ChatGPT

VolunteerScheduler Module Implementation Plan

This plan outlines building a Decidim engine (gem) called `decidim-volunteer_scheduler` that adds volunteer scheduling with referrals, rewards, tasks, dashboards, etc. Use the Decidim generators to scaffold and follow Decidim's conventions and architecture [1] [2]. Begin by running:

```
decidim --component volunteer_scheduler --external
bundle install
```

This creates a new gem with the `decidim-volunteer_scheduler` engine. In the generated gemspec, set `s.add_dependency 'decidim-core', '>= 0.28'` (or the Decidim version used) [3]. Add other needed Decidim module dependencies (e.g. `decidim-meetings` if reusing invites). Use `bundle install` to confirm dependencies [3]. Initialize a git repo and publish to GitHub as usual for Decidim modules. Then set up the development app:

- Run `bundle exec rake test_app` to create a dummy Decidim app for testing [4].
- In the dummy app, add `gem 'decidim-volunteer_scheduler', path: ../decidim-volunteer_scheduler` to its Gemfile, run migrations, and install assets:

```
bundle exec rails decidim_volunteer_scheduler:install:migrations
bundle exec rails decidim_volunteer_scheduler:webpacker:install
bundle exec rails db:migrate
```

These commands mirror other modules' install steps [5]. Also ensure your Decidim app has ActionCable enabled by uncommenting `require "action_cable/engine"` in `config/application.rb` [6]. For code style, follow Decidim's linters (RuboCop, ESLint) [7] and naming conventions (namespace all code under `Decidim::VolunteerScheduler`).

Architectural Overview

The module will consist of several interconnected subsystems:

- **Referral & Commission System:** Tracks multi-level referrals and distributes rewards.
- **Scicent Token & Reward System:** Manages token rewards for actions, integrated with referrals/XPs.
- **XP & Leveling System:** Assigns XP for volunteer activities and unlocks privileges at thresholds.
- **Task Management:** Defines task templates and assignments for volunteers.
- **Dashboards & UI:** Builds admin and volunteer interfaces for tasks, stats, and referrals.
- **Background Jobs:** Offloads commission calculations, XP multipliers, etc.
- **Invitations Integration:** Hooks into Decidim's invites to recruit volunteers.
- **Real-time Updates:** Uses ActionCable (or Turbo Streams) for live updates.

- **Settings, Permissions, Notifications:** Uses Decidim settings for config, permission classes for security, and Decidim events for notifications.

Each will be implemented mostly by **extending Decidim's core patterns** (component registration, concerns, events, jobs, etc.) rather than reinventing.

1. Referral & Commission System

Data Models: Create models in `app/models/decidim/volunteer_scheduler/` such as `Referral`, `Commission`, and possibly `UserReferral` or linking to `Decidim::User`. A `Referral` record links a *referrer* user and a *referred* user (or email). Use hierarchical relationships (e.g. using the `ancestry` gem or parent_id up to 5 levels) to model the 5-level tree. Each `Referral` should include: referrer_id, referee_id (user or email), level (1–5), and commission amount (if any). Include `Decidim::Traceable` in these models so changes are logged [8].

Referral Flow: When a user signs up or is invited through the referral link, record the chain of referrals. For each level up to 5, create a `Commission` or update the referrer's pending reward. Each time a new volunteer joins via an invite link, trigger a background job to assign commissions (see Jobs section). The commission percentages (e.g. 10% to level 1, 5% to level 2, etc.) should be configurable via component settings (see Settings section).

Commands and Controllers: Implement a command (in `app/commands/decidim/volunteer_scheduler/`) to process a referral registration. In controllers (inherited from `Decidim::VolunteerScheduler::ApplicationController`), use `enforce_permission_to` to guard actions. A custom `Decidim::VolunteerScheduler::Permissions` class should allow volunteers to view only their own referrals or allow admins full access [9] [10]. For example, allow a user to `:read` their own referrals and allow admins to `:manage` all.

Audit Trail: By including `Decidim::Traceable` in `Referral` and `Commission`, each create/update will be versioned [8]. Use `Decidim.traceability.with_context(author: current_user)` in your create/update logic so the author is recorded. Customize presenters if needed so logs read clearly (see Traceable docs).

2. Scicent Token & Reward Tracking

Treat *Scicent tokens* as a point balance per user. Create a `Reward` or `TokenBalance` model linked to `Decidim::User`. Record transactions (earnings/spends). Integrate this with referrals and tasks: whenever a user earns XP or referral commission, also credit tokens. For flexibility, define token values in settings (e.g. how many tokens per XP or per commission). Include `Traceable` on reward transactions.

