ChatGPT

Coop 400 Riel Members Database and Scorecards – Project Overview and Plan

Executive Summary

The **Coop 400 Riel Members Database and Scorecards** project will create a unified, user-friendly system to manage member information, track tasks/responsibilities, and facilitate cooperative decision-making. The system's **target audiences** include all *members (residents)* who can view their profiles, tasks, and votes; *administrators* who manage data, run polls, and monitor community progress; and *observers* (e.g. auditors or interested partners) with read-only access to public dashboards. By centralizing data and automating workflows, the system will improve transparency, accountability, and engagement – factors often called "the lifeblood of an engaged and satisfied membership" [1]. High-level benefits include streamlined member administration, easier tracking of shared tasks and goals, and data-driven insights (through scorecards and dashboards) to help the co-op operate more efficiently.

Business Requirements

- **Member Profiles & Units:** Each member (resident) needs a profile storing personal and contact information, unit/lot assignment, membership status, and roles within the coop. A central member database should make personal data easily accessible and up-to-date [2]. Administrators must be able to add, edit, and search profiles; members should be able to view and update their own details; observers may have view-only access.
- **Unit Mapping:** The system should map members to their physical units or shares. This supports tracking which tasks and responsibilities are assigned to each unit (e.g. maintenance chores, rent/apartment data).
- **Task & Responsibility Tracking:** Members and admins can create and assign tasks (maintenance tasks, communal duties, projects) to individuals or units. The system tracks task status (e.g. To Do, In Progress, Done), deadlines, and outcomes. Task dashboards allow members to see their pending and completed tasks, and admins to monitor overall task fulfillment.
- **Voting & Decision-Making:** The system must support cooperative voting and polling. Members can vote in Board elections or on co-op proposals. This includes scheduling ballots, recording votes, and securely tallying results. An integration with open-source tools like Loomio (for collaborative decision-making) can be considered.
- **Role Definitions:**
- *Residents (Members):* Can log in to view their own profile, unit information, personal scorecard (see below), and assigned tasks; participate in polls; and receive notifications.
- *Administrators:* Have full access to manage member and unit data, create and assign tasks, set up polls, and view all dashboards/scorecards. They may also generate reports (e.g. member activity, voting results).
- *Observers:* Have read-only access to certain dashboards or scorecards (for transparency), but cannot modify data. This role could include board members, auditors, or external stakeholders.

- **Scorecards (Participation Analytics):** A core feature is *member scorecards* – personalized summaries of each member's participation and responsibilities. For example, a scorecard might track attendance at co-op meetings, number of community tasks completed, and other contribution metrics. Scorecards help recognize active members and highlight areas for improvement. At an aggregate level, analytics modules will visualize coop-wide KPIs (e.g. average task completion rate, meeting turnout) to support data-driven governance. These dashboards will include charts, tables, and drill-down reports.

Technical Requirements

- **Backend (Python Stack):** A Python-based backend is preferred. Frameworks like **FastAPI** or **Django** (with Django REST Framework) are well-suited. FastAPI offers very high performance and async support for handling many concurrent requests [3], which is ideal for real-time features. Django provides a robust "batteries-included" ecosystem (ORM, admin UI, auth) for faster development of CRUD and admin interfaces. The backend will expose a RESTful JSON API (or GraphQL) for all data operations (members, tasks, votes, scorecards). Background processing (e.g. sending reminders or recomputing scores) can be handled by a task queue or scheduler such as Celery or APScheduler (an open-source Python job scheduler that scales from simple to large deployments [4]).
- **Frontend Architecture:** A modern JavaScript single-page application (SPA) is recommended for a responsive UI. Options include **React**, **Vue.js**, or similar. For example, React "simplifies UI development" and enjoys a strong community [5]. Vue.js is another popular choice for its ease of use and performance. The frontend will consume the API for data, display member dashboards and scorecards, and update in real time (via WebSockets or polling). Alternatively, a no-code/low-code platform (e.g. Appsmith, Budibase) could be used for rapid prototyping, though a custom JS frontend offers maximum flexibility. UI frameworks (like Tailwind CSS or Material-UI) will ensure a professional look.
- **Real-Time Features:** To keep users informed, certain data should update in real time. For example, when a task is completed or a vote is cast, dashboards and the concerned users' views should refresh automatically. This can be implemented with WebSockets or Server-Sent Events. FastAPI (built on ASGI) and Django Channels both support WebSockets out of the box. FastAPI is explicitly noted as being *ideal for real-time applications* thanks to async support and WebSockets [6].
- **API & Webhooks:** The system will provide a well-documented JSON API. This allows external services or automations (like the open-source workflow tool **n8n**) to integrate via REST calls or webhooks. For example, n8n workflows could be triggered when a new member is added, sending welcome emails or updating external systems. We will ensure secure token-based API access (e.g. OAuth2 or API keys) and support Webhooks for event-driven automations.
- **Authentication & RBAC:** All users will authenticate securely (e.g. via JWT tokens, or integrate an open-source identity provider like Keycloak for single sign-on). Role-Based Access Control (RBAC) will enforce permissions. Libraries like **Casbin** provide a flexible, open-source way to define and enforce RBAC/ACL policies across services [7]. For instance, Casbin can manage which user roles can create tasks, view reports, or manage members.
- **Notifications & Reminders:** The system will send notifications (email or in-app) for events like upcoming task deadlines, meeting schedules, or voting reminders. This requires a background job system. For example, **APScheduler** can schedule emails or notifications at set times [4]. Alternatively, n8n or similar tools could handle recurring reminders. Integration with mail or chat (Slack, Discord) should be supported using open-source libraries or via n8n connectors.

