
GardenHub Documentation

Release 1.1

HarvestHub

Oct 12, 2018

Contents:

1	Developer Guide	3
1.1	Models	4
1.2	Order states	8
1.3	Views	8
1.4	Deployment (WIP)	12
2	User Guide	13
2.1	Client view vs admin panel	13
2.2	Inviting new users	16
2.3	Managing plots/gardens	19
3	Indices and tables	21
	Python Module Index	23

GardenHub is an open source web application that facilitates food distribution for community gardens. It's written in [Python](#) using the [Django web framework](#).

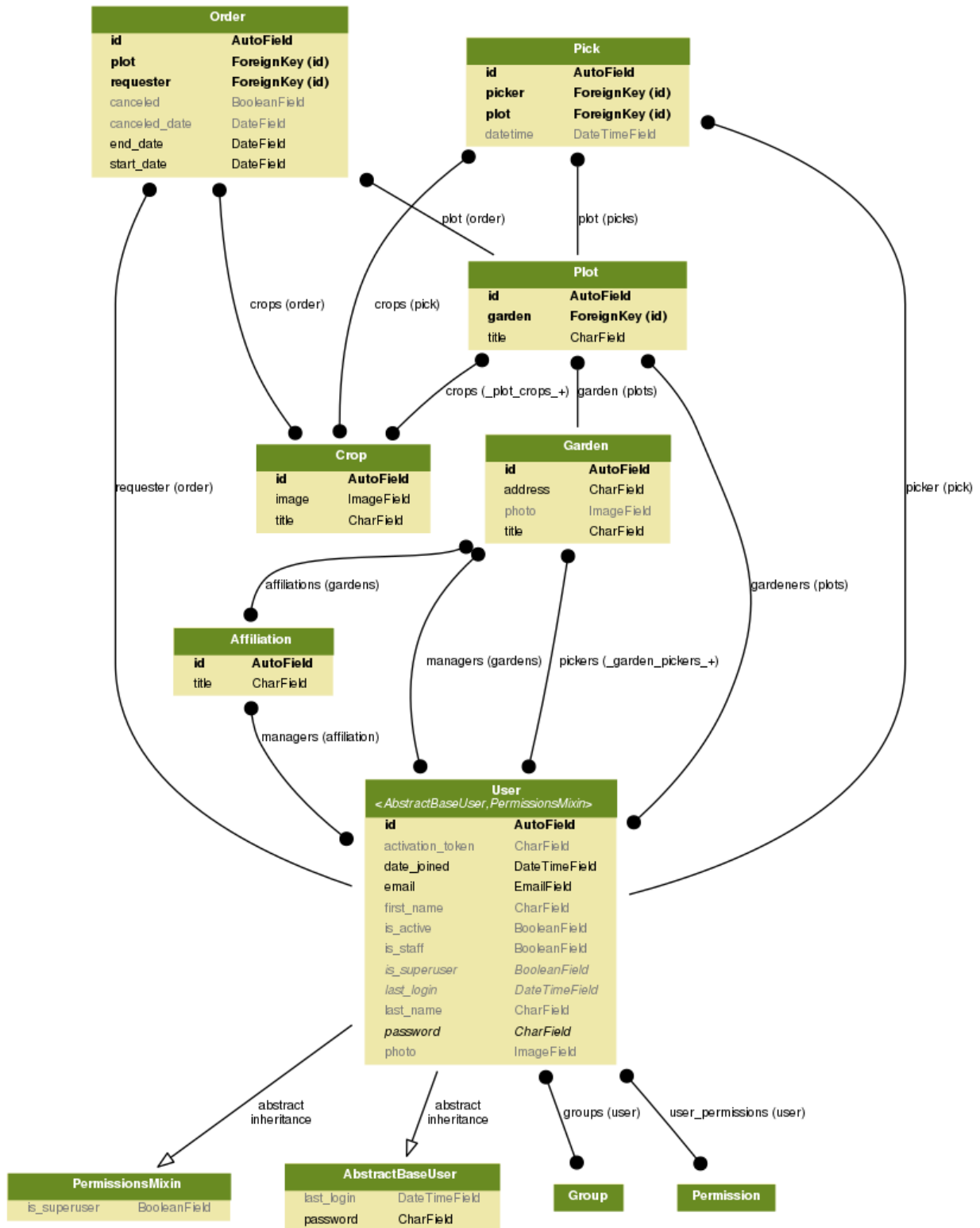
[GardenHub's source code](#) may be found on [GitHub](#).

CHAPTER 1

Developer Guide

This section will explain how to hack on GardenHub.

1.1 Models



1.1.1 Querying objects

class gardenhub.managers.**OrderQuerySet** (*model=None, query=None, using=None, hints=None*)

Custom QuerySet for advanced filtering of orders.

