# NPO Reach README

## Account Model

Customers, i.e. Non-Profits, are represented in the system as Accounts. The system is multi-tenant, so all other top level objects (e.g. Users, Classes, etc.) are always scoped by Account. In addition, the application offers a recurring billing model that allows for multiple billing plan levels that are charged at varying rates. Customers can create Accounts with a one-month free trial without collecting payment information. When the trial is about to expire, the system will send an email alert. Such alerts are also sent when the billing level gets changed, when a payment is collected, or when an Account is canceled. The account model makes a call to has\_subscription, which loads the Saas custom module (found in the lib folder), and sets up thing like possible user limits for accounts. This functionality, along with the payment gateway functionality is customized from the SaaS Rails Kit.

## Devise and Inherited Resources

The project uses the Devise gem for basic security functionality, and the Inherited Resources gem, well, just because it’s cool and saves lots of repetitive coding. Check out the README’s for these if you’re not familiar with them. The authentication\_keys token on the user model ensures that users are always scoped by email address AND account ID. Scope is always restricted to the current account, unless you are the superuser (i.e. the administrator for ALL Accounts).

## Roles and Privileges

The project implements an authorization scheme that will look familiar, as it is based on many common practices. The complexities and custom functionality tend to lie in the relationship between privileges and menu items. In a nutshell, the model structure looks like this:

* Users have many Roles through Assignments
* A Role has many Grants (of Privileges)
* A Privilege is Granted to one or more Roles
* Users have Privileges through Roles

Privileges are named rights to do something to a resource. They consist of a name, a description, a category, the controller, and the action(s) that are allowed.

On the User model, the “can” method is the way to check whether a user has the right to do something, and this method is called by the check\_authorization before\_filter on the Application Controller. The “can” method is defined as follows:

|  |
| --- |
| def can? (action, controller)  self.owner? || is\_global\_privilege(controller, action) ||  roles.includes(:privileges).for(action,controller).any?  end |

The current controller and action for a page are passed into this function, and

Is\_global\_privilege references an array of “GLOBAL\_PRIVILEGES” (defined on the Privilege model) that are always available to everyone. These include signing in, signing out, accessing the dashboard, accessing help, and editing your personal information.

“for” is a named scope on the Role model, which is implemented as follows.

|  |
| --- |
| scope :for, lambda{|controller, action|  where("privileges.controller = ? AND ? & privileges.actions >  0", controller,  Privilege::CONTROLLER\_ACTIONS[controller][action]) |

Valid actions for Privileges are stored as a single integer in the database, which is the sum of integer representations of ALL of the actions that the privilege refers to. Those integer representations are articulated in the CONTROLLER\_ACTIONS hash on the Privilege model. CONTROLLER\_ACTIONS is just a mapping between the string names of the actions and the integers that represent them in the database. The “for” named scope takes the string controller and action name as an argument, then looks up the corresponding integer in CONTROLLER\_ACTIONS, and passes that to a where clause that does the bitwise comparison to determine if the user’s Roles contain the privileges necessary to perform the passed in controller and action.

The superuser defines Default Roles (drawing upon a global set of privileges) that are inherited when a new Account is created. This is to allow us to introduce new out-of-the-box Roles in the future without having to necessarily change code.

Following is a list of default roles and their associated privileges that are seeded in the system by seeds.rb and copied to the account upon its creation.

|  |  |  |  |
| --- | --- | --- | --- |
| **Role** | **Category** | **Privilege (Controller)** | **Actions** |
| System Administrator | Administration | Manage People (families) | index = 1, new = 2, create = 4, edit = 8, update = 16, destroy = 32 (Total: 63) |
|  | Administration | Manage Roles (roles) | index = 1, new = 2, create = 4, edit = 8, update = 16, destroy = 32 (Total: 63) |
|  | Administration | Manage Settings (accounts) | Settings = 2048, update\_settings = 4096 (Total: 6144) |
|  | Administration | Manage Account Info (accounts) | show = 1, edit = 2, update = 4, destroy = 8, change\_owner = 16, plans = 32, plan = 64, billing = 128, paypal = 256, plan\_paypal = 512, cancel = 1024 (Total: 2047) |
| Family Member | Personal | Personal Info (users) | index = 1, new = 2, create = 4, edit = 8, update = 16, destroy = 32, profile = 64 |
|  | Personal | Change Password (users) | password = 128 |
|  | Personal | Add Members (families) | my\_family = 64 |

