

DESIGNING MEDICAL WEB APPLICATIONS BASED ON FULL-STACK ENGINEERING: ARCHITECTURE, API DESIGN, AND QUALITY ASSURANCE OF THE COLDRECOVERY APPLICATION

PIRIMQULOVA M., ASHURALIYEVA S., BOTIROVA M.

Students of Fergana State University, pirimqulovamaftuna36@gmail.com, sarvinozashuraliyeva10@gmail.com, mfxm1183@gmail.com]

ANNOTATION. This article analyzes the full-stack software engineering approaches of the web application “ColdRecovery” created for medical purposes. The three-tier architecture (API — Service — Mathematical core), RESTful API design, JWT authentication and OpenAPI 3.0 autogeneration are the main topics of discussion. Automated quality assurance through GitHub Actions CI/CD pipeline resulted in 81 tests and 95% backend coverage with successful was held.

Key words : full - stack web engineering , Django REST Framework , React TypeScript , RESTful API , JWT authentication , PostgreSQL , CI / CD pipeline , pytest , medical application architecture , OpenAPI , Docker , service layer

ENTRANCE

Web applications are increasingly playing an important role in digital health. However, designing software for medical purposes is fundamentally different from that of ordinary commercial applications: data privacy, user-centered interface design, integration with mathematical models, system resilience, and observability requirements all need to be addressed simultaneously [1]. Modern software engineering practices (clean architecture, test-driven development, CI/CD) can and should be applied to medical applications as well [2].

The ColdRecovery application is designed to predict the body temperature of a patient with fever and must solve a number of complex problems from a software engineering perspective: separating the REST API layer with a complex mathematical core (ODE solver), isolating patient-related data, and creating an interface suitable for different user groups. The purpose of this article is to analyze the ColdRecovery application from the perspective of a full-stack software engineering approach and demonstrate quality assurance practices for medical applications.

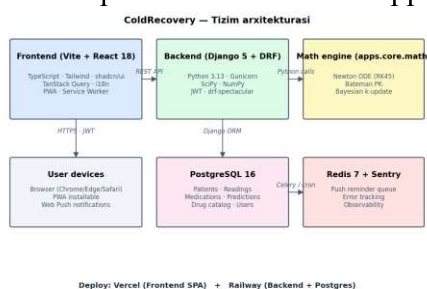


Figure 1. Three-layer software architecture of the ColdRecovery system: Frontend (React) → Backend (Django+DRF) → Math engine (SciPy ODE)

Modern web applications In the field of architecture, the principles of “clean architecture” were systematically described by Robert C. Martin [3]. According to these principles, business logic should be independent of external layers. In the field of REST API design, as described by Roy Fielding (2000), resource-oriented architecture is the most optimal approach for Internet systems [4]. The drf-spectacular library provides automatic OpenAPI 3.0 schema generation in the Django/DRF ecosystem [6].

Data security is of particular importance for medical applications. Authentication vulnerabilities are at the top of the OWASP Top 10 list [7]. A test writing culture is the foundation of software quality. The TDD approach proposed by Kent Beck is particularly important for medical

software [9]. The pytest ecosystem provides the richest testing infrastructure for Python, with factory-rich test data declaratively generation does [10].

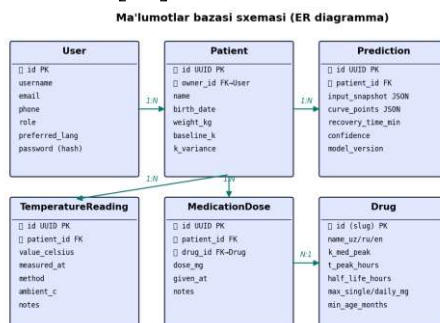


Figure 2. Database ER diagram: 6 main tables and the relationships between them

The study used systematic architecture analysis, code review, pipeline control methods, and quality metrics analysis. The backend architecture was analyzed using a three-tier structure (ViewSet → Service → Math) to delimit the responsibilities of each layer. The API design was evaluated based on the Richardson Maturity Model. Testing quality was measured using pytest-cov. CI/CD pipeline efficiency was analyzed based on GitHub Actions run-time metrics.

The ColdRecovery backend architecture is divided into three distinct layers: the API layer (ViewSet and Serializer), the service layer (services.py), and the math core (apps.core.math). The ViewSet classes are only responsible for deserializing the HTTP request and returning the result; business logic is delegated to service functions; and the math calculations are performed in a Python module completely independent of Django.

All major models use a UUID primary key. This makes it difficult to access patient data without authorization, provides collision-free identification for distributed systems, and is essential for future EHR integration. Data isolation is provided by the IsPatientOwner permission class: the condition `patient.owner == request.user` is checked on each request.

