MIOE
IDEAS TO IMPACT

YAPAY ZEKA • MÜHENDİSLİK • GERÇEK DEĞER

AI Çağında Uzman Yazılım Mühendislerinin Elde Ettiği İlerleme ve Sonuçlar

Getrac, yapay zekâyı doğru bir mühendislik disipliniyle birleştiren uzman bir ekibin, gerçek dünyada nasıl ölçeklenebilir ve güvenli bir ürün ortaya çıkardığının somut örneği.



Daha Hızlı Geliştirme
AI destekli üretkenlik artışı

Daha Güvenli Sistem
Test, güvenlik ve çok kiracılı mimari

Daha Yüksek Kalite
Otomatik test ve regresyon koruması

Daha Büyük Etki
Gerçek müşteri değeri ve ölçeklenebilirlik

Getrac Nedir?

Getrac, işletmelerin müşteri, teklif, sipariş, gönderim, finans ve operasyonlarını tek platformda yöneten, çok kiracılı (multi-tenant) bir iş uygulamasıdır. Gerçek dünya ihtiyaçları için geliştirilen, güvenli, ölçeklenebilir ve entegre bir ekosisteme sahiptir.

Zengin Entegrasyon Ekosistemi

Zoho · DIA · Google Workspace · Microsoft 365 · WhatsApp · Netgsm · iyzico · OpenAI / Anthropic · ve daha fazlası

Modern ve Güvenli Mimari

- React Tabanlı Web Uygulaması
- Cloudflare + Nginx
- Fastify API (Modüler Monolit)
- PostgreSQL + Row Level Security (RLS)
- Entegrasyon Adaptörleri
- Arka Plan İşleri (Scheduler, Poller, Outbox)

Mühendislik Disiplini

- Net geliştirme politikaları
- Migration ve veri yönetimi
- Güvenlik ve tenant izolasyonu
- Otomatik testler (unit, integration, e2e)
- CI/CD ve önizleme ortamları
- Dokümantasyon ve alan notları

Somut Sonuçlar

- **644 commit** test branch'inde main'e göre ileride
- **70+ API modülü** modüler ve sürdürülebilir yapı
- **154+ kütüphane modülü** yeniden kullanılabilir bileşenler
- **Yüzlerce otomatik test** gerçek PostgreSQL ile tenant izolasyonu
- **Üretim ortamında canlı kullanım** İstanbul Local Zone'da güvenli altyapı

Öne Çıkan Kazanımlar

- **Daha Hızlı Ürün Geliştirme** AI destekli analiz, tasarım ve kodlama
- **Daha Az Hata** Otomatik testler ve regresyon koruması
- **Daha Yüksek Güvenlik** PostgreSQL RLS ve tenant izolasyonu
- **Daha İyi Dokümantasyon** AI destekli, güncel ve kapsamlı dokümanlar
- **Daha Yüksek İş Değeri** Müşteriye daha hızlı ve kaliteli çözümler

AI Destekli Geliştirme Nasıl İşliyor?

1. Net kurallar ve politika (Claude.md, skills, alan notları)
2. AI destekli analiz ve tasarım
3. Kod, test ve dokümantasyon üretimi
4. Otomatik doğrulamalar (CI)
5. İnceleme ve kalite kontrol
6. Staging / Prod'a güvenli geçiş
7. Üretimden öğrenim, yeni iyileştirmeler

MIOE
IDEAS TO IMPACT

İnsan uzmanlığı × Yapay zeka gücü = **Daha büyük etki**

Daha akıllı ürünler.
Daha güçlü işletmeler.
Daha iyi bir gelecek.

MIOE

AI ÇAĞINDA YAZILIM GELİŞTİRME

Deneyimli Yazılım Mühendisleri ile Daha Hızlı, Daha Güvenli, Daha Büyük Etki

Getrac, deneyimli yazılım mühendislerinin AI araçlarını doğru bir geliştirme politikası ve güçlü mühendislik disiplini ile kullanarak neler başarabileceğinin somut bir örneği.



Hızlandırılmış Geliştirme

AI destekli geliştirme ile fikirden üretime çok daha kısa sürede.

