Open Labs Share

Next-Gen Learning Platform: Microservices Meets Education

Kirill Efimovich, Aleliya Turushkina, Mikhail Trifonov, Nikita Maksimenko, Timur Salakhov, Ravil Kazeev, Kirill Shumskiy

📋 Agenda

- Problem Statement & Technical Vision 🕶
- Team intro 👨‍💻
- Demo 🎥
- Frontend Engineering & UX 🔚
- Advanced System Architecture ⛩
- Backend Microservices Deep Dive 🤿
- ML integration 🌀
- DevOps & Cloud Infrastructure ☁
- Discussion ⚔

🚨 The Problem: Skills Gap in Tech Education

Engineering challenges in education technology:

- 🍾 **Scalability bottlenecks** in traditional learning management systems
- 🚧 **Limited real-world project experience** due to academic focus on theory
- 💼 **Inefficient mentor-learner matching** without automated skill assessment
- 💌 **Poor feedback loops** between industry needs and educational content

💎 **Our engineering mission:** Build a distributed, scalable platform that efficiently connects industry experts with aspiring developers and young professionals through hands-on technical projects and education.

🔰 Meet the team

- 🌁 **Kirill Efimovich (PM/DevOps)** - *Project Leadership & DevOps Engineer*
- 🔒 **Mikhail Trifonov** - *Backend Engineer & Authentication Systems*
- 🏗️ **Nikita Maksimenko** - *Backend Engineer & API System*
- 📚 **Timur Salakhov** - *Backend Engineer & Content Systems*
- 💌 **Ravil Kazeev** - *Backend Engineer & Feedback Systems*
- 🤖 **Kirill Shumskiy** - *ML & Backend Engineer*
- 🎨 **Aleliya Turushkina** - *Designer & Frontend Engineer*

🚚 Product Vision

Open Labs Share - A modern learning platform that bridges the gap between academia and industry through hands-on technical collaboration.

We revolutionize education by:

- 📝 **Streamlined Content Creation:** Lab and Article systems enabling experts to publish and maintain high-quality learning materials
- 🎯 **Guided Learning Experience:** Structured submission and review process that provides meaningful feedback
- 🧠 **Smart Community Building:** AI-enhanced assistance for better feedback and user experience
- 🐍 **Marimo elements:** Interactive Python notebook execution with widgets for deep understanding of the material

👾 Technical Vision

Open Labs Share - A microservices-driven learning ecosystem with AI-powered assistance.

- 🏗️ **Microservices:** Separate services for labs, articles, feedback etc.
- 🚀 **gRPC:** Fast and reliable inter-service communication
- 🧠 **ML Services:** ML powered feedback and chat

Live Technical Demo: Core Features

Interactive walkthrough of platform capabilities:

1. 🔐 **Secure Authentication:** OAuth2/JWT with multi-factor authentication demo
2. 🔍 **Intelligent Lab Discovery:** ML-powered recommendations and search
3. ⚡ **Advanced Development Workflow:** Real-time collaboration and submission pipeline
4. 🧠 **Intelligent Review Engine:** AI-assisted peer matching and quality scoring
5. 📊 **Analytics Dashboard:** Real-time metrics and performance insights

ЗДЕСЬ ДОЛЖНО БЫТЬ ДЕМО

🌻 Frontend Architecture

A modern web app that lets users explore and review labs and articles through an interactive, user-friendly interface, bridging the gap between complex backend systems and user-friendly experience.

Aleliya Turushkina (Frontend Engineer)

🍄 Frontend: Main user interface for the Open Labs Share

- 📁 Handles user authentication, profile management, and navigation
- 📖 Enables users to:
  - Browse, upload, and review labs and articles
  - Participate in peer review and feedback
  - Interact with real-time features (e.g., chat, notifications)

🌷 Frontend: Tech Stack & Connections

- ☀️ **Frontend:** React, Vite, Tailwind CSS, React Router
- 🌱 **Component Libraries:** React PDF Viewer, Markdown/KaTeX
- 🌍 **API Integration:**
  - Communicates with backend via REST API through the API Gateway
  - Auth, Labs, Articles, Submissions, Feedback, and ML services
  - Real-time and file download support from MinIO