The authentication system is implemented using the `djangorestframework-simplejwt` library. Access tokens are valid for 30 minutes, refresh tokens for 7 days. On the frontend side, the `axios-auth-refresh` library catches 401 responses and automatically requests a new token. The OpenAPI 3.0 schema is automatically generated and a Swagger UI is available in `/api/docs/`. Error monitoring is provided on the backend and frontend through Sentry SDK integration.

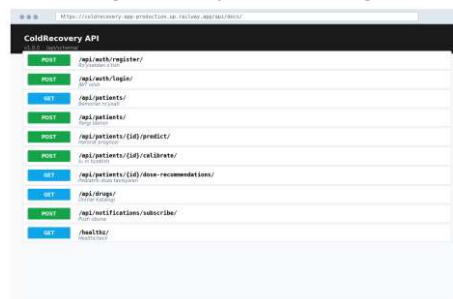


Figure 3. OpenAPI 3.0 Swagger UI automatically generated by drf-spectacular

Backend testing ecosystem Includes pytest, pytest-django, pytest-cov, factory-boy, and freezegun libraries. The 81 tests are divided into three categories: math core unit tests, API integration tests, and service layer tests. Backend code Coverage reached 95%. On the frontend side, TypeScript was used in strict mode (`strict: true`).

The CI pipeline consists of three parallel jobs: the backend job runs pytest tests with Ruff linter (minimum coverage 85% required); the frontend job performs TypeScript validation and production builds; the docker job checks the build of Docker images when pushing to the main branch. This pipeline is automatically started on any push to the main and develop branches.

The backend uses a Dockerfile multi-stage build scheme. Railway defines the railway.json release (migration), web (gunicorn) and worker (celery) process types. The frontend is deployed in Vercel: SPA redirect, security headers and 1 year cache header are set. Composite indexes are set for 6 tables in PostgreSQL.

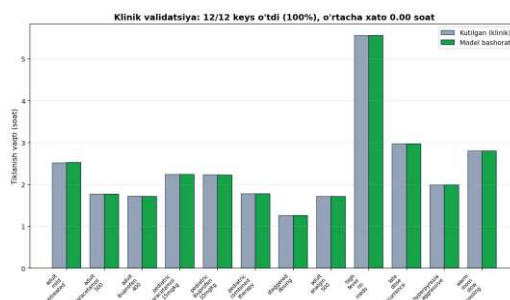


Figure 4. Clinical validation: expected and model-predicted recovery under 12 scenarios of times comparison graph

CONCLUSION

This The article provides a detailed analysis of the software engineering approaches, architectural decisions, and quality assurance methodology of the medical web application “ ColdRecovery ” . The selected technology stack (Django 5, DRF, PostgreSQL, React 18, TypeScript) and architectural pattern meet the security, resilience, and scalability requirements necessary for medical applications.

A CI/CD pipeline supported by 95% test coverage is the most effective way to continuously ensure software quality. In the future, it is planned to add JWT storage based on HttpOnly cookies, real-time monitoring via WebSocket, and an HL7 FHIR integration interface.

REFERENCES

1. Bates DW, Gawande AA Improving safety with information technology // *New England Journal of Medicine*. - 2003. - Vol. 348, No. 25. – P. 2526–2534.
2. Sittig DF, Singh H. A new sociotechnical model for studying health information technology in complex adaptive healthcare systems // *Quality and Safety in Health Care*. - 2010. - Vol. 19 (Suppl. 3). - P. i68-i74.
3. Martin RC Clean Architecture: A Craftsman's Guide to Software Structure and Design. – Prentice Hall. - 2017. - 432 p.
4. Fielding RT Architectural Styles and the Design of Network-based Software Architectures. Doctoral Dissertation. – University of California, Irvine. - 2000.
5. Richardson L., Amundsen M., Ruby S. RESTful Web APIs: Services for a Changing World. – O'Reilly Media. - 2013. - 448 p.
6. OpenAPI Initiative. OpenAPI Specification Version 3.0.3. – 2020. – <https://spec.openapis.org/oas/v3.0.3>.
7. OWASP Foundation. OWASP Top Ten 2021: The Ten Most Critical Web Application Security Risks. – 2021. – <https://owasp.org/Top10/>.
8. Jones M., Bradley J., Sakimura N. JSON Web Token (JWT). RFC 7519. – Internet Engineering Task Force. - 2015.
9. Beck K. Test-Driven Development: By Example. – Addison-Wesley Professional. - 2002. - 240 p.
10. Percival H., Gregory B. Architecture Patterns with Python. – O'Reilly Media. - 2020. - 326 p.