70 API modülü

154 kütüphane modülü

Güçlü Güvenlik

Çok katmanlı **tenant** izolasyonu ve testlerle güvenli büyüme.

PostgreSQL Row Level Security (RLS)

Otomatik tenant testleri

CI ile güvenlik kontrolleri

Gerçek Kurumsal Ölçek

Modüler monolit mimari ile sade ama güçlü bir yapı.

React SPA + Fastify API

PostgreSQL + RLS

70+ iş senaryosu

10+ harici entegrasyon (Zoho, DİA, Google, Microsoft, WhatsApp, iyzico, OpenAI, vs.)

Mühendislik Disiplini

AI araçları, net geliştirme politikası ve kapsamlı test stratejisi ile.

- Katmanlı test yapısı
- Veritabanı migration disiplini
- CI/CD ile otomasyon
- Gerçek hata örneklerinden kalıcı kurallar

Somut Sonuçlar

Deneyimli **mühendisler** + AI araçları = yüksek etki.

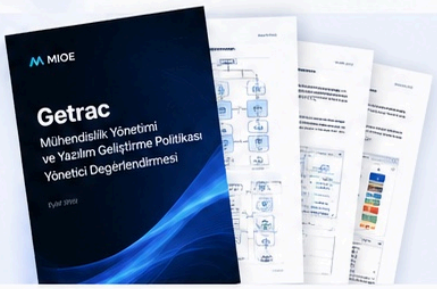
644 commit ilerleme (test → main)

9 sayfalık yönetici raporu

90 günlük yol haritası

Detaylı Değerlendirme: Getrac Mühendislik Yönetimi Raporu

Getrac'ın mimari yapısı, geliştirme süreçleri, güvenlik yaklaşımı, test stratejisi ve AI native geliştirme modeli üzerine hazırladığımız kapsamlı yönetici değerlendirmesini paylaşıyoruz.



Raporu İnceleyin (PDF)

Gerçek problemler
Gerçek çözümler
Daha iyi bir gelecek

MIOE | AI ile daha fazla değer üreten, daha güçlü yazılım ekipleri.

ENGINEERING GOVERNANCE

& SOFTWARE DEVELOPMENT POLICY

Executive Assessment

A C-level review of how getrac governs software change, protects tenant data, verifies quality, and moves code from idea to production.

STRONG	STRONG	GAP	GAP
Engineering discipline	Policy-as-code	Merge governance	Review independence

Prepared from the current repository policy, CI/CD workflows, branch rules, tests and deployment controls on the test integration branch.

Assessment baseline: mioe-tr/getrac · test branch · 21 September 2026

Executive summary

getrac demonstrates a mature engineering culture inside the repository. The project systematically converts operational failures and domain rules into executable tests, migration checks, tenant-isolation controls and deployment verification. Its principal weakness is not software craftsmanship; it is governance at the GitHub merge boundary.

BOARD-LEVEL CONCLUSION

The SDLC is stronger than the repository governance that protects it.

The next maturity step is to formalize approvals, required checks, ownership and release gates around the engineering controls already in place.

01 Policy is executable

Many documented rules are backed by CI tests rather than relying on developer memory.

02 Tenant safety is defense-in-depth

Application org filtering is reinforced by PostgreSQL RLS and restricted-role integration tests.

03 GitHub controls lag the SDLC

Main is protected from deletion/non-fast-forward changes, but required PR approvals and status checks are not enforced by the current ruleset.

04 Independent review is limited

Recent PRs can be deeply documented and heavily tested, yet still have no requested human reviewer.

What leadership should decide

- Whether test and main should become protected promotion boundaries with mandatory successful checks.
- Which change categories require independent approval: security, tenancy, finance, infrastructure, privacy and destructive data operations.
- Whether the repository's AI-oriented operating rules should be promoted into an official human-readable engineering policy.

1 Current software delivery operating model

The repository implements a pragmatic integration-branch model. Feature and fix branches flow into test, which is deployed to staging; main represents production. The process is reinforced by CI, preview environments and smoke tests.

