frepple-data-admin

Release 1.0.0

https://frepple.org

Sep 28, 2021

CONTENTS

A django-based framework for rapid development of data management applications.

Spreadsheet are omnipresent in any company. When the process grows beyond the limits of Excel, you need a tool to manage your data in a more scalable, robust, secure and collaborative way.

FrePPLe data admin provides a framework for developing such data management applications with a low code, short learning curve approach.

Home page: https://frepple.org/

Live demo: https://demo.frepple.com/

Source code: https://github.com/frePPLe/frepple-data-admin

User forum: https://github.com/frePPLe/frepple-data-admin/discussions

Documentation: https://frepple-data-admin.readthedocs.io/en/latest/

Release notes: https://frepple-data-admin.readthedocs.io/en/latest/release-notes.html

CHAPTER

ONE

GETTING STARTED

This chapter aims to get you started quickly and easily.

1.1 Installation

- *Prerequisites*
- *Install the Python package*
- *Create a PostgreSQL database user*

1.1.1 Prerequisites

You will need:

- Python from https://www.python.org/ (any version >= 3.6)
- PostgreSQL from https://www.postgresql.org/ (any version >= 9)

1.1.2 Install the Python package

Download the source code from github https://github.com/frePPLe/frepple-data-admin into a local folder on your machine.

Open a command prompt in that folder and install the third party Python packages data-admin depends on. Using a Python virtual environment is supported.

```
pip3 install -r requirements.txt
```

1.1.3 Create a PostgreSQL database user

Next, create a database user for data admin. From a psql prompt or pgadmin, you can do this with the following SQL command:

```
create role frepple with login superuser password 'frepple';
```

The role name, password and privileges can be changed to your taste. The above is just a quick default to get started with.

1.2 Running the example app

In this section, we'll run the example application that comes with the installation.

- *Edit the djangosettings.py configuration file*
- example_initialize
- *Run the web server*

1.2.1 Edit the djangosettings.py configuration file

The python package provides a configuration file "djangosettings.py". Open this file with a text editor and review the following sections:

- DATABASES:

  The minimal content for this block is as follows. The USER and PASSWORD should match the database user you created in the previous step.

```
DATABASES = {
  "default": {
    "ENGINE": "django.db.backends.postgresql",
    # Database name
    "NAME": "data_admin",
    # Role name when using md5 authentication.
    # Leave as an empty string when using peer or ident authencation.
    "USER": "frepple",
    # Role password when using md5 authentication.
    # Leave as an empty string when using peer or ident authencation.
    "PASSWORD": "frepple",
    # When using TCP sockets specify the hostname, the ip4 address or the ip6␣
↪address here.
    # Leave as an empty string to use Unix domain socket ("local" lines in pg_hba.
↪conf).
    "HOST": "",
    # Specify the port number when using a TCP socket.
    "PORT": "",
    "OPTIONS": {},
    "CONN_MAX_AGE": 60,
    "TEST": {
      "NAME": "test_data_admin"  # Database name used when running the test suite.
      },
    "FILEUPLOADFOLDER": os.path.normpath(
      os.path.join(FREPPLE_LOGDIR, "data", "default")
      ),
    "SECRET_WEBTOKEN_KEY": SECRET_KEY,
    }
  }
```

- INSTALLED_APPS:

  This setting configures the apps that will be deployed on your web server.

  The minimal content for this block is as follows. Notice the "example1" app at a specific place in the list.

```
INSTALLED_APPS = (
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "data_admin.boot",
    "data_admin_examples.example1",   # <<< The example app
    "data_admin.execute",
    "data_admin.common",
    "django_filters",
    "rest_framework",
    "django_admin_bootstrapped",
    "django.contrib.admin",
)
```

1.2.2 Initialize the database

With the following commands we will create a database, build all database tables and load some sample data.

```
>> frepplectl.py createdatabase

    Executing SQL statement: create database "data_admin" encoding = 'UTF8'


>> frepplectl.py migrate

    Operations to perform:
      Apply all migrations: admin, auth, common, contenttypes, example1,␣
↪execute
    Running migrations:
      Applying contenttypes.0001_initial... OK
      Applying contenttypes.0002_remove_content_type_name... OK
      Applying auth.0001_initial... OK
      Applying auth.0002_alter_permission_name_max_length... OK
      Applying auth.0003_alter_user_email_max_length... OK
      Applying auth.0004_alter_user_username_opts... OK
      Applying auth.0005_alter_user_last_login_null... OK
      Applying auth.0006_require_contenttypes_0002... OK
      Applying auth.0007_alter_validators_add_error_messages... OK
      Applying auth.0008_alter_user_username_max_length... OK
      Applying auth.0009_alter_user_last_name_max_length... OK
      Applying auth.0010_alter_group_name_max_length... OK
      Applying auth.0011_update_proxy_permissions... OK
      Applying common.0001_initial... OK
      Applying admin.0001_initial... OK
      Applying admin.0002_logentry_remove_auto_add... OK
      Applying admin.0003_logentry_add_action_flag_choices... OK
      Applying example1.0001_initial... OK
      Applying execute.0001_initial... OK
```

(continues on next page)

(continued from previous page)

```
>> frepplectl.py loaddata example1

     Installed 29 object(s) from 1 fixture(s)
```

1.2.3 Run the web server

Now, we can run the web server and use data-admin from your browser. If all goes well, you will see a message with the URL.

```
>> frepplectl.py runserver

     INFO Watching for file changes with StatReloader
     Performing system checks...

     System check identified no issues (1 silenced).
     Django version 2.2.17, using settings 'data_admin.settings'
     Starting development server at http://127.0.0.1:8000/
     Quit the server with CTRL-BREAK.
```

You can now open your favorite browser on http://127.0.0.1:8000/. A default user **admin** is created automatically with password **admin**.

1.3 Your first app

In this section, you'll start building your own app.

- *Initialize your app*
- *Register your app*
- *Define the database models*
- *Create tables and fields in the database*
- *Define a REST API for your models*
- *Create editing forms for your models*
- *Define new reports*
- *Register the URLs of the new reports*
- *Add the reports to the menu*
- *Add demo data*
- *Add custom administration commands*
- *Add unit tests*
- *Even more information!*

1.3.1 Initialize your app

In the previous section you used the example app. In your installation folder you should be able to find its source code in the folder data_admin_examples/example1. You can also see it online at https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin_examples/example1

An app is nothing more than a python package with a specifically structured content. The steps will walk you through the process of adding your own app.

Create a new folder my_app in your data_admin_examples folder and add an empty __init__.py file in it.

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
```

1.3.2 Register your app

Next, register the app in the web server.

Open the djangosettings.py file and add a new line in the INSTALLED_APPS sections. Note that the ordering of the apps is important - apps higher in the list can override functionality of apps lower in the list.

```
INSTALLED_APPS = (
    "django.contrib.auth",
    "django.contrib.contenttypes",
    "django.contrib.messages",
    "django.contrib.staticfiles",
    "data_admin.boot",
    "data_admin_examples.example1",    # <<< The example app
    "data_admin_examples.my_app",    # <<< Your own app
    "data_admin.execute",
    "data_admin.common",
    "django_filters",
    "rest_framework",
    "django_admin_bootstrapped",
    "django.contrib.admin",
)
```

1.3.3 Define the database models

Add a file called **models.py** to describe new database models. It defines the database tables, their fields and indexes.

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
```

A minimal example looks as follows. An online copy is available on https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin/docs/getting_started/my_app/models.py

```
from django.db import models
from django.utils.translation import ugettext_lazy as _
from data_admin.common.models import AuditModel

class My_Model(AuditModel):
    # Database fields
    name = models.CharField(_("name"), max_length=300, primary_key=True)
    charfield = models.CharField(
        _("charfield"),
        max_length=300,
        null=True,
        blank=True,
        help_text=_("A sample character field"),
    )
    booleanfield = models.BooleanField(
        _("booleanfield"),
        blank=True,
        default=True,
        help_text=_("A sample boolean field"),
    )
    decimalfield = models.DecimalField(
        _("decimalfield"),
        max_digits=20,
        decimal_places=8,
        default="0.00",
        help_text=_("A sample decimal field"),
    )

    class Meta(AuditModel.Meta):
        db_table = "my_model"  # Name of the database table
        verbose_name = _("my model")  # A translatable name for the entity
        verbose_name_plural = _("my models")  # Plural name
        ordering = ["name"]
```

This file only declares the model structure. The actual table will be created in a later step.

You can find all details on models and fields on https://docs.djangoproject.com/en/2.2/ref/models/fields/

1.3.4 Create tables and fields in the database

Now we create database tables in the PostgreSQL database for each of your models. This is done by in two steps.

In the **first step** we generate a Python file that defines the evolution of your database model.

```
frepplectl makemigrations my_app

   Migrations for 'my_app':
     data_admin_examples\my_app\migrations\0001_initial.py
       - Create model My_Model
```

The command created a new folder in your app:

```
data_admin_examples
   |- example1
   |- my-app
   |   |- __init__.py
   |   |- models.py
   |   |- migrations
   |       |- __init__.py
   |       |- 0001_initial.py
```

It is very important to run the makemigration script after EVERY update of the models.py file. For every change an extra migration file is generated.

In a **second step** you will actually execute the migrations generated in the previous step and create the database tables. This command will incrementally bring the database schema up to date. The database schema migration allows upgrading between different versions of your app without loss of data and without recreating the database from scratch.

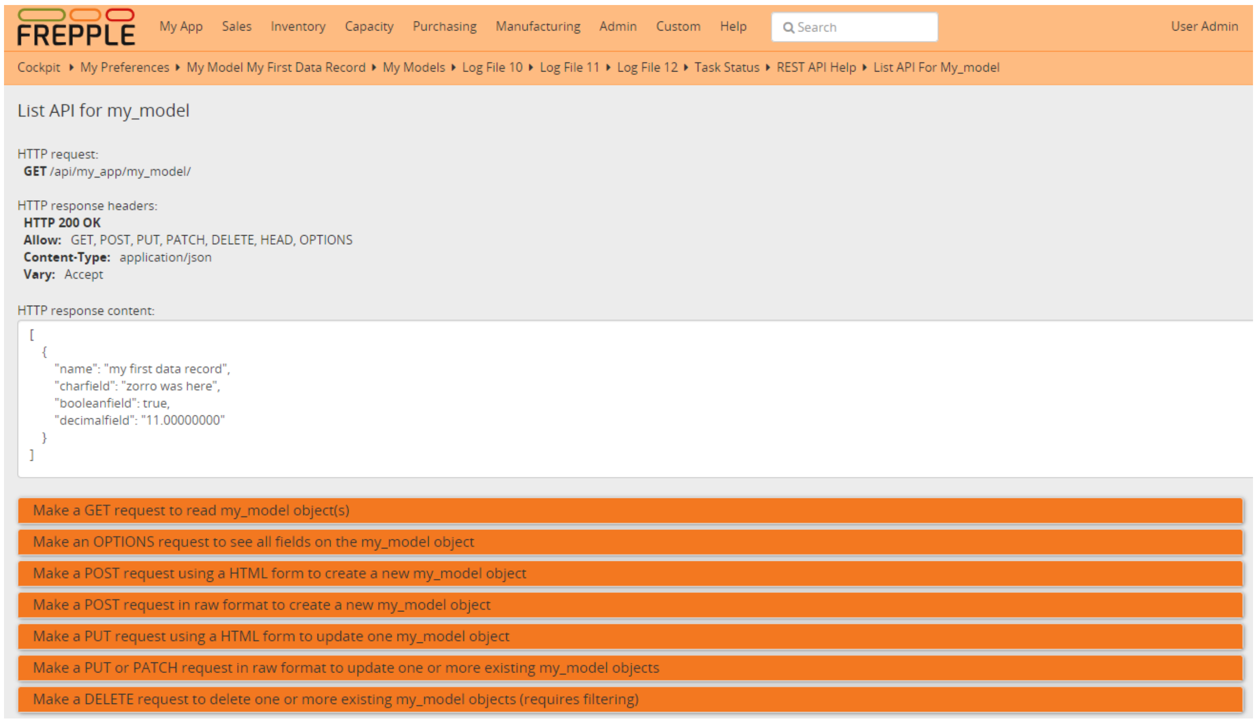
```
frepplectl.py migrate

   Operations to perform:
     Apply all migrations: admin, auth, common, contenttypes, example1, execute, my_app
   Running migrations:
     Applying my_app.0001_initial... OK
```

The first step is done by the developer that is updating the models.py file. The second step is executed by everybody that is installing your app (or upgrading it to a new release). You can find all details on migrations on https://docs.djangoproject.com/en/2.2/topics/migrations/

1.3.5 Define a REST API for your models

The file **serializers.py** defines a REST API for your models. You can explore the REST API from the menu "help/REST API help". An online copy is available on https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin/docs/getting_started/my_app/serializers.py

```
data_admin_examples
   |- example1
   |- my-app
   |   |- __init__.py
   |   |- models.py
   |   |- migrations
   |   |   |- __init__.py
   |   |   |- 0001_initial.py
   |   |- serializers.py
```


FREPPLE
My App
Sales
Inventory
Capacity
Purchasing
Manufacturing
Admin
Custom
Help
Search
User Admin
Cockpit
My Preferences
My Model My First Data Record
My Models
Log File 10
Log File 11
Log File 12
Task Status
REST API Help
List API For My_model

List API for my_model

HTTP request:
GET /api/my_app/my_model/

HTTP response headers:
```
HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept
```

HTTP response content:
```
[
  {
    "name": "my first data record",
    "charfield": "zorro was here",
    "booleanfield": true,
    "decimalfield": "11.00000000"
  }
]
```

Make a GET request to read my_model object(s)
Make an OPTIONS request to see all fields on the my_model object
Make a POST request using a HTML form to create a new my_model object
Make a POST request in raw format to create a new my_model object
Make a PUT request using a HTML form to update one my_model object
Make a PUT or PATCH request in raw format to update one or more existing my_model objects
Make a DELETE request to delete one or more existing my_model objects (requires filtering)

```python
from django_filters import rest_framework as filters
from rest_framework_bulk.drf3.serializers import BulkListSerializer, BulkSerializerMixin
from data_admin.common.api.views import (
    frePPleListCreateAPIView,
    frePPleRetrieveUpdateDestroyAPIView,
)
from data_admin.common.api.serializers import ModelSerializer
from .models import My_Model

class MyModelFilter(filters.FilterSet):
    class Meta:
        model = My_Model
        fields = {
            "name": ["exact", "in", "contains"],
            "charfield": ["exact", "contains"],
            "booleanfield": ["exact"],
            "decimalfield": ["exact", "in", "gt", "gte", "lt", "lte"],
            "source": ["exact", "in"],
            "lastmodified": ["exact", "in", "gt", "gte", "lt", "lte"],
        }
        filter_fields = ("name", "charfield", "booleanfield", "decimalfield")


class MyModelSerializer(BulkSerializerMixin, ModelSerializer):
    class Meta:
        model = My_Model
        fields = ("name", "charfield", "booleanfield", "decimalfield")
        list_serializer_class = BulkListSerializer
        update_lookup_field = "name"
        partial = True
```

(continues on next page)

(continued from previous page)

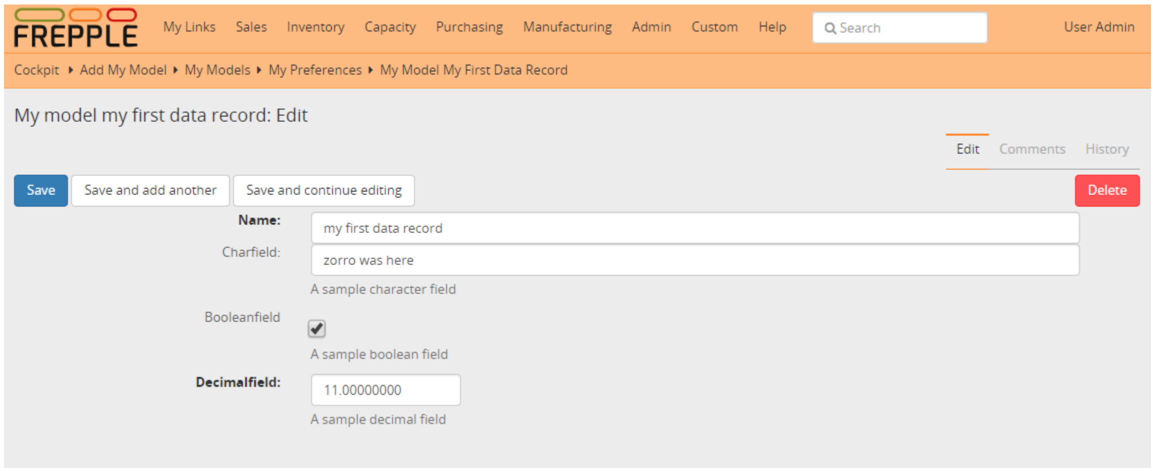
```
class MyModelSerializerAPI(frePPleListCreateAPIView):
    queryset = My_Model.objects.all()
    serializer_class = MyModelSerializer
    filter_class = MyModelFilter
```

You can find all details on creating REST APIs on https://www.django-rest-framework.org/

1.3.6 Create editing forms for your models

The file **admin.py** defines a form to edit objects of your models. An online copy is available on https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin/docs/getting_started/my_app/admin.py

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
   |    |- migrations
   |    |    |- __init__.py
   |    |    |- 0001_initial.py
   |    |- serializers.py
   |    |- admin.py
```



My model my first data record: Edit

Name: my first data record

Charfield: zorro was here

A sample character field

Booleanfield ✓

A sample boolean field

Decimalfield: 11.00000000

A sample decimal field

```
from django.utils.translation import gettext_lazy as _
from data_admin.admin import data_site
from data_admin.common.adminforms import MultiDBModelAdmin
from .models import My_Model

@admin.register(My_Model, site=data_site)
class My_Model_Admin(MultiDBModelAdmin):
    model = My_Model
    fields = ("name", "charfield", "booleanfield", "decimalfield")
    save_on_top = True
    # Defines tabs shown on the edit form
```

(continues on next page)

(continued from previous page)

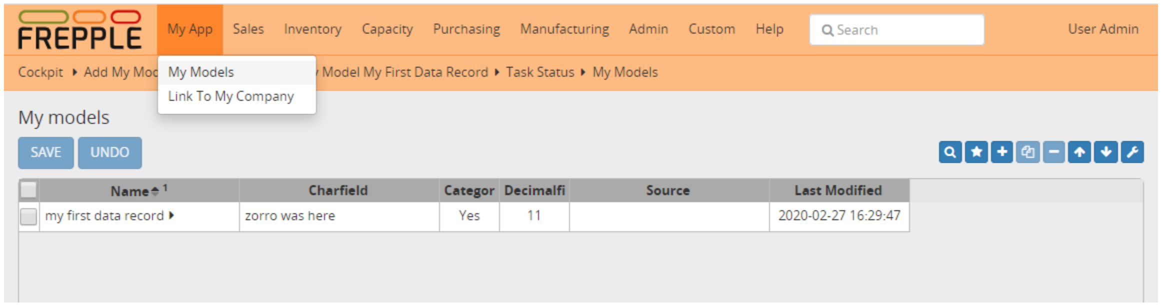
```
    tabs = [
        {
            "name": "edit",
            "label": _("edit"),
            "view": "admin:my_app_my_model_change",
            "permissions": "my_app.change_my_model",
        },
        {
            "name": "comments",
            "label": _("comments"),
            "view": "admin:my_app_my_model_comment",
        },
        {
            "name": "history",
            "label": _("History"),
            "view": "admin:my_app_my_model_history",
        },
    ]
```

You can find all details on admin forms on https://docs.djangoproject.com/en/2.2/ref/contrib/admin/

1.3.7 Define new reports

New reports are defined in a file **views.py**. The classes in this file typically will run SQL statements to retrieve data from the database, apply the correct business logic and return HTML code to the user's browser.

In this example we will inherit from a class that allows us to display an editable grid for our new model. An online copy is available on https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin/docs/getting_started/my_app/views.py



	Name ⬍ 1	Charfield	Categor	Decimalfi	Source	Last Modified
☐	my first data record ▸	zorro was here	Yes	11		2020-02-27 16:29:47

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
   |    |- migrations
   |    |    |- __init__.py
   |    |    |- 0001_initial.py
   |    |- serializers.py
   |    |- admin.py
   |    |- views.py
```

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

from data_admin.common.report import (
    GridReport,
    GridFieldText,
    GridFieldNumber,
    GridFieldBoolNullable,
    GridFieldLastModified,
)
from .models import My_Model

class MyModelList(GridReport):
    """
    This report show an editable grid for your models.
    You can sort data, filter data, import excel files, export excel files.
    """
    title = _("My models")
    basequeryset = My_Model.objects.all()
    model = My_Model
    frozenColumns = 1
    rows = (
        GridFieldText(
            "name",
            title=_("name"),
            key=True,
            formatter="detail",
            extra='"role":"my_app/my_model"',
        ),
        GridFieldText("charfield", title=_("charfield")),
        GridFieldBoolNullable("booleanfield", title=_("category")),
        GridFieldNumber("decimalfield", title=_("decimalfield")),
        GridFieldText("source", title=_("source")),
        GridFieldLastModified("lastmodified"),
    )
```

More advanced views can also separate the python business logic from the HTML rendering. This example app doesn't explore this.

See *this page* for more details on the structure of the report code.

1.3.8 Register the URLs of the new reports

The url where the report is published is defined in the file **urls.py**. An online copy is available on https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin/docs/getting_started/my_app/urls.py

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
   |    |- migrations
   |    |    |- __init__.py
```

(continues on next page)

(continued from previous page)

```
   |    |    |- 0001_initial.py
   |    |- serializers.py
   |    |- admin.py
   |    |- views.py
   |    |- urls.py
```

```
from django.conf.urls import url
from .views import MyModelList
from .serializers import MyModelSerializerAPI

# Automatically add these URLs when the application is installed
autodiscover = True

urlpatterns = [
    # Model list reports, which override standard admin screens
    url(
        r"^data/my_app/my_model/$",
        MyModelList.as_view(),
        name="my_app_my_model_changelist",
    ),
    # URLs for the REST API
    url(r"^api/my_app/my_model/$", MyModelSerializerAPI.as_view()),
]
```

You can find more detailed information on https://docs.djangoproject.com/en/2.2/topics/http/urls/

1.3.9 Add the reports to the menu

The menu is defined in the file **menu.py**. In the screenshot above you can see your own menu. With the menu, the users have access to the reports, views and urls you defined in the previous steps.

An online copy is available on https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin/docs/getting_started/my_app/menu.py

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
   |    |- migrations
   |    |    |- __init__.py
   |    |    |- 0001_initial.py
   |    |- serializers.py
   |    |- admin.py
   |    |- views.py
   |    |- urls.py
   |    |- menu.py
```

```
from django.utils.translation import ugettext as _
from data_admin.menu import menu
from .models import My_Model
```

(continues on next page)

(continued from previous page)

```
from .views import MyModelList

menu.addGroup("my_menu", label=_("My App"), index=1)
menu.addItem(
    "my_menu",
    "my_model",
    url="/data/my_app/my_model/",
    report=MyModelList,
    index=100,
    model=My_Model,
)
menu.addItem(
    "my_menu",
    "google",
    url="http://google.com",
    window=True,
    label=_("link to my company"),
    prefix=False,
    index=300,
)
```

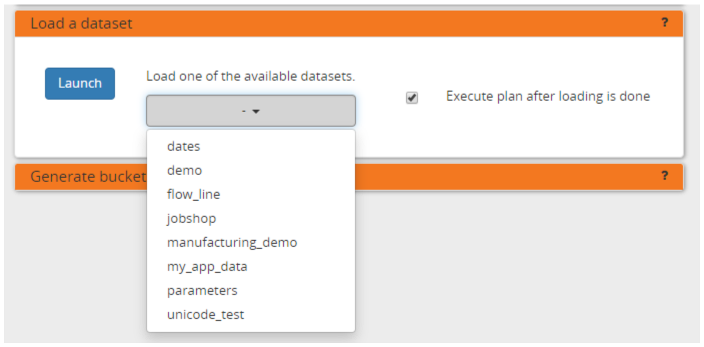
1.3.10 Add demo data

In the subfolder **fixtures** you can define demo datasets that can be loaded with the command “frepplectl loaddata” or interactively in the execution screen.

Fixtures are text files in JSON format. They can be loaded from the command line, from the execution screen (see the “my_app_data” entry in the screenshot below) or through a web API.

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
   |    |- migrations
   |    |    |- __init__.py
   |    |    |- 0001_initial.py
   |    |- serializers.py
   |    |- admin.py
   |    |- views.py
   |    |- urls.py
   |    |- menu.py
   |    |- fixtures
   |    |    |- my_app_data.json
```

```
[
{"model": "my_app.my_model", "fields": {"name": "sample #1", "charfield": "A",
↪"booleanfield": true, "decimalfield": 999.0}},
{"model": "my_app.my_model", "fields": {"name": "sample #2", "charfield": "B",
↪"booleanfield": false, "decimalfield": 666.0}}
]
```



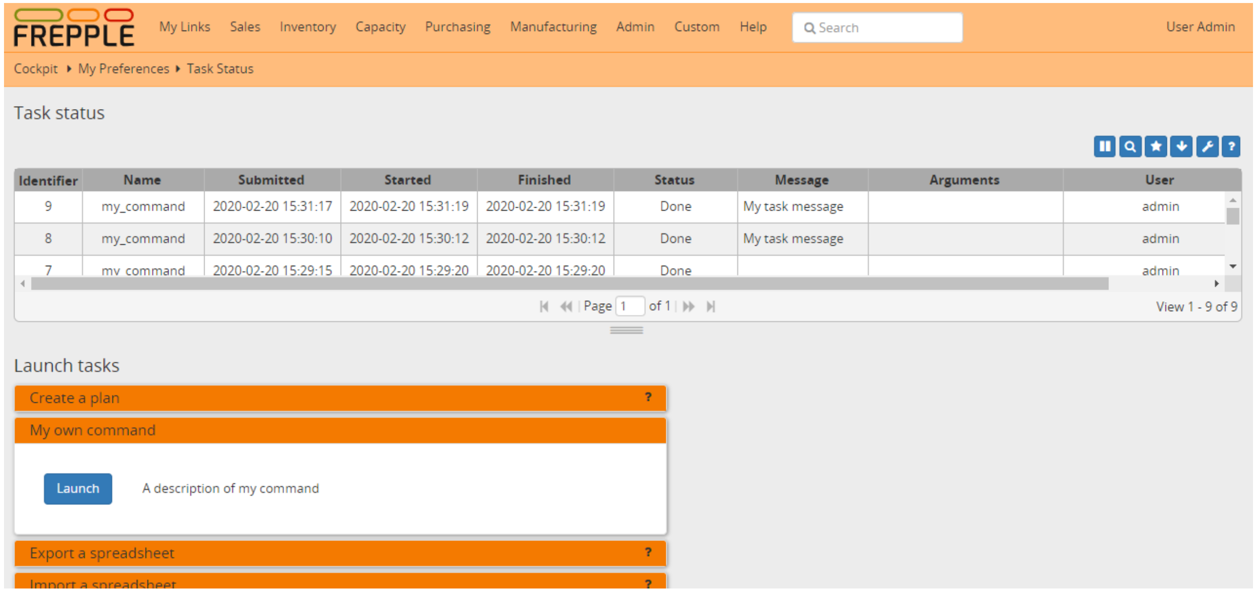
You can find more detailed information on https://docs.djangoproject.com/en/2.2/howto/initial-data/

1.3.11 Add custom administration commands

Files in the folder **management/commands** define extra commands. You can execute the custom commands from the command line, through a web API or interactively from the execution screen.

```
# Run from the command line
frepplectl my_command
```

```
# Web API of the command
POST /execute/api/my_command/
```



Identifier	Name	Submitted	Started	Finished	Status	Message	Arguments	User
9	my_command	2020-02-20 15:31:17	2020-02-20 15:31:19	2020-02-20 15:31:19	Done	My task message		admin
8	my_command	2020-02-20 15:30:10	2020-02-20 15:30:12	2020-02-20 15:30:12	Done	My task message		admin
7	my_command	2020-02-20 15:29:15	2020-02-20 15:29:20	2020-02-20 15:29:20	Done			admin

A description of my command

Simplified, the code for a command looks as follows. An online copy is available on https://github.com/frePPLe/frepple-data-admin/tree/master/data_admin/docs/getting_started/my_app/management/commands/my_command.py

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
   |    |- migrations
```

(continues on next page)

(continued from previous page)

```
  |    |    |- __init__.py
  |    |    |- 0001_initial.py
  |    |- serializers.py
  |    |- admin.py
  |    |- views.py
  |    |- urls.py
  |    |- menu.py
  |    |- fixtures
  |    |    |- my_app_data.json
  |    |- management
  |         |- __init__.py
  |         |- commands
  |              |- __init__.py
  |              |- my_command.py
```

```
class Command(BaseCommand):
    # Help text shown when you run "frepplectl help my_command"
    help = "This command does ..."