## Menus

Menus are generated by querying the MenuItem model and filtering the results based on the privileges (ultimately, the controller/action combinations) for the current user and grouping them by the “category” attribute of MenuItem. This happens as follows: If the session does not contain a :current\_menu entry, the current\_menu(current\_user) method is called on the MenuItem model. This method first checks to see if the user is the owner of the account, if so, then all menu\_items are added to the available list. If not, then for each menu\_item, the can? Method on the user model is called, passing in the controller and action associated with the menu\_item. The “can” method returns true if EITHER the controller/action combination is a “global privilege”, as defined in the GLOBAL\_PRIVILEGES array on the privilege model, OR the user’s roles include privileges for that controller/action combination. As mentioned earlier, the “for” call is defined as a named scope on the Role model, and performs a bitwise comparison of the unique identifying integer defined for that controller/action combo in Privilege::CONTROLLER\_ACTIONS against the actions integer stored with the privilege in the database. When the list of privileges for that user has been compiled, each are written out into an array of string arrays, grouped by category, and returned as the “current\_menu” (and placed on the session). When the menu is rendered, the category is displayed as a large button, and the submenu extends out to the right from the category, displaying the various menu items that are available to the user. The image and text for the category are retrieved from Privilege::CATEGORIES, and the link that is associated with the category button itself is derived from a lookup to Privilege::ROOT\_MENU\_ACTIONS in the helper method link\_for\_privilege(priv\_controller) in application\_helper.rb. Clicking on one of the submenu buttons invokes the action associated with the menu item.

## Child Menus

Menu Items can have Child Menu Items. When a menu item is clicked, the children are selected and rendered as left menu items, providing that the is\_collection attribute is true. Left\_menu is an empty method in application\_controller. Whichever controller is associated with a given two-column layout page implements the left\_menu function to retrieve the appropriate left\_menu.

Following is a table of menu items. Left Menu Items are described below in the Two Column Layout section.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Category** | **Menu Items** | **Child Menu Items** | **Is Collection?** | **Controller** | **Action** |
| Administration | People | N/A | True | families | index |
|  |  | View Families | True | families | index |
|  |  | - New Family | False | families | new |
|  |  | - Edit Family | False | families | edit |
|  |  | Import Family(ies) | True | csv | import |
|  |  | Manage System Users | True | users | index |
|  |  | - New User | False | users | new |
|  |  | - Edit User | False | users | edit |
|  |  | Family Members | False | families | members |
|  |  | - Add Member | False | users | new |
|  |  | - Edit Member | False | users | edit |
|  | Account | N/A | True | accounts | edit |
|  |  | Modify Organization Name | True | accounts | edit |
|  |  | Change Billing Plan | True | accounts | plan |
|  |  | Edit Payment Information | True | accounts | billing |
|  |  | Edit Account Settings | True | accounts | settings |
|  |  | Cancel Your Subscription | True | accounts | cancel |
|  |  | Import Data | True | csv | import |
|  |  | Change Owner | True | accounts | change\_owner |
|  | Roles | N/A | True | roles | index |
|  | Settings | N/A | True | accounts | settings |
| Personal | Profile | N/A | True | users | profile |
|  | Change Password | N/A | True | Users | password |
|  | Family Members | N/A | True | families | my\_family |
|  |  | - Add Member | False | users | add |
|  |  | - Edit Member | False | users | edit |

## Two Column Layout

The two column layout supports a set of sub menu items beyond the single level of menu items listed by category in the horizontal menu at the top. This is invoked by using the ‘two\_column’ layout, and overriding the left\_menu function that is defined as an empty function (a la an abstract class) in application\_controller.rb. The left\_menu function simply returns a menu\_array, which is an array of two element arrays. Each two element array consists of a string (the menu text) and a link (the link to be redirected to when the menu item is clicked). The left\_menu function is implemented in any controller that needs one. Help text can also be displayed on a per controller basis in the form of a post-it that will appear underneath the left menu. This is implemented in a similar way. The controller declares a private function called “help\_text”, which overrides an empty declaration in application\_controller.rb. In the left menu view, a helper function (in application\_helper.rb) called “helpful\_information(help\_text)” is called unless help\_text is nil. So, if help\_text is set in the current controller, this helper function will display the appropriate help text for that controller. It will do so UNLESS the help\_text is nil (not defined in the current controller) OR a user\_preference has previously been set for that particular help. The user preference is set when a user sees the help, then clicks on the X button in the upper right corner of the help. This is done via an ajax call, and the help is then immediately hidden. **Open question**: Provide a screen where a user can manage all of his/her help post-its? That would be the only way for a user to see the help again after hiding it the first time.

