which have ended already.

The queryset can be further filtered to a range of dates between `start_at` and `ends_at`.

The queryset can be further filtered by passing a `q` parameter. The result queryset can be ordered.

**Examples

```
GET /api/metrics/cowork/churned?o=created_at&ot=desc HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "created_at": "2016-01-14T23:16:55Z",
            "ends_at": "2017-01-14T23:16:55Z",
            "description": null,
            "organization": {
                "slug": "xia",
                "printable_name": "Xia Lee"
            },
```

```
            "plan": {
                "slug": "open-space",
                "title": "Open Space",
                "description": "open space desk, High speed internet
                            - Ethernet or WiFi, Unlimited printing,
                            Unlimited scanning, Unlimited fax service
                            (send and receive)",
                "is_active": true,
                "setup_amount": 0,
                "period_amount": 17999,
                "interval": 4,
                "app_url": "http://localhost:8020/app"
            },
            "auto_renew": true
        }
    ]
}
```

GET /api/metrics/:organization/coupons/:coupon/

class saas.api.metrics.**CouponUsesAPIView**(***kwargs*)

Queries a page (PAGE_SIZE records) of Coupon usage.

The queryset can be filtered to a range of dates ([start_at, ends_at]) and for at least one field to match a search term (q).

The result queryset can be ordered by passing an o (field name) and ot (asc or desc) parameter.

**Examples

```
GET /api/metrics/cowork/coupons/DIS100/ HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "user": {
                "slug": "xia",
                "email": "xia@localhost.localdomain",
                "full_name": "Xia Doe",
                "created_at": "2012-09-14T23:16:55Z"
            },
            "plan": "basic",
            "created_at": "2014-01-01T09:00:00Z"
        }
    ]
}
```

GET /api/metrics/:organization/customers/

class saas.api.metrics.**CustomerMetricAPIView**(***kwargs*)

Produce revenue stats

**Examples

```
GET /api/metrics/cowork/customers HTTP/1.1
```

```
{
    "title": "Customers"
    "table": [
        {
            "key": "Total # of Customers",
            "values": [
                ["2014-10-01T00:00:00Z", 15],
                ["2014-11-01T00:00:00Z", 17],
                ["2014-12-01T00:00:00Z", 19],
                ["2015-01-01T00:00:00Z", 19],
                ["2015-02-01T00:00:00Z", 25],
                ["2015-03-01T00:00:00Z", 29],
                ["2015-04-01T00:00:00Z", 37],
                ["2015-05-01T00:00:00Z", 43],
                ["2015-06-01T00:00:00Z", 46],
                ["2015-07-01T00:00:00Z", 48],
                ["2015-08-01T00:00:00Z", 54],
                ["2015-08-06T05:20:24.537Z", 60]
            ]
        },
        {
            "key": "# of new Customers"
            "values": [
                ["2014-10-01T00:00:00Z", 2],
                ["2014-11-01T00:00:00Z", 2],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 6],
                ["2015-02-01T00:00:00Z", 4],
                ["2015-03-01T00:00:00Z", 8],
                ["2015-04-01T00:00:00Z", 6],
                ["2015-05-01T00:00:00Z", 3],
                ["2015-06-01T00:00:00Z", 2],
                ["2015-07-01T00:00:00Z", 6],
                ["2015-08-01T00:00:00Z", 7],
                ["2015-08-06T05:20:24.537Z", 0]
            ]
        },
        {
            "key": "# of churned Customers"
            "values": [
                ["2014-10-01T00:00:00Z", 0],
                ["2014-11-01T00:00:00Z", 0],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 0],
                ["2015-02-01T00:00:00Z", 0],
                ["2015-03-01T00:00:00Z", 0],
                ["2015-04-01T00:00:00Z", 0],
                ["2015-05-01T00:00:00Z", 0],
                ["2015-06-01T00:00:00Z", 0],
                ["2015-07-01T00:00:00Z", 0],
                ["2015-08-01T00:00:00Z", 1],
                ["2015-08-06T05:20:24.537Z", 60]
            ]
        },
        {
            "key": "Net New Customers",
            "values": [
                ["2014-10-01T00:00:00Z", 2],
```

```
                ["2014-11-01T00:00:00Z", 2],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 6],
                ["2015-02-01T00:00:00Z", 4],
                ["2015-03-01T00:00:00Z", 8],
                ["2015-04-01T00:00:00Z", 6],
                ["2015-05-01T00:00:00Z", 3],
                ["2015-06-01T00:00:00Z", 2],
                ["2015-07-01T00:00:00Z", 6],
                ["2015-08-01T00:00:00Z", 6],
                ["2015-08-06T05:20:24.537Z", -60]
            ]
        }
    ],
    "extra": [
        {
            "key": "% Customer Churn",
            "values": [
                ["2014-10-01T00:00:00Z", 0],
                ["2014-11-01T00:00:00Z", 0.0],
                ["2014-12-01T00:00:00Z", 0.0],
                ["2015-01-01T00:00:00Z", 0.0],
                ["2015-02-01T00:00:00Z", 0.0],
                ["2015-03-01T00:00:00Z", 0.0],
                ["2015-04-01T00:00:00Z", 0.0],
                ["2015-05-01T00:00:00Z", 0.0],
                ["2015-06-01T00:00:00Z", 0.0],
                ["2015-07-01T00:00:00Z", 0.0],
                ["2015-08-01T00:00:00Z", 2.08],
                ["2015-08-06T05:20:24.537Z", 111.11]
            ]
        }
    ]
}
```

