orthoplex

Development Challenge

Back-end (Laravel + SQL)

Last Updated:

2025-09-21

Requirements

- **Stack:** Laravel 11+, MySQL or similar
- **Focus:** Robust auth, multi-tenancy, analytics, reliability, tests, and DX

Challenge

1. **Project Setup & DX**
   - Provide Docker (or Sail) with **MySQL**, **Redis**, and **Mailhog**; one-command bootstrap.
   - .env.example complete; **config caching** and **route caching** must not break the app.
   - Publish an **OpenAPI (Swagger) spec** for all endpoints (/docs route acceptable).
2. **Auth & Accounts**

- Keep JWT auth, but add:
  - **Email verification** (mandatory before login).
  - **2FA (TOTP)** with backup codes.
  - **Brute-force protection** (login throttling + lockout window).
  - **Passwordless "magic link"** login as an alternate flow (expires; one-time use).
  - **Idempotency keys** for POST /api/register and POST /api/login (avoid double processing).

3. **RBAC & Policies**

- Replace simple is_admin with **roles & permissions**:
  - Roles: owner, admin, member, auditor.
  - Permissions derived via Gates/Policies; include at least: users.read, users.update, users.delete, users.invite, analytics.read.
  - Support **org-level invitations** by email; accept token to join.

4. **User Lifecycle & Profiles**

- CRUD as before, plus:
  - **Soft delete** with **restore** (/api/users/{id}/restore).
  - **GDPR export** (/api/users/{id}/export) – generate a ZIP (JSON files) via queued job; email when ready; downloadable once.
  - **GDPR delete request** queue; owner/admin approval flow.

5. **Login Analytics (accurate & scalable)**
   - Maintain login_events table + **daily aggregates** table:
     - On login: write event (queued), **update users.last_login_at** and **increment users.login_count** transactionally.
     - Nightly job: roll up per-org and per-user counts to login_daily(user_id, org_id, date, count).
   - Endpoints:

- GET /api/users/top-logins?window=7d|30d (per org; from aggregates, fall back to events).
- GET /api/users/inactive?window=hour|day|week|month (org-scoped; cursor-paginated).

6. Querying & Pagination

- **Cursor-based pagination** everywhere lists appear. Stable sort keys.
- **Advanced filters** on /api/users:
  - RSQL-like syntax (e.g., name==*jo*;verified==true;created_at>=2025-01-01).
  - Server-side validation of filter AST; prevent unsafe fields.
- **Sparse fieldsets** (?fields=id,name,email) and **includes** (?include=orgs,roles).

7. Consistency & Concurrency

- Use **optimistic locking** on users via version or Eloquent's updated_at precondition.
- Document your approach to **eventual consistency** between login_events and aggregates.

8. Webhooks & Integrations

- Outbound **webhook** when:
  - user verified, user invited, user deleted/restored, login recorded (batched).
- **HMAC-SHA256 signed** with per-org secret; deliver **at-least-once** using queue + retry + DLQ.
- Inbound **"org provisioning"** endpoint that creates an org + owner from a SaaS partner, protected by API key and signature.

9. Security

- CORS, strict JSON output, no HTML in API responses.
- Rate-limit sensitive routes; **per-IP and per-user** buckets.
- **Secrets management** via env; never commit secrets.
- **Audit log** table for admin actions (who, when, what resource, old vs new snapshot).
- **Validation** via Form Requests; consistent error envelope; map exceptions.

Time Commitment

We understand this is a comprehensive challenge that goes beyond a typical coding test. We expect this project to take **6-8 hours** to complete, and we want you to approach it as you would a real-world enterprise application.

Our Philosophy:

- We value quality over speed - take the time needed to build something you're proud of.
- **Leverage the ecosystem** - Use existing packages, libraries, and open-source solutions where appropriate. We want to see your ability to integrate and architect, not reinvent the wheel.
- **Document your decisions** - Include a README explaining your approach, package choices, and trade-offs.
- **It's okay to be incomplete** - If you run out of time, prioritize core functionality and document what you would implement next.

Optional Features (Bonus)

1. **Search**: Add MySQL full-text (or Meilisearch/Scout) on users' name/email with relevance sorting.
2. **Resilience**: Implement **saga/compensation** pattern for multi-step org provisioning (create org, owner, default roles; rollback on failure).
3. **API Keys**: Org-scoped API keys with **scope restrictions** and rotation; separate from user JWTs.
4. **Rate-limit analytics**: Per-org rate metrics, surfaced in an admin endpoint.
5. **Internationalization**: Response messages localized; Accept-Language respected.
6. **K6/Gatling**: Provide a simple load test script and a baseline report.

Evaluation Criteria

1. **Functionality (40%):**
   - Does the API meet all the required functionality outlined in the challenge?
   - Is the API able to create, retrieve, update, and delete users correctly?
   - Does the API implement authentication using JWT?
2. **Code Quality (30%):**
   - Is the code well-structured, organized, and easy to read?
   - Does the code follow best practices and established coding conventions?
   - Are appropriate design patterns and separation of concerns used?
3. **Error Handling and Validation (15%):**
   - Does the API handle errors gracefully and provide meaningful error messages?
   - Does the API validate user input and prevent common security vulnerabilities?
4. **Bonus Challenges (15%):**
   - Did you attempt any of the bonus challenges?
   - How well did you implement the optional features?

orthoplex

Thank you for your interest in our company

Best of luck with this challenge!