# VolunteerScheduler Module Overview

We implement **Decidim::VolunteerScheduler** as a namespaced Rails engine (decidim module) following the latest Decidim **develop**-branch conventions. The module lives in its own gem/engine (e.g. decidim-volunteer\_scheduler) with isolated namespace Decidim::VolunteerScheduler[[1]](https://docs.decidim.org/en/develop/develop/modules#:~:text=module%20Decidim%20module%20Verifications%20module,%3A%3ARails%3A%3AEngine%20isolate_namespace%20Decidim%3A%3AVerifications%3A%3AMyVerifier). In the component manifest we register the engine and a separate admin engine:

Decidim.register\_component(:volunteer\_scheduler) do |component|  
 component.engine = Decidim::VolunteerScheduler::Engine  
 component.admin\_engine = Decidim::VolunteerScheduler::AdminEngine  
 # ... settings/hooks below ...  
end

This ensures the module’s public UI and its admin backend are mounted properly[[2]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=component). We also enable component hooks and data export/import if needed (beyond Phase 1) by adding code in the Decidim::VolunteerScheduler::Engine and manifest.

All core models (below) include Decidim::Resourceable so they can be treated as Decidim resources and can use cards/cells as UI fragments[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work). We isolate their database tables (e.g. via migrations prefixed volunteer\_scheduler\_). Each model has its own migration file defining its attributes (XP amounts, statuses, foreign keys, etc).

## Data Models

We define the following primary models (each Decidim::VolunteerScheduler::<Model>) with fields and relations:

* **VolunteerProfile** – tracks each volunteer’s stats. Fields include:
* xp (integer, default 0) – total earned experience points.
* level (integer, default 1) – volunteer’s current level.
* referral\_code (string) – a unique code or token for referrals.
* user\_id – reference to the Decidim user/participant (e.g. Decidim::User or related participant model).  
  Relationships: belongs\_to :user; has\_many :task\_assignments; has\_many :scicent\_transactions; has\_many :referrals\_as\_referrer (see Referral below). Include Decidim::Resourceable so it can appear in cards/views[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work).
* **TaskTemplate** – defines a volunteer task opportunity. Fields include:
* title and description – human-readable task details.
* xp\_reward (integer) – XP awarded upon completion.
* level\_required (integer) – minimum volunteer level to see/accept it.
* category (string) – optional categorization (e.g. “administration”, “outreach”, etc).
* frequency (string, enum: "daily" or "weekly") – whether this task recurs daily or weekly.  
  This model has\_many :task\_assignments. Also include Decidim::Resourceable so it can be registered as a component resource and have a UI card[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work). For example:

class Decidim::VolunteerScheduler::TaskTemplate < Decidim::ApplicationRecord  
 include Decidim::Resourceable  
 attribute :title, :string  
 attribute :description, :text  
 attribute :xp\_reward, :integer, default: 0  
 attribute :level\_required, :integer, default: 1  
 attribute :category, :string  
 attribute :frequency, :string # "daily" or "weekly"  
 has\_many :task\_assignments, class\_name: "Decidim::VolunteerScheduler::TaskAssignment"  
end

* **TaskAssignment** – records when a volunteer accepts and submits a task. Fields:
* volunteer\_profile\_id – which volunteer.
* task\_template\_id – which task.
* status (enum string: "pending", "approved", "rejected", default "pending").
* submitted\_at, reviewed\_at (timestamps of submission and admin review).
* reviewer\_id – optional admin user who approved/rejected.  
  Relations: belongs\_to :volunteer\_profile, belongs\_to :task\_template. Include Decidim::Resourceable. We enforce the typical follow-up flow: volunteer creates (:new/:create) assignment to take a task, then updates (submits) it, and an admin edits it to approve/reject.
* **Referral** – manages a 5-level referral tree. Each record links a *referrer* to a *referred* volunteer. Fields:
* referrer\_id, referred\_id – both are VolunteerProfile IDs.
* level (integer 1–5) – the referral depth (1 = direct referral, up to 5).  
  Associations:
* belongs\_to :referrer, class\_name: "Decidim::VolunteerScheduler::VolunteerProfile"  
  belongs\_to :referred, class\_name: "Decidim::VolunteerScheduler::VolunteerProfile"
* By storing the level, we can easily manage commission calculations. Business logic should ensure the tree depth does not exceed 5.
* **ScicentTransaction** (custom named) – ledger of XP and commission earnings. Fields include:
* volunteer\_profile\_id.
* xp\_amount (integer) – XP gained.
* sale\_amount (decimal) – if applicable (for fundraising tasks).
* commission (decimal) – referral commission amount.
* description (string) – e.g. “Task completed: XYZ” or “Referral bonus”.  
  Relations: belongs\_to :volunteer\_profile.