🌼 Frontend: Problems & Solutions

❌ Problems	✅ Solutions
❌ Lack of viewing of the user's submission and feedback	✅ Downloading files directly from MinIO
❌ Inability to view Markdown using a dark theme	✅ Implemented theme switching and style across the entire platform

Advanced System Architecture

Architecture designed for high availability and horizontal scaling

🔐 Authentication Service

Stateless JWT-based authentication microservice providing enterprise-grade security for the entire Open Labs Share ecosystem 🛡️

Mikhail Trifonov (Backend Engineer)

Authentication Service: Primary Use Case

Handles all authentication flows and token lifecycle management for secure access control 🔑

- 🔑 **User Authentication:** `sign-in/sign-up` with users-service gRPC calls 👤
- 🎟️ **JWT Generation:** Creates access & refresh `tokens` with user claims 🔐
- ✅ **Token Validation:** `Verifies` signatures, expiration, and blacklist status 🛡️
- 🚪 **Session Management:** Logout with token `blacklisting` for security 🏴‍☠️
- 🛡️ **Security Gateway:** Validates all API requests for `protected` resources 🦝

Authentication Service: Tech Stack & Connections

Java Spring with gRPC communication and no database 😎

- 🏗️ **Java 21 + Spring Boot 3.5:**

  ↳ `REST` controller for endpoints

- 🔐 **Spring Security + JWT:**

  ↳ Token generation with signing and validation, refresh token support

- 🚀 **gRPC Server/Client:**

  ↳ High-performance calls to Users Service and token validation for API Gateway

- 💾 **In-Memory Blacklist:**

  ↳ Storage for invalidated tokens for logout functionality

- 📚 **OpenAPI Docs:** Auto-generated REST API documentation

  ↳ Interactive API documentation for frontend integration and testing

Authentication Service: Problems & Solutions

❌ **Problems**	✅ **Solutions**
❌ Same JWT is still available after user logout	✅ Token `blacklisting` and invalidating on logout 📝
❌ User data consistency between Auth and Users Services	✅ Single `source of truth` in Users Service, Auth Service fetches on-demand and do not store any users data 👮 🤝 👨
❌ Username changes makes current JWT invalid	✅ Token `reissue` logic if that preserves user sessions seamlessly 🛂

👥 Users Service

> Single source of truth for all user data with profile management and points system 📊

Mikhail Trifonov (Backend Engineer)

Users Service: Primary Use Case

Manages all user data, credentials, and points for solving & reviewing labs 🎯

- 🆕 **User Registration:** Creates new user accounts 👤
- 🔐 **Credential Management:** Stores bcrypt-hashed passwords, validates username/email and password
- 👤 **Profile Operations:** CRUD for user profiles ✏️
- 🏆 **Points System:** Tracks labs solved/reviewed counts & points balance 💵💰💳
- 📈 **Data Integrity:** Single source of truth for all user-related information 🐟

Users Service: Tech Stack & Connections

Java with PostgreSQL persistence and gRPC API ☕🐘

- 🔧 **Java 21 + Spring Boot 3.5:**

  ↳ `REST` controllers and JPA repositories for user management

- 🗄 **PostgreSQL:**

  ↳ Stores user data, credentials, points, and labs solved/reviewed counts

- 📋 **Flyway:**

  ↳ Database schema versioning and `migration` management

- ⚡ **gRPC Server:**

  ↳ Provides `API` for user validation, data retrieval, and points updates to other microservices

Users Service: Problems & Solutions

❌ Problems	✅ Solutions
❌ Create-drop strategy in ORM caused inconsistency when all containers restarted	✅ Flyway for SQL tables creation instead of auto-creation by ORM. Validate strategy 🦜🦅🐦‍⬛🐦
❌ Points system requiring strict control on changes due to its "*money*" purpose	✅ Transactional methods to prevent inconsistency in balance and counters 🤑

📥 API Gateway 📤

> The API Gateway centrally coordinates frontend REST API requests, executes business logic, and routes them to backend microservices via gRPC