The following fields enable filtering orders via `Order.objects`. For example,

```
from gardenhub.models import Order

# All open orders
orders = Order.objects.open()

# All open orders that haven't been picked today
orders = Order.objects.open().unpicked_today()
```

active()

Orders that are happening right now.

closed()

Orders that have finished or were canceled.

inactive()

All orders that aren't happening right now.

open()

Orders that have not finished but also may not have begun.

picked_today()

Orders that have at least one Pick from today.

unpicked_today()

Orders that have no Picks from today.

upcoming()

Orders that have not yet begun but are scheduled to happen.

class gardenhub.managers.**UserManager**

Custom User manager because the custom User model only does auth by email.

get_or_invite_users (*emails, request*)

Gets or creates users from the list of emails. When a user is created they are sent an invitation email.

1.1.2 Full model reference

class gardenhub.models.**Affiliation** (**args, **kwargs*)

A group of affiliated gardens.

exception **DoesNotExist**

exception **MultipleObjectsReturned**

class gardenhub.models.**Crop** (**args, **kwargs*)

A crop represents an item that may be picked, such as a zucchini or an orange. Crops are stored in a master list, managed by the site admin, and may be listed on Orders or Picks.

exception **DoesNotExist**

exception **MultipleObjectsReturned**

class gardenhub.models.**Garden** (**args, **kwargs*)

A whole landscape, divided into many plots. Managed by Garden Managers.

exception DoesNotExist

exception MultipleObjectsReturned

class gardenhub.models.**Order** (*args, **kwargs)

A request from a Gardener or Garden Manager to enlist a particular Plot for picking over a specified number of days.

exception DoesNotExist

exception MultipleObjectsReturned

get_picks ()

Return the list of Picks that occurred on this Order's plot within this Order's timeframe.

get_status_icon ()

Returns a Semantic UI status icon class string depending on this order's status. This may be removed in the future.

is_active ()

Whether this Order is active.

is_closed ()

Whether this Order is closed.

is_open ()

Whether this Order is open.

progress ()

Percentage this order is complete, as a decimal between 0-100.

was_picked_today ()

True if at least one Pick was submitted today for the Order's Plot.

class gardenhub.models.**Pick** (*args, **kwargs)

A submission by a picker signifying that certain Crops have been picked from a particular Plot at a particular time.

exception DoesNotExist

exception MultipleObjectsReturned

inquirers ()

People to notify about this pick.

class gardenhub.models.**Plot** (*args, **kwargs)

Subdivision of a Garden, allocated to a Gardener for growing food.

exception DoesNotExist

exception MultipleObjectsReturned

class gardenhub.models.**User** (*args, **kwargs)

Custom user class for GardenHub users. This is necessary because we want to authorize users by their email address (and provide a few extra fields).

exception DoesNotExist

exception MultipleObjectsReturned

can_edit_garden (garden)

Can the given user manage this garden? True if the user is listed in Garden.managers for that garden. False otherwise.

can_edit_order (*order*)

Can the given user manage this order? True if the user can edit Order.plot for that order. False otherwise.

can_edit_plot (*plot*)

Can the given user manage this plot? True if the user is listed in Plot.gardeners for that plot, OR the user is listed in Garden.managers for the garden in Plot.garden. False otherwise.

clean ()

Hook for doing any extra model-wide validation after clean() has been called on every field by self.clean_fields. Any ValidationError raised by this method will not be associated with a particular field; it will have a special-case association with the field defined by NON_FIELD_ERRORS.

email_user (*subject, message, from_email=None, **kwargs*)

Send an email to this user.

get_full_name ()

Return the first_name plus the last_name, with a space in between.

get_gardens ()

Return all the Gardens the given user can edit.

get_orders ()

Return all Orders for the given user's Plots and Gardens.

get_peers ()

Return all the Users within every Plot and Garden that you manage, except yourself.

get_picker_gardens ()

Return all Gardens where the user is in Garden.picker.

get_picker_orders ()

Return all Orders this user is assigned to fulfill.

get_plots ()

Return all the Plots the given user can edit. Users can edit any plot which they are a gardener on, and any plot in a garden they manage.

get_short_name ()

Return the short name for the user.

has_orders ()

Determine whether the user has any orders at all.

is_anything ()

GardenHub is only useful if the logged-in user can manage any garden or plot. If not, that is very sad. :(

is_garden_manager ()

A garden manager is someone who facilitates renting Plots of a Garden out to Gardeners. Any person who is set as Garden.manager on at least one Garden.

is_gardener ()

A gardener is someone who rents a garden Plot and grows food there. Gardeners are assigned to Plot.gardener on at least one Plot. GM's of a garden with plots are also considered gardeners.

is_order_picker (*order*)

Is the user assigned as a picker on this order? True if the user is listed in Order.plot.garden.pickers. False otherwise.

is_picker ()

A picker is someone who is assigned to fulfill Orders on a Garden. They will submit Picks over the duration of the Orders.

1.2 Order states

Orders are often queried by one of more possible states they can be in. Below is a chart of all the possible states.