- **Data Export & Reporting:** Administrative users should be able to export data (e.g. member lists, voting results) in CSV/Excel format. Built-in reporting for common queries (membership growth, task logs, etc.) will also be provided. Data visualization libraries (Chart.js, D3, or Python Plotly) will enable custom reports on web pages if needed.

Open-Source Stack Recommendation

- **Backend Framework:** Use **FastAPI** (for high performance and modern Python async features) or **Django** (for rapid development and built-in admin). Both are open-source and have large communities [3] [8]. For example, FastAPI is "designed for high performance" and is one of the fastest Python frameworks available [3].
- **Database & ORM:** Use **PostgreSQL** as the relational database (open-source, reliable, and scalable). It supports complex queries and can store JSON if needed. The backend can use SQLAlchemy (or Django ORM) for database access. SQLAlchemy is a mature ORM with full Python support.
- **Frontend Framework:** Use an open-source JavaScript framework like **React** or **Vue.js**. React is highly popular (simplifies UI development and offers strong community support [5]). Vue.js is also widely adopted for its simplicity. For UI components, consider frameworks like Material-UI (React) or Vuetify (Vue). If a low-code approach is preferred, tools like Appsmith or Budibase could provide quick web interface building, though with some flexibility trade-offs.
- **Containerization & CI/CD:** Containerize all services with **Docker**. Use **Docker Compose** to orchestrate development (web, API, DB, worker, etc.), and Kubernetes for production if needed. For continuous integration/continuous deployment, use **GitHub Actions**. A typical pipeline builds Docker images, runs automated tests, and deploys to staging/production on push or tag. For example, GitHub Actions can integrate with Docker to "automatically trigger a deployment pipeline to build your Docker image, run tests, push it to a container registry (like GHCR) and deploy it" upon commits [9] [10]. This ensures changes are deployed quickly and reliably.
- **Data Visualization & Dashboards:** Implement scorecard analytics using open-source BI tools. **Metabase** (Apache license) is an excellent choice: it lets you "easily create and share interactive dashboards" with many chart types [11], and it supports 15+ visualization options (bar/line charts, maps, etc.). Metabase can connect directly to the Postgres database to auto-generate charts and even allow non-technical users to build queries. Alternatively, **Apache Superset** or **Grafana** can be used. These tools will power the cooperative's dashboards (e.g. showing task completion rates or meeting attendance trends) with minimal coding.
- **Scorecard & Dashboard Tools:** Beyond general BI tools, consider specialized libs or apps for scorecards. For example, build front-end charts using Chart.js or D3 for embedded scorecards in the app. If using Nextcloud or a similar platform, the **Deck** app (a kanban board) can handle tasks in a visual way; note Nextcloud Deck is an open-source kanban tool (if relevant). However, custom web UIs built in React/Vue can provide more flexibility for scorecard visualization.
- **Open-Source Integrations:** Leverage community tools:
- **Loomio:** An open-source decision-making platform (for collaborative proposals and voting). Loomio can integrate via API so members can deliberate or vote on issues. Loomio "helps [cooperatives] create a more engaged and collaborative culture, build trust and coordinate action" [12].
- **n8n:** Open-source workflow automation (100k+ GitHub stars, 500+ integrations) [13] [14]. Use n8n to automate processes (e.g. when a vote ends, n8n can alert admins or archive results).
- **Casbin:** For RBAC as noted, Casbin is an efficient open-source library supporting many access control models [7].