We add Rails migrations to create tables for each model, including foreign keys (e.g. volunteer\_profile\_id on assignments/transactions, etc.), and indexes on frequently queried fields. All models reside under the decidim\_volunteer\_scheduler database schema (prefix) to avoid conflicts.

## Permissions

We leverage Decidim’s built-in permission system to secure actions[[4]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=Permissions)[[5]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=). A custom permission class Decidim::VolunteerScheduler::Permissions < Decidim::DefaultPermissions is created in app/permissions/decidim/volunteer\_scheduler/permissions.rb. In it, we enforce rules such as:

* Public (anonymous) users can only view **Available Tasks** listings if tasks are public, but cannot take or submit tasks.
* A volunteer user can manage their own TaskAssignment (create/view/edit until approved). Typically: allow :read on TaskTemplate to all, allow :create on TaskAssignment for signed-in users, and allow :read/:update on a TaskAssignment only if it is authored\_by? (belongs to) that user. We can use authorable or check assignment.volunteer\_profile.user == current\_user.
* Admins (component admins or higher roles) can :read, :update, :destroy any TaskTemplate or TaskAssignment. They can bulk-edit statuses.
* Similarly, only the volunteer themselves can view their VolunteerProfile (XP, referrals), while admins can view/modify all profiles if needed.

In controllers we use enforce\_permission\_to to apply this permission chain[[5]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=). For example:

class TaskAssignmentsController < Decidim::VolunteerScheduler::ApplicationController  
 include NeedsPermission  
 register\_permissions(::Decidim::VolunteerScheduler::Permissions, ::Decidim::Permissions)  
  
 def index  
 enforce\_permission\_to :read, :task\_assignment  
 # ...  
 end  
  
 def create  
 enforce\_permission\_to :create, :task\_assignment  
 # ...  
 end  
  
 def show  
 enforce\_permission\_to :read, :task\_assignment, my\_object: @task\_assignment  
 # ...  
 end  
end

This ensures only authorized users can perform actions. (See the Decidim docs for custom permissions classes[[5]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=).)

## Task Assignment Workflow

We implement a **follow-up style workflow** for tasks: volunteers “take” a task (creating a TaskAssignment), then later mark it complete, after which an admin reviews it. This parallels Decidim’s idea of components where resources can be approved. For example:

1. **Available Tasks**: volunteer views tasks (TaskTemplate) filtered by their level (see below) and chooses one.
2. **Accept Task**: on click, a new TaskAssignment is created (status=pending).
3. **Submit Work**: volunteer edits the assignment (e.g. uploading proof or entering a report) and marks it “submitted” (we set submitted\_at).
4. **Admin Review**: in the admin panel, the component admin sees pending assignments. They can approve or reject each (or select multiple for bulk actions). On approval, status is set to approved, and reviewed\_at/reviewer\_id are recorded. On rejection, status=rejected.
5. **XP & Referral Handling**: when an assignment is approved, we immediately create a ScicentTransaction giving the volunteer the XP (xp\_amount = task.xp\_reward) and also allocate referral commission up the chain (up to 5 levels) according to configured percentages (below). The volunteer’s VolunteerProfile.xp is incremented accordingly, and we recalc their level if thresholds crossed. (Hooking into callbacks or service objects here.)

No external JS frameworks are used – the accept/submit flows are standard Rails forms/links. The UI uses ERB and view cells (see below).

## Task Filtering & Scheduling

Each TaskTemplate can be marked for **daily** or **weekly** scheduling. We implement scopes like:

scope :daily, -> { where(frequency: 'daily') }  
scope :weekly, -> { where(frequency: 'weekly') }

When generating the *Available Tasks* list for a volunteer, we filter by their VolunteerProfile.level (only tasks where level\_required <= volunteer.level) and by frequency (e.g. show daily tasks each day, weekly tasks once a week – this logic could be time-based or simply flagged by the admin in the UI). For example, in a cell or controller:

tasks = TaskTemplate.where("level\_required <= ?", volunteer.level)  
 .where(frequency: 'daily').shuffle.first(10)

The UI can randomize or paginate these tasks as needed.