State:	open	closed	upcoming	active	inactive
start date	Any	Past	Future	Up through today	Future
end date	Future	Past	Future	Today forward	Past
canceled	No	Coerced	No	No	Coerced
comparison	and	and	and	and	or

Legend:

- **canceled = No** means that if the order is canceled it *cannot* be this state.
- **canceled = Coerced** means that the order will automatically be in this state by virtue of the fact it is canceled.
- **comparison** describes how the start date and end date of the order are compared. For instance, an open order can have any start date **and** an end date in the future. An inactive order can have a start date in the future **or** an end date in the past.

1.2.1 QuerySet filtering

Corresponding to the table above, orders have custom QuerySet functions for each of the states. For instance, `Order.objects.open()` or `Order.objects.active()`.

1.3 Views

```
class gardenhub.views.AccountRemoveView(**kwargs)
```

```
    Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.views.generic.  
    edit.DeletionMixin, django.views.generic.base.TemplateView
```

Remove the logged-in user's GardenHub account.

```
    delete (request, *args, **kwargs)
```

Call the `delete()` method on the fetched object and then redirect to the success URL.

```
class gardenhub.views.AccountSettingsView(**kwargs)
```

```
    Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.views.generic.  
    edit.FormView
```

Account settings screen for the logged-in user.

```
    form_class
```

alias of `gardenhub.forms.AccountSettingsForm`

```
    form_valid (form)
```

If the form is valid, redirect to the supplied URL.

```
class gardenhub.views.AccountView(**kwargs)
```

```
    Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.views.generic.  
    base.TemplateView
```

Profile edit screen for the logged-in user.

```
class gardenhub.views.ApiCrops (**kwargs)
    Bases:      django.contrib.auth.mixins.LoginRequiredMixin,  gardenhub.mixins.
                UserCanEditPlotMixin, django.views.generic.detail.DetailView

    Return JSON about crops.

    model
        alias of gardenhub.models.Plot

class gardenhub.views.GardenDetailView (**kwargs)
    Bases:      django.contrib.auth.mixins.LoginRequiredMixin,  gardenhub.mixins.
                UserCanEditGardenMixin, django.views.generic.detail.DetailView

    View a single garden.

    model
        alias of gardenhub.models.Garden

class gardenhub.views.GardenListView (**kwargs)
    Bases:      django.contrib.auth.mixins.LoginRequiredMixin,  django.views.generic.
                list.ListView

    A list of all gardens the logged-in user can edit.

    get_queryset ()
        Return the list of items for this view.

        The return value must be an iterable and may be an instance of QuerySet in which case QuerySet specific
        behavior will be enabled.

class gardenhub.views.GardenUpdateView (**kwargs)
    Bases:      django.contrib.auth.mixins.LoginRequiredMixin,  gardenhub.mixins.
                UserCanEditGardenMixin, django.views.generic.edit.UpdateView

    Edit form for an individual garden.

    form_class
        alias of gardenhub.forms.GardenForm

    form_valid (form)
        If the form is valid, save the associated model.

    model
        alias of gardenhub.models.Garden

class gardenhub.views.HomePageView (**kwargs)
    Bases:      django.contrib.auth.mixins.LoginRequiredMixin,  django.views.generic.
                base.TemplateView

    Welcome screen with calls to action.

class gardenhub.views.LogoutView (**kwargs)
    Bases:      django.contrib.auth.views.LogoutView

    Logs out the user and redirects them to the login screen.

class gardenhub.views.OrderCancelView (**kwargs)
    Bases:      django.contrib.auth.mixins.LoginRequiredMixin,  django.contrib.auth.
                mixins.UserPassesTestMixin, django.views.generic.edit.DeleteView

    delete (request, *args, **kwargs)
        Call the delete() method on the fetched object and then redirect to the success URL.
```

model

alias of *gardenhub.models.Order*

class gardenhub.views.**OrderCreateView** (**kwargs)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.contrib.auth.mixins.UserPassesTestMixin, django.views.generic.edit.CreateView

This is a form used to submit a new order. It's used by gardeners, garden managers, or anyone who has the ability to edit a plot.

form_class

alias of gardenhub.forms.OrderForm

form_valid (form)

If the form is valid, save the associated model.

get_form (*args, **kwargs)

Return an instance of the form to be used in this view.

get_form_kwargs ()

Return the keyword arguments for instantiating the form.

model

alias of *gardenhub.models.Order*

class gardenhub.views.**OrderDetailView** (**kwargs)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.contrib.auth.mixins.UserPassesTestMixin, django.views.generic.detail.DetailView

Review an individual order that's been submitted. Anyone who can edit the plot may view or cancel these orders.

model

alias of *gardenhub.models.Order*

class gardenhub.views.**OrderListView** (**kwargs)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.views.generic.list.ListView

Manage orders page to view all upcoming orders.

get_queryset ()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

class gardenhub.views.**PasswordResetConfirmView** (**kwargs)

Bases: django.contrib.auth.views.PasswordResetConfirmView

form_valid (form)

If the form is valid, redirect to the supplied URL.

class gardenhub.views.**PasswordResetView** (**kwargs)

Bases: django.contrib.auth.views.PasswordResetView

form_valid (form)

If the form is valid, redirect to the supplied URL.