Nikita Maksimenko (Backend Engineer)

API Gateway: Primary Use Case

Centralized entry point and request orchestration for all client interactions 🌐

- 🌐 **Centralized Entry Point:** Serves as the unified access layer for all client REST API requests
- 🔀 **Request Routing:** Directs incoming requests to the appropriate microservice (`auth`, `user`, `article`, `lab`) via gRPC
- 🔒 **Authentication & Security:** Validates JWT tokens and user's permissions
- 📝 **Cross-Cutting Concerns:** Handles logging, request tracing, and error handling for all API traffic
- 🧠 **Business Logic Execution:** Aggregating data and enforcing business rules beyond simple routing

API Gateway: Tech Stack & Connections

Java Spring Boot with REST-to-gRPC translation ☕🔄

- 📥 **REST API:** Receive data from frontend via REST

  ↳ REST is the simplest and most widely supported method for web communication

- 🛡️ **Security Layer:** Intercept incoming REST requests for authentication and authorization

  ↳ Ensures secure access and centralized permission checks

- 🔀 **gRPC Client:** Route requests internally to backend microservices via gRPC

  ↳ gRPC provides high-speed, type-safe, and scalable service-to-service communication

- 📤 **Response Handling:** Return responses to the client through the API Gateway

  ↳ Centralizes response handling and error management

- 🧑‍💻 **Technology Stack:** Java 21, Spring Boot 3 (Web, AOP, Doc OpenAPI), gRPC

  ↳ Ensures a secure, efficient, and maintainable technology stack for all platform components

API Gateway: Problems & Solutions

❌ Problems	✅ Solutions
❌ Too many services and people to communicate with	✅ Create clear rules of communication and define issue execution order for efficient collaboration
❌ Unclear models from both frontend and backend	✅ Establish detailed requirements for each request step to ensure consistency and clarity
❌ Lack of data checks on frontend	✅ Use Jackson validators in request models to enforce data integrity before processing

📚 Articles Service

The central repository for all scientific articles and research papers, enabling authors to publish content and students to access educational materials.

Timur Salakhov (Backend Engineer)

Articles Service: Primary Use Case

Manages all articles & assets metadata 🗄️

- 📝 **Articles Operations:** Provides CRUD for articles details
- 🗂️ **Content Management:** Handles articles assets in independent storage system
- ⚙️ **Metadata Management:** Organizes and updates metadata for articles and its assets
- 🔍 **Searching:** Provides articles searching based on its title and abstract

Articles Service: Tech Stack & Connections

Python-based microservice with PostgreSQL and MinIO storage 🐍

- 🐍 **Programming Language:** Python 3.12
- 🔄 **Inter-service Communication:** gRPC

Service Integrations:

- 🚪 **API Gateway:** Receive and return data in gRPC format
- 🗄️ **PostgreSQL Database:** Store all articles and its assets metadata
- ☁️ **MinIO Storage System:** Store all articles assets

Articles Service: Problems & Solutions

❌ Problems	✅ Solutions
❌ Difficult interaction with database via SQL queries	✅ Use `SQLAlchemy` ORM system for convenient and flexible database interaction
❌ Need to organize article files systematically	✅ Created structured MinIO bucket organization: `articles/article_id/article.pdf`
❌ Large files causing timeout issues during upload	✅ Implemented streaming gRPC uploads for efficient file transfer
❌ Frontend does searching across articles	✅ Moved searching on service and built text search functionality on articles with pagination

📚 Labs Service

The central repository for all laboratory work and student submissions, enabling teachers to create assignments and students to submit solutions with comprehensive grading and feedback.