## Volunteer Dashboard (Cells & Views)

We build a **Volunteer Dashboard** page using Decidim’s cell/view-model system[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work). This page includes multiple components:

* **Available Tasks**: a cell listing current tasks (as above) the volunteer can take. Each row has an “Accept” button to create an assignment.
* **My Assignments**: a cell listing this volunteer’s open TaskAssignments (status pending or rejected), and those in progress. Each entry might link to a detail/edit page.
* **Failed Assignments**: a cell specifically for assignments that were rejected, so volunteers can see which tasks they need to retry.
* **Referral Link**: a cell showing the volunteer’s unique referral URL or code (VolunteerProfile.referral\_code) to invite others. Clicking the link could automatically record referrals when new users sign up with that code.
* **XP & Level Status**: a cell displaying the volunteer’s total XP and current level, plus a progress bar toward the next level threshold. We can compute the percentage based on our configured thresholds.

Each of these UI components is implemented as a Rails cell (subclassing Decidim::ViewModel) in app/cells/decidim/volunteer\_scheduler/. For example, a DashboardCell or multiple smaller cells. In our engine initializer, we add the cell view paths so Decidim can find them[[6]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=,module%3E%2Flib%2Fdecidim%2F%3Cmodule%3E%2Fengine.rb):

initializer "decidim\_volunteer\_scheduler.add\_cells\_view\_paths" do  
 Cell::ViewModel.view\_paths << File.expand\_path("#{Decidim::VolunteerScheduler::Engine.root}/app/cells")  
 Cell::ViewModel.view\_paths << File.expand\_path("#{Decidim::VolunteerScheduler::Engine.root}/app/views")  
end

We use card\_for @task\_template or custom partials to render task and profile information[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work). All UI uses ERB templates (no React or Vue).

## Admin Interface

The module includes an **admin panel** (backoffice) under the Decidim admin dashboard. The admin engine (e.g. Decidim::VolunteerScheduler::Admin::TaskTemplatesController) handles CRUD for TaskTemplate and reviewing TaskAssignments. We rely on Decidim’s admin layout. Key admin features:

* **TaskTemplate Management**: Admins can create/read/update/delete task templates, setting title, XP, level requirement, etc. These controllers inherit from Decidim::VolunteerScheduler::Admin::ApplicationController. Permission is open to component admins.
* **Assignment Review**: Admins see a list of pending assignments. They can select one or many and mark them approved or rejected. We implement batch actions in the controller (e.g. approve\_all). Each approval triggers XP and commission logic.
* **Leaderboard**: A view (e.g. Admin::LeaderboardsController#index) that shows top volunteers by XP. This simply queries VolunteerProfile.order(xp: :desc) and displays rank, name, and points.
* **Settings Page**: In the admin panel for this component (via the Configure button), we present the component’s **Settings UI** (see next section).

We use Decidim’s component.admin\_engine to mount these routes[[2]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=component). The admin controllers also use enforce\_permission\_to :manage, :resource or similar to restrict to admins.

## Configurable Settings

Via Decidim’s component settings system[[7]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=,), we add global settings (changeable in the admin UI) for key parameters:

Decidim.register\_component(:volunteer\_scheduler) do |component|  
 # ...  
 component.settings(:global) do |settings|  
 settings.attribute :xp\_per\_task, type: :integer, default: 10  
 settings.attribute :referral\_commission\_pct, type: :integer, default: 5  
 settings.attribute :level\_thresholds, type: :text, default: "100,300,600,1000"   
 # ... others as needed ...  
 end  
end

* **XP per task**: default XP to award (though each task can override via its own xp\_reward).
* **Referral Commission %**: the commission rate for referrals. We can apply this per level or flat (e.g. 5% for any referred sale).
* **Level Thresholds**: a comma-separated list or JSON that defines XP needed for each level. The UI shows the current level as the highest threshold crossed.

Each setting will have I18n labels and help text (like decidim.components.volunteer\_scheduler.settings.global.xp\_per\_task). This lets admins configure the reward system without code changes[[7]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=,).