class gardenhub.views.**PickCreateView** (**kwargs)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.contrib.auth.mixins.UserPassesTestMixin, django.views.generic.edit.CreateView

Form enabling a picker to submit a Pick for a given plot.

form_valid (*form*)

If the form is valid, save the associated model.

get_context_data (***kwargs*)

Insert the form into the context dict.

get_form_kwargs ()

Return the keyword arguments for instantiating the form.

model

alias of *gardenhub.models.Pick*

class gardenhub.views.**PlotCreateView** (***kwargs*)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.contrib.auth.mixins.UserPassesTestMixin, django.views.generic.edit.CreateView

form_class

alias of gardenhub.forms.PlotForm

form_valid (*form*)

If the form is valid, save the associated model.

model

alias of *gardenhub.models.Plot*

class gardenhub.views.**PlotListView** (***kwargs*)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, django.views.generic.list.ListView

A list of all plots the logged-in user can edit.

get_queryset ()

Return the list of items for this view.

The return value must be an iterable and may be an instance of *QuerySet* in which case *QuerySet* specific behavior will be enabled.

class gardenhub.views.**PlotUpdateView** (***kwargs*)

Bases: django.contrib.auth.mixins.LoginRequiredMixin, gardenhub.mixins.UserCanEditPlotMixin, django.views.generic.edit.UpdateView

Edit form for an individual plot.

form_class

alias of gardenhub.forms.PlotForm

form_valid (*form*)

If the form is valid, save the associated model.

get_form (**args, **kwargs*)

Return an instance of the form to be used in this view.

model

alias of *gardenhub.models.Plot*

gardenhub.views.**account_activate_view** (*request, token*)

When a new user is invited, an email call to action will send them to this view so they can fill out their profile and activate their account.

1.4 Deployment (WIP)

This guide will walk you through deploying an instance of GardenHub online. It will make a lot of choices for you, with the trade-off of not requiring advanced knowledge in order to do this. Familiarity with the Linux terminal is required.

1.4.1 1. Getting the server

If you don't have one, create an account with [DigitalOcean](#). Then **create a new droplet**. It will cost you at least \$10/mo, possibly more depending on the size of your userbase.

- You will want to select the latest LTS, 64-bit Ubuntu as your OS. At the time of writing, that is 16.04 x64. The LTS version is the one that ends in .04, not .10.
- Choose **at least 2GB of memory** if you intend to run this application seriously. If not, you may be able to get away with less. With many users, you will want to increase the memory so the app doesn't become slow. You can always do this later.
- Choose a datacenter region that is close to the majority of your users, if possible.
- Enable IPv6.
- Set the hostname to the domain name you intend to use, such as `gardenhub.io`.
- You can leave the other options alone. Click "Create".

Next, wait a minute for the droplet to provision. Once it's done, you'll get an IP address. You can use that to shell into the server via your terminal. Replace the IP address with your droplet's:

```
ssh root@765.432.1
```

The password will be provided by DigitalOcean.

1.4.2 2. Provisioning the server

First, let's install [Dokku](#). It will let us push the GardenHub repo up to the server via git. While ssh'd into the server, run this:

```
wget https://raw.githubusercontent.com/dokku/dokku/v0.11.3/bootstrap.sh
sudo DOKKU_TAG=v0.11.3 bash bootstrap.sh
```

Once it completes, visit your server's IP address in your web browser and follow the instructions. Be sure to tick the virtual hosts option and enter the domain name you intend to use with the app.

TODO: Write the rest of this

This is a walkthrough for garden managers and administrators of the app. This guide will show you how to perform common administrative tasks.

2.1 Client view vs admin panel

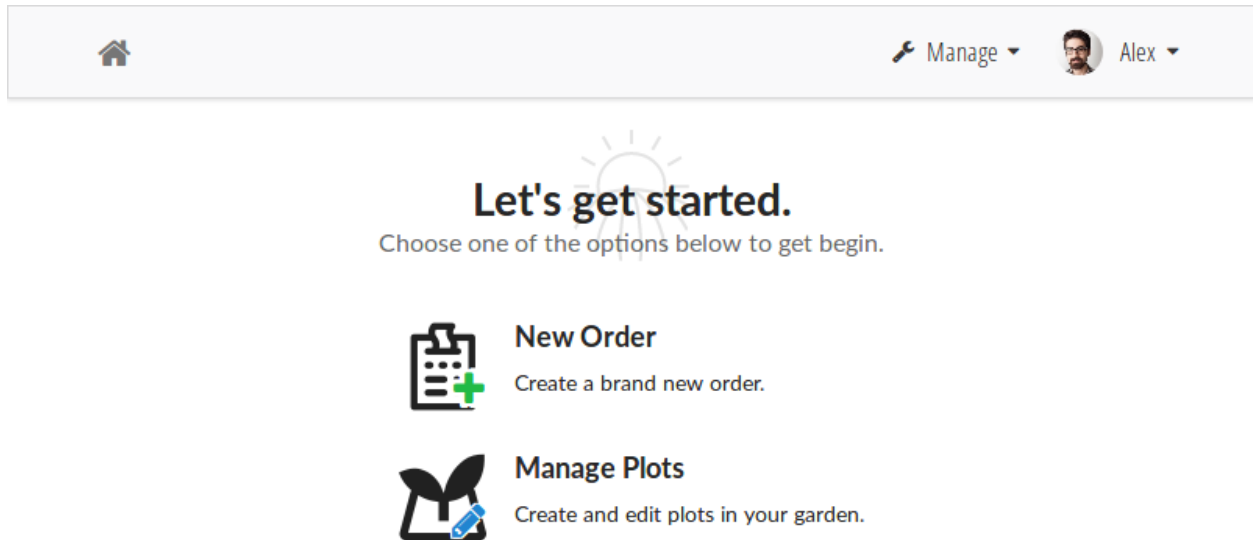
There are two main parts of the site that are used to manage data: the **client view** and the **admin panel**.