GET /api/metrics/:organization/funds/

class saas.api.metrics.**RevenueMetricAPIView**(***kwargs*)

Produces sales, payments and refunds over a period of time.

**Examples

```
GET /api/metrics/cowork/funds/ HTTP/1.1
```

```
{
    "title": "Amount",
    "scale": 0.01,
    "unit": "usd",
    "table": [
        {
            "key": "Total Sales",
            "values": [
                ["2014-10-01T00:00:00Z", 1985716],
                ["2014-11-01T00:00:00Z", 3516430],
                ["2014-12-01T00:00:00Z", 3279451],
                ["2015-01-01T00:00:00Z", 3787749],
                ["2015-02-01T00:00:00Z", 4480875],
                ["2015-03-01T00:00:00Z", 5495920],
```

```
                ["2015-04-01T00:00:00Z", 7678976],
                ["2015-05-01T00:00:00Z", 11064660],
                ["2015-06-01T00:00:00Z", 10329043],
                ["2015-07-01T00:00:00Z", 11444177],
                ["2015-08-01T00:00:00Z", 10274412],
                ["2015-08-06T04:59:14.721Z", 14106288]
            ]
        },
        {
            "key": "New Sales",
            "values": [
                ["2014-10-01T00:00:00Z", 0],
                ["2014-11-01T00:00:00Z", 0],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 0],
                ["2015-02-01T00:00:00Z", 0],
                ["2015-03-01T00:00:00Z", 0],
                ["2015-04-01T00:00:00Z", 0],
                ["2015-05-01T00:00:00Z", 0],
                ["2015-06-01T00:00:00Z", 0],
                ["2015-07-01T00:00:00Z", 0],
                ["2015-08-01T00:00:00Z", 0],
                ["2015-08-06T04:59:14.721Z", 0]
            ]
        },
        {
            "key": "Churned Sales",
            "values": [
                ["2014-10-01T00:00:00Z", 0],
                ["2014-11-01T00:00:00Z", 0],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 0],
                ["2015-02-01T00:00:00Z", 0],
                ["2015-03-01T00:00:00Z", 0],
                ["2015-04-01T00:00:00Z", 0],
                ["2015-05-01T00:00:00Z", 0],
                ["2015-06-01T00:00:00Z", 0],
                ["2015-07-01T00:00:00Z", 0],
                ["2015-08-01T00:00:00Z", 0],
                ["2015-08-06T04:59:14.721Z", 0]
            ]
        },
        {
            "key": "Payments",
            "values": [
                ["2014-10-01T00:00:00Z", 1787144],
                ["2014-11-01T00:00:00Z", 3164787],
                ["2014-12-01T00:00:00Z", 2951505],
                ["2015-01-01T00:00:00Z", 3408974],
                ["2015-02-01T00:00:00Z", 4032787],
                ["2015-03-01T00:00:00Z", 4946328],
                ["2015-04-01T00:00:00Z", 6911079],
                ["2015-05-01T00:00:00Z", 9958194],
                ["2015-06-01T00:00:00Z", 9296138],
                ["2015-07-01T00:00:00Z", 10299759],
                ["2015-08-01T00:00:00Z", 9246970],
                ["2015-08-06T04:59:14.721Z", 12695659]
            ]
```

```
        },
        {
            "key": "Refunds",
            "values": [
                ["2014-10-01T00:00:00Z", 0],
                ["2014-11-01T00:00:00Z", 0],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 0],
                ["2015-02-01T00:00:00Z", 0],
                ["2015-03-01T00:00:00Z", 0],
                ["2015-04-01T00:00:00Z", 0],
                ["2015-05-01T00:00:00Z", 0],
                ["2015-06-01T00:00:00Z", 0],
                ["2015-07-01T00:00:00Z", 0],
                ["2015-08-01T00:00:00Z", 0],
                ["2015-08-06T04:59:14.721Z", 0]
            ]
        }
    ],
}
```