FEATURE / FIX → TEST → STAGING → MAIN / PROD

Control model

Control layer	What it does	Observed strength	Executive implication
Repository policy	Global engineering rules, domain notes, task-specific skills	High	Institutional knowledge is captured close to the code.
Automated tests	Unit, integration, RLS, migration, E2E and policy regression gates	High	Many critical rules are executable and repeatable.
CI/CD	Build, checks, isolated previews, staging deployment, smoke verification	High	Delivery quality is automated beyond basic compilation.
GitHub governance	Branch rules, required reviews, required checks	Low–Medium	Documented process can still be bypassed by sufficiently privileged users.

Policy hierarchy observed

CLAUDE.md Global engineering handbook and operating constraints.

.claude/skills/* Task-specific execution checklists for DB, UI and AI-related work.

docs/alan-notlari/* Domain-specific rationale, edge cases and historical lessons.

tests + CI Executable policy enforcement.

GitHub rules Outer governance boundary - currently narrower than the internal SDLC.

2 Where the engineering policy is strong

Regression knowledge becomes permanent

Production and UX failures are repeatedly turned into named automated gates: tenant-join checks, transaction-context checks, send-confirmation tests, modal layering tests, embedded tenant-data tests and deployment smoke verification.

Tenant isolation is treated as a system property

Tenant filters exist at application level, while PostgreSQL RLS adds a second boundary. Tests use a NOSUPERUSER/NOBYPASSRLS role so they cannot accidentally pass by bypassing RLS.

Database changes follow explicit discipline

Raw parameterized SQL is intentional. Migrations, transaction context, RLS activation order and schema documentation are treated as controlled engineering concerns rather than incidental implementation details.

Deployment verifies reality, not intent

After deployment, the environment is smoke-tested on the host. This addresses the common failure mode where a workflow reports green while the service failed to start or a migration failed.

External and destructive actions have safety semantics

Customer-facing sends require preview and explicit confirmation; destructive deletes require impact visibility and explicit consent. These are backed by regression gates.

3 Governance gaps and enterprise risk

The dominant risks are process-governance risks, not evidence of weak implementation quality. They matter because an organization should not depend on every contributor voluntarily following a process that the platform could enforce.

Risk	Likelihood	Impact	Current state	Management response
Merge without required quality gates	Medium	High	CI exists, but the current main ruleset does not require successful checks or PR approval.	Make checks and PR review mandatory on promotion branches.
Correlated AI error	Medium	High	AI can produce code, tests, docs and rationale within the same change.	Require independent review for high-impact domains.
Policy fragmentation	Medium	Medium	Critical guidance is distributed across CLAUDE.md, skills and domain notes.	Create one canonical Development Policy + explicit hierarchy.
Documentation drift	High	Medium	Some older architecture documents conflict with current RLS implementation.	Classify docs as authoritative / generated / guidance / historical and gate freshness.
Large promotion batches	Medium	High	test can diverge substantially from main before production promotion.	Reduce release batch size; use release manifests/tags and explicit promotion criteria.

Critical governance observation

GitHub currently protects main against deletion and non-fast-forward updates, but the observed active ruleset does not itself require reviews or passing status checks. No CODEOWNERS or pull-request template was present on the reviewed branch.

4 Executive maturity view

The assessment below is qualitative. It describes maturity of the operating system around software delivery, not developer performance or product quality.

Dimension	Observed maturity	Evidence	Next threshold
Engineering standards	Advanced	Detailed global/domain rules + recurring regression gates	Make policy canonical and tool-neutral.
Quality assurance	Advanced	Large unit/integration suites, RLS checks, E2E, smoke tests	Risk-based test ownership and measurable coverage targets.
Data / tenant safety	Advanced	RLS + application filtering + restricted-role testing	Formal security review for bypass paths and privileged operations.
CI/CD	Advanced	Preview/staging automation and host-level smoke verification	Formal release promotion and rollback controls.
Change governance	Developing	PR-centered workflow, but no observed mandatory review/check rules	Required checks, approvals, CODEOWNERS and protected test/main.
Documentation governance	Developing–Established	Generated docs exist; some stale narrative docs remain	Authoritative hierarchy + freshness ownership.
Independent assurance	Developing	Deep self-verification; limited reviewer assignment	Second-person or independent agent review on critical change classes.