    # Define optional and required arguments
    def add_arguments(self, parser):
        parser.add_argument(
            "--my_arg",
            dest="my_arg",
            type=int,
            default=0,
            help="an optional argument for the command",
        )

    # The busisness logic of the command goes in this method
    def handle(self, *args, **options):
        print("This command was called with argument %s" % options["my_arg"])

    # Label to display on the execution screen
    title = _("My own command")

    # Sequence of the command on the execution screen
    index = 1

    # This method generates the text to display on the execution screen
    @staticmethod
    def getHTML(request):
        context = RequestContext(request)
        template = Template(
            """
            {% load i18n %}
            <form class="form" role="form" method="post"
               action="{{request.prefix}}/execute/launch/my_command/">{% csrf_token %}
            <table>
            <tr>
              <td style="padding:15px; vertical-align:top">
              <button  class="btn btn-primary" id="load" type="submit">{% trans "launch
↪"|capfirst %}</button>
```

(continues on next page)

(continued from previous page)

```
              </td>
              <td style="padding:15px">
              A description of my command
              </td>
            </tr>
            </table>
            </form>
            """
        )
        return template.render(context)
```

You can find more detailed information on https://docs.djangoproject.com/en/2.2/howto/custom-management-commands/

1.3.12 Add unit tests

Unit tests are defined in the file **tests.py**. They are executed when you run the command:

```
# Run the test
frepplectl test freppledb.my_app
```

The code for a unit test looks as follows:

```
data_admin_examples
   |- example1
   |- my-app
   |    |- __init__.py
   |    |- models.py
   |    |- migrations
   |    |    |- __init__.py
   |    |    |- 0001_initial.py
   |    |- serializers.py
   |    |- admin.py
   |    |- views.py
   |    |- urls.py
   |    |- menu.py
   |    |- fixtures
   |    |    |- my_app_data.json
   |    |- management
   |    |    |- __init__.py
   |    |    |- commands
   |    |        |- __init__.py
   |    |        |- my_command.py
   |    |- tests.py
```

