Project analysis and improvements

stronaszkoly-api-testing.zip
Archiwum zip

Analyze this project, suggest improvement, fixes and find bugs. Please give me a long rapport. What I want is for you to check if something is wrong, something should be streamlined, something should be redesigned, changed, or you see some things that should be deleted / not used or other things that disrupt this project and will greatly benefit it. Just look at this whole code and analyze everything and give all around suggestions.

Prośba o GPT-5 Pro
Reasoned for 20m 37s

Below is a comprehensive engineering review of your project (`stronaszkoly-api-testing`). I've focused on correctness, security, maintainability, and developer experience. Where possible, I point to concrete files/lines, highlight specific bugs, and include suggested code changes you can apply immediately.

Executive summary (top action items)

P0 – Fix now

1. **Session token generation is not cryptographically strong**
   `v1/login` creates session tokens with `Math.random()` → replace with `crypto.randomBytes`.
2. **Rate limiters defined but not used on v1**
   `loginLimiter` and `refreshLimiter` exist but are not attached to `/v1` routes.
3. **Leaky auth surface (password in URL)**
   `v1/login` accepts `username/password` via query params. Remove this path.
4. **CORS too permissive with credentials**
   `cors({ origin: true, credentials: true })` reflects any `Origin` and allows cookies. Lock this down via allow-list.
5. **No CSRF protection on cookie-based mutating endpoints**
   Introduce a CSRF defense (double-submit or SameSite=None + token).
6. **Approval links never expire in enforcement**
   You store `expiresAt` but don't enforce it in `GET/POST /v1/approvals/:token`.
7. **Admin-level writes not restricted**
   `/v1/overrides` write is protected by `requireAuth` but not by admin check.
8. **File persistence is non-atomic**
   `data.json` and `overrides.json` are updated with plain `writeFileSync` (risk of corruption). Switch to atomic writes.

9. **Jobs invoke `pip install` on every run**
   `/v1/jobs/*-scrape` installs dependencies each time. Add caching/venv or containerize.
10. **Monolithic server file (~48 kB)**
    Split `server/server.js` into routers: auth, timetable, attendance, approvals, jobs, overrides, legacy.

P1 – Next 1–2 sprints

- Unify and de-duplicate API docs (you have *two* Swagger UIs: Express `/docs` and static `/public/docs/index.html`).
- Restrict/relocate Python scripts and generated data out of `/public` (avoid serving internals by default).
- Add body size limits (`express.json({ limit: '64kb' })`) and per-route rate limits (jobs, approvals).
- Align OpenAPI with the actual v1 routes; validate requests server-side (e.g., `zod/express-zod-api`).
- Add E2E tests (Playwright is installed) and unit tests for reducers and server routes (Jest/Vitest + Supertest).

P2 – Later

- Replace the file-store with SQLite (e.g., `better-sqlite3`) or Postgres for robustness & concurrency.
- Move to typed backend (TypeScript) for the server.
- Create a small admin UI for overrides/backups/jobs with explicit permissions.

Architecture & repo layout

- **Front-end**: Vite + React 19 (lazy routes), nice UI primitives, and a solid `src/features` structure.
- **Server**: Express (ESM) with Helmet, cookie sessions, bearer auth (API keys), attendance state, timetable reader/writer, scraping jobs, “remote approvals”, and OpenAPI draft served via Swagger UI.
- **Public assets**: Also store Python scrapers and generated JSON in `/public`, which may be served by Vite in dev.

  Suggestion: Treat `/public` as truly public (only static assets required by the browser). Move scrapers and internal JSON (except those intentionally downloaded by the client) to `/tools` or `/data`.