## Notifications

We use Decidim’s **internal notifications** (in-app tracker) for user alerts[[8]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=To%20publish%20an%20event%20to,should%20be%20used). Whenever a volunteer earns XP or a commission, we publish a Decidim event:

Decidim::EventsManager.publish(  
 event: "decidim.events.volunteer\_scheduler.assignment\_approved",  
 event\_class: Decidim::VolunteerScheduler::AssignmentApprovedEvent,  
 resource: @task\_assignment,  
 affected\_users: [@volunteer\_profile.user],  
 followers: []  
)

Decidim will then create internal notifications for the affected users (the volunteer, possibly their referrers). We intentionally *do not* send emails for most updates. However, per requirements we do send email only for: the welcome message to a new volunteer, notifications of perks unlocked, and a monthly summary. Those few email events can also be implemented via Decidim::Notifications or custom mailers. All other alerts (task approved, XP gained, referral bonus) appear in the **Bell icon** notifications inside Decidim[[8]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=To%20publish%20an%20event%20to,should%20be%20used)[[9]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=ActiveSupport%3A%3ANotifications,resource). This leverages Decidim’s EventPublisherJob system, where publishing an event as above triggers both email (if enabled) and in-app notification jobs[[9]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=ActiveSupport%3A%3ANotifications,resource)[[10]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=Decidim%3A%3AEventsManager.publish%28). We set followers: [current\_user] as needed so the user sees it.

## Extension Points

The code is organized for extensibility:

* **Callbacks/Hooks**: After a TaskAssignment is approved, a service object (e.g. VolunteerScheduler::AssignmentApprover) handles awarding XP and referral commission, but can be extended or overridden in future. We add a placeholder for a webhook call (e.g. for external payment) like ExternalWebhooks::Notify.call(transaction).
* **Console and Seeds**: We include seed data files under db/fixtures so developers can load sample tasks.
* **Mailer Stubs**: We include a stub mailer WelcomeMailer for the welcome email, though this can be fully implemented later.

All classes and methods are documented with comments. We adhere to Decidim conventions (e.g. resources go under Decidim::VolunteerScheduler, all queries scoped to the component, etc). No external JS libraries are used; all front-end is ERB + Cells.

## Summary

In summary, **Phase 1** of volunteer\_scheduler includes a clean, namespaced Rails engine with models (VolunteerProfile, TaskTemplate, TaskAssignment, Referral, ScicentTransaction), controllers/views for volunteers and admins, and fully configurable settings. It uses Decidim’s framework for components, resources, cells, permissions, and notifications[[7]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=,)[[5]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=)[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work)[[8]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=To%20publish%20an%20event%20to,should%20be%20used). This provides a working core system that can be extended in future phases (e.g. richer referral logic, webhooks, multi-language text) while leveraging Decidim best practices throughout.

**Sources:** Decidim official docs on module development, components, settings, permissions, view models (cells) and notifications[[7]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=,)[[5]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=)[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work)[[8]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=To%20publish%20an%20event%20to,should%20be%20used).

[[1]](https://docs.decidim.org/en/develop/develop/modules#:~:text=module%20Decidim%20module%20Verifications%20module,%3A%3ARails%3A%3AEngine%20isolate_namespace%20Decidim%3A%3AVerifications%3A%3AMyVerifier) Modules :: Decidim Docs

<https://docs.decidim.org/en/develop/develop/modules>

[[2]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=component) [[7]](https://docs.decidim.org/en/develop/develop/components.html#:~:text=,) Components :: Decidim Docs

<https://docs.decidim.org/en/develop/develop/components.html>

[[3]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=In%20your%20%60decidim,concern%20for%20this%20to%20work) [[6]](https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html#:~:text=,module%3E%2Flib%2Fdecidim%2F%3Cmodule%3E%2Fengine.rb) View Models (a.k.a. Cells) :: Decidim Docs

<https://docs.decidim.org/en/develop/develop/view_models_aka_cells.html>

[[4]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=Permissions) [[5]](https://docs.decidim.org/en/develop/develop/classes/permissions.html#:~:text=) Permissions :: Decidim Docs

<https://docs.decidim.org/en/develop/develop/classes/permissions.html>

[[8]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=To%20publish%20an%20event%20to,should%20be%20used) [[9]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=ActiveSupport%3A%3ANotifications,resource) [[10]](https://docs.decidim.org/en/develop/develop/notifications.html#:~:text=Decidim%3A%3AEventsManager.publish%28) Notifications :: Decidim Docs

<https://docs.decidim.org/en/develop/develop/notifications.html>