GET /api/metrics/:organization/plans/

class saas.api.metrics.**PlanMetricAPIView**(***kwargs*)

Produce plan stats

**Examples

```
GET /api/metrics/cowork/plans HTTP/1.1
```

```
{
    "title": "Active Subscribers",
    "table": [
        {
            "is_active": true,
            "key": "open-space",
            "location": "/profile/plan/open-space/",
            "values": [
                ["2014-09-01T00:00:00Z", 4],
                ["2014-10-01T00:00:00Z", 5],
                ["2014-11-01T00:00:00Z", 6],
                ["2014-12-01T00:00:00Z", 6],
                ["2015-01-01T00:00:00Z", 6],
                ["2015-02-01T00:00:00Z", 9],
                ["2015-03-01T00:00:00Z", 9],
                ["2015-04-01T00:00:00Z", 9],
                ["2015-05-01T00:00:00Z", 11],
                ["2015-06-01T00:00:00Z", 11],
                ["2015-07-01T00:00:00Z", 14],
                ["2015-08-01T00:00:00Z", 16],
                ["2015-08-06T05:37:50.004Z", 16]
            ]
        },
        {
            "is_active": true,
            "key": "open-plus",
            "location": "/profile/plan/open-plus/",
            "values": [
```

```
                ["2014-09-01T00:00:00Z", 7],
                ["2014-10-01T00:00:00Z", 8],
                ["2014-11-01T00:00:00Z", 9],
                ["2014-12-01T00:00:00Z", 9],
                ["2015-01-01T00:00:00Z", 12],
                ["2015-02-01T00:00:00Z", 13],
                ["2015-03-01T00:00:00Z", 18],
                ["2015-04-01T00:00:00Z", 19],
                ["2015-05-01T00:00:00Z", 19],
                ["2015-06-01T00:00:00Z", 20],
                ["2015-07-01T00:00:00Z", 23],
                ["2015-08-01T00:00:00Z", 25],
                ["2015-08-06T05:37:50.014Z", 25]
            ]
        },
        {
            "is_active": true,
            "key": "private",
            "location": "/profile/plan/private/",
            "values": [
                ["2014-09-01T00:00:00Z", 3],
                ["2014-10-01T00:00:00Z", 3],
                ["2014-11-01T00:00:00Z", 3],
                ["2014-12-01T00:00:00Z", 3],
                ["2015-01-01T00:00:00Z", 6],
                ["2015-02-01T00:00:00Z", 7],
                ["2015-03-01T00:00:00Z", 10],
                ["2015-04-01T00:00:00Z", 15],
                ["2015-05-01T00:00:00Z", 16],
                ["2015-06-01T00:00:00Z", 17],
                ["2015-07-01T00:00:00Z", 17],
                ["2015-08-01T00:00:00Z", 18],
                ["2015-08-06T05:37:50.023Z", 18]
            ]
        }
    ],
    "extra": [
        {
            "key": "churn",
            "values": [
                ["2014-09-01T00:00:00Z", 0],
                ["2014-10-01T00:00:00Z", 0],
                ["2014-11-01T00:00:00Z", 0],
                ["2014-12-01T00:00:00Z", 0],
                ["2015-01-01T00:00:00Z", 0],
                ["2015-02-01T00:00:00Z", 0],
                ["2015-03-01T00:00:00Z", 0],
                ["2015-04-01T00:00:00Z", 0],
                ["2015-05-01T00:00:00Z", 0],
                ["2015-06-01T00:00:00Z", 0],
                ["2015-07-01T00:00:00Z", 0],
                ["2015-08-01T00:00:00Z", 1],
                ["2015-08-06T05:37:50.031Z", 1]
            ]
        }
    ]
}
```

9.4 Search API

At times, we might be looking to grant a `User` permissions to an `Organization` through a `Role` (manager, etc.), or we might be looking to request access to an `Organization` on behalf of a `User`. Both features might benefit from an auto-complete suggestions list. The two following API end point will list all `Organization` and `User` in the database regardless of their associations.

class `saas.api.organizations.`**`OrganizationListAPIView`**(***kwargs*)

Queries all `Organization`.