Security review

1) Session tokens (critical)

Where: `server/server.js`, in `v1.post('/login', ...)`
Issue: Tokens are built with `Math.random().toString(36)` concatenations (predictable).
Fix:

```diff
- const token = Math.random().toString(36).slice(2) + Math.random().toString(36).slice(2);
+ const token = crypto.randomBytes(32).toString('base64url');
```

2) Rate limiting not wired for v1 (critical)

Where: `server/server.js` defines `loginLimiter` / `refreshLimiter` but never uses them.
Fix:

```diff
- v1.post('/login', (req, res) => {
+ v1.post('/login', loginLimiter, (req, res) => {

- v1.post('/refresh', requireAuth, async (req, res) => {
+ v1.post('/refresh', requireAuth, refreshLimiter, async (req, res) => {
```

3) Credentials via query string (critical)

Where: `v1.post('/login' ...)` accepts `username/password` from `req.query`.
Risk: Secrets may be logged/intercepted; caches and proxies may store them.
Fix: Remove query path entirely; only allow JSON body.

```diff
- const qUser = req.query && typeof req.query.username === 'string' ? req.query.username : undefined;
- const qPass = req.query && typeof req.query.password === 'string' ? req.query.password : undefined;
- const userIn = String(username ?? qUser ?? '').trim().toLowerCase();
- const passIn = String(password ?? qPass ?? '');
+ const userIn = String(username ?? '').trim().toLowerCase();
+ const passIn = String(password ?? '');
```

4) CORS configuration (high)

Where: `app.use(cors({ origin: true, credentials: true }))`
Risk: With credentials enabled, reflecting arbitrary origins is dangerous.
Fix: Gate by env allow-list.

```diff
- app.use(cors({ origin: true, credentials: true }));
+ const ALLOWED_ORIGINS = (process.env.ALLOWED_ORIGINS || '')
+   .split(',')
+   .map(s => s.trim())
+   .filter(Boolean);
+ app.use(cors({
+   origin: (origin, cb) => {
+     if (!origin) return cb(null, true);
+     if (ALLOWED_ORIGINS.includes(origin)) return cb(null, true);
+     return cb(new Error('Not allowed by CORS'), false);
+   },
+   credentials: true
+ }));
```

5) CSRF on cookie-based writes (high)

Where: Any mutating route when authenticating via cookie (`requireAuth`).
Fix: Add a CSRF token strategy:

- Send `Set-Cookie: csrf=<random>; SameSite=Lax; Secure`
- Require header `X-CSRF-Token` to match cookie value on `POST`/`PUT`/`PATCH`/`DELETE`.
- Alternatively, prefer Bearer tokens for all mutating calls in browsers.

6) Approval token expiry not enforced (high)

Where: `/v1/approvals/:token` (`GET` and `POST`)
Issue: `expiresAt` is returned but never validated.
Fix:

```diff
const item = db.approvals.find(a => a.tokenHash === tokenHash);
if (!item) return problem(...not_found...);
+ if (Date.now() > item.expiresAt) {
+   item.status = 'expired';
+   saveDb(db);
+   return problem(res, 410, 'approvals.expired', 'Gone', 'Token expired');
+ }
```

7) Admin-only operations (high)

Where: `/v1/overrides`
Issue: Anyone logged-in can `PUT /v1/overrides`.
Fix:

```diff
- v1.put('/overrides', requireAuth, (req, res) => {
+ v1.put('/overrides', requireAuth, (req, res) => {
+   if (req.userId !== 'admin') return problem(res, 403, 'auth.forbidden', 'Forbidden', 'Tylko administrator');
    ...
```

✅ **Admin checks already exist** for `/v1/jobs/*` and `/v1/timetable/*`, which is good.

8) Body size limits & general hardening (medium)

- Add `app.use(express.json({ limit: '64kb' }))`
- Consider `compression()` for docs and larger JSONs.
- Always validate request shapes (`zod`) and fail fast with consistent RFC7807 responses.

API design & correctness

- **Legacy `/api` vs `/v1`**
  You have a catch-all legacy handler and still a few live `/api/*` routes (`/api/attendance/summary`,

  `/api/apikeys*`). This is coherent with the comments, but confusing.

  Suggestion:

  - Keep the legacy catch-all.
  - Remove the remaining live legacy endpoints, or mark them deprecated in logs and plan removal.

- **Inconsistent “resource vs envelope” responses**
  Some endpoints return `{ ok: true, data }`, others return the raw map (e.g., `/v1/teachers`). That’s acceptable if documented, but make the convention explicit (e.g., “Simple dictionary resources return bare objects; everything else returns `{ data }`”).