## User Preferences

User preferences are name/value pairs stored in the user\_preferences table along with an optional seq\_no to support arrays of preferences of the same type. One intrinsic user preference is the MY\_LINKS preference that takes as its value a string that can be divided into three substrings delimited by ‘#’. The first substring is the controller for the link, the second substring is the action for the link, and the third substring is the user-readable text for the link. By default, these are the menu items that a user has access to. However, if the user modifies the list by changing its order, deleting one or more items or adding new items (by clicking on an Add to Favorites button present on any page), then the list is stored in the database as MY\_LINKS preferences.

## Export to CSV/Excel

Export uses the built-in CSV module and is called via a button in the menu bar (shared/\_menubar.html.erb). The menu bar calls “indexpath”, which is the controller action that is passed into the partial followed by the suffix csv. This calls the CSV format on the “indexpath” action. So, in the appropriate controller, the CSV option must be enabled. The CSV option will call the export\_csv function, which is defined in application\_controller.rb. Arguments are an array of columns (which will be headers) and an array of objects. A filename is also passed in as the third argument which will be used in conjunction with the date to create a unique file name. Export\_csv in turn calls csv\_for, which creates the CSV, then export\_csv sets the csv content type to excel and calls “send\_data”, which displays the file.

## Styles

Global styles are defined in the application style sheet, with styles that are specific to controllers defined in controller specific style sheets. There are global styles defined for Lists, Forms, and Buttons. Buttons have gradients that are defined using the –webkit-gradient style, which takes a From color and a To color. There are three states of the buttons: Normal, Hover, and Active which is the style shown when the button is clicked. Normal and Active just reverse the colors and change the box shadow slightly.

## Workflow for setting up a new account

1. A new customer browses the public site and selects free trial (or a paid subscription)
2. The new customer fills in information about the account and about the administrative user (the Account owner) for the account.
3. Account owners are exempted from the authorization checking, and are therefore “account superusers”. It is possible to grant other user semi-superuser privileges (except ownership changing) through roles (e.g. the System Administrator default role or a custom role).
4. When the customer saves the information, a new account is created, a new user is added to that account, and a default set of administrative privileges are granted to that user. Also, a default set of Roles are assigned to the account so that they can be granted to other users that the administrative user may create.
5. The account owner for the new account signs in, and can begin adding new users and granting them Roles. The Administrative user can also define new Roles that are specific to her account if she likes (this should probably be pretty unusual).
6. The account owner also sets account settings to personalize their account, e.g. the term used to describe a semester or a student, and the email address(es) to be used for alerts.
7. The account owner can optionally reassign “ownership” of the account to another person. When the change is saved, only the new owner (or the superuser) can then reverse the change.

## Steps to add a new listing page:

1. Controller: Add respond\_to :js, :only => :index and helper\_method :sort\_column
2. Controller: In protected section, add:

def collection

@<collection name> ||= end\_of\_association\_chain.search(params[:search]).order(sort\_column + ' ' + sort\_direction).paginate(:per\_page => 15, :page => params[:page])

end

1. Controller: in private section add:

private

def sort\_column

<model name>.column\_names.include?(params[:sort]) ? params[:sort] : "1"

end

1. Model: add the following and adjust the search:

def self.search(search)

if search #&& column\_name && self.column\_names.include?(column\_name)

where('name LIKE ?', "%#{search}%")

else

scoped

end

end

1. Add index.js.erb with the ajax call: $('#roles').html("<%=j render 'role\_list' %>"); (substitute the right ID’s)
2. Add a partial \_role\_list.html.erb to display the table and facilitate ajax
3. Add the shared menubar partial to the index.html.erb file

## Subscriptions

Billing options will include yearly at $1,500.00 per year, or monthly at $150.00 per month. The customer will be able to sign up for a free 30-day trial prior to being billed. When they sign up, they will not be asked for a credit card, but will be notified to convert 2 weeks in, 1 week out from the end, and daily for the last 3 days. To sign up for the trial, they will need to provide the name and address of their organization, as well as the name and email address of the account owner. If the customer signs up for a paid plan, their credit card or paypal account will be charged immediately.

## User Limits

The account model calls has\_subscription, which looks for \*\_limit fields in the subscription table (and the subscription\_plan table, which populates the value in the subscriptions table upon account creation). If there is a value in the limit field, this value is checked by this function in the User controller and in the New User view. If the number of users is updated in the plan after a subscription is created, that subscription is not affected.

## Miscellaneous

Made a change to rails3\_acts\_as\_paranoid.rb 0.1.3 to eliminate DEPRECATION warnings. #alias\_method :foreign\_key, :primary\_key\_name unless respond\_to?(:foreign\_key)

alias\_method :foreign\_key, :primary\_key\_name unless instance\_methods.include?(:foreign\_key)

warnings