In the UI (volunteer dashboard), show current token balance. If there's an external token system, the module could either just track balances or interface via API to a blockchain – but focus on internal ledger.

Integration with Referrals

When the referral job computes commissions, also create corresponding token rewards. For example, when a referrer gets a 5-point commission, give them X tokens. Use jobs (below) to create `Reward` records.

Integration with Tasks

Optionally, awarding tokens for task completion: when a volunteer completes an assigned task, trigger a reward. This can be done in the task completion command or via an event (see Notifications).

3. XP-Based Leveling System

Implement a leveling mechanism parallel to tokens. Create a `Level` or simply store `xp` on the user's volunteer profile (e.g. extend `Decidim::User` via a decorator or a separate `Profile` model). Define XP thresholds for levels (e.g. level 2 at 100 XP, level 3 at 250 XP, etc.) in component settings.

Earning XP: Assign XP for actions: e.g. successful referrals, completed tasks. After awarding XP, check if the user crosses a level boundary. If so, "unlock" capabilities. This may mean granting a role or showing new UI elements. You can implement an event like `decidim.events.volunteer.level_up`, publish it via `Decidim::EventsManager.publish`, and have subscribers (or use `NotificationGenerator`) to send an alert [11].

Unlocks: Decide what capabilities unlock (e.g. higher commission rate, moderator privileges, ability to create tasks). Use Decidim permissions classes to grant new permissions when a user is in a certain level (e.g. check in `Permissions#permissions` the user's XP or level).

4. Task Templates and Assignments

Define a new component within the module, e.g. `Decidim::VolunteerScheduler::Tasks`. Use the Decidim component pattern: in the engine's registration (in `volunteer_scheduler/engine.rb`), call `Decidim.register_component(:volunteer_scheduler)` and configure `component.settings` etc [12] if the tasks are part of a participatory space. If tasks are global, you can just create standard Rails controllers (no Decidim component wrapper).

Models: Create `TaskTemplate` (fields: title, description, default XP or tokens reward, requirements) and `TaskAssignment` (fields: user_id, task_id, status). Use `Decidim::Traceable` on these.

Admin UI: In `app/controllers/decidim/volunteer_scheduler/admin/`, create controllers for managing templates and overseeing assignments. Use view cells or conventional Rails views (Decidim's CSS classes) to match style.

Volunteer UI: In participant-facing controllers, list available tasks and allow a volunteer to request assignment. Once assigned, volunteers can mark a task as complete (possibly with admin approval). Trigger XP/token rewards on completion.

Questionnaires (Optional): If sensitive info needed, leverage Decidim's questionnaire component by attaching a form to task acceptance (similar to how TimeTracker does [13]).

Notifications: When tasks are assigned or completed, publish Decidim events for notifications [11] so users are notified by email or internal message. For example, on assignment publish `decidim.events.volunteer.task_assigned` with the task resource.

5. Dashboards and Interfaces

Volunteer Dashboard: Add a link in the user account menu (via menu customization) to "My Volunteer Dashboard." Build a page showing: - Current XP and level, token balance. - Referral tree or summary (e.g. how many referrals at each level). - Available tasks and status of assignments. - Recent activity log (using traceable history or custom summaries).

Use Decidim view hooks or cells for consistent UI. You may create a `Decidim::VolunteerScheduler::DashboardCell` to encapsulate rendering logic. Follow Decidim's styling (Bootstrap classes, etc.).

Admin Panel: In the Decidim Admin interface, add a "Volunteer Scheduler" section (via `config/admin_navigation.yml` or Rails routes) where admins can: - Configure global settings (see below). - View all referrals, commissions, tasks. - Manually trigger jobs or audits. - Manage invitations and check referral links.

For index pages, use Decidim's filters and table views to let admins search volunteers, referrals or tasks.

Always scope data to the current Organization. Prefix controllers with `Decidim::VolunteerScheduler::Admin::` and views under `app/views/decidim/volunteer_scheduler/admin/...` to match Decidim's layout.

6. Background Jobs

For any heavy or scheduled processing, use ActiveJob (which is supported by Decidim) [14]. Decidim is agnostic to backend; delayed_job is simplest (DB-backed) [14], but Sidekiq/Redis is fine.