```
class SimpleTest(TestCase):
  def test_basic_addition(self):
    self.assertEqual(1 + 1, 2)    # Just making sure
```

You can find more detailed information on https://docs.djangoproject.com/en/2.2/topics/testing/overview/

1.3.13 Even more information!

Data-admin is based on django web application framework. You can dig deeper by visiting https://www.djangoproject.com, checking out the full documentation and follow a tutorial.

Another good approach is to study the way the standard apps in frePPLe are structured. The full source code of the Community Edition is on https://github.com/frePPLe/frepple/tree/master/freppledb

CHAPTER

TWO

USER GUIDE

This section has instructions to navigate through the user interface.

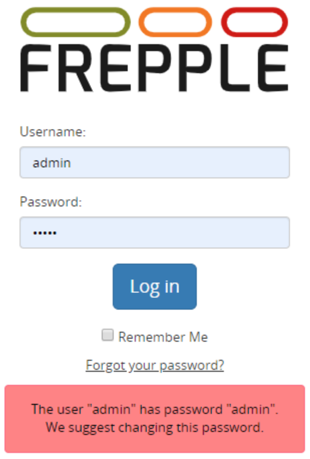
2.1 Logging in and logging out

You can log in using your user name or your email address.

A default user is created after installation: user name **admin** and password **admin**.

Danger: For security reasons, it is highly recommended to change the password of this user.

Links to log out are provided in the user menu and in the upper right corner, next to your user name.



The user "admin" has password "admin".
We suggest changing this password.

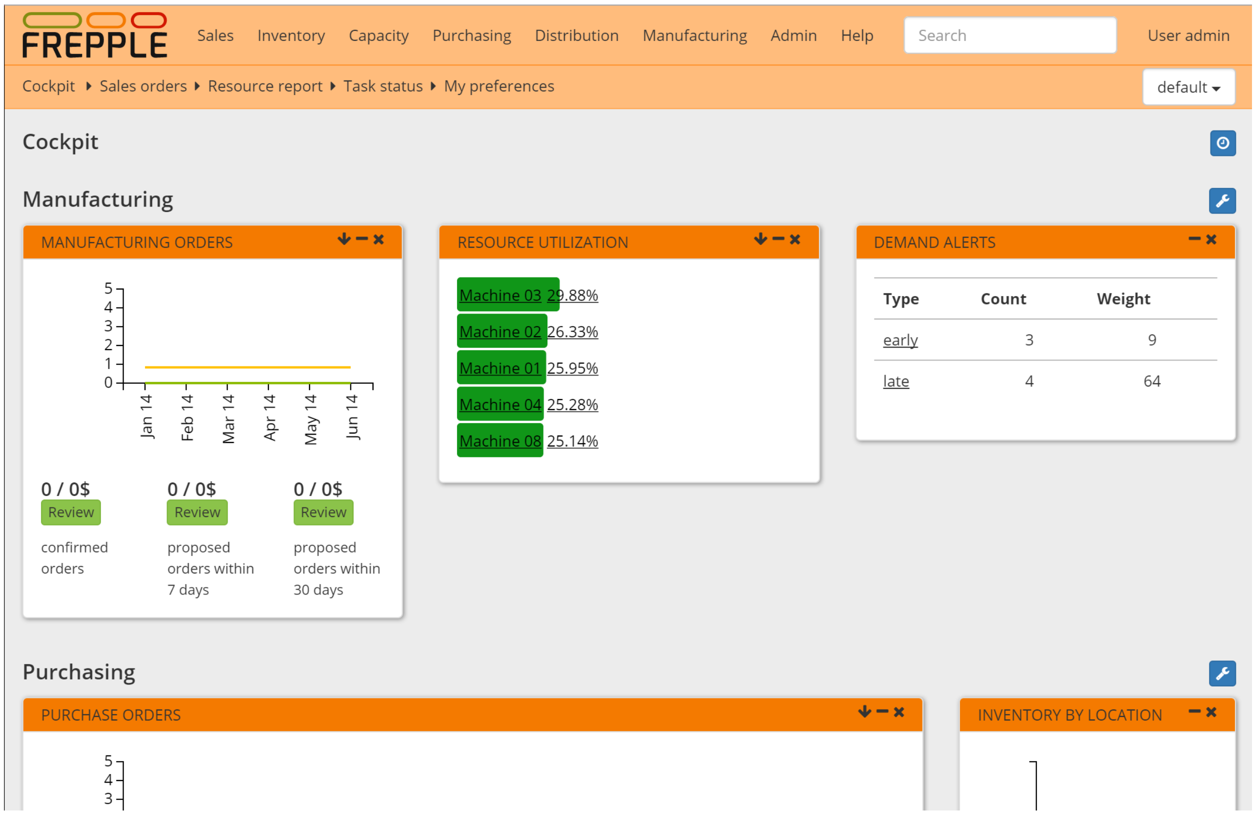
When the "remember me" box is checked, your user session session will be persisted in your browser after you close the browser window. You will only have to log in again after some time of inactivity (3 days by default, configurable by an administrator with the setting SESSION_COOKIE_AGE).
Security sensitive deployments should set this setting equal to 0, which disables this feature and forces users to log in for every browser session.

The "forgot your password" feature will send an email to reset your password. The administrator will need to update the configuration with the connection details of an SMTP mail server for this feature to work.

2.2 Home screen

The home page of is a dashboard with widgets that a planner uses in his daily planning activities. It is an efficient starting point for the common activities.

The home screen is configurable by every user to meet his/her requirements and taste.



0 / 0$ Review confirmed orders

0 / 0$ Review proposed orders within 7 days

0 / 0$ Review proposed orders within 30 days

Machine 03	29.88%
Machine 02	26.33%
Machine 01	25.95%
Machine 04	25.28%
Machine 08	25.14%

Type	Count	Weight
early	3	9
late	4	64

The following widgets are available:

- **Inbox**
  An overview of notifications and messages.
- **News**
  This widget picks up the latest news topics from frepple.com.
- **Recent actions**
  Shows your most recent editing actions.

2.3 Changing your password

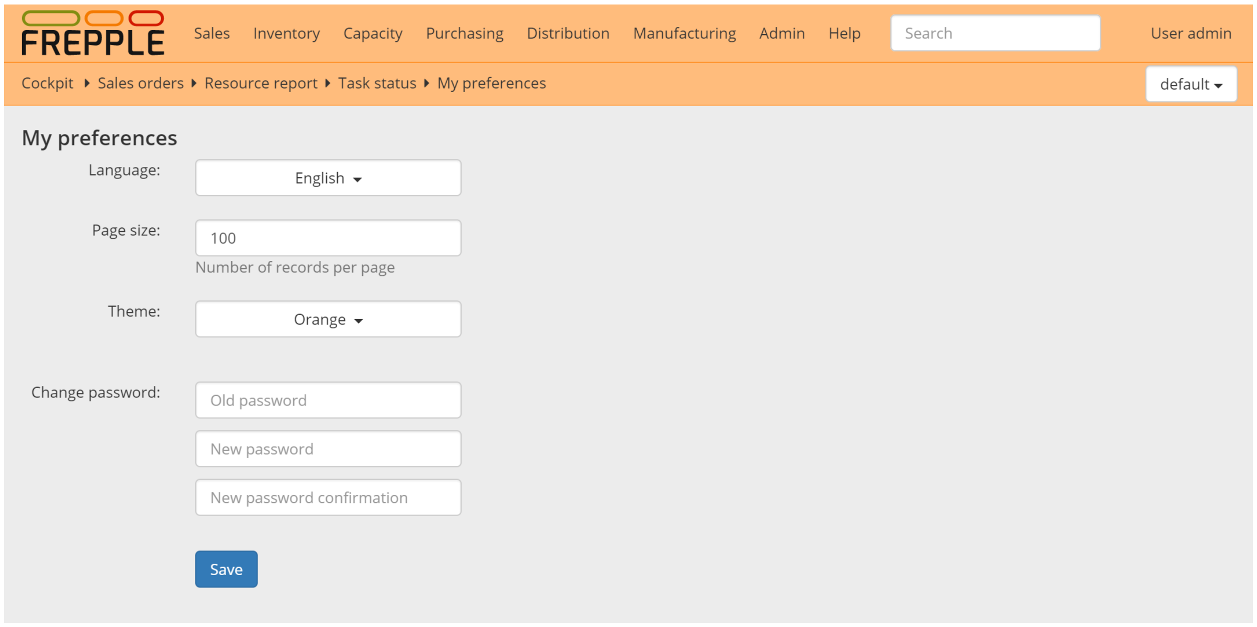
From the top right of the screen "My Preferences" or from the menu bar a screen can be opened where you can change your password.

Users often choose poor passwords. We enforce a certain validation rules for passwords:

- Passwords need to be at least 8 characters.
- Passwords cannot be similar to the user name, email address, first name or last name.
- Passwords cannot be entirely numeric.

- Password cannot be part of a list of commonly used passwords.

An administrator can use the user administration screen to change passwords as well.



My preferences

Language: English

Page size: 100
Number of records per page

Theme: Orange

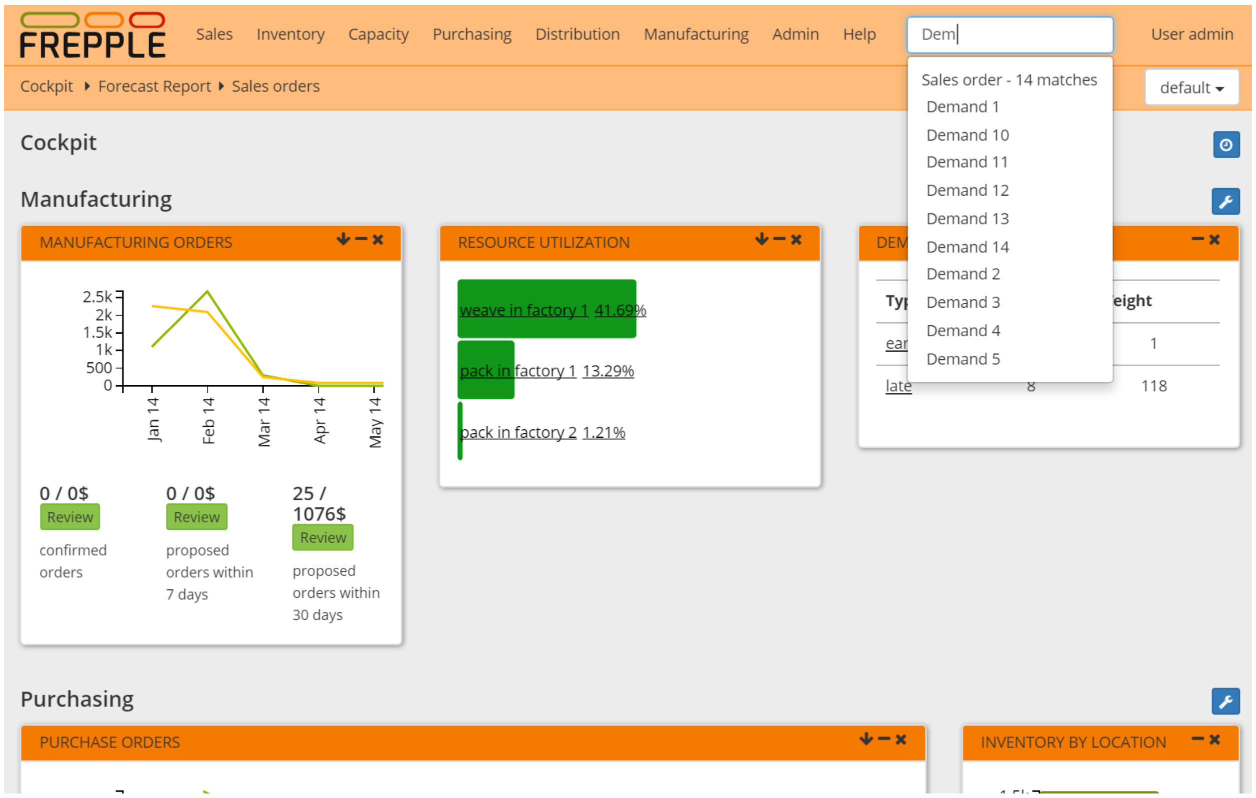
Change password: Old password / New password / New password confirmation

2.4 Navigation

Navigating the user interface is easy and intuitive.

- **Menu bar**
  Doesn't need explanation. . .
  Screens to which you have no permissions will not be shown in the menu.
- **Jump search**
  Enter 2 or more characters in the search box, and a list of matching objects is shown to you.
- **Breadcrumbs**
  The breadcrumbs allow you to navigate with a single click to a screen you visited before.
- **Detail links**
  The modelling objects have an triangular icon next to them. Clicking on it will open a screen with a selection of detailed reports on the object.

The screenshot below illustrates each of these methods.



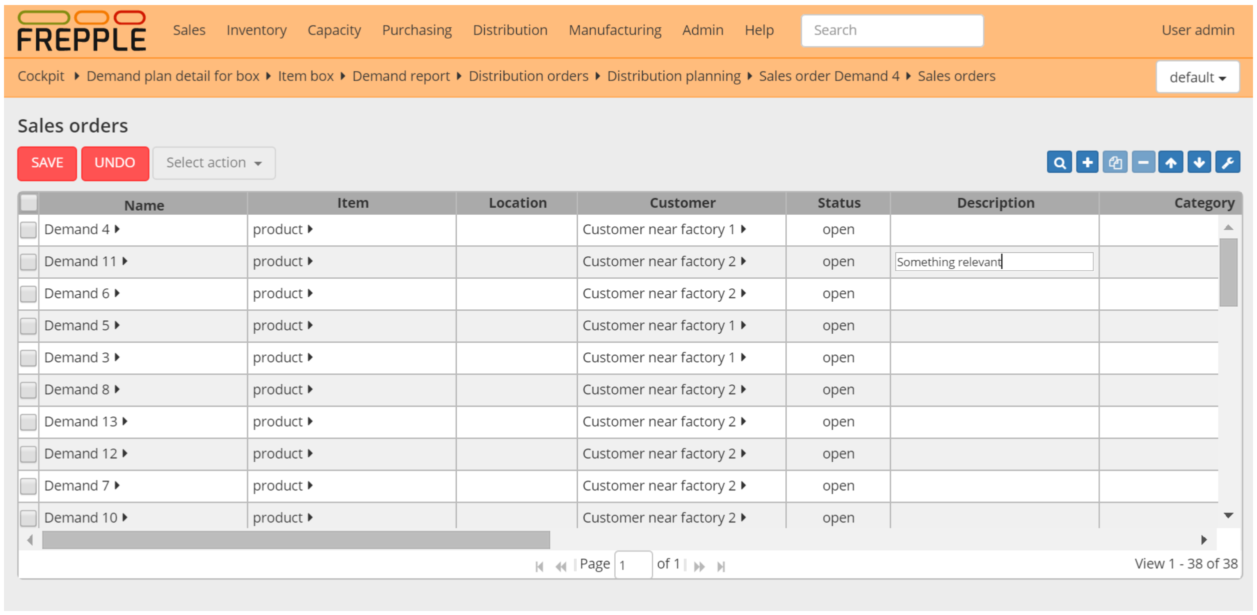
2.5 Data maintenance

FrePPLe has fully integrated data maintenance capabilities, including an audit trail of all changes.

When you don't have edit or add permissions this options will automatically be hidden from the screens.

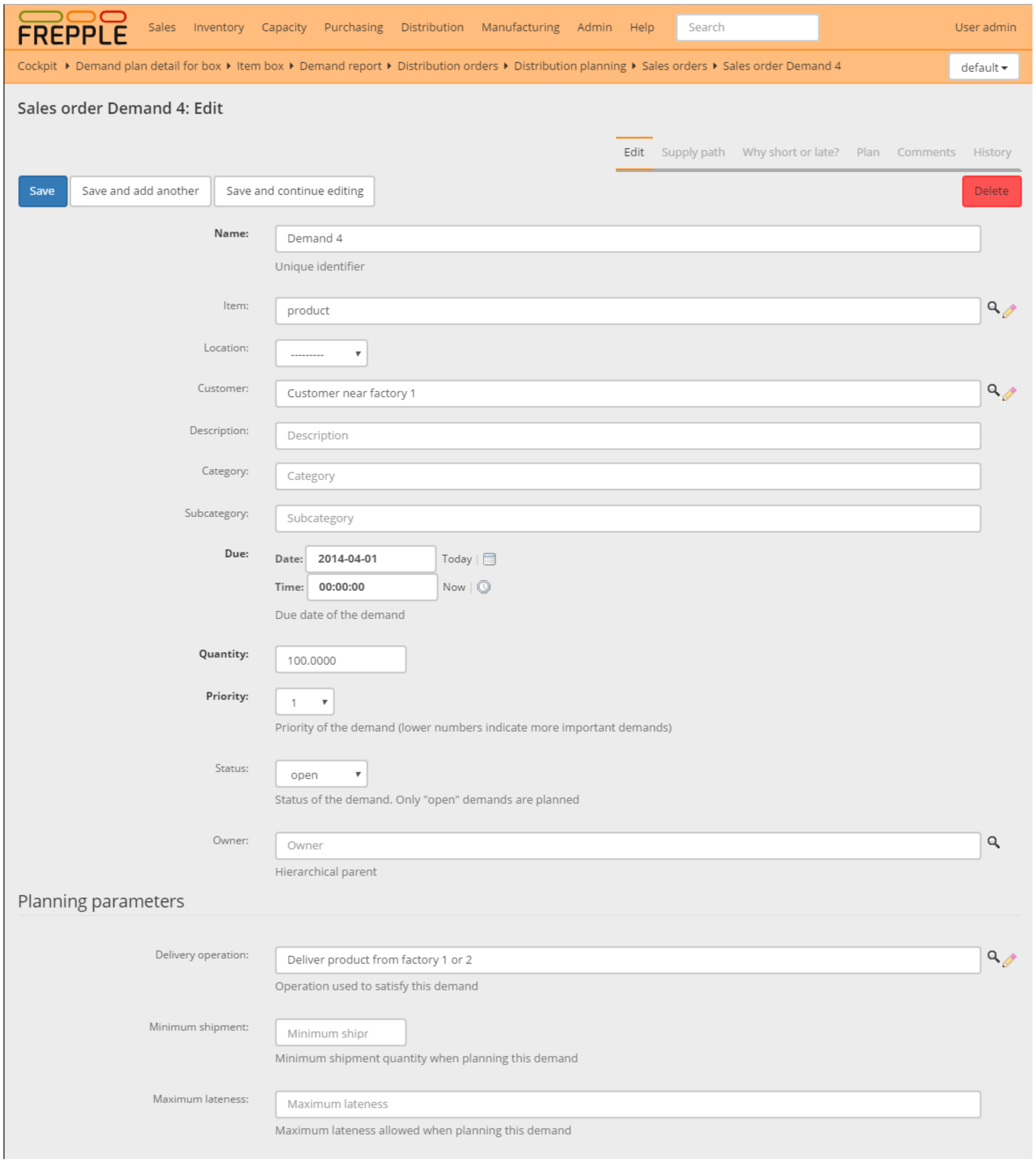
Data can be edited in different ways:

- **Edit data in the grid**
  Updated cells are marked in bold and the save icon will turn red. Hit the save icon to store the changes on the server. Or hit cancel to restore the original values.
  You can use the plus and minus icon to add and remove rows.
  You can also select one or more rows and duplicate them.
- **Edit the data in a form**
  Each entity has an edit form.
- **Import an Excel file or CSV-text file**
  For mass changes to the data, it'll be easier to export the data to Excel, apply the changes in Excel and upload the new file.
  See *Importing data*.


Sales
Inventory
Capacity
Purchasing
Distribution
Manufacturing
Admin
Help
Search
User admin
Cockpit ▸ Demand plan detail for box ▸ Item box ▸ Demand report ▸ Distribution orders ▸ Distribution planning ▸ Sales order Demand 4 ▸ Sales orders
default
SAVE
UNDO
Select action
Page 1 of 1
View 1 - 38 of 38

Sales orders

	Name	Item	Location	Customer	Status	Description	Category
	Demand 4 ▸	product ▸		Customer near factory 1 ▸	open		
	Demand 11 ▸	product ▸		Customer near factory 2 ▸	open	Something relevant	
	Demand 6 ▸	product ▸		Customer near factory 2 ▸	open		
	Demand 5 ▸	product ▸		Customer near factory 1 ▸	open		
	Demand 3 ▸	product ▸		Customer near factory 1 ▸	open		
	Demand 8 ▸	product ▸		Customer near factory 2 ▸	open		
	Demand 13 ▸	product ▸		Customer near factory 2 ▸	open		
	Demand 12 ▸	product ▸		Customer near factory 2 ▸	open		
	Demand 7 ▸	product ▸		Customer near factory 2 ▸	open		
	Demand 10 ▸	product ▸		Customer near factory 2 ▸	open		



Sales order Demand 4: Edit

Name:	Demand 4 Unique identifier
Item:	product
Location:	---------
Customer:	Customer near factory 1
Description:	Description
Category:	Category
Subcategory:	Subcategory
Due:	**Date:** 2014-04-01 Today **Time:** 00:00:00 Now Due date of the demand
Quantity:	100.0000
Priority:	1 Priority of the demand (lower numbers indicate more important demands)
Status:	open Status of the demand. Only "open" demands are planned
Owner:	Owner Hierarchical parent

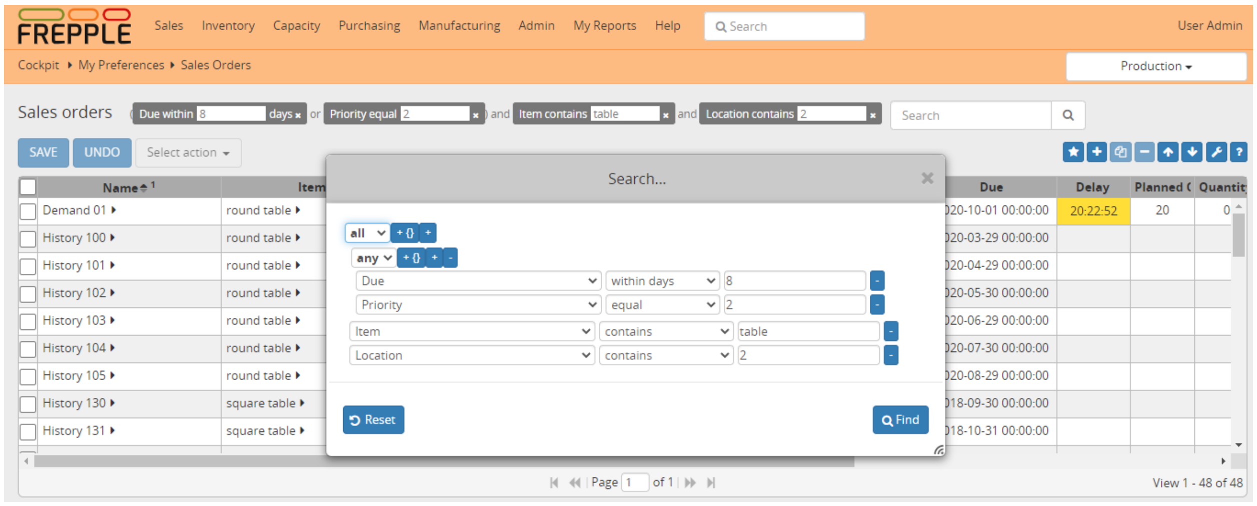
Planning parameters

Delivery operation:	Deliver product from factory 1 or 2 Operation used to satisfy this demand
Minimum shipment:	Minimum shipr Minimum shipment quantity when planning this demand
Maximum lateness:	Maximum lateness Maximum lateness allowed when planning this demand

2.6 Filtering data

In all screens a filter can be defined with the search box displayed next to the title.

- A search on a text field can be added by entering the search term in the input box, and then selecting the field where to search on.
- For more complex filters, you click on the search icon next to the input box. An rich expression editor pops up.
- Existing values are displayed next to the title. Filter values can be edited, which allows quick re-filtering. When the filter value is empty, that filter is inactivate.



Sales orders (Due within 8 days or Priority equal 2) and Item contains table and Location contains 2

all + {} +
any + {} + -

Due	within days	8	-
Priority	equal	2	-
Item	contains	table	-
Location	contains	2	-

Name 1	Item	Due	Delay	Planned	Quantit
Demand 01	round table	020-10-01 00:00:00	20:22:52	20	0
History 100	round table	020-03-29 00:00:00			
History 101	round table	020-04-29 00:00:00			
History 102	round table	020-05-30 00:00:00			
History 103	round table	020-06-29 00:00:00			
History 104	round table	020-07-30 00:00:00			
History 105	round table	020-08-29 00:00:00			
History 130	square table	018-09-30 00:00:00			
History 131	square table	018-10-31 00:00:00			

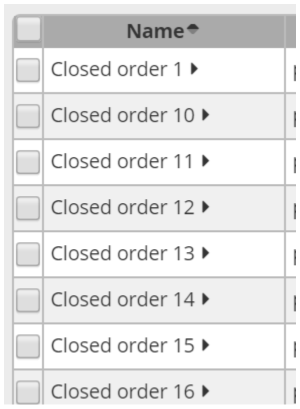
Page 1 of 1 View 1 - 48 of 48

2.7 Sorting data

Clicking on a column header will sort the sort the data in ascending order. Another click will sort in descending order. A third click deactivates the sort.

You can sort on up to 3 fields simultaneously.

The sorting of a screen is automatically stored. When you reopen the same report later on, it will open with the same sorting as when you left it.



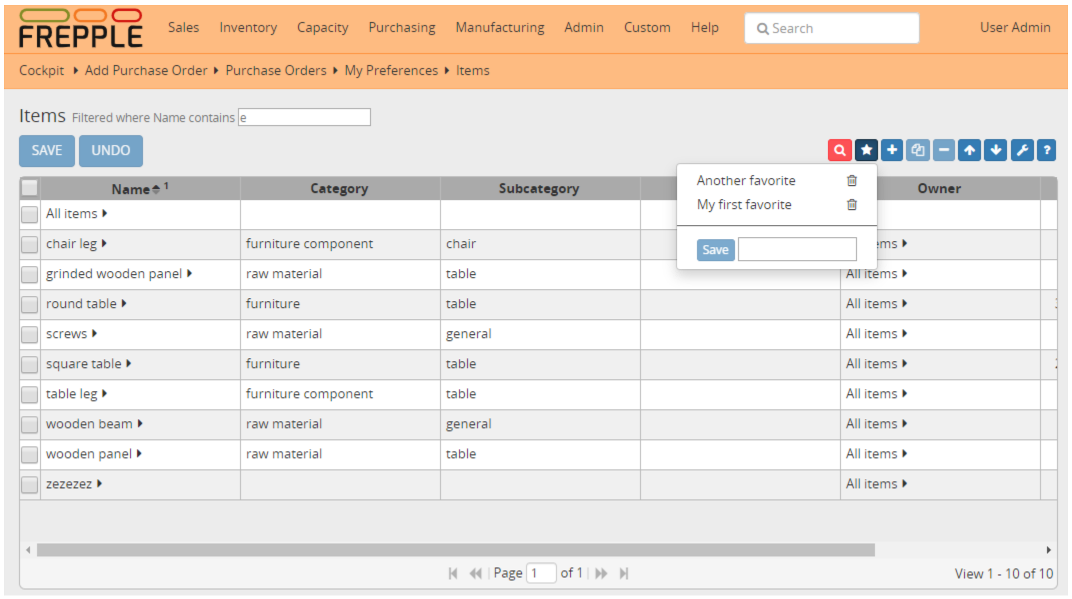
Name
Closed order 1
Closed order 10
Closed order 11
Closed order 12
Closed order 13
Closed order 14
Closed order 15
Closed order 16

2.8 Favorites

All screens allow you to manage favorite settings with the star icon in the tool bar on the upper left. This allows you to quickly jump back to frequently report settings, and can be huge time saver in your daily workflows.

A favorite remembers the following information: - filters - sorting fields and direction - displayed columns, their order and width

Favorites are private for each user.



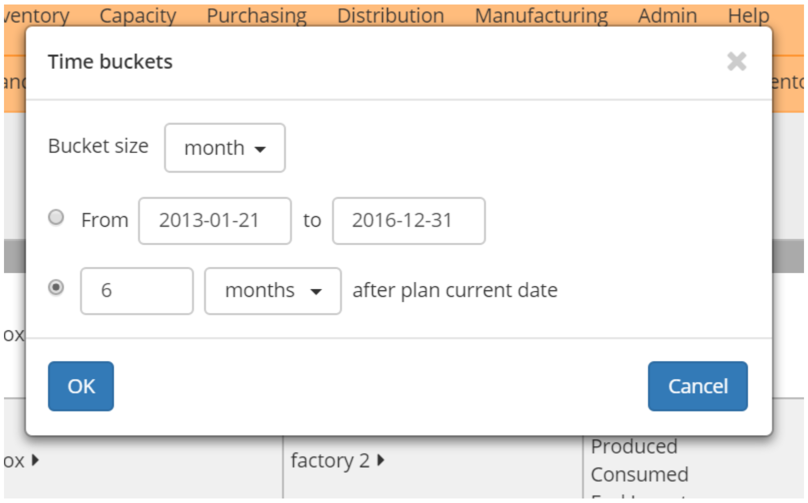
Items Filtered where Name contains e

Name	Category	Subcategory		Owner
All items ▸				
chair leg ▸	furniture component	chair		ms ▸
grinded wooden panel ▸	raw material	table		All items ▸
round table ▸	furniture	table		All items ▸
screws ▸	raw material	general		All items ▸
square table ▸	furniture	table		All items ▸
table leg ▸	furniture component	table		All items ▸
wooden beam ▸	raw material	general		All items ▸
wooden panel ▸	raw material	table		All items ▸
zezezez ▸				All items ▸

Page 1 of 1 View 1 - 10 of 10

2.9 Selecting time buckets

A number of reports show results in a bucketized format. The buckets can be selected using on the clock icon in the upper right corner.



Time buckets

Bucket size: month

( ) From 2013-01-21 to 2016-12-31

(●) 6 months after plan current date

The popup window allows you to define the horizon you want to report for:

- **The size of the report buckets**:
  By default we create buckets for “days”, “weeks”, “months”, “quarter” and “year”.
  Weeks are starting on Monday.
  Months are calendar months, and a week thus spans across 2 months.
- The **start and end date of the horizon** can be specified in 2 ways:
  - As absolute dates.
  - Relative to the current date.

The report horizon then always starts at the current date of the plan. This value is specified in the Parameter table as "currentdate", or is automatically set to the system clock if the parameter is absent or incorrectly formatted.

The end date of the reporting horizon is equal to the current date plus the specified offset. When the offset is specified in weeks or months, the end date is rounded up to the start date of the following week or month. This avoids reporting on partial buckets.

The settings you select are saved and will apply to all bucketized reports you open.

Tip: An administrator can update the buckets by manipulating the records in the buckets and bucketdetail tables: In this way you can set up the buckets that are relevant for your organization, and also give the buckets the labels you prefer.

2.10 Exporting data

You can export the data either a) as a **native Excel workbook** or b) as a **CSV-formatted text file** or c) **as a data source URL** by clicking on the download arrow on the upper right.

For report with time buckets, two structures are available:

- **Table**:
  Uses the same layout as shown on the screen. The time buckets are shown as columns in the CSV-file.
- **List**:
  A separate line is generated for each time bucket. This flat format can be more appropriate for further processing by other tools.

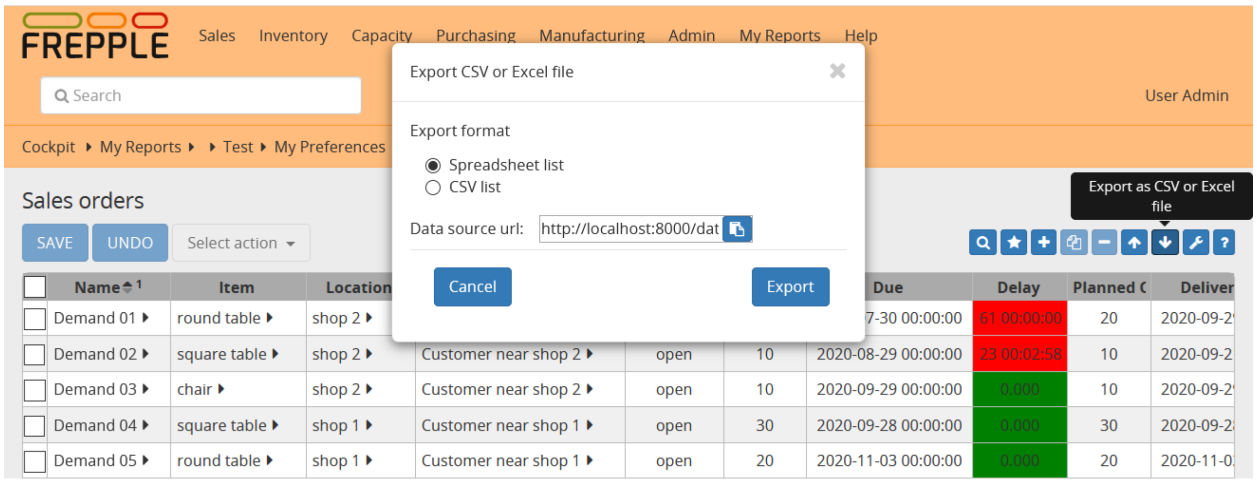
The export is not limited to the page currently displayed on the screen, but all pages in the filtered selection will be exported.

A couple of notes on the CSV-format:

- The separator in the CSV-files varies with the chosen language: If in your language a comma is used as a decimal separator for numbers, the CSV file will use a semicolon (;) as delimiter. Otherwise a comma (,) is used. See http://en.wikipedia.org/wiki/Decimal_mark
- The date format exported and imported by frePPLe is 'YYYY-MM-DD HH:MM:SS'. Microsoft Excel tends to export dates in your local format, which can cause problems when you save the file again and try to importing it back in frePPLe. The best approach is to import the cells as text to avoid any conversion.
- The export process will encode the data file in the encoding defined by the setting CSV_CHARSET (default UTF-8).

Tip: Exporting to Excel format avoids these common pitfalls from the CSV text-files.

In the *Execution screen* you also have the capability to export all entities into a single Excel workbook.



Export CSV or Excel file

Export format

- [✓ Spreadsheet list]
- CSV list

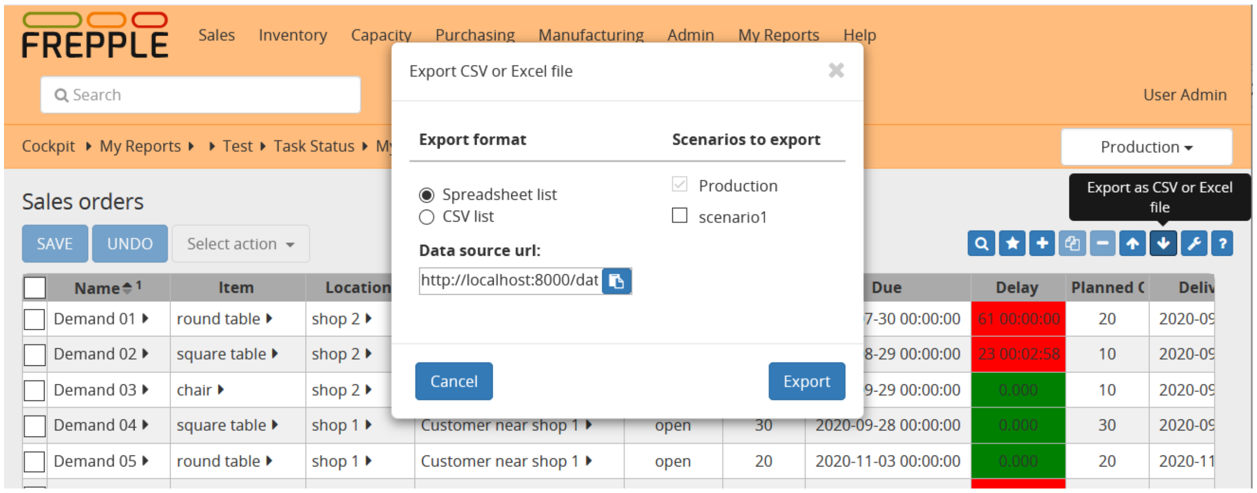
Data source url: http://localhost:8000/dat

Name 1	Item	Location				Due	Delay	Planned (	Deliver
Demand 01	round table	shop 2				7-30 00:00:00	61 00:00:00	20	2020-09-2
Demand 02	square table	shop 2	Customer near shop 2	open	10	2020-08-29 00:00:00	23 00:02:58	10	2020-09-2
Demand 03	chair	shop 2	Customer near shop 2	open	10	2020-09-29 00:00:00	0.000	10	2020-09-2
Demand 04	square table	shop 1	Customer near shop 1	open	30	2020-09-28 00:00:00	0.000	30	2020-09-2
Demand 05	round table	shop 1	Customer near shop 1	open	20	2020-11-03 00:00:00	0.000	20	2020-11-0

If the user has permissions on other scenarios for the view he/she is trying to export, another window dialog will be displayed with these scenarios.

By default only current scenario will be selected and the user cannot disable this selection.

If the user selects extra scenarios then the generated spreadsheet or CSV file will contain data for all selected scenarios. An extra column "scenario" is added in the file to clearly identify to which scenario a data row belongs to.



Export CSV or Excel file

Export format

- [✓ Spreadsheet list]
- CSV list

Data source url: http://localhost:8000/dat

Scenarios to export

- [✓ Production]
- scenario1

Name 1	Item	Location				Due	Delay	Planned (	Deliv
Demand 01	round table	shop 2				7-30 00:00:00	61 00:00:00	20	2020-09
Demand 02	square table	shop 2				8-29 00:00:00	23 00:02:58	10	2020-09
Demand 03	chair	shop 2				9-29 00:00:00	0.000	10	2020-09
Demand 04	square table	shop 1	Customer near shop 1	open	30	2020-09-28 00:00:00	0.000	30	2020-09
Demand 05	round table	shop 1	Customer near shop 1	open	20	2020-11-03 00:00:00	0.000	20	2020-11

2.11 Data source URL

With this option we provide a URL where external applications have on line access to the report data.

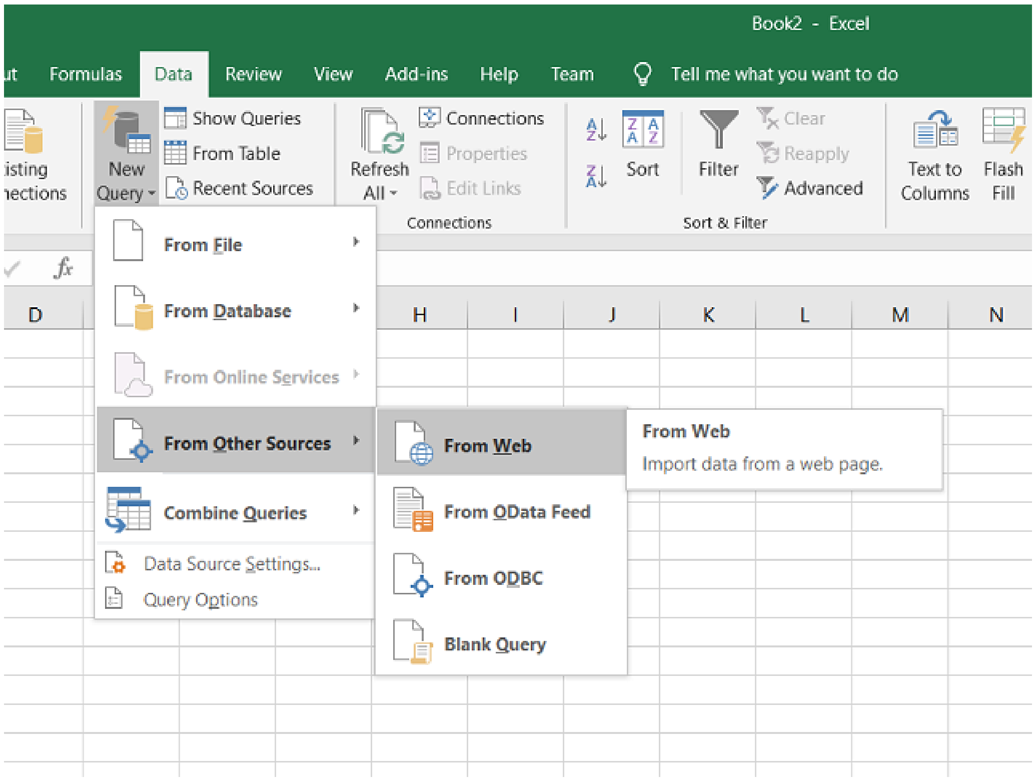
The data source URL can be pasted in any kind of reporting software supporting this format (Excel, Google sheets…). This is pretty convenient if you are exporting the page you are visiting on a regular basis as refreshing your data in your software will take no longer than a click.

The video below demonstrates how to pull frePPLe data in an Excel spreadsheet.

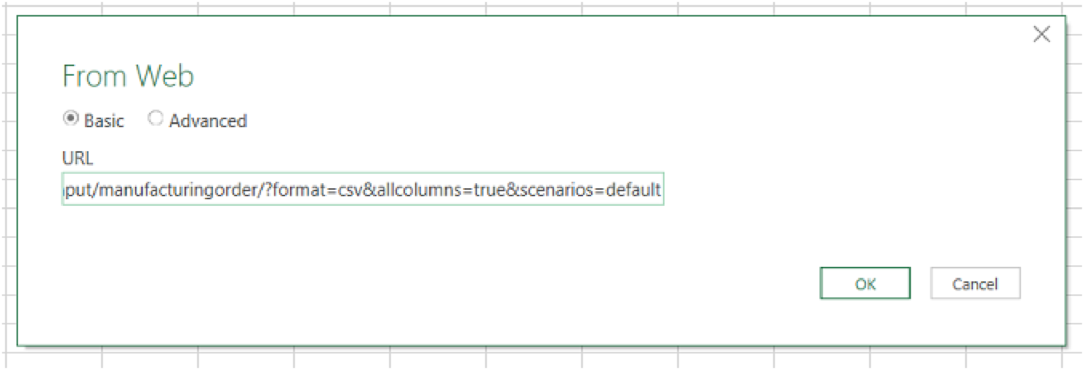
Below are the steps to configure a data source URL in Excel:

- Select the scenarios you wish to export (by default, only current scenario is selected).