There are some things that can only be accomplished within the client view, and others that can only be accomplished in the admin panel.

2.1.1 Client view

The client view is what end-users will see. This design is geared towards gardeners and pickers. It is the default view that will show when visiting the GardenHub app. In the client view, **people can only see plots and gardens they are assigned to**, even if they are an administrator. This is by design. Administrators are treated just like any other user in this view with no special capabilities.

The client view is the only place that users can invite other users to GardenHub via email.



2.1.2 Admin panel

The admin panel is a tool for directly editing data in GardenHub's database. You can use this to manually create users, manage gardens and plots, and edit the master list of crops. Administrators can access the entire site's database through this view, regardless of whether they're assigned to those gardens or plots.


Django administration

WELCOME, ALEX. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Site administration


AUTHENTICATION AND AUTHORIZATION

Groups


+ Add  Change

GARDENHUB


Affiliations

+ Add  Change


Crops

+ Add  Change


Gardens

+ Add  Change


Orders

+ Add  Change


Picks

+ Add  Change

Plots

+ Add  Change

Users

+ Add  Change

Recent actions

My actions

 User

 User

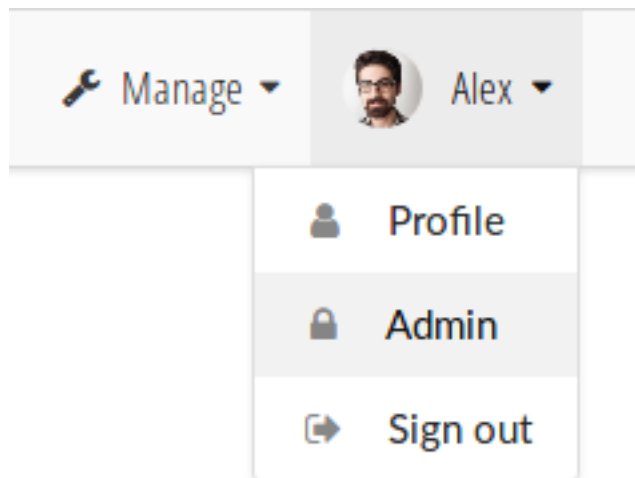
 User

 Alex's Garden [1]
Plot

 Alex's Garden
Garden

 acorn squash
Crop

The admin panel can be accessed by first logging into GardenHub, then going under your user dropdown in the top navigation and clicking “Admin”. Note that only superusers and staff members can access the admin panel.



2.2 Inviting new users

For now, GardenHub is an invite-only platform. Anyone can invite other people to co-manage resources they can edit (such as plots or gardens). To invite a new user, you must first create the plot or garden you would like that user to manage, then add that user's email address into the “managers” (or “gardeners”) field.

Email invitations only work in the [client view](#).

2.2.1 Invite a new user to a plot

First, create a new plot, or edit an existing plot. Type the email address of the user you'd like to invite into the “gardeners” field. You will need to click away after so the email becomes enclosed in a gray box as shown below. Then you may click “Save plot” and an invitation will be sent to the email address(es) you entered.

Administrators must first [add themselves to the plot](#) before they can edit it in the client view.

Manage Alex

Garden*
The garden this plot is in.

Alex's Garden

Name
The plot's name is probably a number, like 11. The plot should be clearly labeled with a sign.

1

Gardeners
List the people who should manage this plot, or enter an email address to invite someone new.

Alex Gleason × you@email.com ×

Crops
List the crops growing on this plot.