- **Commission Job:** When a new referral or user signup happens, enqueue a job to compute and credit commissions up the chain. This job will look up the referrers, calculate each level's commission (using settings %), create `Commission` records, and credit tokens/XPs.
- **XP Multipliers:** If you allow time-based or event-based XP multipliers (e.g. double XP weekends), a scheduled job can apply multipliers or reset counters.
- **Notification Job:** Use `Decidim::EventsManager` with `force_send: true` if you want immediate notification bypassing preferences [11]. Otherwise, rely on `EventPublisherJob` in Decidim Core to process events asynchronously.
- **Invites Processing:** If volunteers sign up via invites, a job can periodically expire old invites or send reminders.

Ensure to configure an ActiveJob backend. For simplicity, add the `delayed_job` gem and set `ActiveJob::Base.queue_adapter = :delayed_job` in an initializer, as Decidim suggests [14]. Index your database fields (e.g. user_id, email) on jobs to maintain performance.

7. Invitation System Integration

Decidim already includes an invitation feature for meetings/conferences [15]. If volunteers are organized under a "meeting" or event, the module can reuse this: e.g. admins invite emails to volunteer events just as meetings do. The invites allow existing or new users. You can either: - **Reuse Meetings Invites:** Require `decidim-meetings` as a dependency and use its invites controllers/models to send referral/invites. The referral code could piggyback on the invite code.
- **Custom Invite:** Or, copy the pattern: create a `VolunteerInvite` model similar to `Decidim::Meetings::Invite`. Use Devise's `devise_invitable` or simply mailers to send invitation links to join Decidim with tracking of the referrer.

In either case, tie the invite email's signup link to record a `Referral`. On acceptance, the invite's token triggers the 5-level commission logic. Document clearly how the invite flow connects with referral logic. Always respect Decidim's email templates (use the I18n keys under `decidim.volunteer_scheduler.*`).

8. Real-Time Updates (ActionCable/UI)

For live updates (e.g. new tasks appear, referral credit updated), use Rails ActionCable or Turbo Streams. Decidim Notify uses ActionCable by default [16]. Ensure the app's `cable` config is set (prefer Redis over Postgres NOTIFY for payload size) [16]:

```ruby
# config/initializers/volunteer_scheduler.rb
Decidim::VolunteerScheduler.configure do |config|
  config.cable_adapter = "redis"
  config.cable_url = "redis://localhost:6379/1"
end
```

(This mirrors `decidim-notify` recommendations [16].) In JS, open a channel (e.g. `DecidimVolunteerSchedulerChannel`) that listens for broadcasts. Broadcast updates from backend when events occur (e.g. `ActionCable.server.broadcast "volunteer_notifications_#{user.id}", message`). For UI, wrap dynamic parts in Turbo Frames or use Stimulus controllers to handle incoming messages and update the DOM. Keep the UX consistent with Decidim's admin/participant theming.

ActionCable Setup: Make sure your Decidim app includes ActionCable (see Setup). Use `rails decidim_volunteer_scheduler:webpacker:install` to set up any JS packages. If using Hotwire/Turbo (Decidim 0.28+ uses Turbo), you can broadcast Turbo Stream templates from Rails jobs.

9. Settings, Permissions, and Notifications

Settings: Use Decidim's Settings API for configurable values [12]. Within `Decidim.register_component(:volunteer_scheduler)`, add:

```
component.settings(:global) do |settings|
  settings.attribute :referral_commission_pct, type: :integer, default: 10
  settings.attribute :referral_levels, type: :integer, default: 5
  settings.attribute :xp_for_task, type: :integer, default: 10
  # ... other settings like token conversion rates, level thresholds, etc.
end
```

For step-level (per space) settings, use `component.settings(:step)` similarly [17]. Provide I18n labels for each setting under `decidim.components.volunteer_scheduler.settings.*`. These settings let administrators adjust the behavior without code changes.

Permissions: As shown in the Permissions docs [9] [10], create `app/permissions/decidim/volunteer_scheduler/permissions.rb` inheriting `Decidim::DefaultPermissions`. In `permissions`, check user roles or referral ownership. For example, allow users to view/claim only their own tasks/referrals, and allow admins full access. In component registration or controllers, use `include NeedsPermission` and `register_permissions` as shown [18] [19]. Always enforce permissions in controllers with `enforce_permission_to :action, :resource`.