- Click on the "Copy to clipboard" button so that frePPLe copies the data source URL into your clipboard.

- Open a new spreadsheet in Excel, move to the "Data" menu and select "New Query", then "From Other Sources" and finally "From Web". Note that, in some old versions of Excel, this requires the installation of the Power Query module.

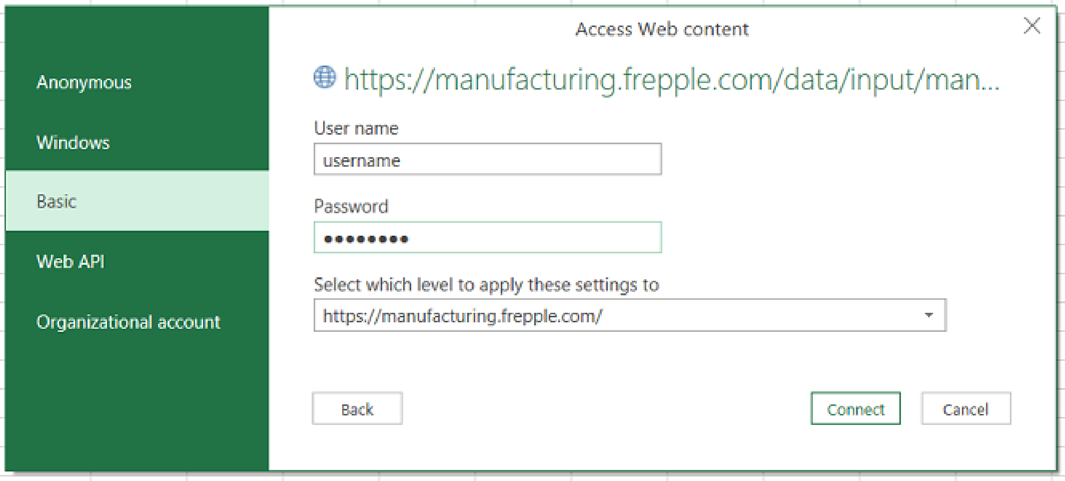


- Paste the URL in the clipboard in the field and press ok.



put/manufacturingorder/?format=csv&allcolumns=true&scenarios=default

- Your credentials need to be populated in Excel so that it can connect to your frePPLe instance. Choose the "basic" option and enter your frePPLe username and password. Note that, as long as the connection uses the https protocol (which is always the case for the frePPLe cloud users), the crendentials are encrypted.



Access Web content

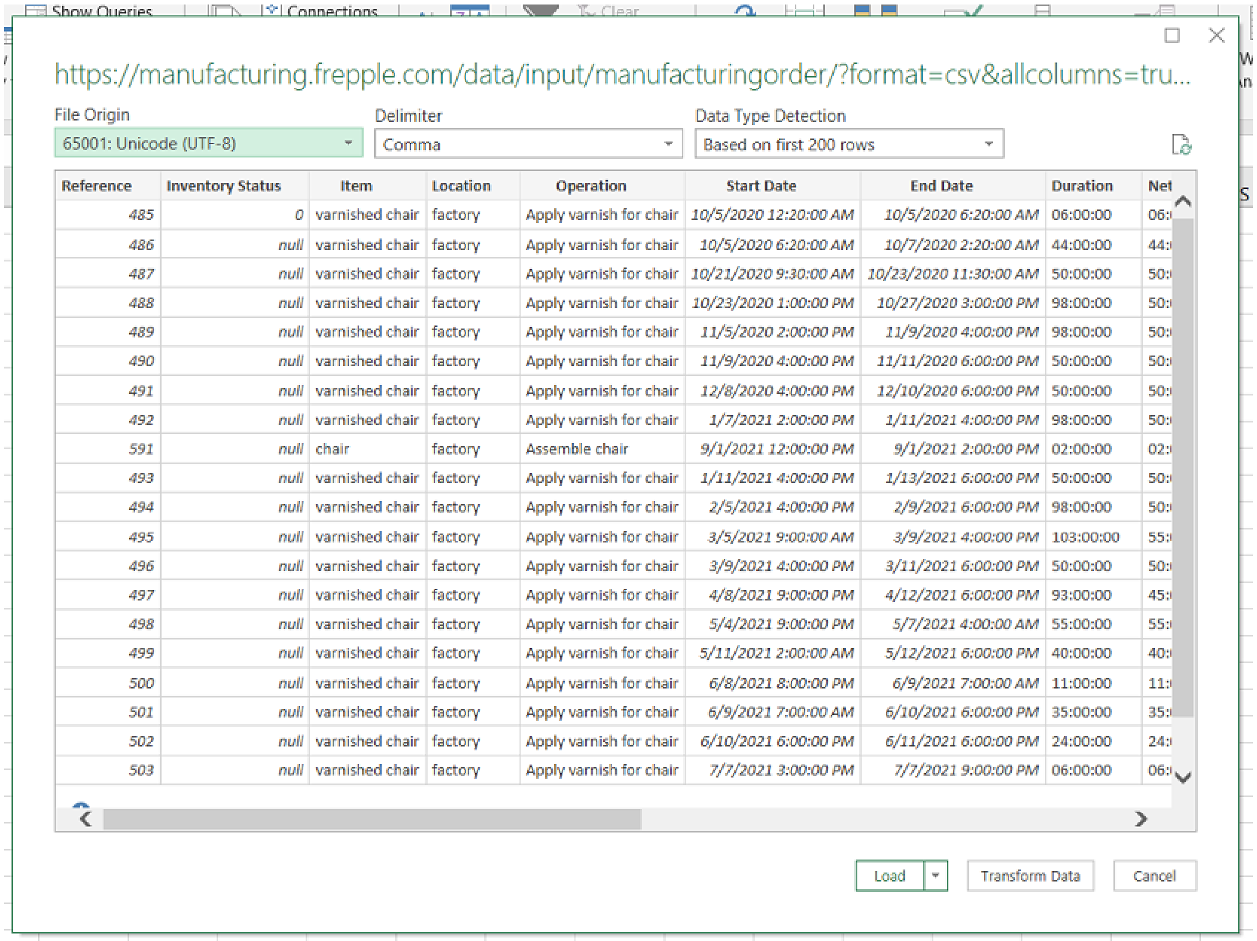
https://manufacturing.frepple.com/data/input/man...

User name: username

Password: ●●●●●●●●●

Select which level to apply these settings to: https://manufacturing.frepple.com/

* If Excel has been able to connect to your frePPLe instance, Excel should display data from your report in a dedicated window:



https://manufacturing.frepple.com/data/input/manufacturingorder/?format=csv&allcolumns=tru...

File Origin: 65001: Unicode (UTF-8)

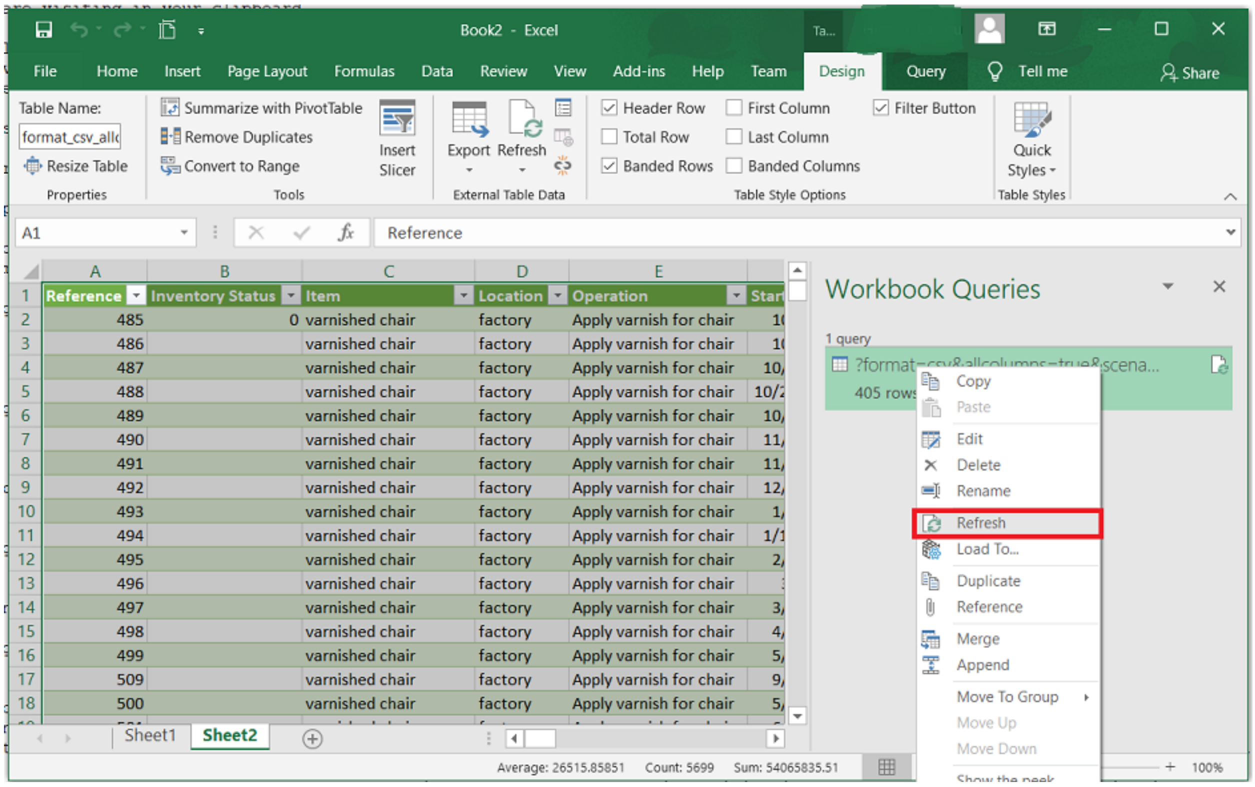
Delimiter: Comma

Data Type Detection: Based on first 200 rows

Reference	Inventory Status	Item	Location	Operation	Start Date	End Date	Duration	Net
485	0	varnished chair	factory	Apply varnish for chair	10/5/2020 12:20:00 AM	10/5/2020 6:20:00 AM	06:00:00	06:
486	null	varnished chair	factory	Apply varnish for chair	10/5/2020 6:20:00 AM	10/7/2020 2:20:00 AM	44:00:00	44:
487	null	varnished chair	factory	Apply varnish for chair	10/21/2020 9:30:00 AM	10/23/2020 11:30:00 AM	50:00:00	50:
488	null	varnished chair	factory	Apply varnish for chair	10/23/2020 1:00:00 PM	10/27/2020 3:00:00 PM	98:00:00	50:
489	null	varnished chair	factory	Apply varnish for chair	11/5/2020 2:00:00 PM	11/9/2020 4:00:00 PM	98:00:00	50:
490	null	varnished chair	factory	Apply varnish for chair	11/9/2020 4:00:00 PM	11/11/2020 6:00:00 PM	50:00:00	50:
491	null	varnished chair	factory	Apply varnish for chair	12/8/2020 4:00:00 PM	12/10/2020 6:00:00 PM	50:00:00	50:
492	null	varnished chair	factory	Apply varnish for chair	1/7/2021 2:00:00 PM	1/11/2021 4:00:00 PM	98:00:00	50:
591	null	chair	factory	Assemble chair	9/1/2021 12:00:00 PM	9/1/2021 2:00:00 PM	02:00:00	02:
493	null	varnished chair	factory	Apply varnish for chair	1/11/2021 4:00:00 PM	1/13/2021 6:00:00 PM	50:00:00	50:
494	null	varnished chair	factory	Apply varnish for chair	2/5/2021 4:00:00 PM	2/9/2021 6:00:00 PM	98:00:00	50:
495	null	varnished chair	factory	Apply varnish for chair	3/5/2021 9:00:00 AM	3/9/2021 4:00:00 PM	103:00:00	55:
496	null	varnished chair	factory	Apply varnish for chair	3/9/2021 4:00:00 PM	3/11/2021 6:00:00 PM	50:00:00	50:
497	null	varnished chair	factory	Apply varnish for chair	4/8/2021 9:00:00 PM	4/12/2021 6:00:00 PM	93:00:00	45:
498	null	varnished chair	factory	Apply varnish for chair	5/4/2021 9:00:00 PM	5/7/2021 4:00:00 AM	55:00:00	55:
499	null	varnished chair	factory	Apply varnish for chair	5/11/2021 2:00:00 AM	5/12/2021 6:00:00 PM	40:00:00	40:
500	null	varnished chair	factory	Apply varnish for chair	6/8/2021 8:00:00 PM	6/9/2021 7:00:00 AM	11:00:00	11:
501	null	varnished chair	factory	Apply varnish for chair	6/9/2021 7:00:00 AM	6/10/2021 6:00:00 PM	35:00:00	35:
502	null	varnished chair	factory	Apply varnish for chair	6/10/2021 6:00:00 PM	6/11/2021 6:00:00 PM	24:00:00	24:
503	null	varnished chair	factory	Apply varnish for chair	7/7/2021 3:00:00 PM	7/7/2021 9:00:00 PM	06:00:00	06:

* At this point, you have the choice to either load your data or most likely transform your data to select which columns to display, in which order and possibly add calculated columns. Note that all the columns of the view are available before transformation and all crosses also in case of a pivot report (reports where time bucket is a column such as inventory or capacity reports) so exporting your data using the data source URL will differ from the columns selection you have on frePPLe. Once transformed, you can then load your data.

- The magic happens when the report you have exported using the data source URL is modified in frePPLe, you have the possibility in Excel to refresh your data in a simple click using the “refresh” option. This will automatically create a connection to your frePPLe instance and update the data in the spreadsheet.



	A	B	C	D	E	
1	Reference	Inventory Status	Item	Location	Operation	Start
2	485	0	varnished chair	factory	Apply varnish for chair	1(
3	486		varnished chair	factory	Apply varnish for chair	1(
4	487		varnished chair	factory	Apply varnish for chair	10/
5	488		varnished chair	factory	Apply varnish for chair	10/2
6	489		varnished chair	factory	Apply varnish for chair	10/
7	490		varnished chair	factory	Apply varnish for chair	11/
8	491		varnished chair	factory	Apply varnish for chair	11/
9	492		varnished chair	factory	Apply varnish for chair	12/
10	493		varnished chair	factory	Apply varnish for chair	1/
11	494		varnished chair	factory	Apply varnish for chair	1/1
12	495		varnished chair	factory	Apply varnish for chair	2/
13	496		varnished chair	factory	Apply varnish for chair	
14	497		varnished chair	factory	Apply varnish for chair	3/
15	498		varnished chair	factory	Apply varnish for chair	4/
16	499		varnished chair	factory	Apply varnish for chair	5/
17	509		varnished chair	factory	Apply varnish for chair	9/
18	500		varnished chair	factory	Apply varnish for chair	5/

2.12 Importing data

You can upload data files in a) **native Excel format** or b) **CSV format**. Clicking on the import arrow in the upper right corner allows you to select the file to upload. Note that you can also drag and drop a file into the dialog box. FrePPLe automatically detects which of the two data formats you’re using.

The upload icon will only be available when you have add-permission on the data object you’re uploading. A message is shown when you don’t have this permission.

The first line in the data file should contain the field names (not case sensitive). To get a sample of the input format you can first create a export: the format of the export file is such that it can be reread into frePPLe.

The dialog screen shows a checkbox *First delete all existing records AND ALL RELATED TABLES* meaning that you can choose to delete the existing contents of the table before uploading the new data. When the option is selected, dependent tables are also erased: e.g. if you select this option when uploading the *resource* table, also the *operationresource* table will be erased since the second table references the first one.

Caution: If you’re not very familiar with the relation between the objects in frePPLe’s data model, you probably shouldn’t use the *First delete all existing records AND ALL RELATED TABLES* option.

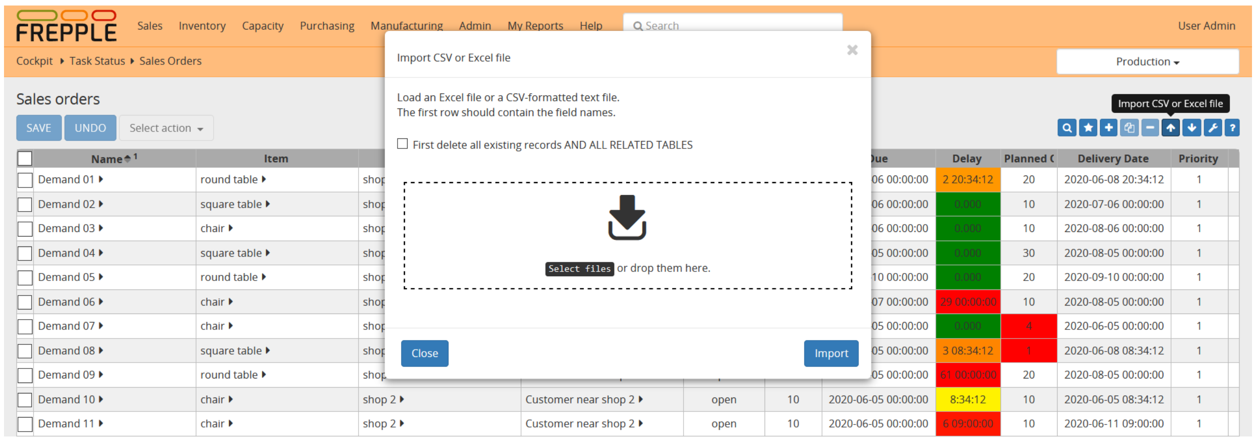
After the upload, the number of data rows loaded will be displayed. When data errors were found during the loading process the details will be shown as well.

A couple of notes on the CSV-format:

- The separator in your CSV-files varies with the chosen language: If in your language a comma is used as a decimal separator for numbers, the CSV file will use a semicolon (;) as delimiter. Otherwise a comma (,) is used. See http://en.wikipedia.org/wiki/Decimal_mark
- The date format expected by frePPLe is 'YYYY-MM-DD HH:MM:SS'.
- The data file is expected to be encoded in the character encoding defined by the setting CSV_CHARSET (default UTF-8).

Tip: Importing a native Excel file avoids these common pitfalls from the CSV text-files.

In the Admin/Execute menu you also have the capability to import multiple tables from a single Excel workbook using the *Import a spreadsheet* option.



Import CSV or Excel file

Load an Excel file or a CSV-formatted text file.
The first row should contain the field names.

☐ First delete all existing records AND ALL RELATED TABLES

Select files or drop them here.

Name	Item					Due	Delay	Planned (	Delivery Date	Priority
Demand 01	round table	shop				06 00:00:00	2 20:34:12	20	2020-06-08 20:34:12	1
Demand 02	square table	shop				06 00:00:00	0.000	10	2020-07-06 00:00:00	1
Demand 03	chair	shop				06 00:00:00	0.000	10	2020-08-06 00:00:00	1
Demand 04	square table	shop				05 00:00:00	0.000	30	2020-08-05 00:00:00	1
Demand 05	round table	shop				10 00:00:00	0.000	20	2020-09-10 00:00:00	1
Demand 06	chair	shop				07 00:00:00	29 00:00:00	10	2020-08-05 00:00:00	1
Demand 07	chair	shop				05 00:00:00	0.000	4	2020-06-05 00:00:00	1
Demand 08	square table	shop				05 00:00:00	3 08:34:12	1	2020-06-08 08:34:12	1
Demand 09	round table	shop				05 00:00:00	61 00:00:00	20	2020-08-05 00:00:00	1
Demand 10	chair	shop 2	Customer near shop 2	open	10	2020-06-05 00:00:00	8:34:12	10	2020-06-05 08:34:12	1
Demand 11	chair	shop 2	Customer near shop 2	open	10	2020-06-05 00:00:00	6 09:00:00	10	2020-06-11 09:00:00	1

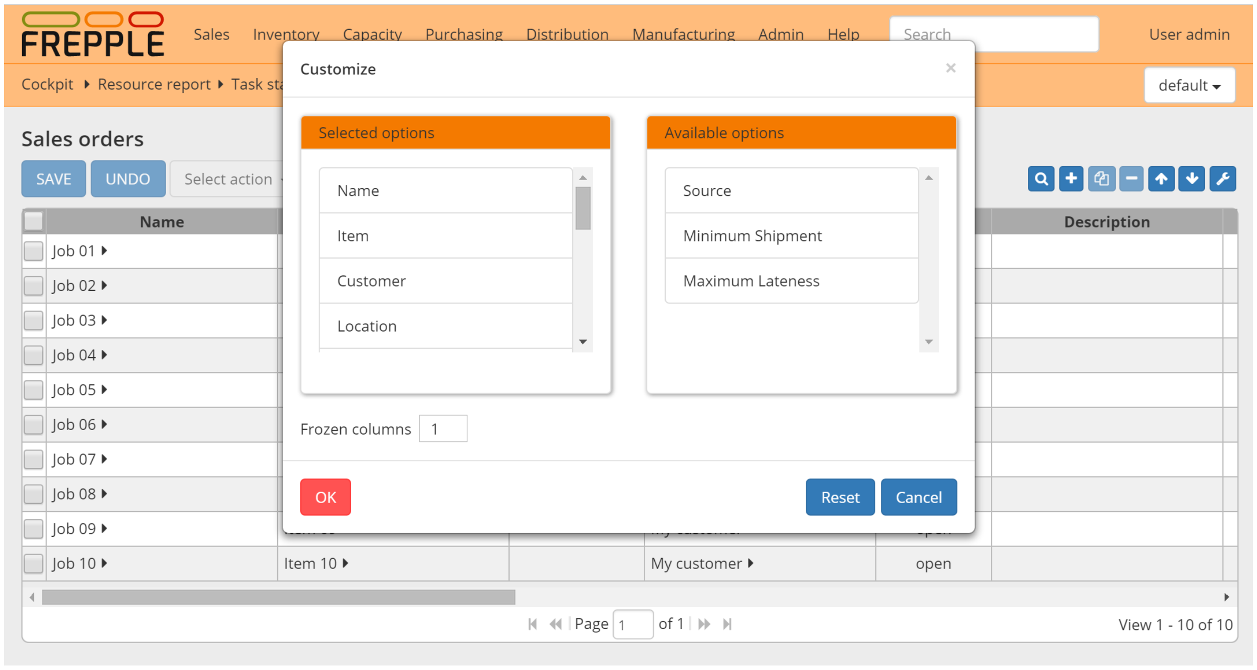
2.13 Customizing a report

Each user can update the configuration of the reports to suite his needs and taste.

Click the customize icon in the tool bar to bring up a popup window where the user can update:

- **Fields to be shown**:
  Click on a field to toggle its visibility.
  Note that the key field of a table cannot be hidden.
- **Order of the fields**:
  Drag and drop a visible field to the desired position.
- **Number of frozen columns**:
  Select the number of columns that stay in place when you scroll to the right.
- **Column width**:
  Changing the column width is not done from the popup window, but directly in the report itself.
  Hover the mouse over the boundary between 2 columns, holds the mouse down and drag to the correct width.
- **Filters, sorting and paging**:
  Filtering criteria, *sorting* and page number are preserved. When you re-open the report later on it will opens exactly as when you left it.

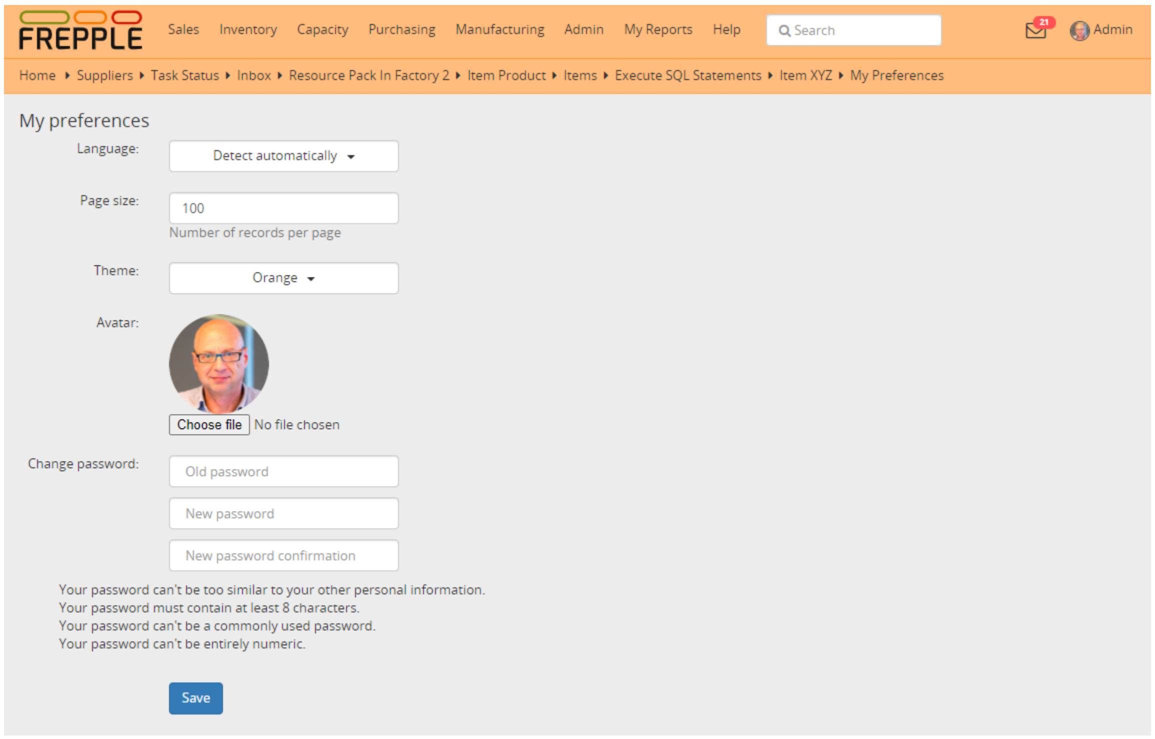
The settings are stored on the server when you hit the OK button.



2.14 User preferences

For each user frePPle stores a number of personal settings and preferences.

- **Language**:
  Specifies the language of the user interface.
  By default frePPLe will detect the preferred language of your web browser and use that.
  You can override this and force a particular language.
- **Page size**:
  Number of records to fetch in a single page from the server.
  The default value is 100. Values lower than 25 are not accepted.
  Selecting a high value can slow down the display of the reports.
- **Theme**:
  Theme for the user interface.
  If your system administrator has configured the system to support only a single theme (by editing the THEMES setting in the djangosettings.py file), this option will not be available to the users.
- **Avatar**:
  A small picture of yourself.
  The uploaded picture must 1) be square (ie same height and width), 2) be in jpeg, png or gif format and 3) be smaller than 100kB.
- **Password**:
  To change the password enter the current one and twice the new value.


Sales
Inventory
Capacity
Purchasing
Manufacturing
Admin
My Reports
Help
Search
Admin
Home ▸ Suppliers ▸ Task Status ▸ Inbox ▸ Resource Pack In Factory 2 ▸ Item Product ▸ Items ▸ Execute SQL Statements ▸ Item XYZ ▸ My Preferences
Choose file
Save

My preferences

Language: Detect automatically

Page size: 100
Number of records per page

Theme: Orange

Avatar:
No file chosen

Change password:
Old password
New password
New password confirmation

Your password can't be too similar to your other personal information.
Your password must contain at least 8 characters.
Your password can't be a commonly used password.
Your password can't be entirely numeric.

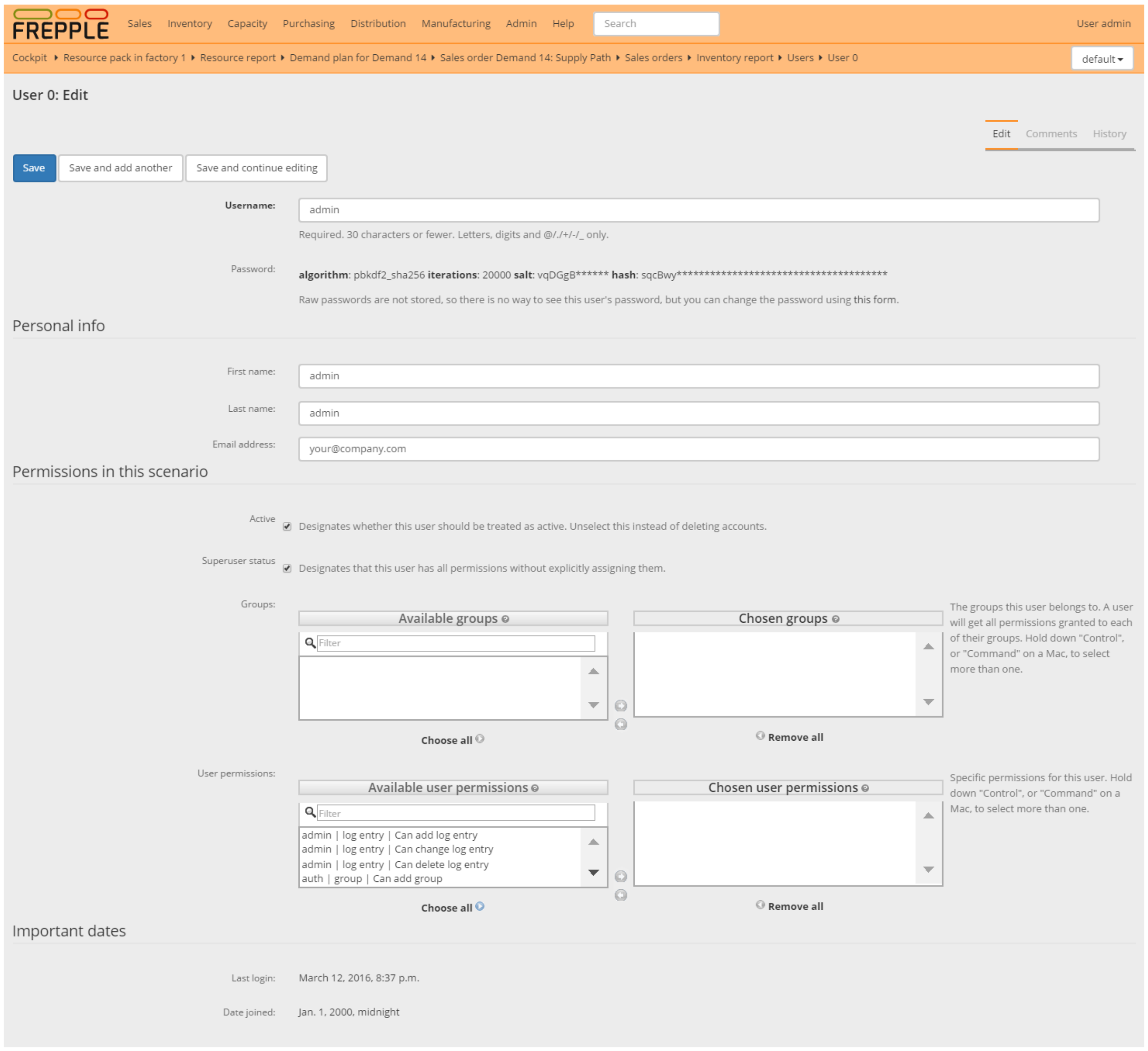
2.15 User permissions and roles

An administrator can create login accounts, set their password and permissions.

Groups are a generic way of categorizing users and apply permissions to those users. A user in a group inherits all the permissions granted to that group. A user can belong to any number of groups.

The user name, email, first name, last name, password are properties that are defined system-wide.

The active flag, superuser flag, assigned groups and user permissions can be defined per scenario. When the user is marked active in a scenario, the scenario will appear in the dropdown list on the top right of the screen.


FREPPLE
Sales Inventory Capacity Purchasing Distribution Manufacturing Admin Help
Search
User admin
Cockpit ▸ Resource pack in factory 1 ▸ Resource report ▸ Demand plan for Demand 14 ▸ Sales order Demand 14: Supply Path ▸ Sales orders ▸ Inventory report ▸ Users ▸ User 0
default
Edit
Comments
History
Save
Save and add another
Save and continue editing
Available groups
Chosen groups
Filter
Choose all
Remove all
Available user permissions
Chosen user permissions
Choose all
Remove all

User 0: Edit

Username: admin

Required. 30 characters or fewer. Letters, digits and @/./+/-/_ only.

Password: **algorithm**: pbkdf2_sha256 **iterations**: 20000 **salt**: vqDGgB****** **hash**: sqcBwy**************************************

Raw passwords are not stored, so there is no way to see this user's password, but you can change the password using this form.

Personal info

First name: admin

Last name: admin

Email address: your@company.com

Permissions in this scenario

Active ✓ Designates whether this user should be treated as active. Unselect this instead of deleting accounts.

Superuser status ✓ Designates that this user has all permissions without explicitly assigning them.

Groups: The groups this user belongs to. A user will get all permissions granted to each of their groups. Hold down "Control", or "Command" on a Mac, to select more than one.

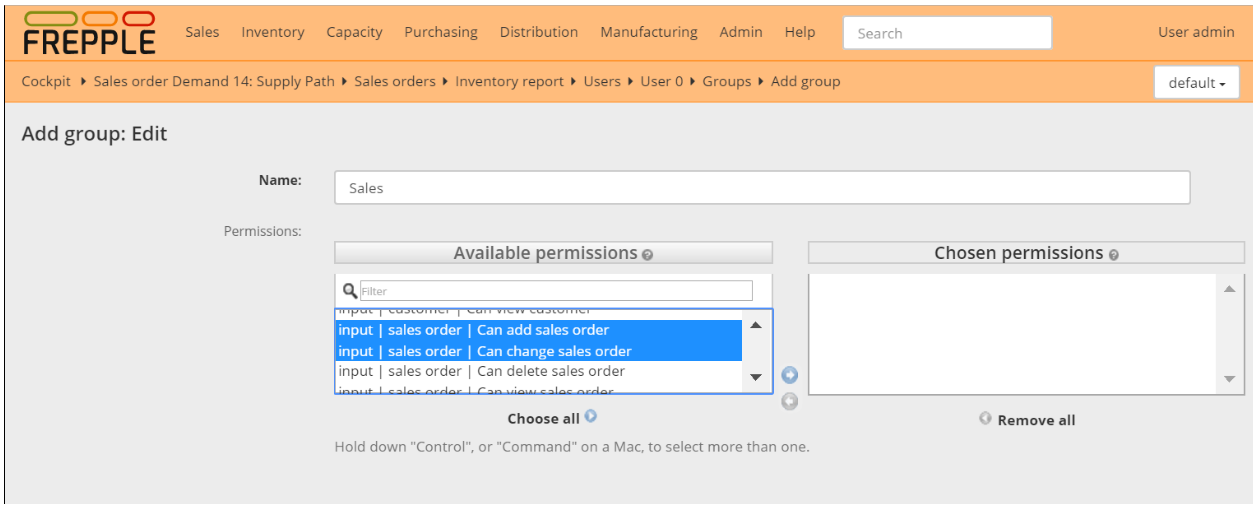
User permissions: Specific permissions for this user. Hold down "Control", or "Command" on a Mac, to select more than one.

admin | log entry | Can add log entry
admin | log entry | Can change log entry
admin | log entry | Can delete log entry
auth | group | Can add group

Important dates

Last login: March 12, 2016, 8:37 p.m.

Date joined: Jan. 1, 2000, midnight


FREPPLE
Sales Inventory Capacity Purchasing Distribution Manufacturing Admin Help
Search
User admin
Cockpit ▸ Sales order Demand 14: Supply Path ▸ Sales orders ▸ Inventory report ▸ Users ▸ User 0 ▸ Groups ▸ Add group
default
Available permissions
Chosen permissions
Filter
Choose all
Remove all

Add group: Edit

Name: Sales

Permissions:

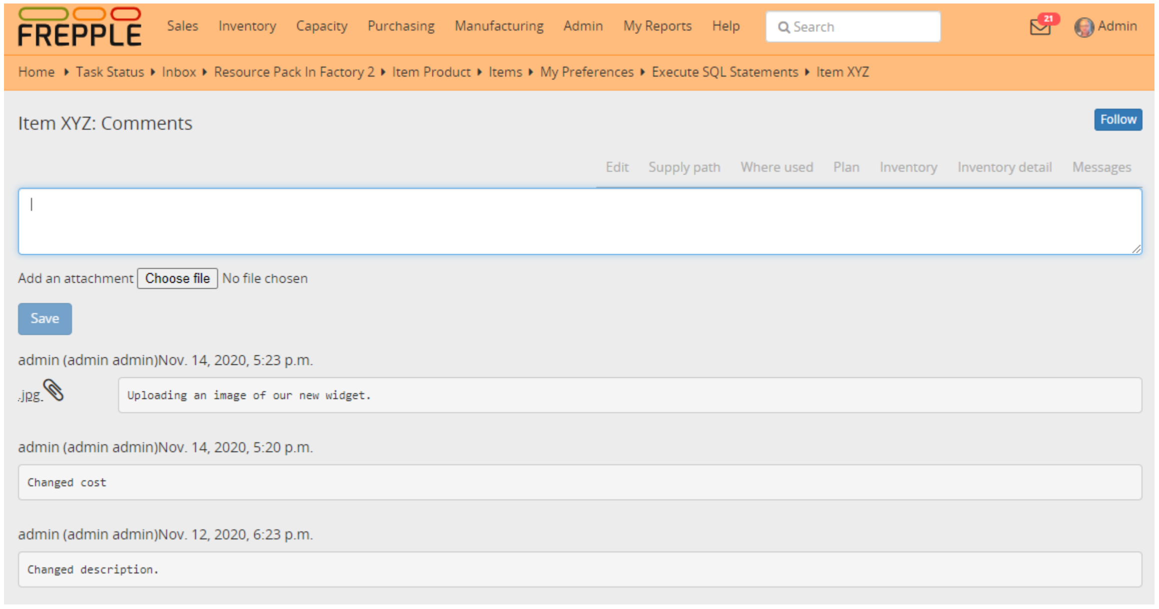
input | customer | Can view customer
input | sales order | Can add sales order
input | sales order | Can change sales order
input | sales order | Can delete sales order
input | sales order | Can view sales order

Hold down "Control", or "Command" on a Mac, to select more than one.

2.16 Messages

Each object has a message tab where you can:

- Click to "follow" button to get notifications in your inbox when there is activity on the object.
- Read and add comments.
- Upload attachments.
- See the change history of the object.



Item XYZ: Comments

Add an attachment No file chosen

admin (admin admin)Nov. 14, 2020, 5:23 p.m.

.jpg