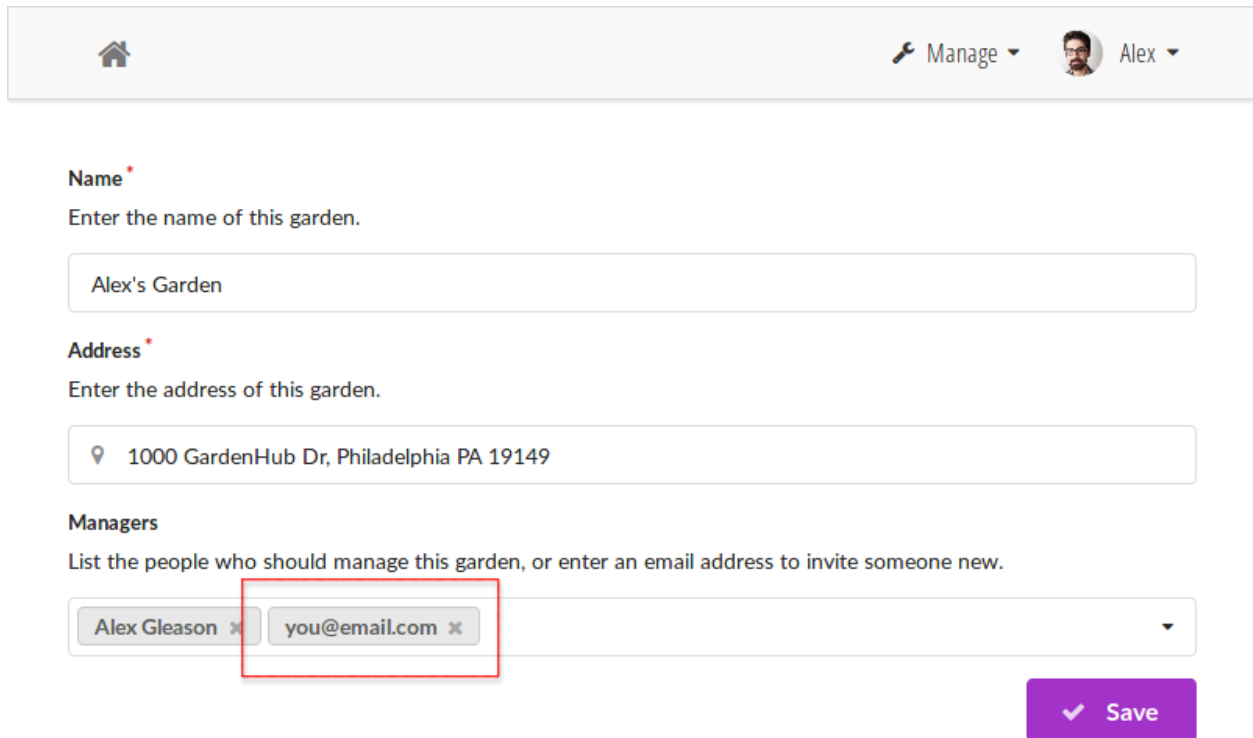
alfalfa spouts × artichoke × bananas ×

✓ Save Plot

2.2.2 Invite a new user to a garden

First, create a new garden, or edit an existing garden. Type the email address of the user you'd like to invite into the “managers” field. You will need to click away after so the email becomes enclosed in a gray box as shown below. Then you may click “Save” and an invitation will be sent to the email address(es) you entered.

Administrators must first [add themselves to the garden](#) before they can edit it in the client view.



The screenshot shows the 'Manage' page for a garden. At the top, there is a navigation bar with a home icon, a 'Manage' button with a wrench icon, and a user profile for 'Alex'. Below the navigation bar, the form is divided into three sections: 'Name', 'Address', and 'Managers'. The 'Name' section has a label 'Name' with a red asterisk and a prompt 'Enter the name of this garden.' followed by a text input field containing 'Alex's Garden'. The 'Address' section has a label 'Address' with a red asterisk and a prompt 'Enter the address of this garden.' followed by a text input field containing '1000 GardenHub Dr, Philadelphia PA 19149'. The 'Managers' section has a label 'Managers' and a prompt 'List the people who should manage this garden, or enter an email address to invite someone new.' Below the prompt is a list of managers: 'Alex Gleason' and 'you@email.com'. The 'you@email.com' entry is highlighted with a red box. To the right of the list is a dropdown arrow. At the bottom right of the form is a purple 'Save' button with a checkmark icon.

Name *

Enter the name of this garden.

Alex's Garden

Address *

Enter the address of this garden.

1000 GardenHub Dr, Philadelphia PA 19149

Managers

List the people who should manage this garden, or enter an email address to invite someone new.

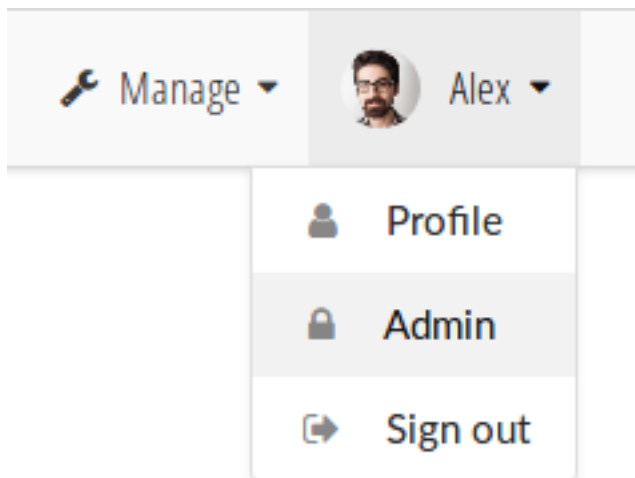
Alex Gleason × you@email.com ×

✓ Save

2.2.3 Create a new user manually

If you'd like to create a new user without inviting them to a plot or garden, you can add them manually. They won't receive an email invitation, so you will need to send them the password you choose.