Timur Salakhov (Backend Engineer)

Labs Service: Primary Use Case

Manages all labs, submissions & educational content 🗄️

- 📚 **Labs Operations:** Provides CRUD for lab assignments with tags
- 📤 **Submissions Management:** Handles submissions with text content and file assets
- 🏷️ **Tag System:** Organizes labs with flexible tagging and search capabilities
- 📊 **Grading System:** Tracks submission status and grade workflow

Labs Service: Tech Stack & Connections

Python with hybrid database architecture and MinIO storage 🐍

- 🐍 **Programming Language:** Python 3.12
- 🔄 **Inter-service Communication:** gRPC

Service Integrations:

- 🚪 **API Gateway:** Single entry point for all requests
- 🗄️ **PostgreSQL Database:** Store labs, submissions, tags, and assets metadata
- 📄 **MongoDB Database:** Store submission text content for flexible storage
- ☁️ **MinIO Storage System:** Store lab and submission assets in organized buckets

Labs Service: Problems & Solutions

❌ Problems	✅ Solutions
❌ Complex data relationships between labs, submissions, and tags	✅ Used `SQLAlchemy` models with proper foreign keys and many-to-many relationships for structured data management
❌ Large submission text content causing database bloat	✅ Implemented hybrid storage: metadata in PostgreSQL, text content in MongoDB for flexibility
❌ Need to organize files systematically	✅ Created structured MinIO bucket organization: `labs/lab_id/` and `submissions/submission_id/`

💬 Feedback Service

The Feedback Service centralizes feedback and discussion by managing detailed lab reviews, supporting file attachments, and powering threaded conversations for both labs and articles

Ravil Kazeev (Backend Engineer)

Feedback Service: Primary Use Case

Comprehensive feedback and discussion management system 💬

- 📝 **Comprehensive Feedback System:** Enables reviewers to create, update, and delete detailed feedback on submissions using Markdown for text and code formatting
- 💬 **Organized Discussion Section:** Powers a threaded commenting system for both labs and articles. Nested replies keep conversations structured and easy to follow
- 📎 **Attachment Handling:** Allows multiple file attachments per feedback entry, using efficient gRPC streaming to handle large uploads and downloads without high memory usage

Feedback Service: Tech Stack & Connections

Go with a multi-storage backend and gRPC API 🐹💾

- 🐹 **Go 1.24:**

  ↳ High-performance, concurrent service ideal for I/O-heavy tasks

- 🗣️ **gRPC Server:**

  ↳ Provides a typed API for feedback, comments, and file streaming

- 🗄️ **Multi-Storage Backend:**

  ↳ **PostgreSQL:** Stores structured feedback metadata

  ↳ **MongoDB:** Stores unstructured comments and feedback content

  ↳ **MinIO:** Object storage for all file attachments

Feedback Service: Problems & Solutions

❌ Problems	✅ Solutions
❌ A single database was inefficient for managing varied data types (metadata, text, and files).	✅ Implemented a **multi-storage architecture**, using the best database for each job: PostgreSQL for metadata, MongoDB for comments, and MinIO for file attachments.
❌ Uploading large files as a single request was unreliable, leading to timeouts and memory errors.	✅ Re-architected attachment handling using **gRPC streaming**, which processes files in small chunks for efficient and robust transfers.
❌ File downloads through the Feedback service would create a bottleneck	✅ Configured the MinIO bucket for **public read access**, allowing the service to provide direct file URLs to the frontend and offload all download traffic.

📓 Marimo Service

> Dual-architecture microservice providing real-time interactive Python notebook execution powered by Marimo library 🐍🟢

Mikhail Trifonov (Backend Engineer)

Marimo Service: Primary Use Case

Interactive code execution and data visualization through cells with Python code 🔬

- 📝 **Notebook Management:** CRUD operations for marimo components linked to labs/articles 🔗
- ⏰ **Session Orchestration:** Start/stop interactive Python sessions with TTL 🪦
- 👟 **Code Execution:** Real-time cell execution with output capture and error handling 🖐
- 📊 **Asset Management:** Upload/download datasets and files for notebook use 🐫
- 🎛 **Interactive Widgets:** Set of basic Marimo input widgets which value can be used in code (sliders, switchers, text fields, etc.) 📟
- 📁 **Cross-cells state memory:** Variables and modules from executed cells are available in other cells 📦