```
Uploading an image of our new widget.
```

admin (admin admin)Nov. 14, 2020, 5:20 p.m.

```
Changed cost
```

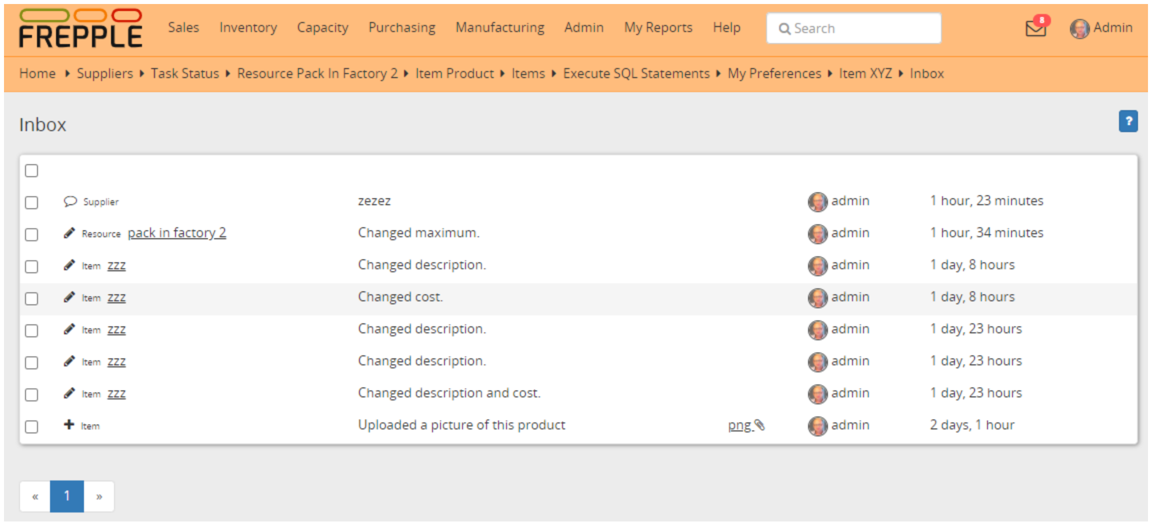
admin (admin admin)Nov. 12, 2020, 6:23 p.m.

```
Changed description.
```

2.17 Inbox

The inbox shows messages about recent activity on objects you are following. It allows you to efficiently collaborate with your colleagues on the plan.

First, you need to follow objects. Whenever there is some activity on that object you will get a message in your inbox.


Sales
Inventory
Capacity
Purchasing
Manufacturing
Admin
My Reports
Help
Search
Admin
Home ▸ Suppliers ▸ Task Status ▸ Resource Pack In Factory 2 ▸ Item Product ▸ Items ▸ Execute SQL Statements ▸ My Preferences ▸ Item XYZ ▸ Inbox
?
« 1 »

Inbox

☐	Supplier	zezez		admin	1 hour, 23 minutes
☐	Resource pack in factory 2	Changed maximum.		admin	1 hour, 34 minutes
☐	Item zzz	Changed description.		admin	1 day, 8 hours
☐	Item zzz	Changed cost.		admin	1 day, 8 hours
☐	Item zzz	Changed description.		admin	1 day, 23 hours
☐	Item zzz	Changed description.		admin	1 day, 23 hours
☐	Item zzz	Changed description and cost.		admin	1 day, 23 hours
☐	Item	Uploaded a picture of this product	png	admin	2 days, 1 hour

2.18 Tasks

FrePPLe provides a list of commands that perform actions on the database, the input data and/or the output data.

The commands can be accessed in three different ways:

- From the execution screen: *Execution screen*
- From the command line: *Command line*
- Through a web-based API: *Remote commands*

This section provides an overview of the available actions:

- Data commands
  - *Group and schedule tasks*
  - *Export a spreadsheet*
  - *Import a spreadsheet*
  - *Export plan result to folder*
  - *Email exported reports*
  - *Import data files from folder*
  - *Scenario management*
  - *Back up database*
  - *Empty the database*
- Administrator commands
  - *Load a dataset in the database*
  - *Generate time buckets*
  - *Create the PostgreSQL database(s)*
  - *Create or migrate the database schema*
  - *Restore a database backup*
  - *Create a new superuser*
  - *Change a user's password*
  - *Remove all database objects*
- Developer commands
  - *Python command prompt*
  - *Database shell prompt*
  - *Run the development web server*
  - *Run the test suite*

The list can be extended with custom commands from extension modules.

2.18.1 Data commands

Group and schedule tasks

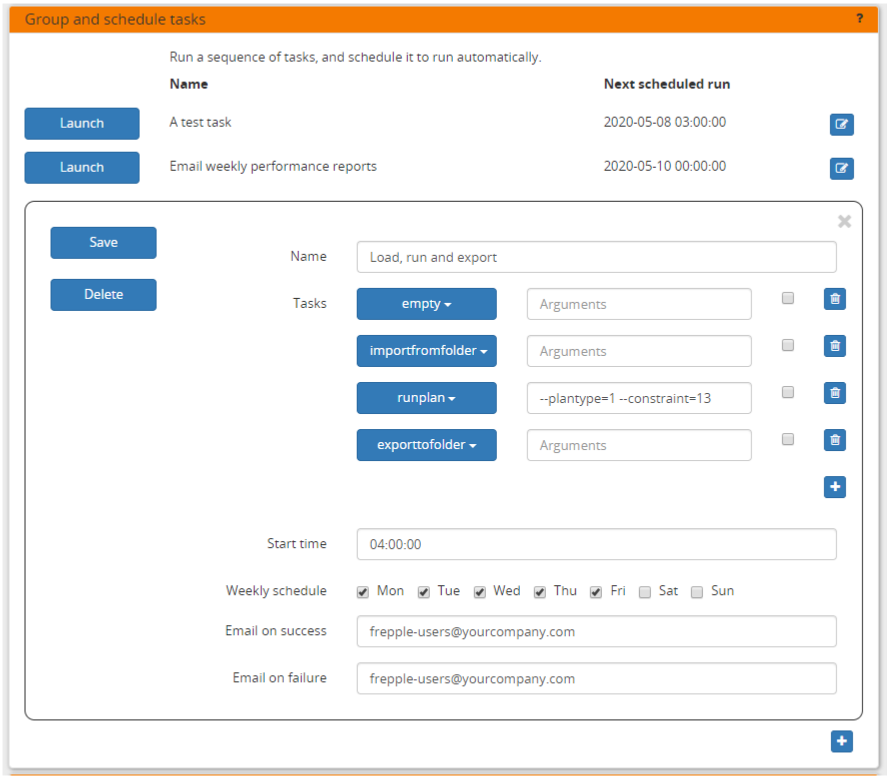
With this option a user can execute a sequence of steps together as a group.

The execution of the task group can be triggered manually. Or it can be scheduled automatically based on a predefined schedule.

Optionally, a email can be sent out upon failure or success of the execution.

For this task to be available some configuration may be required. On Windows this task is a front-end for the Windows Task Scheduler, and you need to assure the user running the web server has access to use it. On Linux this task is a front-end for the at-command, and you need to edit the /etc/at.allow or /etc/at.deny file to grant access for the user running the apache web server.

- Execution screen:



Group and schedule tasks

Run a sequence of tasks, and schedule it to run automatically.

	Name	Next scheduled run
Launch	A test task	2020-05-08 03:00:00
Launch	Email weekly performance reports	2020-05-10 00:00:00

Name: Load, run and export

Tasks:

empty	Arguments	
importfromfolder	Arguments	
runplan	--plantype=1 --constraint=13	
exporttofolder	Arguments	

Start time: 04:00:00

Weekly schedule: [✓ Mon] [✓ Tue] [✓ Wed] [✓ Thu] [✓ Fri] Sat Sun

Email on success: frepple-users@yourcompany.com

Email on failure: frepple-users@yourcompany.com

- Command line:

```
frepplectl scheduletasks --schedule=my_task_sequence
```

- Web API:

```
POST /execute/api/scheduletasks/?schedule=my_task_sequence
```

Export a spreadsheet

This task allows you to download the complete model as a single spreadsheet file. The spreadsheet can be opened with Excel or Open Office.

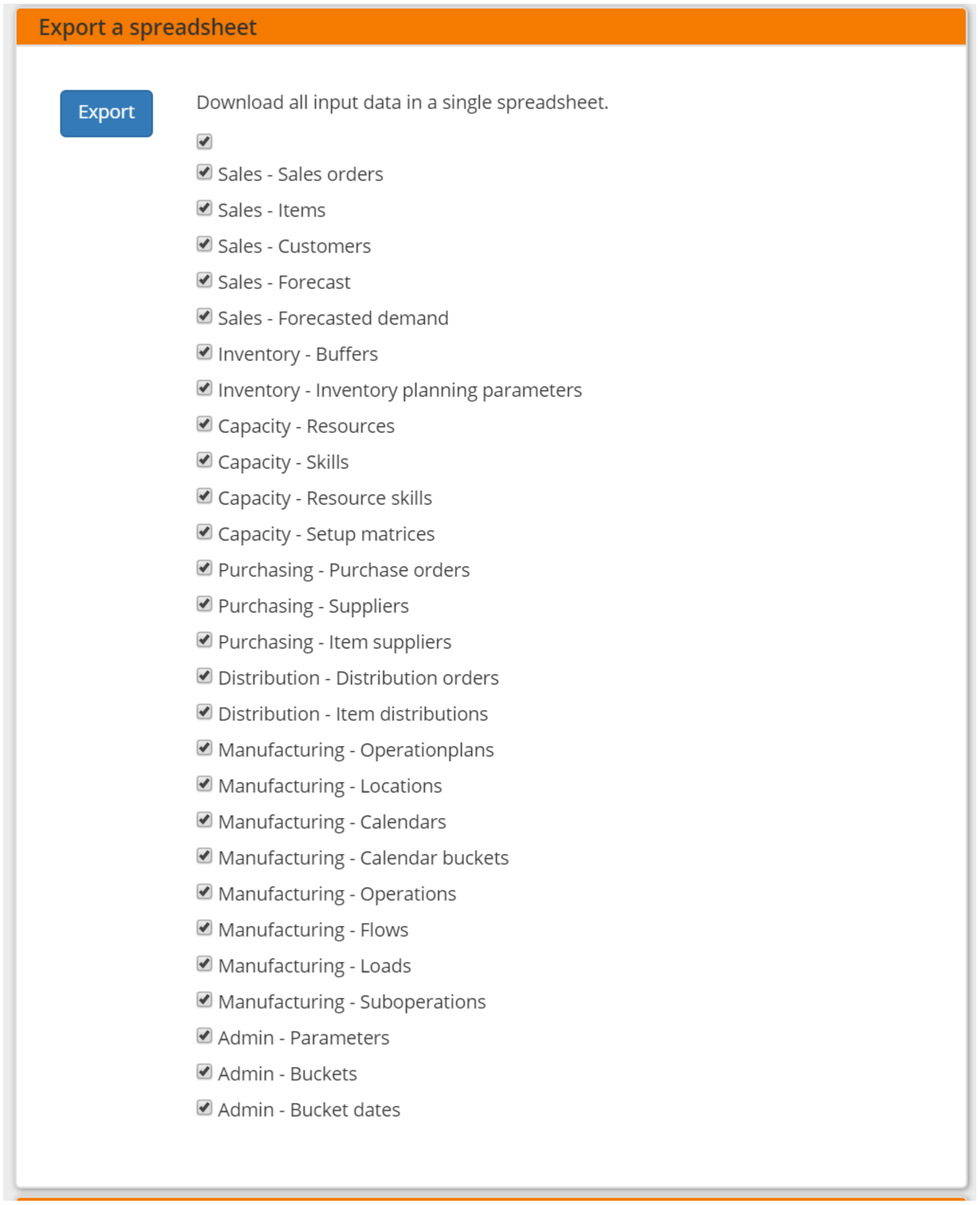
A separate sheet in the workbook is used for each selected entity.

The exported file can be imported back with the task described just below.

Optionally, you can make your dataset anonymous during the export to hide sensitive company data. All entities then get a new name during the export. It remains ABSOLUTELY NECESSARY to carefully review the generated spreadsheet and to remove any sensitive data that is still left, such as descriptions, categories, custom attributes, cost information.

This command is available only in the user interface:

- Execution screen:



Download all input data in a single spreadsheet.

- ✓
- ✓ Sales - Sales orders
- ✓ Sales - Items
- ✓ Sales - Customers
- ✓ Sales - Forecast
- ✓ Sales - Forecasted demand
- ✓ Inventory - Buffers
- ✓ Inventory - Inventory planning parameters
- ✓ Capacity - Resources
- ✓ Capacity - Skills
- ✓ Capacity - Resource skills
- ✓ Capacity - Setup matrices
- ✓ Purchasing - Purchase orders
- ✓ Purchasing - Suppliers
- ✓ Purchasing - Item suppliers
- ✓ Distribution - Distribution orders
- ✓ Distribution - Item distributions
- ✓ Manufacturing - Operationplans
- ✓ Manufacturing - Locations
- ✓ Manufacturing - Calendars
- ✓ Manufacturing - Calendar buckets
- ✓ Manufacturing - Operations
- ✓ Manufacturing - Flows
- ✓ Manufacturing - Loads
- ✓ Manufacturing - Suboperations
- ✓ Admin - Parameters
- ✓ Admin - Buckets
- ✓ Admin - Bucket dates

Import a spreadsheet

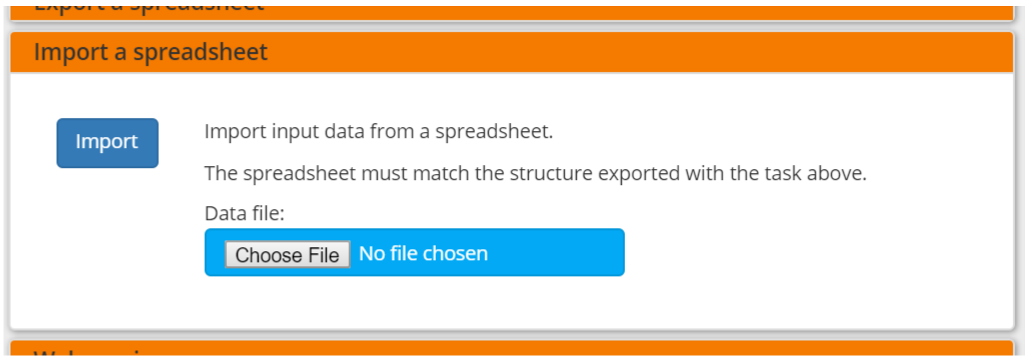
This task allows you to import an Excel spreadsheet.

A separate sheet in the workbook is used for each selected entity.

The sheet must have the right names - in English or your language. The first row in each sheet must contain the column names.

This command is available only in the user interface:

- Execution screen:



Import a spreadsheet

Import input data from a spreadsheet.

The spreadsheet must match the structure exported with the task above.

Data file:

Export plan result to folder

This task allows exporting data to a set of files in CSV or Excel format. The purpose of this task is to help the exchange of information with other systems.

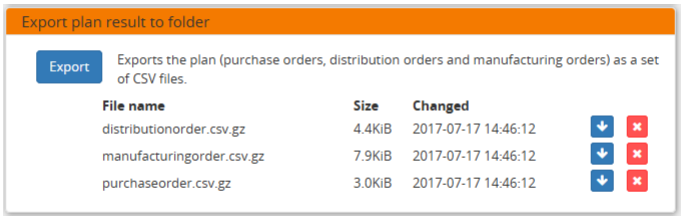
The command can easily by customized to export the results you need.

The files are all placed in a folder UPLOADFILEFOLDER/export/, which can be configured per scenario with the UPLOADFILEFOLDER value in the djangosettings.py file.

The exported files can be accessed from the user interface, or through over a HTTP(S) web interface.

This command is available in the user interface, the command line and the web API:

- Execution screen:



Export plan result to folder

Exports the plan (purchase orders, distribution orders and manufacturing orders) as a set of CSV files.

File name	Size	Changed
distributionorder.csv.gz	4.4KiB	2017-07-17 14:46:12
manufacturingorder.csv.gz	7.9KiB	2017-07-17 14:46:12
purchaseorder.csv.gz	3.0KiB	2017-07-17 14:46:12

- Command line:

```
frepplectl exporttofolder
```

- Web API:

```
Export the planning result files:
POST /execute/api/exporttofolder/
```

(continues on next page)

(continued from previous page)

```
Retrieve one of the exported files:
GET /execute/downloadfromfolder/1/<filename>/
```

Email exported reports

Reports that have been exported using *Export plan result to folder* command can be emailed to one or more recipients.

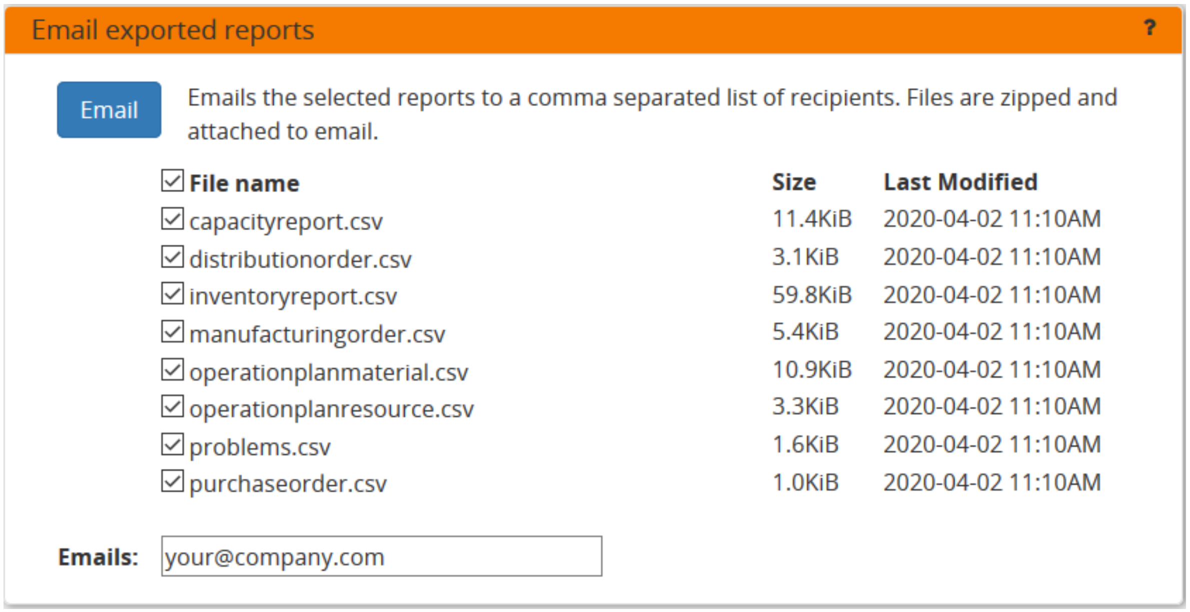
Recipients have to be separated by a comma in the *Emails* field.

Selected reports are zipped into a *reports.zip* file that is attached to the email.

In order to have this command working, the EMAIL parameters in the djangosettings.py file must be properly configured.

This command is available in the user interface, the command line and the web API:

- Execution screen:



Emails the selected reports to a comma separated list of recipients. Files are zipped and attached to email.

✓ File name	Size	Last Modified
✓ capacityreport.csv	11.4KiB	2020-04-02 11:10AM
✓ distributionorder.csv	3.1KiB	2020-04-02 11:10AM
✓ inventoryreport.csv	59.8KiB	2020-04-02 11:10AM
✓ manufacturingorder.csv	5.4KiB	2020-04-02 11:10AM
✓ operationplanmaterial.csv	10.9KiB	2020-04-02 11:10AM
✓ operationplanresource.csv	3.3KiB	2020-04-02 11:10AM
✓ problems.csv	1.6KiB	2020-04-02 11:10AM
✓ purchaseorder.csv	1.0KiB	2020-04-02 11:10AM

Emails: your@company.com

- Command line:

```
frepplectl emailreport [--sender] --recipient --report
```

- Web API:

```
Email exported reports:
POST /execute/api/emailreport/?recipient=recipient1,recipient2...&report=report1,
↪report2,report3...
```

Import data files from folder

This task allows importing data from a set of CSV-formatted files (eventually GZ-compressed). The purpose of this task is to help the exchange of information with other systems.

The files are all placed in a folder that is configurable per scenario with the UPLOADFILEFOLDER in the djangosettings.py configuration file. The log file records all data imports, in addition to any data errors identified during their processing.

The data files to be imported must meet the following criteria:

- The name must match the data object they store: eg demand.csv, item.csv, item.xlsx, item.csv.gz
  This is important for frePPLe to understand the correct processing order of the files.
- Multiple files for the same entity can be provided. They will be processed in alphabetical order: eg "demand (1).xlsx", "demand (2).csv", "demand.1.csv", "demand.2.csv", "demand.extra.xlsx", "demand.postprocessing.sql"
- The first line of the file should contain the field names. The field name can be in English or the default language configured with the LANGUAGE_CODE setting.

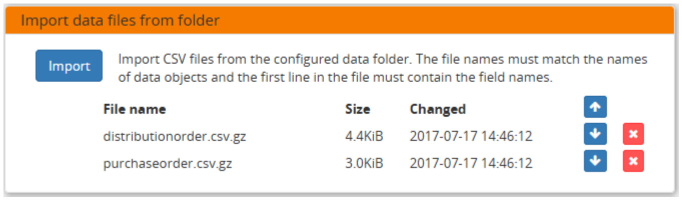
The following file formats are accepted:

- **Excel**:
  The file name must end with .xlsx
- **CSV**:
  The file name must end with .csv (or .csv.gz when compressed with gzip).
  Some specific notes on the CSV format:
  - The separator in your CSV-files varies with the chosen language: If in your language a comma is used as a decimal separator for numbers, the CSV file will use a semicolon (;) as delimiter. Otherwise a comma (,) is used. See http://en.wikipedia.org/wiki/Decimal_mark
  - The date format expected by frePPLe is 'YYYY-MM-DD HH:MM:SS'.
  - The data file is expected to be encoded in the character encoding defined by the setting CSV_CHARSET (default UTF-8).
- **PostgreSQL copy files**:
  The file name must end with .cpy (or .cpy.gz when compressed with gzip).
  Uploading in this format goes MUCH quicker than the other formats. It has some limitations however: a) the validation of the input data is not as extensive as the other formats, b) a single faulty record will abort the upload and c) it only supports adding new records and not updating existing records.
  This method is therefore only recommended for loading very large data files with clean data.
- **SQL**:
  The file name must end with .sql (or .sql.gz when compressed with gzip).
  For security reasons a database role with a minimal set of permissions must be define. The setting DATABASES / SQL_ROLE needs to refer to this role.

In this option you can see a list of files present in the specified folder, and download each file by clicking on the arrow down button, or delete a file by clicking on the red button. The arrow up button will give the user the possibility of selecting multiple files to upload to that folder.

This command is available in the user interface, the command line and the web API:

- Execution screen:


Import data files from folder

Import CSV files from the configured data folder. The file names must match the names of data objects and the first line in the file must contain the field names.

File name	Size	Changed
distributionorder.csv.gz	4.4KiB	2017-07-17 14:46:12
purchaseorder.csv.gz	3.0KiB	2017-07-17 14:46:12

- Command line:

```
frepplectl importfromfolder
```

- Web API:

```
Upload a data file:
POST /execute/uploadtofolder/0/ with data files in multipart/form-data format

Import the data files:
POST /execute/api/importfromfolder/
```

Scenario management

This option allows a user to either create copies of a dataset into a what-if scenario or promote the data from a scenario into *Production* database.

When the data is successfully copied, the status changes from ‘Free’ to ‘In use’. The access to the newly copied scenario is limited to 1) the user who performed the copy plus 2) all superusers of the source scenario.

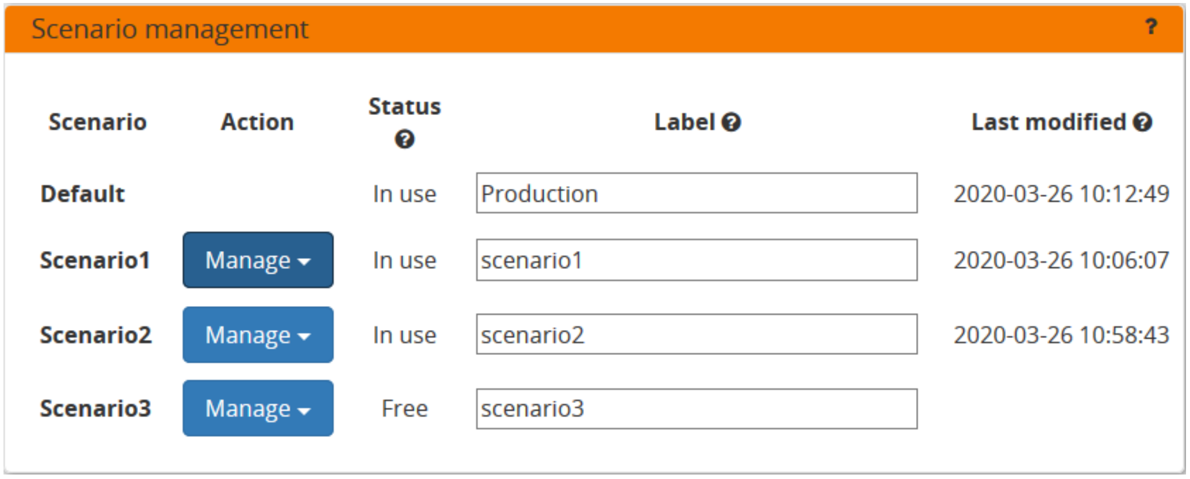
When the user doesn’t need the what-if scenario any more, it can be released again.

Releasing a scenario can be done from any scenario while copying and promoting actions can only be performed from current scenario to destination scenario.

The label of a scenario, which is displayed in the dropdown list in the upper right hand corner, can also be updated here.

This command is available in the user interface, the command line and the web API:

- Execution screen:


Scenario management

Scenario	Action	Status	Label	Last modified
Default		In use	Production	2020-03-26 10:12:49
Scenario1	Manage	In use	scenario1	2020-03-26 10:06:07
Scenario2	Manage	In use	scenario2	2020-03-26 10:58:43
Scenario3	Manage	Free	scenario3	

- Command line:

```
To copy scenario scenario1 into scenario scenario2:
frepplectl scenario_copy [--force --promote] scenario1 scenario2

To release scenario scenario1:
frepplectl scenario_release --database=scenario1
```

- Web API:

```
To copy a scenario (including Production) into another scenario:
* POST /execute/api/scenario_copy/?copy=1&source=scenario1&destination=scenario2&
↪force=1

To release a scenario named scenario1:
* POST /scenario1/execute/api/scenario_copy/?release=1

To promote a scenario named scenario1 into Production (where "default" is the␣
↪Production name):
* POST /execute/api/scenario_copy/?promote=1&source=scenario1&destination=default
```

Back up database

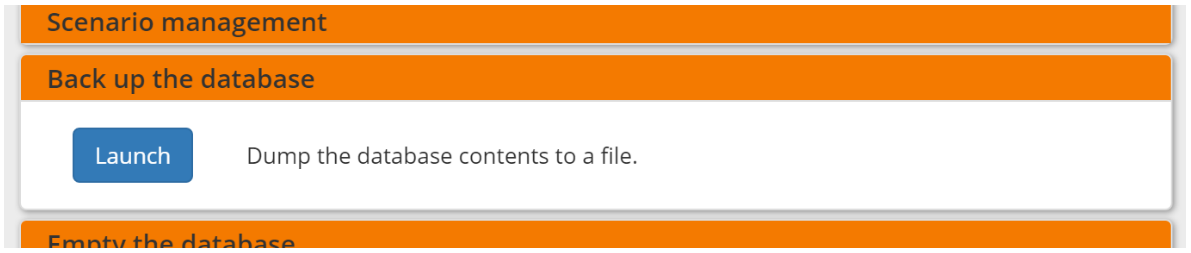
This task dumps the contents of the current database schema to a backup file. The file is created in the log folder configured in the configuration files djangosettings.py. It can be downloaded from the browser.

For security reasons the command is only available to users listed in the setting SUPPORT_ACCOUNTS. By default this is an empty list.

The command also removes dumps older than a month to limit the disk space usage. If you want to keep dumps for a longer period of time, you'll need to copy the backup files to a different location.

This command is available in the user interface, the command line and the web API:

- Execution screen:



- Command line:

```
frepplectl backup
```

- Web API:

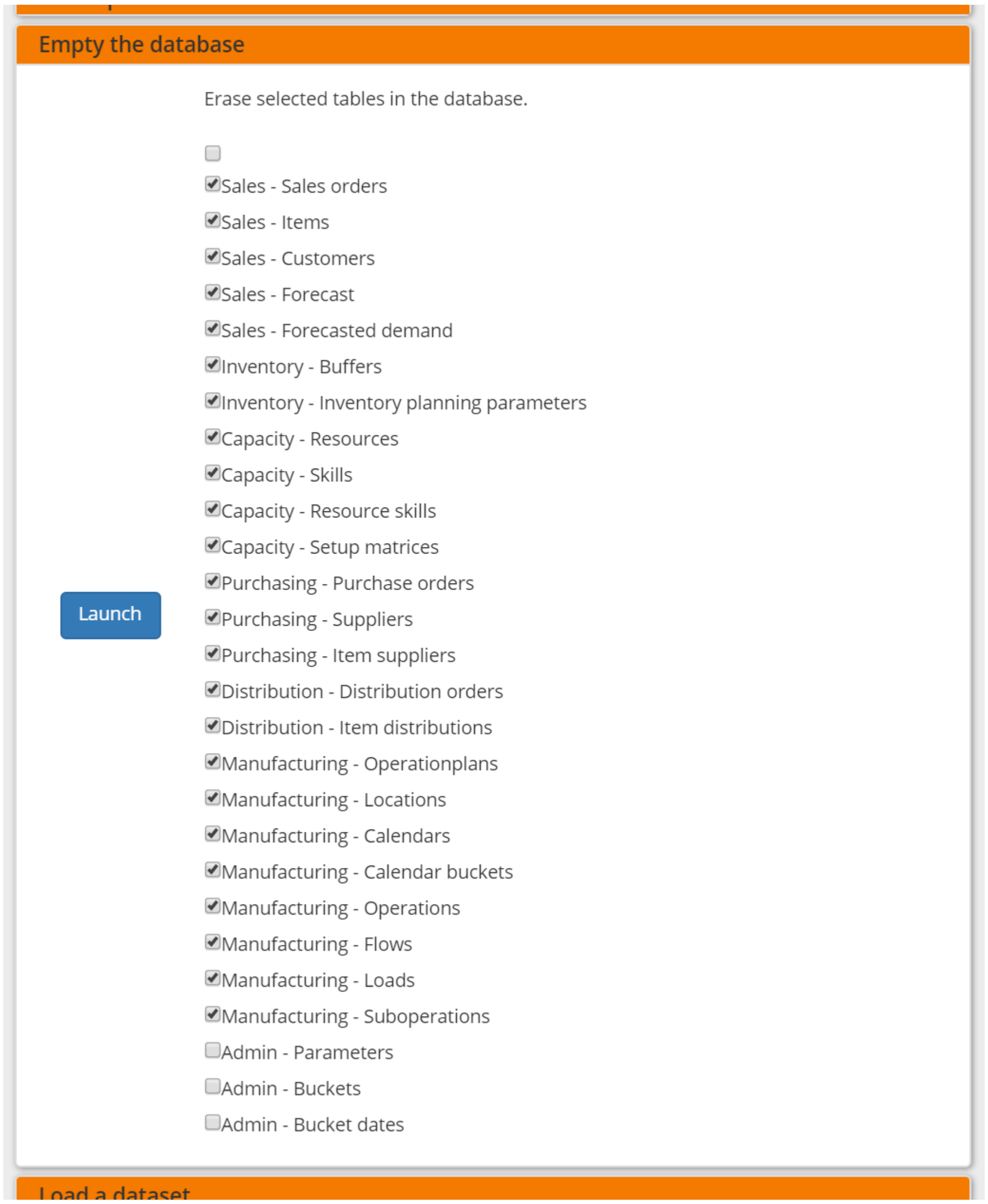
```
Create a backup:
POST /execute/api/backup/

Download the backup file:
GET /execute/logdownload/<task identifier>/
```

Empty the database

This will delete all data from the current scenario (except for some internal tables for users, permissions, task log, etc…).

This command is available in the user interface, the command line and the web API:

- Execution screen:



Empty the database

Erase selected tables in the database.

- [ ]
- [x] Sales - Sales orders
- [x] Sales - Items
- [x] Sales - Customers
- [x] Sales - Forecast
- [x] Sales - Forecasted demand
- [x] Inventory - Buffers
- [x] Inventory - Inventory planning parameters
- [x] Capacity - Resources
- [x] Capacity - Skills
- [x] Capacity - Resource skills
- [x] Capacity - Setup matrices
- [x] Purchasing - Purchase orders
- [x] Purchasing - Suppliers
- [x] Purchasing - Item suppliers
- [x] Distribution - Distribution orders
- [x] Distribution - Item distributions
- [x] Manufacturing - Operationplans
- [x] Manufacturing - Locations
- [x] Manufacturing - Calendars
- [x] Manufacturing - Calendar buckets
- [x] Manufacturing - Operations
- [x] Manufacturing - Flows
- [x] Manufacturing - Loads
- [x] Manufacturing - Suboperations
- [ ] Admin - Parameters
- [ ] Admin - Buckets
- [ ] Admin - Bucket dates

- Command line:

```
frepplectl empty --models=input.demand,input.operationplan
```

- Web API:

```
POST /execute/api/empty/?models=input.demand,input.operationplan
```

2.18.2 Administrator commands

Load a dataset in the database

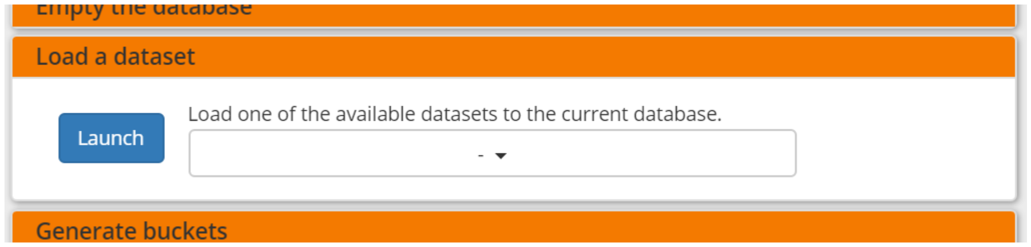
A number of demo datasets are packaged with frePPLe. Using this action you can load one of those in the database.

The dataset is loaded incrementally in the database, **without** erasing any previous data. In most cases you'll want to erase the data before loading any of these datasets.

You can use the dumpdata command to export a model to the appropriate format and create your own predefined datasets.

This command is available in the user interface, the command line and the web API:

- Execution screen:



- Command line:

```
frepplectl loaddata manufacturing_demo
```

- Web API:

```
POST /execute/api/loaddata/?fixture=manufacturing_demo
```

Generate time buckets

Many output reports are displaying the plan results aggregated into time buckets. These time buckets are defined with the tables dates and bucket dates.

This tasks allows you to populate these tables in an easy way with buckets with daily, weekly, monthly, quarterly and yearly granularity. Existing bucket definitions for these granularities will be overwritten.

The following arguments are used:

- Start date, end date:
  Definition of the horizon to generate buckets for.
- Week start: Defines the first date of a week.
- Day name, week name, month name, quarter name, year name:
  Template used to generate a name for the buckets.

  Any character can be used in the names and the following format codes can be used:

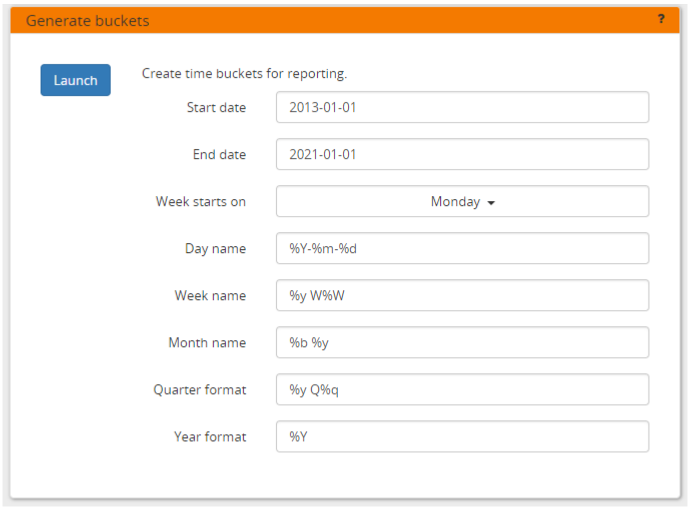
  – %a: Weekday as locale's abbreviated name. Eg: Sun, Mon, . . .

  – %A: Weekday as locale's full name. Eg: Sunday, Monday, . . .

- %w: Weekday as a decimal number, where 0 is Sunday and 6 is Saturday.
- %d: Day of the month as a zero-padded decimal number. Eg: 01, 02, …, 31
- %b: Month as locale's abbreviated name. Eg: Jan, Feb, …
- %B: Month as locale's full name. Eg: January, February, …
- %m: Month as a zero-padded decimal number. Eg: 01, 02, …, 12
- %q: Quarter as a decimal number. Eg: 1, 2, 3, 4
- %y: Year without century as a zero-padded decimal number. Eg: 00, 01, …, 99
- %Y: Year with century as a decimal number. Eg: 2018, 2019, …
- %j: Day of the year as a zero-padded decimal number. Eg: 001, 002, …, 366
- %U: Week number of the year as a zero padded decimal number. Eg: 00, 01, …
- %W: Week number of the year as a decimal number. Eg: 0, 1, …
- %%: A literal '%' character.

This command is available in the user interface, the command line and the web API:

- Execution screen:



Create time buckets for reporting.

Start date	2013-01-01
End date	2021-01-01
Week starts on	Monday
Day name	%Y-%m-%d
Week name	%y W%W
Month name	%b %y
Quarter format	%y Q%q
Year format	%Y

- Command line:

```
frepplectl createbuckets --start=2012-01-01 --end=2020-01-01 --weekstart=1
```

- Web API:

```
POST /execute/api/createbuckets/?start=2012-01-01&end=2020-01-01&weekstart=1
```

Create the PostgreSQL database(s)

This command will create the PostgreSQl databases for frePPLe.

If the database already exists you will be prompted to confirm whether you really to loose all data in the existing database. When confirmed that database will dropped and recreated.

This command is available on the command line only:

```
# Create all scenario databases
frepplectl createdatabase

# Recreate only a single database
frepplectl createdatabase --database=scenario3
```

Create or migrate the database schema

Update the database structure to the latest release

This command is available on the command line only:

```
# Migrate the main database
frepplectl migrate

# Migrate a scenario database
frepplectl migrate --database=scenario1
```

Restore a database backup

This command is available on the command line only:

```
frepplectl restore database_dump_file
```

Create a new superuser

This command creates a new user with full access rights.

This action is possible in the user interface and the command line:

- User interface:

  See *User permissions and roles*

- Command line:

  ```
  frepplectl createsuperuser new_user_name
  ```

Change a user's password

This command changes the password of a certain user.

This action is possible in the user interface and the command line:

- User interface:

  See *Changing your password* and *User permissions and roles*.

- Command line:

  ```
  frepplectl changepassword user_name
  ```

Remove all database objects

This command completely empties all tables in the database, including all log, users, user preferences, permissions, etc…

A complete reset of the database is not very common. In most situations the command described above to empty the database is sufficient. It empties the data tables, but leaves the important configuration information intact.

This command is available on the command line only:

```
frepplectl flush
```

2.18.3 Developer commands

Database shell prompt

This command runs an interactive SQL session on the PostgreSQL database.

```
frepplectl dbshell --database=default
```

Python command prompt

This command runs an interactive Python interpreter session.

```
frepplectl shell
```

Run the test suite

Run the test suite for the user interface.

```
frepplectl test freppledb
```

Run the development web server

Run a development web server, which automatically reloads when code is changed.

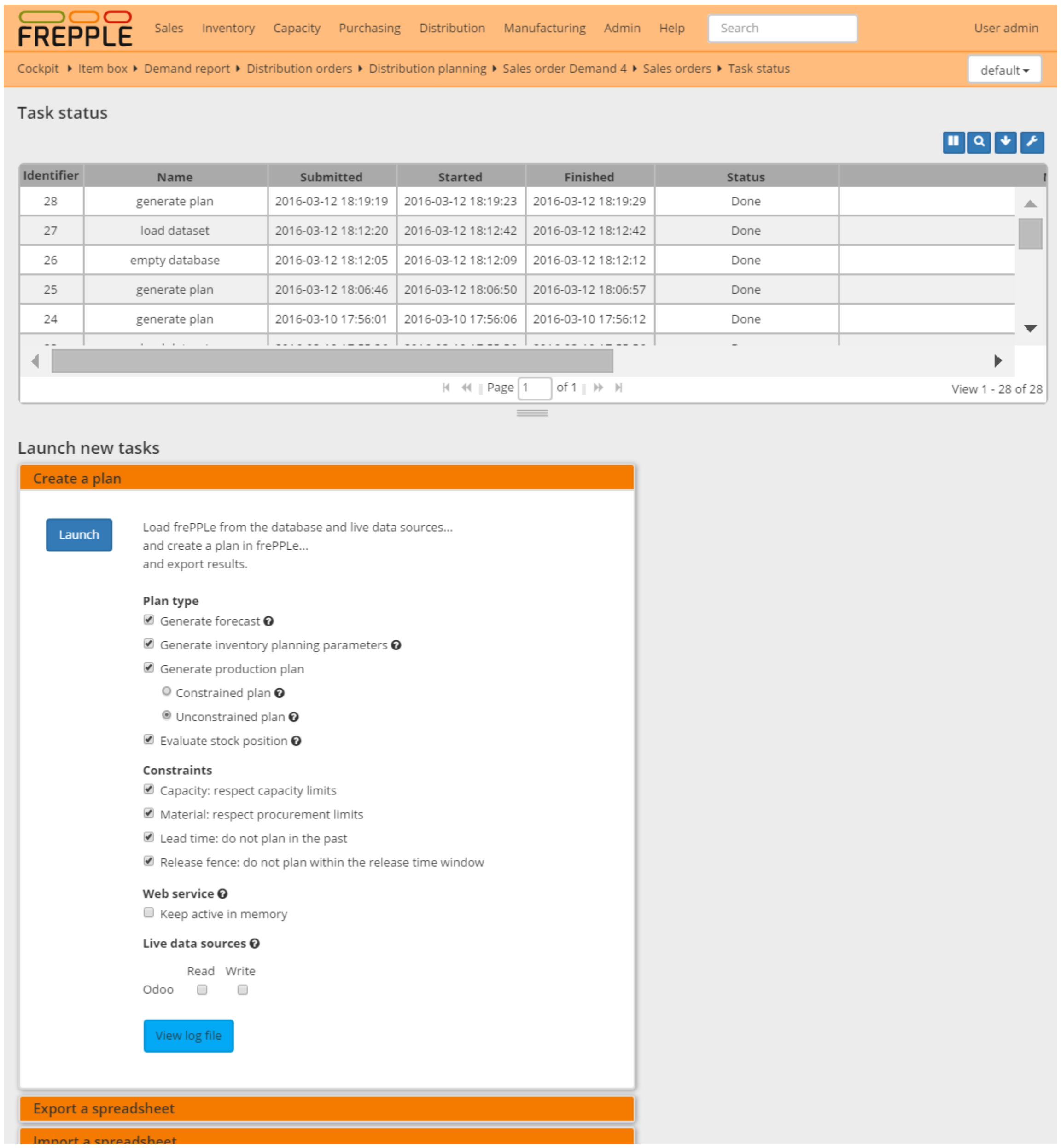
For production use this web server doesn't scale enough.

```
frepplectl runserver
```

2.19 Execution screen

This screen allows you to perform a number of administrative actions.

- *Export a spreadsheet*
- *Import a spreadsheet*
- *Export plan result to folder*
- *Import data files from folder*
- *Scenario management*
- *Back up database*
- *Empty the database*
- *Load a dataset in the database*
- *Generate time buckets*


FREPPLE
Sales Inventory Capacity Purchasing Distribution Manufacturing Admin Help
Search
User admin
Cockpit ▸ Item box ▸ Demand report ▸ Distribution orders ▸ Distribution planning ▸ Sales order Demand 4 ▸ Sales orders ▸ Task status
default ▾

Task status

Identifier	Name	Submitted	Started	Finished	Status
28	generate plan	2016-03-12 18:19:19	2016-03-12 18:19:23	2016-03-12 18:19:29	Done
27	load dataset	2016-03-12 18:12:20	2016-03-12 18:12:42	2016-03-12 18:12:42	Done
26	empty database	2016-03-12 18:12:05	2016-03-12 18:12:09	2016-03-12 18:12:12	Done
25	generate plan	2016-03-12 18:06:46	2016-03-12 18:06:50	2016-03-12 18:06:57	Done
24	generate plan	2016-03-10 17:56:01	2016-03-10 17:56:06	2016-03-10 17:56:12	Done

Launch new tasks

Create a plan

Load frePPLe from the database and live data sources...
and create a plan in frePPLe...
and export results.

Plan type

- [✓] Generate forecast
- [✓] Generate inventory planning parameters
- [✓] Generate production plan
  - ○ Constrained plan
  - ● Unconstrained plan
- [✓] Evaluate stock position

Constraints

- [✓] Capacity: respect capacity limits
- [✓] Material: respect procurement limits
- [✓] Lead time: do not plan in the past
- [✓] Release fence: do not plan within the release time window

Web service

- [ ] Keep active in memory

Live data sources

	Read	Write
Odoo	☐	☐

Export a spreadsheet

Import a spreadsheet

The status section at the top of the screen is refreshed every 5 seconds. You can disable the refreshing by clicking on the autorefresh icon.

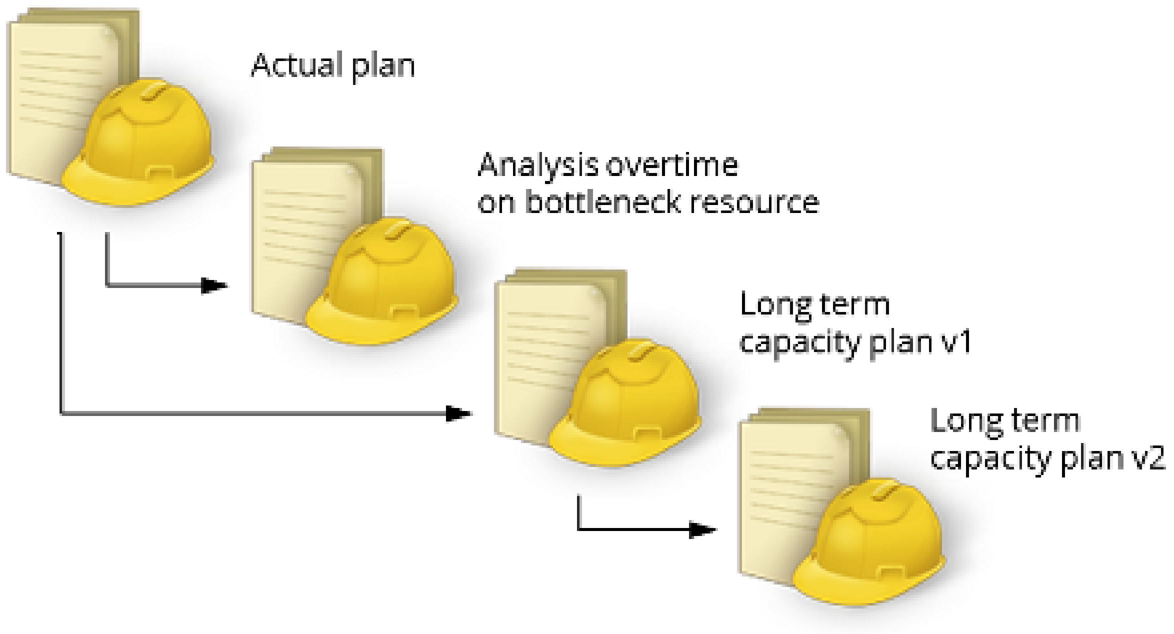
Tasks launched in this screen are all executed asynchronously: when you hit the launch button the task is added to a job queue. A separate worker process will execute the tasks from the queue.

All the actions (and more) can also be performed synchronously from the command line with the frepplectl script: see *Command line*

It is also possible to launch tasks through a web interface: see *Remote commands*

2.20 What-if scenarios

FrePPLe allows users to easily create alternative plans. A scenario is complete sandbox copy of all data, such that any modification done in it doesn't impact the production plan.

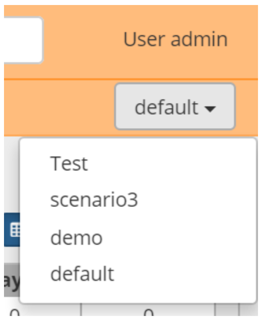

Actual plan
Analysis overtime on bottleneck resource
Long term capacity plan v1
Long term capacity plan v2

- *Selecting a scenario*
- *Scenario management*
- *Access rights and permissions*

2.20.1 Selecting a scenario

When scenarios are in use, a drop down list appears in the upper right corner. It allows you to select the scenario to work in.

Only scenarios to which you have access rights will be shown in the list.



2.20.2 Scenario management

During the installation a number of what-if slots (3 by default) are configured by the administrator. See *this page* for the details.

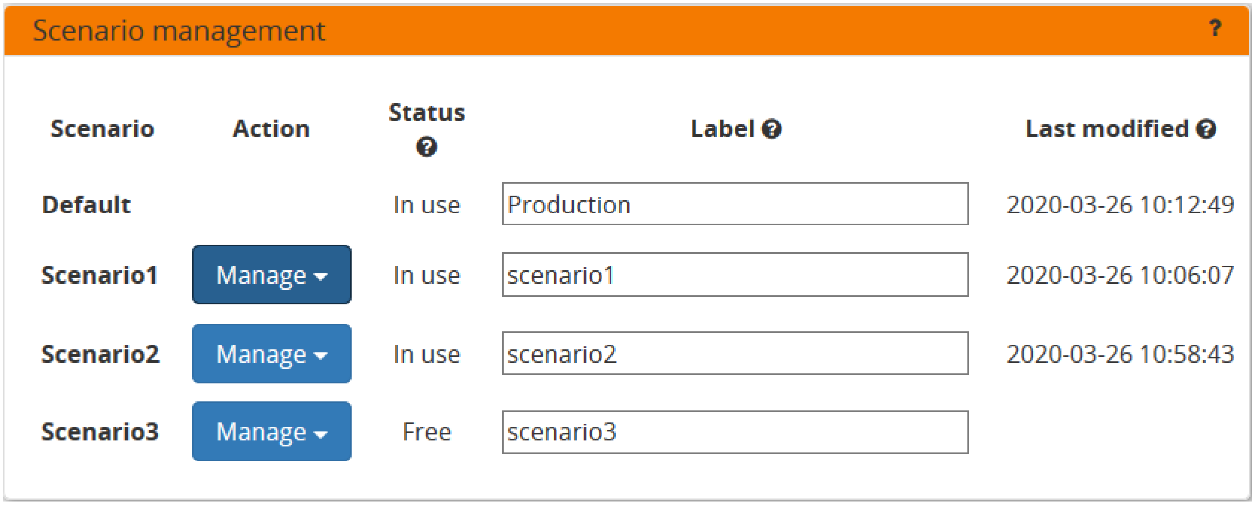
The scenario's can have the following states:

- **Free**: The slot is currently unallocated and available for use.
- **In Use**: Data has been copied into the scenario slot. Users can freely work independently in the scenario, without affecting the main model.

In *the execution screen*, you can change the status of a scenario slot:

- **Copy** is used to duplicate an existing schema into a free slot.
  After copying the scenario slot moves from *free* to *in use*.
- **Release** is used to flag that work on the what-if scenario slot has finished.
  After releasing the scenario slot moves from *in use* to *free* again.

The label shown in the scenario selection dropdown can also be updated in this screen.


Scenario management ?

Scenario	Action	Status	Label	Last modified
Default		In use	Production	2020-03-26 10:12:49
Scenario1	Manage	In use	scenario1	2020-03-26 10:06:07
Scenario2	Manage	In use	scenario2	2020-03-26 10:58:43
Scenario3	Manage	Free	scenario3	

2.20.3 Access rights and permissions

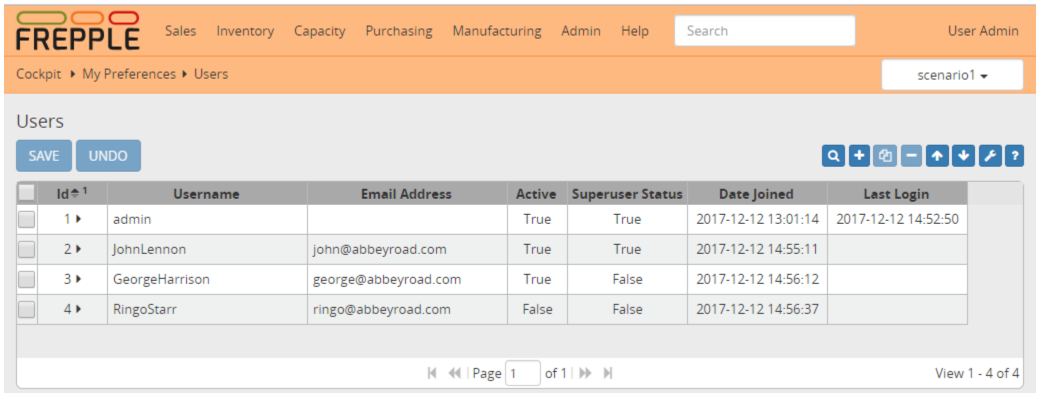
Access rights and permissions can be managed for each scenario individually.

When copying of a new scenario, it will initially be accessible by 1) the user creating the copy and 2) all superusers in the source scenario.

A user must be marked active in a scenario before it will appear in the list of available scenarios.

The superuser status of a user can be different in each scenario. Users can have completely different role and permissions in each scenario.

The list of users and their passwords is always identical in all scenarios.



Users

Id	Username	Email Address	Active	Superuser Status	Date Joined	Last Login
1	admin		True	True	2017-12-12 13:01:14	2017-12-12 14:52:50
2	JohnLennon	john@abbeyroad.com	True	True	2017-12-12 14:55:11	
3	GeorgeHarrison	george@abbeyroad.com	True	False	2017-12-12 14:56:12	
4	RingoStarr	ringo@abbeyroad.com	False	False	2017-12-12 14:56:37	

Page 1 of 1 View 1 - 4 of 4

See *User permissions and roles* for more details on the configuration of access rights.

CHAPTER

THREE

INTEGRATION GUIDE

This chapter describe frePPLe's capabilities to integrate with external systems.

The first sections describe the different ways to integrate your data sources with frePPLe.

The batch command section contains information on how you can automate and schedule tasks in frePPLe.

A final section contains information for people that require a low level interaction with the frePPLe planning engine.

3.1 Excel files

For small models and prototype models the most convenenient way of integrating the data will be to work with excel workbooks.

- Individual data tables can be exported and imported directly from the user interface.
  See *Importing data* and *Exporting data*.
- Multiple data tables can be exported and imported in single workbook. The workbook contains a separate sheet for each table.
  See *Execution screen*.

3.2 CSV text files

FrePPLe can import CSV-formatted files from a configurable data directory. And frePPLe can export its planning results in a set of CSV-formatted files as well.

The files are all placed in a folder that is configurable with the UPLOADFILEFOLDER in the djangosettings.py configuration file. The log files importfromfolder.log and exporttofolder.log record all data imports and file exports, together with any data errors identified during their processing.

The data files to be imported must meet the following criteria:

- The file name must start with the name of the data object they store.
  The file name must end with the extension .csv or or .csv.gz. Files with the extension .csv.gz are expected to compressed in gzip format.
  Some examples of valid file names are item.csv.gz, item.csv, demand.csv, demand.part1.csv
- The first line of the file should contain the field names
- The file should be in CSV format. The delimiter depends on the default language (configured with LANGUAGE_CODE in djangosettings.py). For english-speaking countries it's a comma. For European countries it's a semicolon.
- The file should be encoded in UTF-8 (configurable with the CSV_CHARSET setting in djangosettings.py)

The export can be customized, i.e. export only the relevant data and with the a specific format (file names, dates, separators, …). The customization is done by copying the exporttofolder.py file and updating the SQL statements it contains.

The export and import can be run in 2 ways:

- In the user interface a user can interactively launch the task in the *execution screen*.
- You can run the task from the command line using the *frepplectl utility*.