**Examples

```
GET /api/profile/?o=created_at&ot=desc HTTP/1.1
```

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [{
        "slug": "xia",
        "full_name": "Xia Lee",
        "printable_name": "Xia Lee",
        "created_at": "2016-01-14T23:16:55Z"
    }]
}
```

class `saas.api.users.`**`UserListAPIView`**(***kwargs*)

Queries a page (`PAGE_SIZE` records) of `User`.

The queryset can be filtered to a range of dates ([`start_at`, `ends_at`]) and for at least one field to match a search term (`q`).

Query results can be ordered by natural fields (`o`) in either ascending or descending order (`ot`).

**Examples

```
GET  /api/users/?o=created_at&ot=desc HTTP/1.1
```

responds

```
{
    "count": 1,
    "next": null,
    "previous": null,
    "results": [
        {
            "slug": "alice",
            "email": "alice@djaodjin.com",
            "full_name": "Alice Cooper",
            "created_at": "2014-01-01T00:00:00Z"
        }
    ]
}
```

CHAPTER 10

Database Models

Subscribers and providers are both instances of `Organization`. This is done such that one can be a subscriber to a `Plan` for a service hosted on the broker website as well as itself a provider to other subscribers. (ex: An organization can provide a CRM tool to subscribers while paying another app, also hosted on the broker platform, to display usage analytics of its own product). It is possible to implement a *symmetric double-entry bookkeeping ledger* by having a single model `Organization`.

Typically if you are self-hosting a pure Software-as-a-Service, as opposed to building a marketplace, you will define a single provider which incidently is also the the broker (See *examples*).

A billing profile (credit card and deposit bank account) is represented by an `Organization`. An `Organization` subscriber subscribes to services provided by another `Organization` provider through a `Subscription` to a `Plan`. An `Organization` represents a billing profile. The `processor_card_key` and `processor_deposit_key` fields are respectively used when an organization acts as a subscriber or provider in the subscription relationship.

There are no mechanism provided to authenticate as an `Organization`. Instead `User` authenticate with the application (through a login page or an API token). They are then able to access URLs related to an `Organization` based on their relation with that `Organization` as implemented by a `RoleDescription`. For historical reasons, two roles are often implemented: managers and contributors (for details see *Security*).

`Organization` can be semantically separated in four categories, processor, broker, providers and subscribers.

- subscribers: organizations that subscribe to one or multiple `Plan`.
- providers: organizations that provides plans others can subscribe to.
- broker: The provider that controls the website.
- processor: The organization / *backend* actually processing the charges.

In a pure Software-as-a-Service setup, there is only one provider which is by definition the broker.

In a marketplace setup, there might be multiple providers even though there is only one broker, always. The broker controls the domain name on which the site is hosted.

CHAPTER 11

Processor Backends

There always needs to be a special `Organization`, the processor in your database. The processor represents the payment processor backend in charge and deposit transactions.

`Organization` with pk=1 will be considered to be the default processor. This can be overridden by defining `PROCESSOR_ID` in the settings block.

```
$ cat settings.py

SAAS = {
    'PROCESSOR_ID': 1
}
```

11.1 Razorpay configuration

Install Razorpay pip package

```
$ pip install razorpay
```

Go to your Razorpay dashboard "API Keys", click on "Generate Key", then copy/paste the keys into your project settings.py

```
SAAS = {
    'PROCESSOR': {
        'BACKEND': 'saas.backends.razorpay_processor.RazorpayBackend',
        'PRIV_KEY': "...",
        'PUB_KEY': "...",
    }
}
```

11.2 Stripe configuration

The Stripe backend works in 3 different modes:

- LOCAL
- FORMWARD
- REMOTE

In LOCAL mode, Stripe Customer and Charge objects are created on the Stripe Account identified by settings.PROCESSOR['PRIV_KEY']. All transfers are made to the bank account associated to that account.

In FORWARD mode, Stripe Customer and Charge objects are also created on the Stripe account identified by settings.PROCESSOR['PRIV_KEY'] but each Charge is tied automatically to a Stripe Transfer to a Stripe Connect Account.

In REMOTE mode, Stripe Customer and Charge objects are created on the Stripe Connect Account.

To configure Stripe Connect, follow the instructions at https://stripe.com/docs/connect,

Go to "Account Settings" > "Connect"

Edit the redirect_url and copy/paste the keys into your project settings.py

```
SAAS = {
    'PROCESSOR': {
        'BACKEND': 'saas.backends.stripe_processor.StripeBackend',
        'PRIV_KEY': "...",
        'PUB_KEY': "...",
    # optional
        'CLIENT_ID': "...",
        'MODE': "...",
    }
}
```

CHAPTER 12

Indices and tables

- search

Python Module Index

S

HTTP Routing Table

/api

/app

/billing

/legal

/pricing

/profile

/provider

/redeem

/users

Index