Hackathon Management & Voting System

Technical Documentation

Computer Science Department
SUNY New Paltz

February 2026

Contents

1 Overview

The Hackathon Management & Voting System is a full-stack web application for organizing, judging, and voting on hackathon projects at SUNY New Paltz. It integrates with Hydra SSO for authentication and supports multiple concurrent hackathons.

Repository: `https://github.com/compsci-suny-newpaltz/Hackaton-Voting`
Frontend: Vue.js 3 + Vite + TailwindCSS
Backend: Node.js + Express
Database: SQLite (better-sqlite3)
URL: `https://hydra.newpaltz.edu/hackathons/`

2 Architecture

2.1 Tech Stack

Layer	Technology
Frontend	Vue.js 3, Vue Router, Vite, TailwindCSS, Axios
Backend	Node.js 18, Express 4, better-sqlite3
Auth	Hydra SSO (`np_access` JWT cookie)
File Upload	Multer (single-file replace model)
Deployment	Docker Compose on Hydra (port 45821)

2.2 Project Structure

```
Hackaton-Voting/
  client/            # Vue.js 3 SPA
    src/
      components/    # Reusable UI components
      views/         # Page-level views
      router/        # Vue Router config
  server/
    index.js         # Express entry point
    db/
      schema.sql     # SQLite schema
      init.js        # Database initialization
    routes/
      hackathons.js  # Hackathon CRUD + lifecycle
      projects.js    # Project + team management
      voting.js      # Popular + judge voting
      admin.js       # Admin endpoints
  public/            # Built client assets
  docker-compose.yml
  Dockerfile
```

3 Authentication

The system delegates authentication to Hydra SAML Auth via the `np_access` JWT cookie.

1. User visits `/hackathons/`
2. If no valid `np_access` cookie, redirect to `/login?returnTo=/hackathons/`

3. Hydra performs SAML SSO with Azure AD
4. On success, Hydra sets `np_access` cookie (RS256 JWT)
5. Hackathon app verifies JWT using Hydra's JWKS endpoint

3.1 Admin Access

- Hardcoded admin: `gopeen1@newpaltz.edu`
- Faculty auto-admins: users with `faculty` role in SSO claims
- Manual whitelist: additional admins added via admin dashboard

4 Hackathon Lifecycle

Each hackathon progresses through 7 states:

State	Trigger	Description
`upcoming`	Created, start time in future	Not yet accepting submissions
`active`	Current time $\geq$ start time	Accepting project submissions
`ended`	Current time $\geq$ end time	Submissions closed, judging open
`vote_expired`	Current time $\geq$ vote expiration	All voting closed
`review-period`	Vote expired, review window active	Admin reviews before results
`concluded`	Review period ended	Results publicly visible
`archived`	Admin manually archives	Hidden from default listings

Table 1: Hackathon state machine

Time validation: The system enforces `start_time < end_time < vote_expiration`. The optional review period adds a `review_ends_at` timestamp between vote expiration and public results.

5 Voting Systems

5.1 Popular Vote

- One vote per user per project (enforced by DB uniqueness on `userId + projectId`)
- Users cannot vote for their own project
- Vote counts displayed live with 5-second polling
- Snapshot captured at vote close for historical display
- Admin audit page shows all votes per project

5.2 Judge Voting (Rubric System)

Judges score projects using a customizable category-based rubric:

Category	Description	Weight
Innovation / Creativity	Originality of idea or approach	1.0
Functionality	Working features, reliability	1.0
Design / Polish	UX, accessibility, visual quality	1.0
Presentation / Demo	Communication of idea and goals	1.0

Table 2: Default judge categories (customizable by admin)

- Scores: 1–10 per category with optional comments
- Weights: multiplier 0.1–5.0 per category (admin-editable)
- Judges can edit scores until judging phase closes
- Progress tracking: counter shows `x/y` projects scored
- Saved projects display green outline with checkmark

5.3 Results Calculation

The weighted score for each project is calculated as:

$$Score = \frac{\sum_i w_i \cdot s_i}{\sum_i w_i}$$

where w_i is the category weight and s_i is the average judge score for that category.