Interpretation: getrac already behaves like an organization with strong engineering controls. The missing layer is formal organizational governance around merge authority, approval independence and release promotion.

5 Recommended 90-day governance roadmap

Horizon	Action	Outcome	Owner class
0–30 days	Protect test and main with required PRs, successful status checks and resolved conversations.	The documented SDLC becomes non-bypassable by default.	CTO / Repo Admin
0–30 days	Introduce PR template: scope, risk, migration, tenant impact, tests, rollback, documentation.	Consistent change evidence and faster review.	Engineering Lead
0–30 days	Create CODEOWNERS for security, DB/migrations, infra, billing/accounting and privacy-sensitive paths.	Critical changes automatically reach the right reviewer.	CTO / Domain Owners
31–60 days	Publish docs/development-policy.md as the canonical SDLC policy; make CLAUDE.md an execution companion.	Policy becomes human- and AI-neutral.	Engineering Lead
31–60 days	Classify documentation as authoritative, generated, guidance or historical; add freshness checks where feasible.	Contradictory architecture guidance becomes detectable.	Architecture Owner
31–60 days	Define high-risk changes that require independent approval.	Reduced correlated-error risk, especially for AI-assisted development.	CTO / Security
61–90 days	Formalize test→main promotion: release manifest/tag, rollback plan, deployment evidence and approval.	Production release becomes auditable and repeatable.	Release Owner
61–90 days	Track governance metrics: bypasses, failed promotions, escaped defects, review latency, rollback frequency.	Leadership can measure whether governance adds safety without excessive friction.	CTO / Engineering Ops

Suggested policy statement

“No production-impacting change is considered complete until its required tests, tenant/data-safety checks, documentation obligations, independent approvals and deployment verification have been satisfied by enforceable controls.”

6 Target operating model

The target is not more bureaucracy. It is to preserve the repository's current speed while ensuring that the most important safeguards are automatic, risk-based and auditable.

1	PLAN	Issue / acceptance criteria / risk class
2	BUILD	Branch from test + relevant domain skill/policy
3	VERIFY	Unit + integration + RLS + migration + E2E as applicable
4	REVIEW	Independent review for defined high-risk change classes
5	PREVIEW	Isolated PR environment + evidence
6	PROMOTE	Protected merge to test, then controlled test→main release
7	OBSERVE	Smoke test + monitoring + defect-to-regression feedback loop

Governance principles

Automate the invariant: If a rule matters repeatedly, encode it in CI rather than relying on memory.

Match rigor to risk: Do not impose the same approval burden on copy changes and tenant-data migrations.

Separate authoring from assurance: For high-impact changes, the final approval should not come solely from the same agent or person that authored the change.

Prefer evidence over ceremony: Tests, previews, migration rehearsal and smoke results are stronger than checklist-only compliance.

Keep the source of truth explicit: Generated and authoritative documents must win over stale narrative copies.

Appendix A Evidence reviewed

This assessment is based on repository evidence observed on the getrac test branch and GitHub repository configuration available to the connected account. It is an engineering-governance assessment, not a penetration test, legal audit or formal compliance certification.

Global policy CLAUDE.md and task-specific .claude/skills rules.

Architecture and data policy docs/mimari.md, docs/veri-erisim-katmani.md and current RLS implementation.

Quality controls Frontend tests, backend unit/integration tests, tenant isolation checks, E2E and migration rehearsal.

CI/CD GitHub Actions workflows for CI, previews, staging and production deployment.

Repository governance Active repository ruleset and branch-level configuration accessible through GitHub.

Recent PR evidence Recent test-targeted pull requests, including detailed implementation/test evidence and reviewer assignment state.

Important limitations

- Branch-protection REST endpoints requiring GitHub administration permission were not readable through the connected integration; the active repository ruleset was readable and is the basis for the GitHub-enforcement observation.
- The assessment focuses on documented and observable development policy. It does not independently validate every test, secret-management control, cloud permission or production operational procedure.
- Maturity labels are qualitative management descriptors, not an external certification score.

Final executive takeaway

GETRAC DOES NOT NEED A NEW ENGINEERING CULTURE.

It needs the governance layer that makes its existing engineering culture durable, auditable and scalable beyond the current contributors.