Notifications: Trigger Decidim notifications via events [11]. For any notable change (new referral at level, task assignment, level up), call:

```
Decidim::EventsManager.publish(
  event: "decidim.events.volunteer.XYZ_event",
  event_class: Decidim::VolunteerScheduler::XYZEvent,
  resource: @task_or_user,
  affected_users: [user1, user2],    # who to notify
  followers: [],                    # who follows resource (if any)
  extra: { detail: "..." }
)
```

Use a unique event name (start with `"decidim.events."`) so `EventPublisherJob` picks it up [20] [21]. Define corresponding `XYZEvent` classes under `app/events/decidim/volunteer_scheduler/` inheriting `Decidim::Event`. The payload's `resource` must implement `Decidim::Resource` (usually the model's decorator includes it). Set `affected_users` to the users to notify. This automatically enqueues email and internal notifications respecting users' preferences [11].

E.g., on task completion publish a "volunteer.task_completed" event with the volunteer and task owner (admin) as recipients. On referral, notify the referrer of the earned commission.

10. Testing, Performance, and Production Readiness

Testing: Write RSpec tests for all models, commands, and controllers. Follow Decidim's testing guidelines [22]: use the dummy app (`rake test_app`) and include factories or fixtures. You can run tests via `bundle exec rake test_volunteer_scheduler` (auto-generated by Rails engine) and use parallel tests for speed. Include system tests for UI flows if possible. Maintain high coverage (Decidim modules typically aim for ~80%+).

Performance: Index database fields used in lookups (e.g. `referrer_id`, `user_id`). Cache expensive queries (e.g. referral chains). Batch operations in jobs to avoid N+1 queries. Use eager loading for associated records. If the volunteer list grows large, consider pagination.

Security: Sanitize all inputs and use strong parameter filtering. Leverage Decidim's existing role-based access (admin vs participant) and our permission classes [10]. Ensure no sensitive data leak (e.g. only referrers can see their referrals, not others). Use HTTPS and follow Decidim's security docs for settings (CSRF tokens, CSP, etc.).

Deployment: Provide migrations for all tables (`rails decidim_volunteer_scheduler:install:migrations`). Ensure asset compilation via Webpacker (include any new JS/CSS entrypoints in `config/webpacker.yml`). Add versioning in `lib/decidim/volunteer_scheduler/version.rb`.

Documentation: Include a README with installation steps (gemfile entry, install:migrations, webpacker:install, rake db:migrate) similar to other Decidim modules [23] [24]. Document any decoupled component (e.g. if invites are reused from Meetings, mention that). Provide examples of publishing events or invoking jobs.

By adhering to Decidim's architecture (components, concerns, events, etc.) and using its native features wherever possible, this module will integrate smoothly. All custom code (models, controllers) should live under the `Decidim::VolunteerScheduler` namespace, and views should extend Decidim layouts. This ensures the module can be tested and maintained alongside Decidim releases [25] [7].

Sources: This plan is based on the Decidim developer documentation and existing modules (e.g. Notify, TimeTracker) [6] [8] [9] [11] [4], which demonstrate the patterns above. Follow the cited guidelines for code generation, ActiveJob setup, permissions, traceable audits, and notifications to stay consistent with Decidim best practices.

---

[1] [25] Modules :: Decidim Docs
https://docs.decidim.org/en/develop/develop/modules.html

[2] [3] Creating Decidim modules from scratch | CodiTramuntana
https://coditramuntana.com/en/blog/creating-decidim-modules-from-scratch

[4] [22] How to test Decidim engines :: Decidim Docs
https://docs.decidim.org/en/develop/develop/testing.html

[5] [13] [24] GitHub - Platoniq/decidim-module-time_tracker: A tool for Decidim that allows to track time spent by volunteers doing any arbitrary task
https://github.com/Platoniq/decidim-module-time_tracker

[6] [7] [16] [23] GitHub - Platoniq/decidim-module-notify: A conversation recorder and freezer for Decidim
https://github.com/Platoniq/decidim-module-notify

[8] Activity log :: Decidim Docs
https://docs.decidim.org/en/develop/develop/traceable.html

[9] [10] [18] [19] Permissions :: Decidim Docs
https://docs.decidim.org/en/develop/develop/classes/permissions.html

[11] [20] [21] Notifications :: Decidim Docs
https://docs.decidim.org/en/develop/develop/notifications.html

[12] [17] Components :: Decidim Docs
https://docs.decidim.org/en/develop/develop/components.html

[14] ActiveJob :: Decidim Docs
https://docs.decidim.org/en/develop/services/activejob.html

[15] Registrations :: Decidim Docs
https://docs.decidim.org/en/develop/admin/components/meetings/registrations.html