- **OpenProject:** An open-source project management tool (supports Agile/Scrum/Kanban) [15]. It could be used for internal task tracking or directly integrated for coop projects.
- **Nextcloud (optional):** Provides file sharing and may host Deck (kanban), Contacts, Calendar – all open-source. For example, Deck (Nextcloud app) can manage to-do boards linked to tasks.
- **Authentication:** Consider **Keycloak** (open-source SSO and identity management) for user auth if single-sign-on across apps is needed. Otherwise, use JWT/OAuth libraries in FastAPI/Django.
- **DevOps Tools:** Use **GitHub** or **GitLab** for version control. Code review via pull requests is mandatory. Use automated testing (e.g. pytest) in CI. Set up staging/production environments with infrastructure-as-code (e.g. Terraform or Docker Compose on servers). Backup databases regularly and monitor uptime.

Architecture Overview

The system will follow a modular, service-oriented design. In broad terms, components include:

- **Frontend (Web App):** A single-page application (React/Vue) running in users' browsers (or devices). It interacts with the backend API and renders dashboards/scorecards.
- **Backend API Service:** A Python web service (FastAPI or Django) exposing a RESTful API. It handles all business logic: managing members, tasks, votes, and calculating scores. This service queries the database and returns JSON. It also authenticates users and enforces RBAC (e.g. via Casbin).
- **Database:** A PostgreSQL database stores all data tables (members, units, tasks, votes, logs, etc.). The backend uses an ORM to read/write the database.
- **Background Workers:** A task queue (e.g. Celery with Redis, or APScheduler) runs asynchronous jobs. These jobs handle scheduled tasks like sending reminders, recomputing scorecard metrics overnight, or sending vote result emails. Using a shared datastore, these workers can scale horizontally for performance [4].
- **Automation Service:** An instance of **n8n** or similar runs integration workflows. For example, n8n can poll the backend API for new votes or tasks and trigger emails/notifications or update other systems. The backend exposes webhook endpoints for n8n to call (e.g. when data changes).
- **External Integrations:** Tools like Loomio may be connected via their APIs. For instance, when a cooperative proposal is posted in our system, an n8n workflow could sync it to Loomio for discussion and retrieve voting outcomes.
- **Container Orchestration:** All services (API, frontend, DB, workers, n8n) run in Docker containers. In production, these might run on a Kubernetes cluster or a Docker Swarm to manage scaling, networking, and updates. Docker Compose can orchestrate the setup for development and staging.
- **Networking:** The frontend communicates with the API over HTTPS. The API and workers share a secure connection to the database. Internally, services communicate via a Docker network. Firewalls and SSL ensure security.
- **Diagram (conceptual):**

```
[Browser UI] <--HTTPS/WS--> [API Backend] <--SQL--> [Postgres DB]
     |                           ↑  ⟲
     |                           |  background jobs (Celery/APScheduler)
     |                           |
     +--> [Notifications] <------+  (cron/email jobs)
```

```
          |
          +--> [n8n Automation Workflow] <---> [External (Loomio, Email/Chat)]
```

Fig: Simplified component layout – UI talks to API; API talks to DB; workers and n8n handle background processing.

DevOps and Development Plan

8–12 Week Roadmap (Agile Sprints): We propose an iterative schedule. For example:
- **Sprint 1 (Weeks 1–2):** Set up development environment (Docker Compose), source control, and CI/CD pipeline (GitHub Actions). Establish the database schema (members, units, roles, tasks, votes). Implement basic user authentication and RBAC with Casbin.
- **Sprint 2 (Weeks 3–4):** Develop member profile management (CRUD APIs + UI). Create unit mapping functionality and a simple admin interface. Set up frontend routing (React/Vue components) for login, profile, and admin pages.
- **Sprint 3 (Weeks 5–6):** Implement task management: APIs and UI for creating tasks, assigning to members/units, and updating status. Introduce background jobs for notifications (e.g. APScheduler job that sends email reminders for due tasks).
- **Sprint 4 (Weeks 7–8):** Build voting/poll features: create polls in the API, cast and tally votes, and display results. Optionally integrate Loomio API for decision threads. Begin scorecard calculations (e.g. count tasks completed per member, attendance).
- **Sprint 5 (Weeks 9–10):** Develop scorecard and dashboard UI: charts for participation, task completion, etc., using Chart.js or integrating Metabase/Superset. Refine role permissions. Add user notifications (email or in-app) for task updates and poll outcomes.
- **Sprint 6 (Weeks 11–12):** Testing, user feedback, and deployment. Conduct unit/integration tests, fix bugs. Deploy to a staging environment and run acceptance tests with admin and member users. Finalize CI/CD: automatic deploy to production on tag. Prepare documentation and train administrators.