First, visit the admin panel. You'll need to be a superuser or staff user to do this:



Next, click the “Add” button next to “Users”:

Django administration

WELCOME, ALEX. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Site administration

AUTHENTICATION AND AUTHORIZATION

Groups

[+ Add](#) [Change](#)

GARDENHUB

Affiliations

[+ Add](#) [Change](#)

Crops

[+ Add](#) [Change](#)

Gardens

[+ Add](#) [Change](#)

Orders

[+ Add](#) [Change](#)

Picks

[+ Add](#) [Change](#)

Plots

[+ Add](#) [Change](#)

Users

[+ Add](#) [Change](#)

Recent actions

My actions

✗

User

✗

User

✗

User

+ Alex's Garden [1]

Plot

+ Alex's Garden

Garden

[acorn squash](#)

Crop

Fill out the form. **You must tick the “Active” box to allow the user to log in.** You may also tick “Staff status” and “Superuser status” to grant this user admin privileges.

Django administration

WELCOME, ALEX. [VIEW SITE](#) / [CHANGE PASSWORD](#) / [LOG OUT](#)

Home › Gardenhub › Users › Add user

Add user

First, enter a username and password. Then, you'll be able to edit more user options.

Email address:	<input type="text" value="you@email.com"/>
The user's email address, which is used for notifications and doubles as the username for logging in.	
Password:	<input type="password" value="....."/>
Password confirmation:	<input type="password" value="....."/>
First name:	<input type="text" value="Jane"/>
The user's given name or nickname, sometimes used in a casual context.	
Last name:	<input type="text" value="Doe"/>
The user's family name, sometimes used to distinguish one user from another.	
<input checked="" type="checkbox"/> Active Designates whether this user should be treated as active. Unselect this instead of deleting accounts.	
<input type="checkbox"/> Staff status Designates whether the user can log into this admin site.	
<input type="checkbox"/> Superuser status Designates that this user has all permissions without explicitly assigning them.	

Save and add another

Save and continue editing

SAVE

Finally, click “Save”. The user you created will now be able to log in. Be sure to assign them to any plots or gardens you’d like them to manage.

2.3 Managing plots/gardens

2.3.1 Adding yourself to a plot

If you want the ability to invite people via email to a plot, you will need to first assign yourself to that plot so you can access the plot within the client view.

1. Visit the admin panel.
2. Click “Plots”.
3. Click the name of the plot you’d like to assign yourself to.
4. Under the “Gardeners” field, find your name, and while holding the “Control” key (or “Command” key on MacOS), click it.
5. Click “Save”.

2.3.2 Adding yourself to a garden

If you want the ability to invite people via email to a garden, you will need to first assign yourself to that garden so you can access the garden within the client view.

1. Visit the admin panel.
2. Click “Gardens”.
3. Click the name of the garden you’d like to assign yourself to.
4. Under the “Managers” field, find your name, and while holding the “Control” key (or “Command” key on MacOS), click it.
5. Click “Save”.

CHAPTER 3

Indices and tables

- `genindex`
- `modindex`
- `search`

g

`gardenhub.managers`, 5
`gardenhub.models`, 5
`gardenhub.views`, 8

A

account_activate_view() (in module gardenhub.views), 11
 AccountRemoveView (class in gardenhub.views), 8
 AccountSettingsView (class in gardenhub.views), 8
 AccountView (class in gardenhub.views), 8
 active() (gardenhub.managers.OrderQuerySet method), 5
 Affiliation (class in gardenhub.models), 5
 Affiliation.DoesNotExist, 5
 Affiliation.MultipleObjectsReturned, 5
 ApiCrops (class in gardenhub.views), 8

C

can_edit_garden() (gardenhub.models.User method), 6
 can_edit_order() (gardenhub.models.User method), 6
 can_edit_plot() (gardenhub.models.User method), 7
 clean() (gardenhub.models.User method), 7
 closed() (gardenhub.managers.OrderQuerySet method), 5
 Crop (class in gardenhub.models), 5
 Crop.DoesNotExist, 5
 Crop.MultipleObjectsReturned, 5

D

delete() (gardenhub.views.AccountRemoveView method), 8
 delete() (gardenhub.views.OrderCancelView method), 9

E

email_user() (gardenhub.models.User method), 7

F

form_class (gardenhub.views.AccountSettingsView attribute), 8
 form_class (gardenhub.views.GardenUpdateView attribute), 9
 form_class (gardenhub.views.OrderCreateView attribute), 10
 form_class (gardenhub.views.PlotCreateView attribute), 11

form_class (gardenhub.views.PlotUpdateView attribute), 11
 form_valid() (gardenhub.views.AccountSettingsView method), 8
 form_valid() (gardenhub.views.GardenUpdateView method), 9
 form_valid() (gardenhub.views.OrderCreateView method), 10
 form_valid() (gardenhub.views.PasswordResetConfirmView method), 10
 form_valid() (gardenhub.views.PasswordResetView method), 10
 form_valid() (gardenhub.views.PickCreateView method), 10
 form_valid() (gardenhub.views.PlotCreateView method), 11
 form_valid() (gardenhub.views.PlotUpdateView method), 11