- **Optimistic concurrency**
  `PATCH /v1/attendance/entries` expects `ifMatch` (string) against `entry._v`. That’s good, but the read endpoint does not expose an `ETag` header and uses a **body** field for `ifMatch`.

  Suggestion: Return `version` in each entry and use `If-Match` header (standard) or explicitly document body-level concurrency tokens to avoid confusion.

- **Rate-limit headers**
  `setRateHeaders` sets static values. Either remove them or populate from the actual limiter instance to avoid misleading clients.

- **OpenAPI**
  The `public/openapi.v1.draft.yaml` is a solid start, but it likely diverges from the actual server behavior (e.g., approvals expiry, body shapes).
  Suggestion:

  - Add examples for each route.
  - Run Spectral (`npm run docs:lint`) in CI.
  - Consider generating types for the minimal client (`public/api-client.v1.ts`) from the OpenAPI source.

Data & persistence

- **Atomic writes**
  `saveDb()` and `saveOverrides()` write synchronously to JSON. A process crash mid-write can corrupt the file.

  Fix (example):

  ```js
  import { writeFileSync } from 'node:fs';
  import { renameSync } from 'node:fs';
  function writeFileAtomic(path, data) {
    const tmp = path + '.' + process.pid + '.tmp';
    writeFileSync(tmp, data, 'utf8');
    renameSync(tmp, path);
  }
  function saveDb(db) {
    writeFileAtomic(dbPath, JSON.stringify(db, null, 2));
  }
  function saveOverrides(data) {
  ```

```
    writeFileAtomic(overridesPath, JSON.stringify(data, null, 2));
  }
```

  (You can also use a small dependency like `write-file-atomic`.)

- **Use a database**
  For multi-user attendance, approvals, API keys, and jobs, move to SQLite or Postgres. This unlocks better concurrency, transactions, and queries.

Background jobs & scraping

- **Repeated `pip install`**
  Both `/v1/jobs/timetable-scrape` and `/v1/jobs/articles-scrape` run `pip install -r requirements.txt` before each run.

  Improvements:

  - Bake scrapers into a Docker image with pinned wheels.
  - Or create a persistent venv directory (e.g., `server/.venv`) and install once; then only run scripts.
  - Add execution timeouts and memory caps; capture logs for job diagnosis.

- **Job persistence**
  `JOBS` is an in-memory Map and is lost on restart. If you care about durability, persist basic job metadata to the DB.

- **Relocate scripts**
  Move `public/scraper.py`, `public/article_scraper.py`, `public/requirements.txt` into a `tools/` folder (or a separate service). Keep only the generated artifacts (`timetable_data.json`, `articles.json`) in a non-served directory and expose them via routes.

Front-end observations

- **Good**: Lazy-loaded routes, clean modular components (e.g., news/cards/badges), reducers and hooks split per feature, Zod schemas on the client for runtime validation.

- **Tailwind via CDN**
  You include Tailwind CDN in `index.html`, but you also ship custom CSS. Using CDN means no purge/minification and larger CSS in production.

  Suggestion: Install Tailwind locally and compile via Vite so unused classes are purged. If you prefer not to, consider removing the CDN and rely on your own CSS + small utility classes.

- **Type duplication**
  You have schedule types under both `src/types/schedule.ts` and Zod schemas in `src/lib/api.ts`. Prefer a **single source of truth**: define Zod schemas and export their TypeScript types with `z.infer<...>`, or centralize shared types in `/src/types` and reference them.

- **Docs view duplication**
  `src/Docs.tsx` iframes `/docs/index.html` (from `/public/docs`), while the server also mounts `/docs` via `swagger-ui-express`. Pick one approach to avoid confusion.

- **News / article loading**
  `useArticles` fetches `/articles.json`. That's fine for a first cut. Consider pagination (or incremental rendering) if the file grows large.
- **A11y**
  Lucide icons are decorative—ensure buttons/links have appropriate `aria-label`s for actions (close dialogs, open article, next/prev, etc.). In `NewsSection` the item is a `<button>` with rich content—make sure it's keyboard focusable and has clear focus rings (you already have Tailwind classes that help).