Environments & Workflow: Developers will work in feature branches and submit pull requests for review. Automated tests run on each push. A protected `main` branch holds stable releases. We use GitHub Actions for CI: upon merging, actions will run linting, tests, build Docker images, and (for `main`) deploy to staging. Releases to production are done by tagging; GitHub Actions then deploys to the production cluster.

Local/Dev Setup: Developers can run a local Docker Compose setup that includes the API service, frontend dev server, Postgres, and worker. This replicates the production stack for testing. A separate **staging** server (cloud or coop-hosted) runs the latest code for QA. The **production** environment (e.g. AWS/GCP or coop server) has similar Docker setup behind a load balancer (e.g. Nginx) with SSL termination.

Maintenance: Once live, we will monitor application health (use Prometheus/Grafana for metrics, or hosted monitoring) and set alerts for downtime. Regular database backups and security updates (dependabot for libraries) will be scheduled. The CI/CD pipeline can include security scans (Snyk or OWASP ZAP). We plan for periodic updates (e.g. quarterly releases) that may add features or upgrades, following the same review process.

Fig: Example of an agile sprint planning (Kanban board). We will use similar boards (e.g. in OpenProject or Nextcloud Deck) to organize development tasks into weekly sprints. The image above illustrates team collaboration on tasks, reflecting our iterative development approach.

Success Criteria & KPIs

To measure project success and continued performance, we will define concrete metrics:

- **User Adoption:** Aim for a high percentage of coop members regularly using the system. A target could be ≥*80%* of active members logging in monthly. Track logins and profile updates to measure engagement.
- **Participation Rates:** Monitor task and voting activity. For example, track the percentage of assigned tasks completed on time or the voter turnout in polls. Improvement in these metrics indicates the coop is using the platform.
- **System Reliability:** The system should be robust. We set a target uptime of *99%+* for the backend. API response times should ideally remain under 1 second for common queries. We will track error rates (via logs) and ensure quick resolution.
- **Performance of Real-Time Features:** If real-time updates are enabled, ensure low latency (e.g. websocket updates under 500ms) for user interactions.
- **Data Accuracy:** All reports (scorecards, charts) must accurately reflect the database. Automated tests and data validation ensure integrity. We will audit sample outputs against raw data periodically.
- **User Satisfaction:** Periodically survey members and admins. Positive feedback (e.g. ≥4/5 satisfaction on ease of use and usefulness) indicates success. Comments about transparency ("information is easily accessible") or efficiency gains will be noted.
- **Community Impact (Qualitative):** We expect enhanced transparency (making coop records easy to view) and coordination. As [73] notes, transparency is crucial for engagement [1]. If members report feeling more informed and engaged because of visible dashboards and easy voting, that's a key success marker.

By reaching these targets and continuously iterating based on feedback, the Coop 400 Riel Members Database and Scorecards system will support the cooperative's mission with modern, open-source tools and best practices [11] [7].

Sources: Core recommendations rely on proven open-source solutions and guidelines from the cooperative and software communities [12] [11] [9]. All recommended technologies (Loomio, n8n, Casbin, Metabase, etc.) are open-source, ensuring transparency and cost-effectiveness.

---

[1] A Recipe for Transparency
https://www.cooperative.com/remagazine/articles/Pages/recipe-for-co-op-transparency.aspx

[2] Free Membership Management Software, Members Database
https://www.raklet.com/blog/free-membership-management-software/

[3] [6] [8] Django vs FastAPI in 2024. Choosing the right framework for your... | by Simeon Emanuilov | Medium
https://medium.com/@simeon.emanuilov/django-vs-fastapi-in-2024-f0e0b8087490

[4] GitHub - agronholm/apscheduler: Task scheduling library for Python
https://github.com/agronholm/apscheduler

[5] Angular vs React vs Vue: The Best Framework for 2025 is… | Zero To Mastery
https://zerotomastery.io/blog/angular-vs-react-vs-vue/

[7] Casbin · An authorization library that supports access control models like ACL, RBAC, ABAC for Golang, Java, C/C++, Node.js, Javascript, PHP, Laravel, Python, .NET (C#), Delphi, Rust, Ruby, Swift (Objective-C), Lua (OpenResty), Dart (Flutter) and Elixir | Casbin
https://casbin.org/

[9] [10] How to Build a CI/CD Pipeline with GitHub Actions and Docker
https://runcloud.io/blog/setup-docker-github-actions-ci-cd

[11] Open source business intelligence, dashboards, and data visualizations | Metabase
https://www.metabase.com/

[12] Loomio - make decisions together without meetings
https://www.loomio.com/

[13] [14] Powerful Workflow Automation Software & Tools - n8n
https://n8n.io/

[15] OpenProject - Open Source Project Management Software
https://www.openproject.org/