```
frepplectl exporttofolder

frepplectl importfromfolder
```

3.3 Database access

For large data volumes a data integration with an ETL tool to directly connect to the PostgreSQL database will be the most efficient.

The frePPLe database schema is very transparent and straightforward to allow this type of integration.

Keep in mind however:

- Your ETL-interface will need to capture and handle all types of invalid data. With all other integration methods the frePPLe API layer catches and reports such errors, but not in this integration mode.
- The database schema can change between releases. We provide no guaruntee that your ETL-interface will be future-proof.

3.4 REST API

FrePPLe provides a state of the art REST API (see https://en.wikipedia.org/wiki/Representational_state_transfer) for data exchange with other applications.

Features:

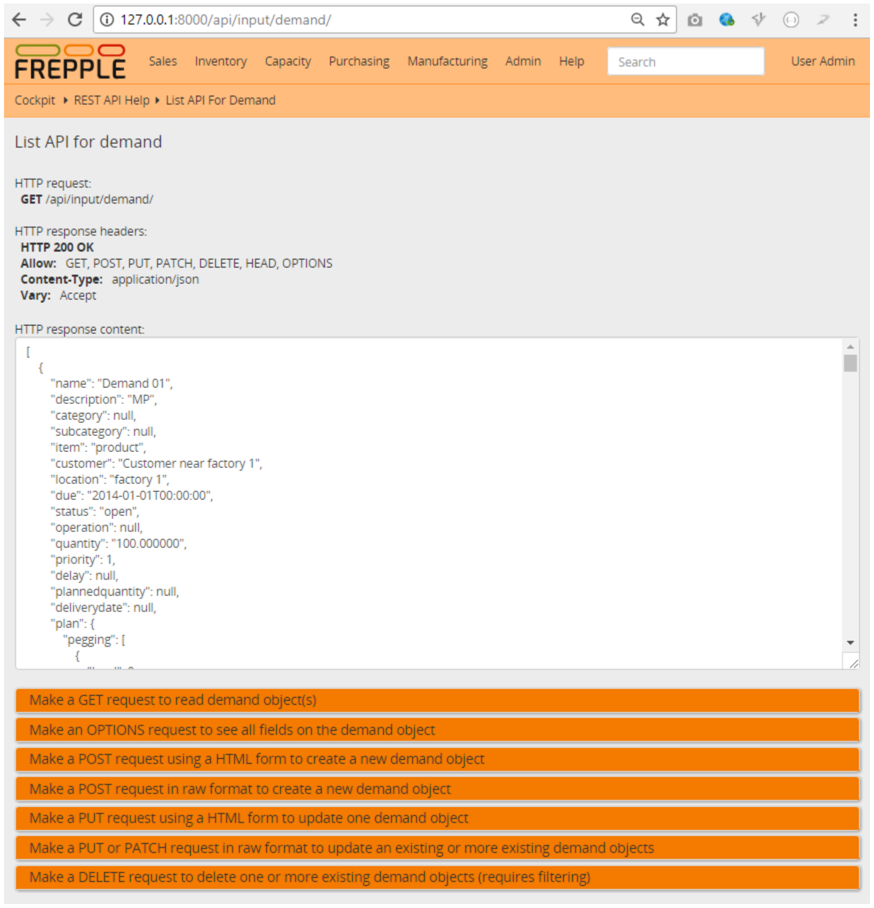
- Fast and efficient.
- Supports multiple formats.
- Open and extendible architecture, leveraging the excellent django REST framework from Tom Christie: see http://www.django-rest-framework.org/

When you access Help in the navigation Menu, you will find a API Help entry. This page allows you to test and experiment with the API from your browser. It shows all URLs and HTTP methods supported by the API.

3.4.1 List API from your browser

The top part of this screen show the headers and content of the HTTP response of the web service.

The different sections below allow you to create new request for the GET, OPTIONS, POST, PUT, PATCH and DELETE HTTP methods. In the next image you can see a list of all demand objects and all the fields of each object.



List API for demand

HTTP request:
GET /api/input/demand/

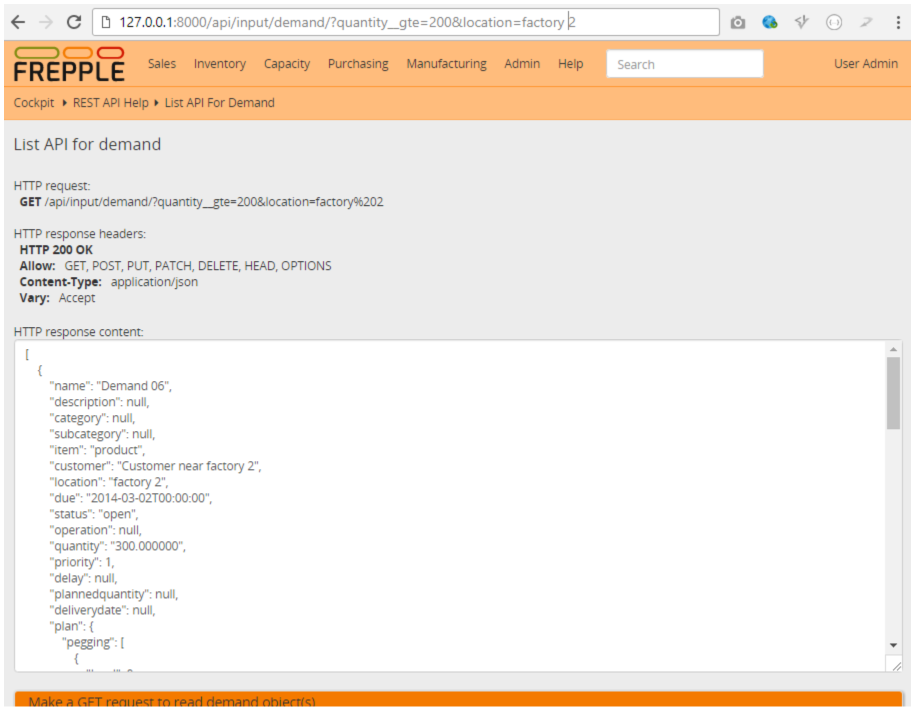
HTTP response headers:
HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

HTTP response content:

```
[
  {
    "name": "Demand 01",
    "description": "MP",
    "category": null,
    "subcategory": null,
    "item": "product",
    "customer": "Customer near factory 1",
    "location": "factory 1",
    "due": "2014-01-01T00:00:00",
    "status": "open",
    "operation": null,
    "quantity": "100.000000",
    "priority": 1,
    "delay": null,
    "plannedquantity": null,
    "deliverydate": null,
    "plan": {
      "pegging": [
        {
```

Make a GET request to read demand object(s)
Make an OPTIONS request to see all fields on the demand object
Make a POST request using a HTML form to create a new demand object
Make a POST request in raw format to create a new demand object
Make a PUT request using a HTML form to update one demand object
Make a PUT or PATCH request in raw format to update an existing or more existing demand objects
Make a DELETE request to delete one or more existing demand objects (requires filtering)

It is also possible to filter GET and DELETE requests by introducing extra parameters in the URL. The first extra parameter in the URL will start with a ? and the following ones start with & Adding ?`quantity__gte=200&location=factory 2` will output all demands with a quantity greater or equal to 200 from location factory 2.


127.0.0.1:8000/api/input/demand/?quantity__gte=200&location=factory 2
FREPPLE
Sales
Inventory
Capacity
Purchasing
Manufacturing
Admin
Help
Search
User Admin
Cockpit ▸ REST API Help ▸ List API For Demand

List API for demand

HTTP request:
GET /api/input/demand/?quantity__gte=200&location=factory%202

HTTP response headers:
HTTP 200 OK
Allow: GET, POST, PUT, PATCH, DELETE, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

HTTP response content:

```
[
  {
    "name": "Demand 06",
    "description": null,
    "category": null,
    "subcategory": null,
    "item": "product",
    "customer": "Customer near factory 2",
    "location": "factory 2",
    "due": "2014-03-02T00:00:00",
    "status": "open",
    "operation": null,
    "quantity": "300.000000",
    "priority": 1,
    "delay": null,
    "plannedquantity": null,
    "deliverydate": null,
    "plan": {
      "pegging": [
        {
```

For number, duration, and date fields the filters are: `__gt`, `__gte`, `__lt`, `__lte`, `__exact`, `__in`.
For strings filters are: `__exact`, `__contains`, `__in`. (including the local primary key field)
For foreign key strings: `__exact`, `__in`.

In the DEMAND table the name field is the local primary key, and location field is a foreign key. In this case you may use `name__contains=06` to get all the demands with 06 on the name, but you cannot do a `location__contains` (as this field is a foreign key) but you may use `location__in=factory 1,factory 2` to list all demands from a list of specific locations.

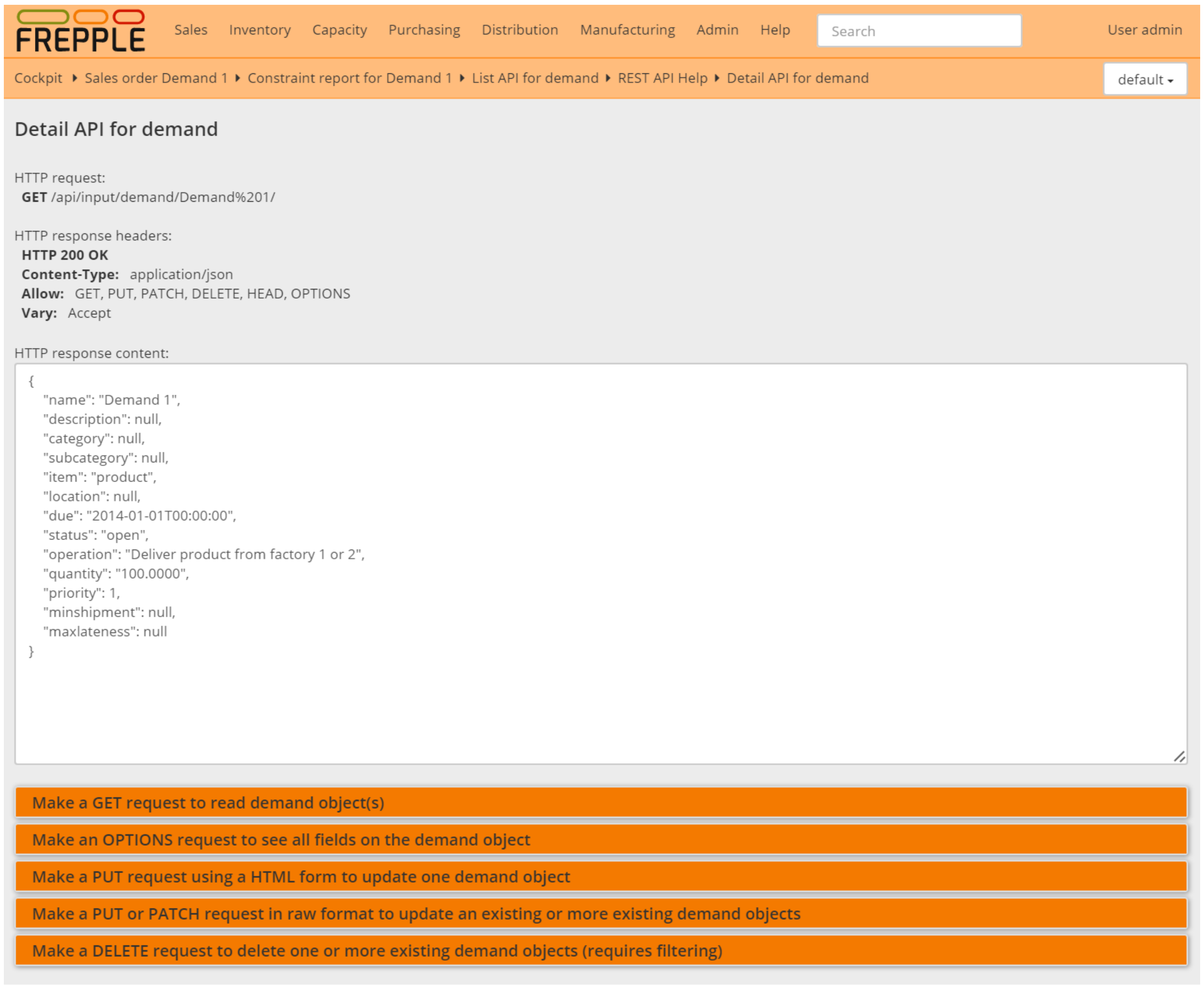
Some models will allow bulk operations. For these models, requests in JSON format have been tested for bulk POST using a LIST as argument, ie. `[``*1 or more objects*]``.` The PUT request requires all fields in the JSON list of objects, and the PATCH request requires the primary key field in each of the objects.

One other option that may be useful for frepple integration and connector troubleshooting to the format of the response. Adding the `format=json` option will dump a JSON to the browser window.

3.4.2 Detail API from your browser

The top part of this screen show the headers and content of the HTTP response of the web service.

The different sections below allow you to create new request for the GET, OPTIONS, PUT, PATCH and DELETE HTTP methods on the selected object.



Detail API for demand

HTTP request:
GET /api/input/demand/Demand%201/

HTTP response headers:
HTTP 200 OK
Content-Type: application/json
Allow: GET, PUT, PATCH, DELETE, HEAD, OPTIONS
Vary: Accept

HTTP response content:

```
{
    "name": "Demand 1",
    "description": null,
    "category": null,
    "subcategory": null,
    "item": "product",
    "location": null,
    "due": "2014-01-01T00:00:00",
    "status": "open",
    "operation": "Deliver product from factory 1 or 2",
    "quantity": "100.0000",
    "priority": 1,
    "minshipment": null,
    "maxlateness": null
}
```

Make a GET request to read demand object(s)
Make an OPTIONS request to see all fields on the demand object
Make a PUT request using a HTML form to update one demand object
Make a PUT or PATCH request in raw format to update an existing or more existing demand objects
Make a DELETE request to delete one or more existing demand objects (requires filtering)

3.4.3 API from the command line

Using tools like “curl”, “wget” or similar you can use the command line to (depending on your permissions) change/read/add/delete data.

To just get a list of all sales orders in JSON format:

```
wget --http-user=admin --http-password=admin http://127.0.0.1:8000/api/input/demand/?
↪format=json

curl -H 'Accept: application/json; indent=4' -u admin:admin http://127.0.0.1:8000/api/
↪input/demand/?format=json
```

To just get a filtered list of sales orders with quantity equal or above 200, and with location *factory 2* (the URL needs escaping, the spaces and & were replaced by `%20` and `\&`) in JSON format:

```
wget --http-user=admin --http-password=admin http://127.0.0.1:8000/api/input/demand/?
↪quantity__gte=200\&location=factory%202\&format=json

curl -H 'Accept: application/json; indent=4' -u admin:admin "http://127.0.0.1:8000/api/
↪input/demand/?quantity__gte=200&location=factory%202&format=json"
```

(continues on next page)

(continued from previous page)

```
```

To just get a list of all sales orders in API format (assuming the user is named "admin" and that the password is also "admin":

```
wget --http-user=admin --http-password=admin http://127.0.0.1:8000/api/input/demand/?
↪format=api

curl -H 'Accept: application/json; indent=4; charset=UTF-8' -u admin:admin http://127.0.
↪0.1:8000/api/input/demand/?format=api
```

To POST a single or multiple records in JSON format it is also straightforward. For a single record POST request:

```
curl -X POST -H "Content-Type: application/json; charset=UTF-8" -d "[{\"keyA0\":\"valA0\
↪", \"keyA1\":\"valA1\"}]" -u admin:admin http://127.0.0.1:8000/api/input/demand/?
↪format=json
```

For a multiple record POST request:

```
curl -X POST -H "Content-Type: application/json; charset=UTF-8" -d "[{\"keyA0\":\"valA0\
↪", \"keyA1\":\"valA1\"},{\"keyB0\":\"valB0\", \"keyB1\":\"valB1\"}]" -u admin:admin␣
↪http://127.0.0.1:8000/api/input/demand/?format=json
```

"key:val" pairs should be separated by a comma, so it is probably easier if you store the data in a file:

```
curl -X POST -H "Content-Type: application/json; charset=UTF-8" --data @json_records_
↪file.txt -u admin:admin http://127.0.0.1:8000/api/input/demand/?format=json
```

To PUT/PATCH a single record in JSON format:

```
curl -X PATCH -H "Content-Type: application/json; charset=UTF-8" -d "{\"key\":\"val\"}" -
↪u admin:admin http://127.0.0.1:8000/api/input/demand/a_demand_id/
```

```
curl -X PUT -H "Content-Type: application/json; charset=UTF-8" --data @json_records_file.
↪txt -u admin:admin http://127.0.0.1:8000/api/input/demand/a_demand_id/
```

PUT requires all fields so "key:val" pairs should be separated by a comma, so it is probably easier if you upload the data from a file like in the POST example.

To PUT/PATCH multiple DEMAND records in JSON format:

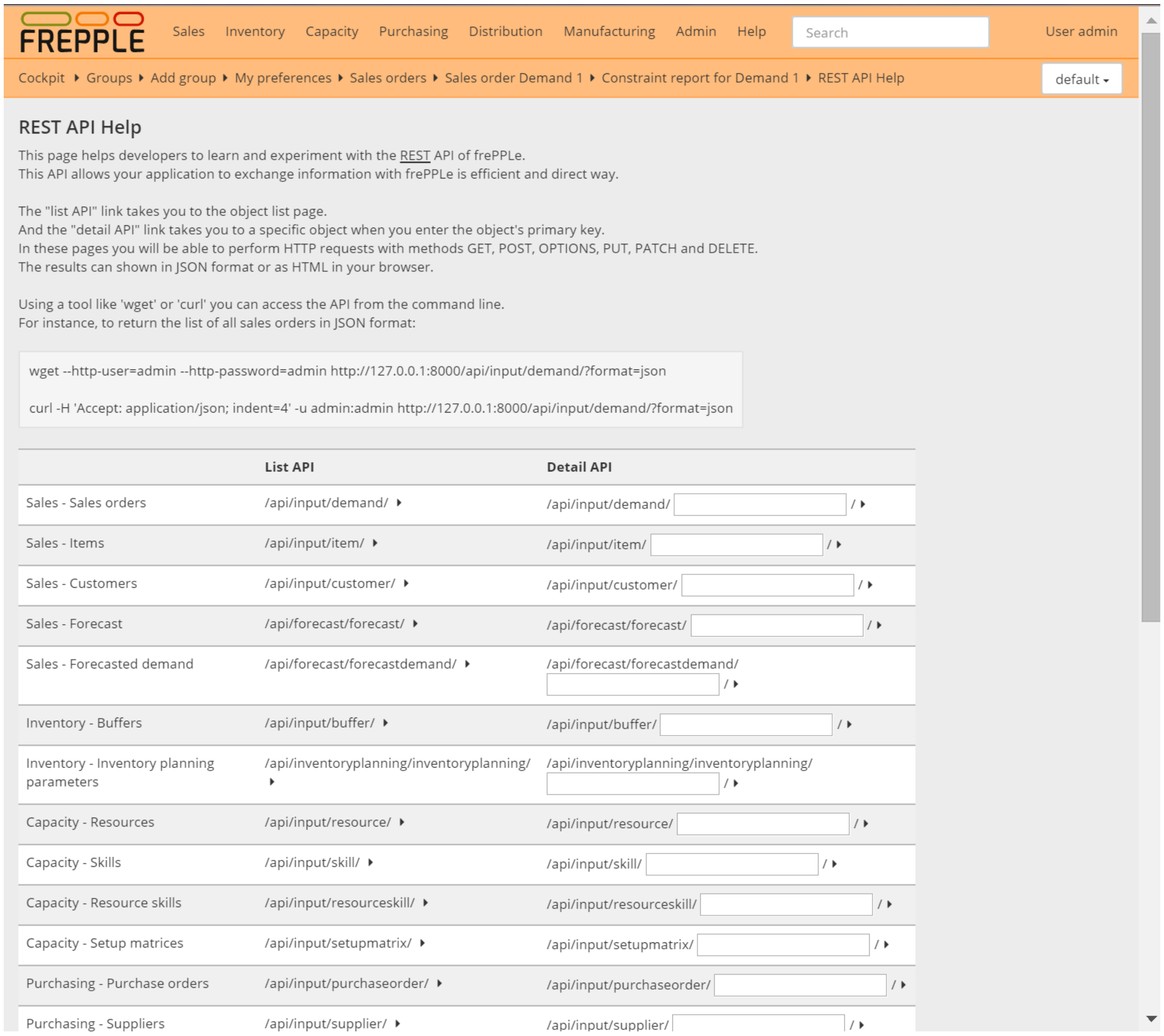
```
curl -X PATCH -H "Content-Type: application/json; charset=UTF-8" -d "[{\"name\":\"a_
↪demand_id1\",\"key\":\"val\"},{\"name\":\"a_demand_id2\",\"key\":\"val\"}]" -u␣
↪admin:admin http://127.0.0.1:8000/api/input/demand/
```

```
curl -X PUT -H "Content-Type: application/json; charset=UTF-8" --data @json_records_file.
↪txt -u admin:admin http://127.0.0.1:8000/api/input/demand/
```

DEMAND primary key field is `name`, so for a PATCH request this field must be present in each object. PUT requires all fields in a so "key:val" pairs should be separated by a comma, so it is probably easier if you upload the data from a file like in the POST example.

To DELETE records a safeguard is in place that prevents deleting all records in a table. So the DELETE request requires that the number of records to be deleted is lower than the number of all records in the table. A DELETE request for one or more records can be done with:

```
curl -X DELETE -H "Content-Type: application/json; charset=UTF-8" -u admin:admin http://
↪127.0.0.1:8000/api/input/demand/?source=ERP
```


FREPPLE
Sales
Inventory
Capacity
Purchasing
Distribution
Manufacturing
Admin
Help
Search
User admin
Cockpit ▸ Groups ▸ Add group ▸ My preferences ▸ Sales orders ▸ Sales order Demand 1 ▸ Constraint report for Demand 1 ▸ REST API Help
default

REST API Help

This page helps developers to learn and experiment with the REST API of frePPLe.
This API allows your application to exchange information with frePPLe is efficient and direct way.

The "list API" link takes you to the object list page.
And the "detail API" link takes you to a specific object when you enter the object's primary key.
In these pages you will be able to perform HTTP requests with methods GET, POST, OPTIONS, PUT, PATCH and DELETE.
The results can shown in JSON format or as HTML in your browser.

Using a tool like 'wget' or 'curl' you can access the API from the command line.
For instance, to return the list of all sales orders in JSON format:

```
wget --http-user=admin --http-password=admin http://127.0.0.1:8000/api/input/demand/?format=json

curl -H 'Accept: application/json; indent=4' -u admin:admin http://127.0.0.1:8000/api/input/demand/?format=json
```

	List API	Detail API
Sales - Sales orders	/api/input/demand/ ▸	/api/input/demand/ / ▸
Sales - Items	/api/input/item/ ▸	/api/input/item/ / ▸
Sales - Customers	/api/input/customer/ ▸	/api/input/customer/ / ▸
Sales - Forecast	/api/forecast/forecast/ ▸	/api/forecast/forecast/ / ▸
Sales - Forecasted demand	/api/forecast/forecastdemand/ ▸	/api/forecast/forecastdemand/ / ▸
Inventory - Buffers	/api/input/buffer/ ▸	/api/input/buffer/ / ▸
Inventory - Inventory planning parameters	/api/inventoryplanning/inventoryplanning/ ▸	/api/inventoryplanning/inventoryplanning/ / ▸
Capacity - Resources	/api/input/resource/ ▸	/api/input/resource/ / ▸
Capacity - Skills	/api/input/skill/ ▸	/api/input/skill/ / ▸
Capacity - Resource skills	/api/input/resourceskill/ ▸	/api/input/resourceskill/ / ▸
Capacity - Setup matrices	/api/input/setupmatrix/ ▸	/api/input/setupmatrix/ / ▸
Purchasing - Purchase orders	/api/input/purchaseorder/ ▸	/api/input/purchaseorder/ / ▸
Purchasing - Suppliers	/api/input/supplier/ ▸	/api/input/supplier/ / ▸

3.5 Command line

Tasks from the execution screen can also be launched from the command line with the frepplectl utility.

Usage:

```
frepplectl subcommand [options] [args]
```

Type 'frepplectl.py help <subcommand>' for help on a specific subcommand.

The options will vary from command to command. There are a number of common options:

- **–database=DATABASE**:

Specifies which scenario database to run the command for. When left unspecified the command will run on the production database.

The database names are defined in the djangosettings.py. Note that they can be different from the name of the database name configured in postgresql.

- **-v VERBOSITY, –verbosity=VERBOSITY**:

  Verbosity level: 0=minimal output, 1=normal output, 2=all output.

- **-h, –help**:

  Show a help message either showing all commands or help on a specific command.

A number of these commands are inherited from the excellent Django web application framework used by frePPLe. More details on the commands can be found on https://docs.djangoproject.com/en/2.2/ref/django-admin/

3.6 Remote commands

Tasks from the execution screen can also be launched and monitored remotely through a web service API.

- *Reference*
- *Authentication*
- *Example*

3.6.1 Reference

The API endpoints can be accessed with any modern web tool or programming language using the following URLs. The examples are using the excellent curl command line tool.

- Run a task on the default database:

```
curl -u <user>:<password> http(s)://<server>:<port>/execute/api/<command>/
  --data "<argument1>=<value1>&<argument2>=<value2>"
```

- Run a task on a scenario database:

```
curl -u <user>:<password> http(s)://<server>:<port>/<scenario>/execute/api/<command>
↪/
  --data "<argument1>=<value1>&<argument2>=<value2>"
```

- Get the status of all running and pending tasks:

```
curl -u <user>:<password> http(s)://<server>:<port>/execute/api/status/
```

- Get the status of a single task:

```
curl -u <user>:<password> http(s)://<server>:<port>/execute/api/status/?id=X
```

- Cancel a waiting or running task:

```
curl -u <user>:<password> http(s)://<server>:<port>/execute/api/cancel/?id=X
```

All these APIs return a JSON object and they are asynchronous, i.e. they don't wait for the actual command to finish. In case you need to wait for a task to finish, you will need to use a loop which periodically polls the /execute/api/status URL to monitor the status.

3.6.2 Authentication

FrePPLe supports 2 methods for authentication of your user in this API:

- **Basic authentication**
  See https://en.wikipedia.org/wiki/Basic_access_authentication for more details.
  With curl you use the argument `-u USER:PASSWORD` on the command line.
- **JSON Web Token**
  See https://jwt.io/ for more details.
  With curl you use the argument `--header 'Authorization:  Bearer TOKEN'` on the command line.

We strongly recommend the use of a HTTPS configuration of the frePPLe server when using this API. Without it your data and login credentials are sent unencrypted over the internet.

3.6.3 Example

To illustrate the above concepts, this section shows a common workflow to upload new data in the frePPLe database and generate a new plan.

- Delete previous data files.
- Upload data files (in csv or excel format).
- Import the data files into frePPLe.
- Regenerate the plan with the new data.

This example uses linux bash and curl, but it can easily be coded in any other modern programming language.