Testing & CI

- **Playwright is in devDependencies** but there are no tests.
  Add basic E2E coverage:
    - Load Hub; open the news overlay; verify close on backdrop click.
    - Auth flow (register → login → get `/v1/users/me` → logout).
    - Attendance add/patch, including version conflict (`409`) path.
- **API unit tests**: Jest or Vitest + Supertest for each route (auth, attendance, timetable lookup, approvals).
- **Spectral lint**: Run `npm run docs:lint` in CI.
- **ESLint**: Already set; add `tsc --noEmit` to catch type issues.

Deployment & configuration

- **Vite proxy**
  You proxy `/api` and `/v1` to `http://localhost:8787` (nice DX). For production, the Express server does not serve your built `dist`. Either:
    - Serve `dist` with a static reverse proxy (Nginx), or
    - Add `app.use(express.static('dist'))` and a catch-all `app.get('*', ...)` to serve SPA.
- **Allowed hosts**
  `vite.config.ts` sets `allowedHosts` to `['.ngrok-free.app']`. Fine for dev; make it environment bound.
- **Environment variables**
  Document and load via `dotenv` in dev: `ADMIN_USER`, `ADMIN_PASS`, `PORT`, `PYTHON_PATH`, `ALLOWED_ORIGINS`.

Concrete bug list & patches

1. **Session token not secure**
   Fix: use `crypto.randomBytes` (see patch above).
2. **Rate-limit not applied**
   Fix: attach `loginLimiter` & `refreshLimiter` to `/v1` (see patch above).
3. **Login accepts query params**
   Fix: remove query param fallback (see patch above).

4. **Approvals expiry not enforced**
   Fix: add `expiresAt` checks to `GET` and `POST /v1/approvals/:token`.
5. **Overrides write open to any authenticated user**
   Fix: require admin check for `PUT /v1/overrides`.
6. **CORS reflection with credentials**
   Fix: add origin allow-list.
7. **Atomic writes**
   Fix: implement atomic `saveDb`/`saveOverrides` as above.
8. **Static rate-limit headers**
   Fix: either remove or wire to limiter state.
9. **JSON body size unlimited**
   Fix: `app.use(express.json({ limit: '64kb' }))`.
10. **Duplicate Swagger UIs**
   Fix: Keep one (prefer server-mounted for uniformity) and remove the other to avoid drift.

Streamlining & cleanups

- Remove legacy `/api/*` implementations that still exist (apikeys/summary) if not strictly needed. Keep the catch-all 410 to guide clients to `/v1`.
- Move Python toolchain and generated content out of `/public` (e.g., to `/tools` & `/data`), and expose via API endpoints. This reduces accidental exposure and clarifies boundaries.
- Split `server/server.js` into modules:
  - `routers/auth.js` (`/v1/users/me`, `/v1/login`, `/v1/logout`, `/v1/register`, `/v1/apikey*`)
  - `routers/attendance.js`
  - `routers/timetable.js`
  - `routers/approvals.js`
  - `routers/jobs.js`
  - `routers/overrides.js`
  - `routers/legacy.js`
- Promote your Zod schemas to shared validation for server and client (build once, consume everywhere).
- Introduce a small permissions helper (`isAdmin(req)`) and a scope matrix to centralize authorization decisions.

Nice touches already present

- Helmet, problem-details responses (RFC7807-like), `X-Request-Id` header, idempotency cache for POSTs, backups for timetable with rotation, attendance reducers with optimistic versioning, and a thoughtful **`resolveCanonicalId`** for timetables (good UX for room/teacher aliases). The project is already well above “prototype” quality.

Closing checklist

- ☐ Patch auth token generation
- ☐ Attach rate-limiters to **`/v1`**
- ☐ Remove query login path
- ☐ Enforce approvals expiry
- ☐ Admin-restrict **`PUT /v1/overrides`**
- ☐ Tighten CORS; add CSRF for cookie flows
- ☐ Add body size limits
- ☐ Make writes atomic
- ☐ Choose one Swagger UI path
- ☐ Move scrapers & generated files out of **`/public`**
- ☐ Add tests (E2E + unit)
- ☐ Consider SQLite/Postgres for persistence
- ☐ Refactor server into routers

If you want, I can produce a set of ready-to-apply patch files (diffs) for the P0 items in a follow-up.