G

Garden (class in gardenhub.models), 5
 Garden.DoesNotExist, 5
 Garden.MultipleObjectsReturned, 6
 GardenDetailView (class in gardenhub.views), 9
 gardenhub.managers (module), 5
 gardenhub.models (module), 5
 gardenhub.views (module), 8
 GardenListView (class in gardenhub.views), 9
 GardenUpdateView (class in gardenhub.views), 9
 get_context_data() (gardenhub.views.PickCreateView method), 11
 get_form() (gardenhub.views.OrderCreateView method), 10
 get_form() (gardenhub.views.PlotUpdateView method), 11
 get_form_kwargs() (gardenhub.views.OrderCreateView method), 10
 get_form_kwargs() (gardenhub.views.PickCreateView method), 11
 get_full_name() (gardenhub.models.User method), 7

`get_gardens()` (gardenhub.models.User method), 7
`get_or_invite_users()` (gardenhub.managers.UserManager method), 5
`get_orders()` (gardenhub.models.User method), 7
`get_peers()` (gardenhub.models.User method), 7
`get_picker_gardens()` (gardenhub.models.User method), 7
`get_picker_orders()` (gardenhub.models.User method), 7
`get_picks()` (gardenhub.models.Order method), 6
`get_plots()` (gardenhub.models.User method), 7
`get_queryset()` (gardenhub.views.GardenListView method), 9
`get_queryset()` (gardenhub.views.OrderListView method), 10
`get_queryset()` (gardenhub.views.PlotListView method), 11
`get_short_name()` (gardenhub.models.User method), 7
`get_status_icon()` (gardenhub.models.Order method), 6

H

`has_orders()` (gardenhub.models.User method), 7
`HomePageView` (class in gardenhub.views), 9

I

`inactive()` (gardenhub.managers.OrderQuerySet method), 5
`inquirers()` (gardenhub.models.Pick method), 6
`is_active()` (gardenhub.models.Order method), 6
`is_anything()` (gardenhub.models.User method), 7
`is_closed()` (gardenhub.models.Order method), 6
`is_garden_manager()` (gardenhub.models.User method), 7
`is_gardener()` (gardenhub.models.User method), 7
`is_open()` (gardenhub.models.Order method), 6
`is_order_picker()` (gardenhub.models.User method), 7
`is_picker()` (gardenhub.models.User method), 7

L

`LogoutView` (class in gardenhub.views), 9

M

`model` (gardenhub.views.ApiCrops attribute), 9
`model` (gardenhub.views.GardenDetailView attribute), 9
`model` (gardenhub.views.GardenUpdateView attribute), 9
`model` (gardenhub.views.OrderCancelView attribute), 9
`model` (gardenhub.views.OrderCreateView attribute), 10
`model` (gardenhub.views.OrderDetailView attribute), 10
`model` (gardenhub.views.PickCreateView attribute), 11
`model` (gardenhub.views.PlotCreateView attribute), 11
`model` (gardenhub.views.PlotUpdateView attribute), 11

O

`open()` (gardenhub.managers.OrderQuerySet method), 5
`Order` (class in gardenhub.models), 6
`Order.DoesNotExist`, 6

`Order.MultipleObjectsReturned`, 6
`OrderCancelView` (class in gardenhub.views), 9
`OrderCreateView` (class in gardenhub.views), 10
`OrderDetailView` (class in gardenhub.views), 10
`OrderListView` (class in gardenhub.views), 10
`OrderQuerySet` (class in gardenhub.managers), 5

P

`PasswordResetConfirmView` (class in gardenhub.views), 10
`PasswordResetView` (class in gardenhub.views), 10
`Pick` (class in gardenhub.models), 6
`Pick.DoesNotExist`, 6
`Pick.MultipleObjectsReturned`, 6
`PickCreateView` (class in gardenhub.views), 10
`picked_today()` (gardenhub.managers.OrderQuerySet method), 5
`Plot` (class in gardenhub.models), 6
`Plot.DoesNotExist`, 6
`Plot.MultipleObjectsReturned`, 6
`PlotCreateView` (class in gardenhub.views), 11
`PlotListView` (class in gardenhub.views), 11
`PlotUpdateView` (class in gardenhub.views), 11
`progress()` (gardenhub.models.Order method), 6

U

`unpicked_today()` (gardenhub.managers.OrderQuerySet method), 5
`upcoming()` (gardenhub.managers.OrderQuerySet method), 5
`User` (class in gardenhub.models), 6
`User.DoesNotExist`, 6
`User.MultipleObjectsReturned`, 6
`userManager` (class in gardenhub.managers), 5

W

`was_picked_today()` (gardenhub.models.Order method), 6