6 Data Model

6.1 Core Tables

Table	Purpose
`hackathons`	Hackathon metadata, times, settings
`projects`	Team submissions with descriptions, files
`team_members`	Project-to-email associations
`popular_votes`	User upvotes (unique per user+project)
`judge_codes`	Secure access codes for judges
`hackathon_judge_categories`	Rubric categories with weights
`judge_category_votes`	Per-category scores from judges
`comments`	Project discussion threads
`audit_log`	Admin action audit trail

Table 3: Database schema overview

6.2 Key Constraints

- `UNIQUE(judge_code_id, project_id, category_id)` — prevents duplicate judge scores
- `CHECK(score >= 1 AND score <= 10)` — enforces score range
- `UNIQUE(user_id, project_id)` on popular votes — one vote per user
- Email domain enforcement: `@newpaltz.edu` and subdomains only

7 API Endpoints

7.1 Hackathon Management

Method	Path	Description
GET	`/hackathons/active`	List active + upcoming hackathons
POST	`/hackathons`	Create hackathon (admin)
PUT	`/hackathons/:id`	Update hackathon (admin)
POST	`/hackathons/:id/archive`	Archive hackathon (admin)

7.2 Project & Team

Method	Path	Description
GET	`/:hid/projects`	List projects
POST	`/:hid/projects`	Create project
POST	`/:hid/projects/:pid/team-members`	Add team member
DELETE	`/:hid/projects/:pid/team-members/:email`	Remove team member

7.3 Voting

Method	Path	Description
POST	`/:hid/projects/:pid/vote`	Cast popular vote
POST	`/:hid/judge/:code/vote`	Submit judge scores
GET	`/:hid/results`	Get weighted results

7.4 Admin

Method	Path	Description
GET	`/admin/hackathons/:id/categories`	List rubric categories
POST	`/admin/hackathons/:id/categories`	Create category
PUT	`/admin/categories/:id`	Update category
DELETE	`/admin/categories/:id`	Delete category
POST	`/admin/hackathons/:id/categories/reorder`	Reorder categories

8 Deployment

8.1 Docker Compose

```
# Build and start
cd /home/infra/Hackaton-Voting
docker compose build
docker compose up -d

# Logs
docker logs hackathons-app --tail=50
```

```
# Initialize database
docker exec hackathons-app npm run init-db
```

8.2 Local Development

```
# Install dependencies
npm install
cd client && npm install && cd ..

# Initialize database
npm run init-db

# Create .env from example
cp .env.example .env

# Run dev servers (backend + Vite frontend)
npm run dev
```

First admin access: The hardcoded admin `gopeen1@newpaltz.edu` and any user with a `faculty` SSO role are automatically administrators. Access the admin dashboard at `/hackathons/admin/dashboard`.

9 Security

- **Authentication:** Delegated to Hydra SSO — no local passwords
- **Email privacy:** Non-team members see masked emails (`j***e@newpaltz.edu`)
- **Judge codes:** Cryptographically secure, hackathon-specific
- **Input validation:** Server-side length limits, domain enforcement, time constraint validation
- **SQL safety:** Parameterized queries via better-sqlite3
- **Audit logging:** All admin actions recorded with timestamp and user

10 Implementation Status

Phase	Description	Status
Phase 0–1	Core constraints, data validation	Complete
Phase 2	Hackathon lifecycle & concurrency	Complete
Phase 3	Project & team management	Complete
Phase 4	Voting & judging overhaul	Complete
Phase 5	UX feedback & navigation	In Progress
Phase 6	Security & permissions	Planned
Phase 7	File & transaction safety	Planned
Phase 8	Performance & scalability	Planned
Phase 9	Mobile & responsive	Planned
Phase 10	Documentation	Planned

Table 8: Development roadmap (see `TODOS.md` for full details)