```
#!/bin/bash

 server="localhost:8000"

 #declare -a filelist=("buffer.csv" "item.csv")
 id=0
 output=""
 result=""

 #check the status of a task
 function checkstatus () {
   id=$1
   if (($id>0));
   then
     output=$(curl -u admin:admin http://$server/execute/api/status/?id=$id);
   else
     output=$(curl -u admin:admin http://$server/execute/api/status/);
   fi
   if [[ $output =~ .*Failed || $output =~ .*Done ]];
   then
     output="break";
   else
     output="wait";
   fi
   echo $output
 }
```

(continues on next page)

(continued from previous page)

```
# you may delete all files or just the ones in the arguments
# you will have to comment the delete all files locationstable
# and uncomment the lines above
function deletefiles () {

  #if you want to delete just the files that you will replace
  # for FILE1 in "${filelist[@]}"; do
  #   FILE2=$(basename "$FILE1")
  #   #spaces should be escaped in the URL
  #   FILE2=${FILE2// /\%20}
  #   result=$(curl -X DELETE -u admin:admin http://$server/execute/
↪deletefromfolder/0/"$FILE2"/);
  # done

  #to delete all files in the folder
  result=$(curl -X DELETE -u admin:admin http://$server/execute/
↪deletefromfolder/0/AllFiles/);
}

function waitTillComplete () {
  id=$1
  until [[ $WAIT -eq 0 ]]; do
    if [[ "$(checkstatus $id)" =~ "break" ]]; then
      #show the result
      echo $(curl -u admin:admin http://$server/execute/api/status/?id=$id);
      break
    fi

    sleep "$WAIT_TIME"
    ((WAIT--))
  done
}

# create the file list
# if the argument is a directory it will add all the files there
# If the arguments are files only these will be added
for FILE0 in "$@"; do
  if [[ -d "${FILE0}" ]]; then
    cd "${FILE0}"
    filelist=(*.csv *.csv.gz *.xlsx)
  else
    filelist=( $filelist "$FILE0" )
  fi
done

#delete files before
echo -e "\n---------------start delete files----------------"
deletefiles
echo "----------------end delete files------------------"

#upload the files in the list
```

(continues on next page)

(continued from previous page)

```
echo -e "\n---------------start upload files----------------"
for FILE1 in "${filelist[@]}"; do
  #get filename without path
  FILE2=$(basename "$FILE1")
  if [[ ! "$FILE2" =~ \*.* ]]; then
    curl -X POST -F "$FILE2=@$FILE1" -u admin:admin http://$server/execute/
↪uploadtofolder/0/
  fi
done
echo -e "\n---------------end upload files------------------"

#import the data in the files
echo -e "\n---------------start import the data----------------"
WAIT_TIME=10 #seconds
WAIT=6 #times
result=$(curl -X POST -u admin:admin http://$server/execute/api/
↪importfromfolder/)
id=$(echo "${result//[!0-9]/}")
waitTillComplete $id
echo "---------------end import the data------------------"

#run the plan
echo -e "\n---------------start planning----------------"
WAIT_TIME=10 #seconds
WAIT=6 #times
result=$(curl -u admin:admin --data "constraint=15&plantype=1&env=fcst,
↪invplan,balancing,supply" http://$server/execute/api/runplan/)
id=$(echo "${result//[!0-9]/}")
waitTillComplete $id
echo "---------------end planning------------------"
```

CHAPTER

FOUR

DEVELOPER GUIDE

This chapter discusses some topics of interest to developers working on extending, customizing or maintaining frePPLe.

4.1 Creating an custom theme

The user interface is styled using bootstrap v3. All variables documented at https://getbootstrap.com/docs/3.4/customize/ are available to you to design your look and feel.

Proceed with the following steps to compile a custom theme:

1. **Install node.js**:

   Download and install the node.js javascript runtime environment from https://nodejs.org/en/.

   The installation also makes the npm command available which we'll use in the following steps.

2. **Install grunt command line tools**:

   Grunt is a javascript task running utility.

   Install it with the command:

   ```
   npm install grunt-cli -g
   ```

3. **Install the javascript dependencies**:

   Compiling the frePPLe and bootstrap styles requires a number of javascript libraries. Install these with the following command:

   ```
   npm install
   ```

4. **Design the LESS files**:

   The styles are defined in the following files. Check out http://lesscss.org/ to learn more about the Less syntax used in these files.

   - *freppledb/common/static/css/frepple.less*:

     Defines the frePPLe specific CSS styles.

   - *freppledb/common/static/css/THEME/variables.less*:

     Defines the configuration of bootstrap for each of the themes. The value of the variables is what a theme unique.

   - *freppledb/common/static/css/THEME/frepple.less*:

     Optionally, you can create files with theme-specific styles that can't be expressed as variable values.

5. **Compile the LESS files**:

   The less files need to be compiled into a CSS stylesheet for each theme. Edit the gruntfile.js file to include your theme in the list of themes, and then run the following command:

```
grunt less
```

In each of the theme folders the file bootstrap.min.css and bootstrap.min.css.map will be generated.

6. **Update djangosettings.py file**:
   New themes are only shown in the user interface when the theme is configured in the setting *THEMES*.
   You can also edit the setting *DEFAULT_THEME* to make your theme the default one.

4.2 Adding or customizing a report

This section describes the different steps to add a new report (or update an existing one) in the user interface. We'll describe both the general case as well as the generic view provided by frePPLe.

The steps outline here are a short and very brief summary of how screens are developed in the Django web application framework. Check out https://www.djangoproject.com for more information and an excellent tutorial.

For clarity and ease of maintenance it is recommended to always add your reports in a custom extension app.

4.2.1 General case

As an example we'll create a report to display some statistics on the size of your model. It will simply display the total number of buffers and operations in your model.

1. **Create a view to generate the data**.
   A view function retrieves the report data from the database (or computes it from another source) and passes a data dictionary with the data to the report template.
   A view is a Python function. Here's the view required for our example, which you can put in a file statistics.py:

```
from freppledb.input.models import *
from django.shortcuts import render_to_response
from django.template import RequestContext
from django.contrib.admin.views.decorators import staff_member_required

@staff_member_required
def MyStatisticsReport(request):
  countOperations = Operation.objects.using(request.database).count()
  countBuffers = Buffer.objects.using(request.database).count()
  return render_to_response('statistics.html',
    RequestContext(request, {
      'numOperations': countOperations,
      'numBuffers': countBuffers,
      'title': 'Model statistics',
    }))
```

   The function decorator staff_member_required is used to assure users are authenticated properly.
   Notice how the first 2 statements in the function use the Django relational mapping to pick the data from the database. This code is translated by the framework in SQL queries on the database.
   The last line in the function passes the data in a dictionary to the Django template engine. The template engine will generate the HTML code returned to the user's browser.

2. **Create a template to visualize the data.**

   The template file statistics.html will define all aspects of the visualizing the results.

   The file templates/statistics.html for our example looks like this:

   ```
   {% extends "admin/base_site_nav.html" %}
   {% load i18n %}
   {% block content %}
   <div id="content-main">
   {% trans 'Number of operations:' %} {{numOperations}}<br>
   {% trans 'Number of buffers:' %} {{numBuffers}}<br>
   </div>
   {% endblock %}
   ```

   Templates are inheriting from each other. In this example we inherit from the base template which already contains the navigation toolbar and the breadcrumbs trail. We only override the block which contains the div-element with the main content.

   Templates use special tags to pick up data elements or to call special functions. The {{ }} tag is used to refer to the data elements provided by the view function. The {% trans %} tag is used to mark text that should be translated into the different languages for the user interface.

3. **Map the view as a URL.**

   To expose the view as a URL to the users you'll need to map it to a URL pattern.

   Edit the definition of the urlpatterns variable in the file urls.py to set up a URL for this example:

   ```
   urlpatterns = patterns('',
    ...
    (r'^statistics.html$', 'statistics.MyStatisticsReport'),
    )
   ```

4. **Update the menu structure.**

   You'll want to add the report also to a menu.

   The following lines in this file menu.py will do this:

   ```
   from django.utils.translation import ugettext as _
   from freppledb.menu import menu
   menu.addItem("admin", "statistics", url="/statistics.html",
     label=_('Model statistics'), index=900)
   ```

4.2.2 Using the frePPLe generic report

FrePPLe uses a standard view for displaying data in a list or grid layout, respectively called ListReport and TableReport. With these views you can add new reports with with less code and more functionality (such as sorting, filtering, pagination, export and import).

The steps for adding the view are slightly different from the generic case.

1. **Define a report class**

   Instead of defining a view function we define a class with the report metadata. See the definition of the report base classes in the file common/report.py to see all available options for the metadata classes.

   For a list report this class has the following structure:

```
from freppledb.common.report import *

 class myReportClass(ListReport):
   template = 'myreporttemplate.html'
   title = 'title of my report'
   basequeryset = ... # A query returning the data to display
   frozenColumns = 1
   rows = (
     ('field1', {
       'filter': FilterNumber(operator='exact', ),
       'title': _('field1'),
       }),
     ('field2', {
       'filter': FilterText(size=15),
       'title': _('field2')}),
     ('field3', {
       'title': _('field3'),
       'filter': FilterDate(),
       }),
     )
```

For a table report this class has the following structure:

```
from freppledb.common.report import *

class myReportClass(TableReport):
  template = 'myreporttemplate.html'
  title = 'title of my report'
  basequeryset = ... # A query returning the data to display
  model = Operation
  rows = (
    ('field1',{
      'filter': FilterNumber(operator='exact', ),
      'title': _('field1'),
      }),
    )
  crosses = (
    ('field2', {'title': 'field2',}),
    ('field3', {'title': 'field3',}),
    )
  columns = (
    ('bucket',{'title': _('bucket')}),
    )

  @staticmethod
  def resultlist1(request, basequery, bucket, startdate, enddate, sortsql='1 asc'):
    ... # A query returning the data to display as fixed columns on the left hand␣
↪side.

  @staticmethod
```

(continues on next page)

(continued from previous page)

```
  def resultlist2(request, basequery, bucket, startdate, enddate, sortsql='1 asc'):
    ... # A query returning the data to display for all cells in the grid.
```

2. **Create a template to visualize the data**.

   For a list report the template has the following structure:

```
{% extends "admin/base_site_list.html" %}
{% load i18n %}

{% block frozendata %}
{% for i in objectlist1 %}
<tr>
<td>{{i.field1}}</td>
</tr>{% endfor %}
{% endblock %}

{% block data %}
{% for i in objectlist1 %}
<tr>
<td>{{i.field2}}</td>
<td>{{i.field3}}</td>
</tr>{% endfor %}
{% endblock %}
```

   For a grid report the template is identical, except that you need to inherit from the admin/base_site_table.html template.

3. **Map the view as a URL**.
   The syntax for adding a report now refers to the generic view, and we pass the report class as an argument

```
urlpatterns = patterns('',
 ...
 (r'^myreport/([^/]+)/$', 'freppledb.common.report.view_report',
   {'report': myReportClass,}),
 ...
 )
```

4. **Update the menu structure**.
   This step is identical to the general case.

4.3 Translating the user interface

This section provides step by step instructions on how to translate the user interface to your favourite language.

Hint: We are very keen on receiving translations for additional languages. And it's an easy way for you to contribute back to the frePPLe community.

4.3.1 For translators

1. Install a translation editor

For the translation process you should install an editor for gettext catalogs (.po files).

Highly recommended is the free Poedit tool.

2. Start translating

Pick up file the translation file of the language you wish to update from the github source code repository. All terms to be translated are collected in this single file.

- French: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/fr/fr.po
- German: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/de/de.po
- Hebrew: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/he/he.po
- Italian: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/it/it.po
- Japanese: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/ja/ja.po
- Dutch: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/nl/nl.po
- Portuguese: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/pt/pt.po
- Brazilian Portuguese: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/pt-br/pt-br.po
- Russian: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/ru/ru.po
- Spanish: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/es/es.po
- Simplified Chinese: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/zh-hans/zh-hans.po
- Traditional Chinese: https://raw.githubusercontent.com/frePPLe/frepple/master/freppledb/locale/zh-hant/zh-hant.po

Open the file with the editor you installed in step 1, and start translating. The terms will already "pre-translated" with an automated translation engine. Your job as a translator is to review the pre-translated strings (marked as "needs work"), change them where needed and remove the "needs work" flag.

Some strings may include HTML tags or Python code, e.g.:

%(title)s for %(entity)s

In this case just copy the entire string and translate "for", resulting in:

%(title)s para %(entity)s

3. Submit your translation

You can submit your translation a) either with a pull request on github, or b) by posting the file to the frePPLe user group.

4.3.2 For developers

1. Extract all translation strings

The translatable strings are present at many places in the source code. A first step consist of collecting of these translatable strings in a single file which translators can update.

The following command runs this string collecting.

```
make extract-translations
```

2. Add support for an additional language

Start by copying the translations files of an existing language. You need to copy a subdirectory from *freppledb/locale* and 2 files from *freppledb/common/static/common/po*.

The possible language codes can be found on the World Wide Web Consortium.

You can contact the frepple team and ask them to help you with a pre-translated file. It will makes the translation job much easier and faster.

If you want to create installation packages including the new language then the installer also needs updating. The files *contrib/installer/parameters.ini* and *contrib/installer/frepple.nsi* need straightforward editing.

To activate it you must also add the new language to *djangosettings.py* (or *bin\djangosettings.py* in the binary Windows installation). Add the new language code and description to the variable LANGUAGES:

```
LANGUAGES = (
  ("en", _("English")),
  ("fr", _("French")),
  ("de", _("German")),
  ("he", _("Hebrew")),
  ("it", _("Italian")),
  ("ja", _("Japanese")),
  ("nl", _("Dutch")),
  ("pt", _("Portuguese")),
  ("pt-br", _("Brazilian Portuguese")),
  ("ru", _("Russian")),
  ("es", _("Spanish")),
  ("zh-hans", _("Simplified Chinese")),
  ("zh-hant", _("Traditional Chinese")),
)
```

3. Let the translators do their work

Commit the changes from the previous step, and let the translators bring the translation files *freppledb/locale/<LANGUAGE>/<LANGUAGE>.po* up to date.

4. Compile the translations

Run the following command to compile the output of the translators in the right format in various data files.

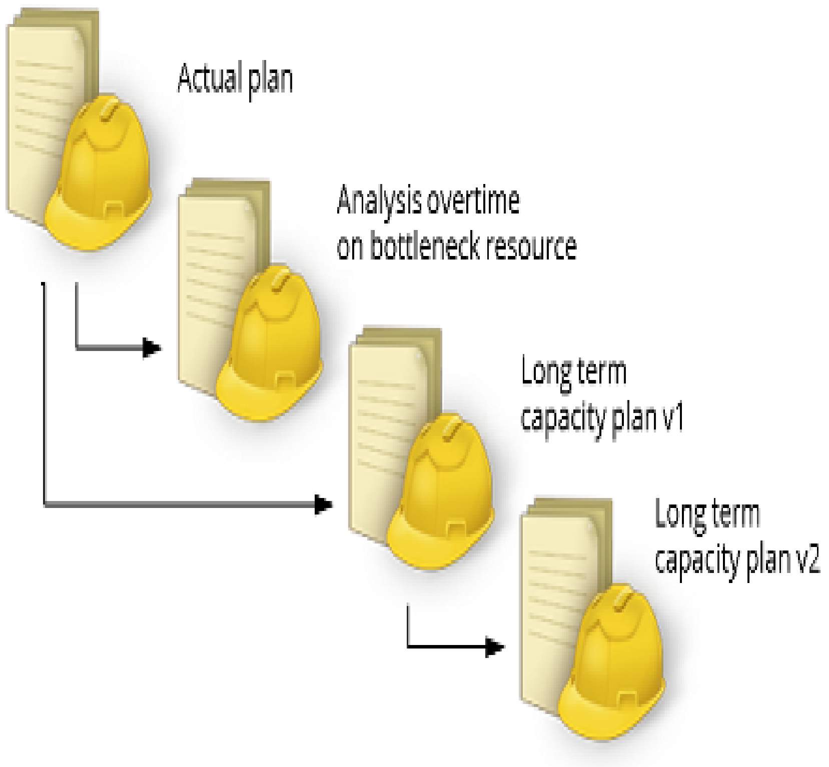
```
make compile-translations
```

4.4 Configuring multiple models in the user interface

FrePPLe supports working with multiple models in the same web application.

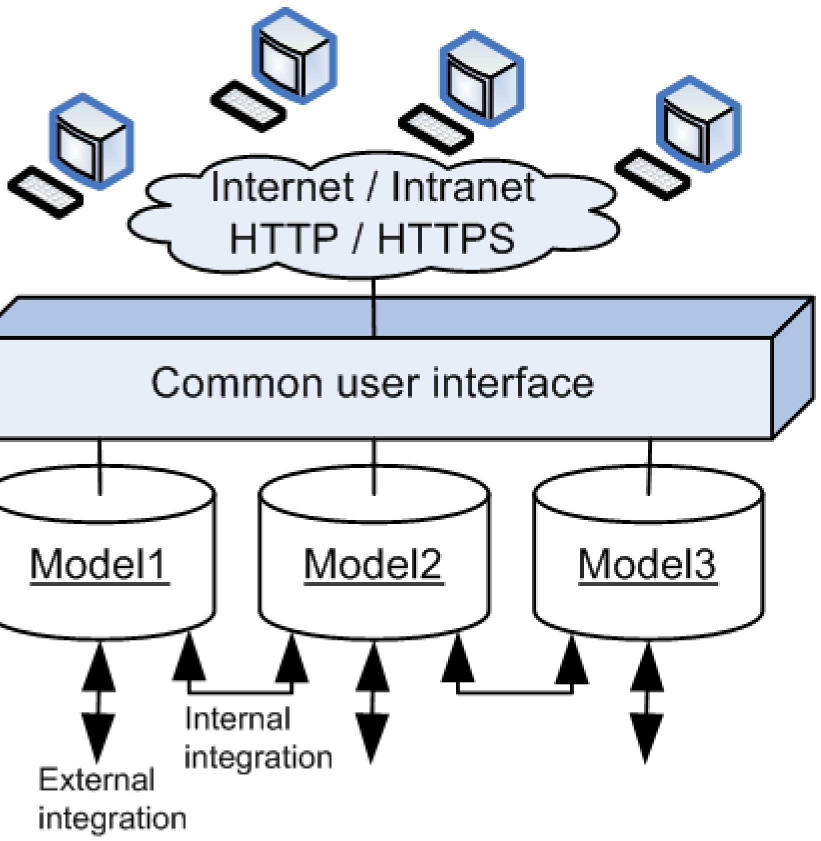
This setup can be useful for the following very typical use cases:

- **What-if models to support scenario analysis.**
  A planner can do all kinds of what-if analysis in a copy of the production model.


Actual plan
Analysis overtime on bottleneck resource
Long term capacity plan v1
Long term capacity plan v2

- **Separate models for separate product lines or different factories.**

  When the interaction between product divisions or plants is relatively low, it might be useful to create seperate models for the planners to work in. This allows their business processes, planner workflows and data to be more loosely coupled.



The following steps are required to configure a multi-model setup.

- **Create additional databases**

  The database administrator needs to create a PostgreSQL database for each model.

  See http://www.postgresqltutorial.com/postgresql-create-database/ for detailed steps.

- **Update the djangosettings.py configuration file**

  The connection details of each schema need to be added as a seperate section for the DATABASES parameter in the file settings.py.

  For instance:

```
DATABASES = {
'default': {
  'ENGINE': 'django.db.backends.postgresql',
  'NAME': 'frepple',
  'USER': '',
  'PASSWORD': '',
  'HOST': '',
  'OPTIONS': {},
  'CONN_MAX_AGE': 60,
  'PORT': '',
  },
'scenario1': {
  'ENGINE': 'django.db.backends.postgresql',
  'NAME': 'scenario1',
  'USER': '',
```

(continues on next page)

(continued from previous page)

```
    'PASSWORD': '',
    'HOST': '',
    'OPTIONS': {},
    'CONN_MAX_AGE': 60,
    'PORT': '',
    },
  }
```

Some guidelines need to be considered when setting up the schemas:

- The number of schemas in the web application is unlimited.

  Extra schema's have NO impact on the performance of the user interface. Only the disk space used by the database will increase.

- One of the schemas MUST be called 'default'.

  All information on user logins, user preferences and browser sessions are stored in this default schema.

- Use short and unambiguous names for the additional schemas.

  Since the names will be used as a prefix in the URLs they should be short and can't contain any special characters.

  Good examples: 'scenario1', 'plant1'…

  Bad examples: 'scenario/1', names with non-ASCII characters, names with spaces…

- The databases can be located on different database servers or database instances, but this is not required. This could be useful for instance to avoid that users running large tasks on what-if scenarios impact the performance of the regular production model.

• **Initialize the new schema(s)**

  If not done yet, the default schema is initialized with the following command. It creates all tables, indices and other database objects.

```
frepplectl migrate
```

  To load the demo data in this database you run:

```
frepplectl loaddata demo
```

  To initialize the additional schemas you copy the default schema with the command below. The command can also be executed from the user interface in the execution screen: see *Execution screen*

```
frepplectl scenario_copy default my_schema
```

  The copy process might take a while for bigger datasets. If it takes too long, you should consider running the copy as an automated batch job during quiet hours.

• **Restart the web server**

  After a change in the djangosettings.py file, the web server needs to be restarted.

• **Review user access and permissions**

  Access rights are controlled for each scenario separately.

  After running the command scenario_copy only 1) the user executing the command and 2) superusers in the source scenario are marked active in the new scenario. Other users can be granted access by marking them active in the new scenario, and by assigning them appropriate privileges in it.

4.5 Upgrade an existing installation

FrePPLe allows migrating an existing installation to a new release without any loss of data.

This page documents the steps for this process.

- *Generic instructions*
- *Debian upgrade script*

4.5.1 Generic instructions

1. **Backup your old environment**

   You're not going to start without a proper backup of the previous installation, are you? We strongly recommend you start off with a backup of a) all PostgreSQL databases and b) the configuration file djangosettings.py.

2. **Upgrade the PostgreSQL database**

   FrePPLe requires postgresql 9.5 or higher. If you're on an older version, upgrading your PostgreSQL database is the first step.

3. **Install the new python package dependencies**

   Different frePPLe releases may require different versions of third party Python libraries.

   Minor releases (ie the third number in the release number changes) never require new dependencies, and you can skip this step.

   The following command will bring these to the right level as required for the new release. Make sure to run it as root user or use sudo (otherwise the packages will be installed locally for that user instead of system-wide), and to replace 5.0.0 with the appropriate release number.

   ```
   sudo -H pip3 install --force-reinstall -r https://raw.githubusercontent.com/frePPLe/
   ↪frepple-data-admin/1.0.0/requirements.txt
   ```

4. **Install the new frePPLe release.**

   ```
   sudo -H pip3 install data-admin
   ```

5. **Update the configuration file djangosettings.py**

   The installation created a new version of the configuration file. Now, you'll need to merge your edits from the old file into the new file.

   In our experience an incorrectly updated configuration file is the most common mistake when upgrading. So, take this edit seriously and don't just use the old file without a very careful comparison.

6. **Migrate the frePPLe databases**

   The following command needs to be repeated for each scenario database (as found in the keys in the DATABASES setting in /etc/frepple/djangosettings.py).

   ```
   frepplectl migrate --database=default
   ```

   On a fresh installation, this command intializes all database objects. When running it on an existing installation it will incrementally update the database schema without any loss of data.

7. **Restart your apache server**

   After a restart of the web server, the new environment should be up and running.

Tip: It is not possible to have multiple versions simultaneously on the same server.

4.5.2 Debian upgrade script

The commands below are a convenience summary of the above steps implemented for a Debian/Ubuntu Linux server.

```
sudo apt-get -y -q update
sudo apt-get -y -q upgrade

# Upgrade of the PostgreSQL database isn't covered in these commands.

sudo -H pip3 install --force-reinstall -r https://raw.githubusercontent.com/frePPLe/
↪frepple-data-admin/1.0.0/requirements.txt

# Download the debian package of the new release here.

sudo -H pip3 install data-admin

# Manually edit the /etc/frepple/djangosettings.py file. The previous line
# keeps the old configuration file, which may not be ok for the new release.

frepplectl migrate --database=default

# Repeat the above line for all scenarios that are in use

sudo service apache2 reload
```

CHAPTER

FIVE

RELEASE NOTES

5.1 2.0.0 (Upcoming release)

- Corrected packaging. The previous release was only working when checking out from git. This new release can also be installed and run from PyPi.

5.2 1.0.0 (2021/04/18)

- Initial release, copied out of frePPLe 6.13.0.