Marimo Service: Tech Stack & Connections

Java for metadata management with Python native code execution ☕🐍

- 🔧 **Java Manager + Python Executor:**

  ↳ Java handles `REST API` and `metadata` while Python `executes` notebooks

- 🗄 **PostgreSQL:**

  ↳ Tracks notebook metadata, user sessions, and execution trails with TTL cleanup

- 📦 **MinIO:**

  ↳ Object storage for notebook `files` and user-uploaded `assets`

- 🔗 **gRPC:**

  ↳ Java Manager ← `execute requests, session management` → Python Executor

- 🐍 **Marimo:** Interactive notebook execution with widgets

  ↳ Interactive notebook execution with ✨ `widgets` ✨

Marimo Service: Problems & Solutions

❌ Problems	✅ Solutions
❌ High load on one service to manage metadata, connections with other services, and execution at the same time	✅ Dual-service architecture for management from execution 2️⃣ ✌️
❌ Managing variables and modules across multiple code cells	✅ Sessions for notebooks to track existing and erased variables/modules 🧹
❌ Marimo widgets incompatibility with our needs and tech	✅ Custom design widgets (but based on Marimo widgets) with configurable behaviour fully under our control 🧩 🕹️

🤖 ML Service

Advanced AI-powered features providing intelligent assistance and automated grading capabilities for enhanced learning experience

Kirill Shumskiy (ML Engineer)

ML Service: Primary Use Case

Two powerful AI enhancements for the learning platform 🧠

- 🔍 **AI RAG Assistant:** Context-aware code and documentation helper, leveraging Retrieval-Augmented Generation (RAG) to deliver accurate, real-time support to students
- ✅ **Autograding:** Automated code assessment system for evaluating submissions instantly — ideal for learning platforms

ML Service: Tech Stack & Connections

FastAPI backend with specialized AI models and infrastructure 🐍🤖

🤖 AI RAG Assistant

- Qwen2.5-Coder-1.5B-Instruct (local inference)
- Qdrant vector store
- BAAI/bge-small-en-v1.5 embeddings

✅ Autograding

- deepseek-r1-distill-llama-70b (groq inference)
- Menagerie dataset: Graded CS1 Assignments for evaluation

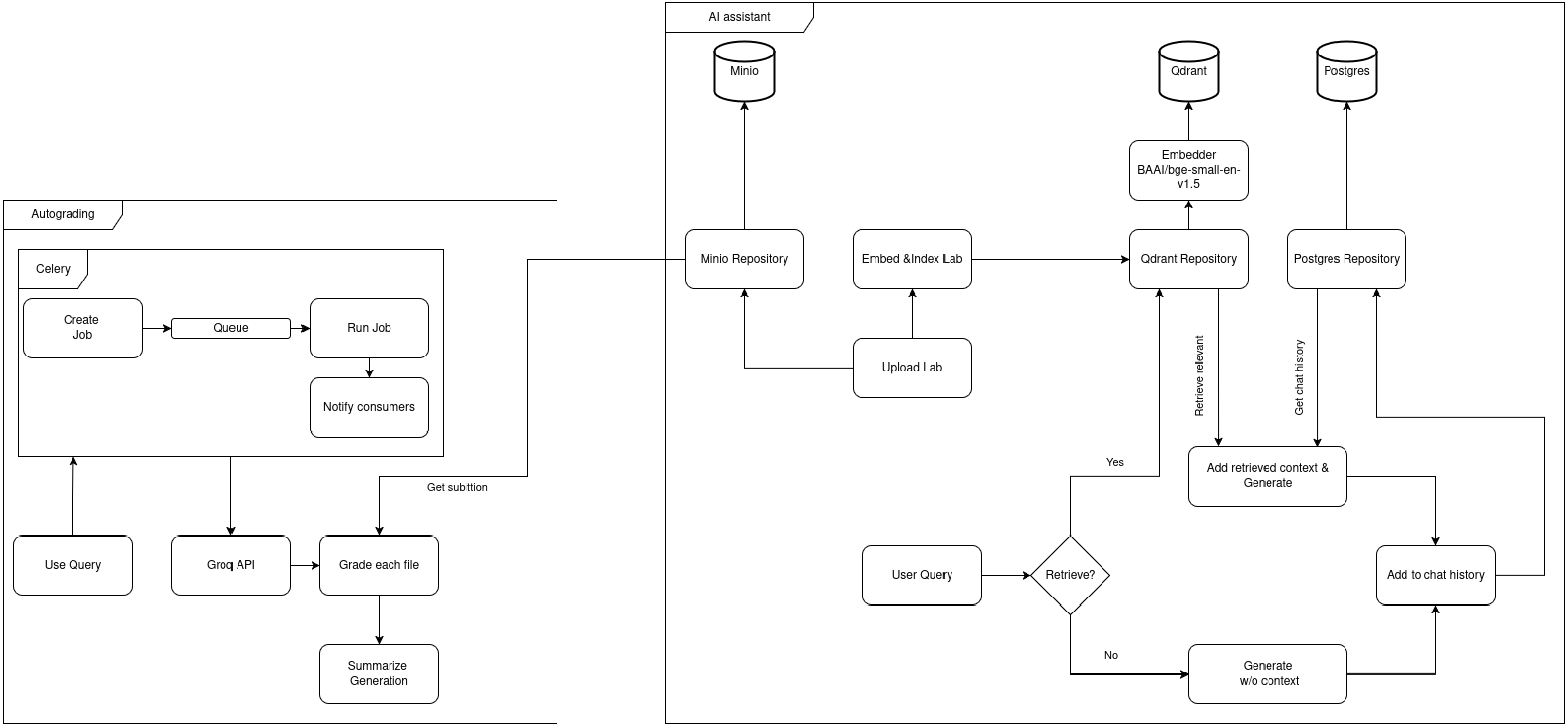
Core Architecture:

🐍 **FastAPI-based backend** with three-layer structure

🥬 **Celery** for asynchronous tasks

🛢️ **Redis** for caching and message broker

ML Service: Infrastructure



🏙️ DevOps & Infrastructure

From manual processes to a fully automated pipeline. Robust DevOps foundation to enable rapid development, consistent testing, and reliable, zero-downtime deployments for our platform.

Kirill Efimovich (DevOps Engineer)

🏛️ DevOps: Primary Use Case

Automated deployment pipeline and infrastructure management 🌆

- 🤖 **Accelerate Delivery:** Fully automate the build, test, and deployment lifecycle
- 📦 **Ensure Stability:** Create reproducible environments with Docker for development and production
- 🧰 **Team help tools** to automate issues managing and PR notifiers to keep the team perfectly synchronized

🛣️ DevOps: Tech Stack & Connections

Key GitHub Actions Workflows: 💫

- 🔧 **Compilation Validation:** Ensures all services compile
- 🏏 **Test Execution:** Runs unit & integration tests
- 🐳 **Docker Build Validation:** Buillds, validates and pushes images to GHCR
- ✈️ **Deployment Automation:** Handles the Blue-Green deployment logic

🛤️ DevOps: Infrastructure

🔵 Green-Blue Strategy 🟢

- 0️⃣ **Zero Downtime:** Updates are seamless
- 🎞️ **Workflow:**
    i. Deploy new version (Green) alongside Production (Blue)
    ii. Test Green environment internally
    iii. Switch HAProxy to route traffic to Green
    iv. Keep Blue for instant rollback

🐧 Server & Networking

- **Host:** Self-managed server on Ubuntu 24.04
- **Specs:** 6-Core CPU, 16GB RAM, 240GB SSD
- **Proxy:** NGINX & HAProxy
- **Access:** CloudPub for public NAT traversal
- **Monitoring:** cAdvisor for container metrics

🏚️ DevOps: Problems & Solutions

❌ Problems	✅ Solutions
❌ University network NAT blocked external access to our self-hosted server.	✅ After issues with Cloudflare, we successfully used **CloudPub** to create a secure tunnel for public access.
❌ The initial CI/CD pipeline was complex and required many iterations to stabilize.	✅ Through persistent, collaborative effort, we developed a set of reliable, modular GitHub Actions workflows.
❌ Risk of downtime during manual deployments.	✅ We fully automated the deployment process and are implementing a **Blue-Green strategy** to ensure zero-downtime updates.

Thank you!

We're glad to hear your questions! 🛒 🤗 🎸