Achei a melhor opção, para realizar a tarefa, utilizar Django REST Framework. Poupando tempo na estruturação do REST, e ainda podemos utilizar class-based views. Assim temos os dispatch methods prontos: get, put, post, delete, patch, dispatch.

http://www.django-rest-framework.org/api-guide/views/

Comportamento

GET: read
PUT: update
POST: create
DELETE: delete
head: headers, no body

Dispatch URL

/api/v1/{endpoint}

Endpoints

GET /produtos/

Retorna uma lista de produtos. {pk:”produto_pk”, nome:”nome_produto”, preco_medio:”preco_medio”,created:”data”,updated:”data”,is_active:”True”}

GET /produtos/<item_pk>/

Retorna toda informação disponível de um produto, mesmo ele deletado (is_active=False). {pk:”produto_pk”, nome:”nome_produto”, preco_medio:”preco_medio”,created:”data”,updated:”data”,is_active:”True”}

POST /produtos/

Cria um novo produto. Objeto JSON com 1 campo obrigatório: **nome** (com 180 caracteres sem quebra de linha), **preco_medio** [**opcional*] (decimal com duas casas), **is_active** [**opcional*] (default True). Em caso de sucesso será retornado um objeto do novo produto criado.

DELETE /produtos/<item_pk>/

Marca produto como não ativo e não apaga o objeto do banco de dados, is_active=False. Assim GET /produtos/ não irá mais mostrar esse produto. Em caso de sucesso será retornado um objeto alterado.

PUT /produtos/<item_pk>/

Modifica um produto existente. É permitido alterar somente alguns campos, partial update. Objeto JSON com pelo menos 1 campo obrigatório: **nome** (com 180 caracteres sem quebra de linha), **preco_medio** [**opcional*] (decimal com duas casas), **is_active** [**opcional*] (bool).

GET /safras/

Retorna uma lista de safras. {pk:”safra_pk”, nome:”nome_safra”, data_inicio:”data”, data_fim:”data”,created:”data”,updated:”data”,is_active:”True”}

GET /safras/<item_pk>/

Retorna toda informação disponível de uma safra, mesmo ela deletada (is_active=False). {pk:”safra_pk”, nome:”nome_safra”, data_inicio:”data”, data_fim:”data”,created:”data”,updated:”data”,is_active:”False”}

POST /safras/

Cria uma nova safra. Objeto JSON com 3 campos obrigatórios: **nome** (com 180 caracteres sem quebra de linha), **data_inicio** (datetime), **data_fim** (datetime), **is_active** [**opcional*] (default True). Em caso de sucesso será retornado um objeto do novo da safra criada.

DELETE /safras/<item_pk>/

Marca safra como não ativo, is_active=False, mas não apaga obejto do banco de dados. Assim GET /safra/ não irá mais mostrar essa safra. Em caso de sucesso será retornado um objeto safra alterado.

PUT /safras/<item_pk>/

Modifica uma safra existente. Objeto JSON com pelo menos 1 campos: **nome** (com 180 caracteres sem quebra de linha), **data_inicio** (datetime), **data_fim**(datetime), **is_active** (bool).

GET /servicos/

Retorna uma lista de servicos. {pk:”servico_pk”, nome:”nome_servico”, data_inicio:”data”, data_fim:”data”,created:”data”,updated:”data”,is_active:”False”}

GET /servicos/<item_pk>/

Retorna toda informação disponível de um servico, mesmo ele deletado (is_active=False). {pk:”servico_pk”, nome:”nome_servico”, data_inicio:”data”, data_fim:”data”,safra:”Safra object”, produtos:”Valores objects”, created:”data”,updated:”data”,is_active:”False”}

POST /servicos/

Cria uma nova safra. Objeto JSON com 5 campos obrigatórios: **nome** (com 180 caracteres sem quebra de linha), **data_inicio** (datetime), **data_fim** (datetime), **safra** (safra object), **produtos** (valores object), **is_active** [**opcional*] (default False). Em caso de sucesso será retornado um objeto do novo do servico criado.

DELETE /servicos/<item_pk>/

Marca servico como não ativo, is_active=False, mas não apaga do banco de dados. Assim GET /servicos/ não irá mais mostrar esse servico. Em caso de sucesso será retornado um objeto servico alterado.

PUT /servicos/<item_pk>/

Modifica um servico existente. Objeto JSON com pelo menos 1 campo obrigatório: **nome** (com 180 caracteres sem quebra de linha), **data_inicio** (datetime), **data_fim** (datetime), **safra** (safra object), **produtos** (valores object), **is_active** [**opcional*] (bool).

Notas

1. Caso um objeto não seja encontrado com <item_pk> indicado será retornado 404.
2. Todas as respostas e envios são JSON objetos.
3. em Serviços save e update methods precisam ter nested relationships é necessário sobrescrever os métodos.

Json / XML / ETC

Incluindo novos parsers e renders podemos facilmente manipular o formato de entrada e saída dos dados.

http://www.django-rest-framework.org/api-guide/renderers/
http://www.django-rest-framework.org/api-guide/parsers/
http://www.django-rest-framework.org/api-guide/settings/

```
REST_FRAMEWORK = {
    'DEFAULT_PARSER_CLASSES': (
        'rest_framework_xml.parsers.XMLParser',
        'rest_framework.parsers.JSONParser',
    ),
    'DEFAULT_RENDERER_CLASSES': (
        'rest_framework_xml.renderers.XMLRenderer',
        'rest_framework.renderers.JSONRenderer',
        'rest_framework.renderers.BrowsableAPIRenderer',

    ),
}
```

Na view (APIView) escolher qual render será utilizado:
renderer_classes = (JSONRenderer,).

Para XML é necessário django rest framework xml:
$ pip install djangorestframework-xml

TODO

- Validação dos models com data de inicio e fim: a data de fim não pode ser menor que a de início.
- http://www.django-rest-framework.org/api-guide/renderers/ fornece especificação para retornar diferentes tipos de dados.
- Autenticação e permissões para utilizar a API
- FIX: Produto.objects.get_or_create(**produto_data) pode retornar mais de um objeto.
- Custo total de um serviço é a soma do preço dos produtos utilizados. Pode ser alterado para outra forma de rateio dos preços.
- Melhorar is_active managers.