Openkoda

Delivering Insurance Innovations Faster with Scalable Insurtech Platform

✓ No Vendor Lock-in ✓ Unlimited Users ✓ Unlimited Customizability

Contents

Authors

Arkadiusz Drysch
CEO at Openkoda
Connect

Michał Głomba
Openkoda Founder
Connect

Arkadiusz Krysik
Marketing Manager at Openkoda

Why “Just Buying a System” Rarely Works in Insurance

If you talk to any CIO at an insurer or MGA, you’ll often hear the same story: “We tried to make the core platform do it... and then we ended up building it ourselves anyway.”

Insurance is one of those domains where the real differentiation hides in the details: product structure, underwriting rules, unusual distribution models, regulatory specifics in each market, legacy processes that still matter for key accounts. A generic policy or claims system can take you only so far before you start hitting the walls.

That’s why so many carriers and MGAs eventually go down the custom software route – not because they want to become software companies, but because off-the-shelf tools stop matching how they actually do business.

Let’s unpack why this happens, what makes custom projects so difficult, and how a new generation of platforms – from big suites like Guidewire to open, code-centric platforms like Openkoda – is trying to change the equation.

Why Insurers and MGAs Get Pushed Toward Custom Software

On paper, the idea of buying a ready-made platform sounds perfect: you get “best practices,” prebuilt modules, and vendor support. In reality, insurers and MGAs quickly discover a few uncomfortable truths:

1. **Products are more unique than vendors assume**
    - Niche commercial lines, parametric covers, embedded products, Takaful models, usage-based insurance, complex facultative reinsurance setups – these rarely map cleanly onto standard product schemas.
    - Even “simple” lines like motor or travel become complex once you add your specific rating logic, partner agreements, and regional regulatory quirks.
2. **Distribution is messy and multi-channel**
    - Brokers, agents, white-label partners, digital direct, aggregators, embedded journeys inside third-party apps – each channel wants slightly different flows and data.
    - Trying to model all of that inside a closed, rigid system quickly becomes a game of compromises and workarounds.

1. **Legacy never completely disappears**
    - Old policy admin or claims systems don't vanish overnight. You need to coexist, integrate, and gradually migrate.
    - This requires very specific integration patterns, data models, and workflows that standard software simply doesn't anticipate.
2. **Regulation moves faster than old architectures**
    - New reporting requirements, ESG disclosures, data residency, evolving ID verification rules – they all demand changes in your systems.
    - If every regulatory tweak becomes a six-month vendor project, your organization starts to feel trapped.

At some point, the technology leadership realizes: *"We need systems that adapt to us, not the other way around."* That's where custom development enters the picture.

The Harsh Reality of Custom Insurance Software Projects

Building your own policy, claims, or distribution platform is empowering... until the bills and deadlines start to stack up.

There are a few repeating patterns:

Scope creep disguised as "just one more rule"

Insurance logic is full of exceptions.

A business user will ask for automatic approvals under a certain threshold but only for selected brokers, or a discount that applies at renewal for a specific partner, or a routing rule that sends high-value claims to a particular team in one region but not another. None of these requests feels big enough to say "no" to; each sounds like "just one more rule." The problem is that every extra nuance touches multiple layers of the system: the underlying data model, the user interface and its validations, the reporting layer, and often the integration logic with billing, accounting, CRM, or external data providers.

What began as a controlled backlog for a minimum viable product gradually inflates into a sprawling scope. Every workshop uncovers new branches in the logic tree.

In most industries this would be labeled classic scope creep; in insurance, it often feels unavoidable because these details genuinely matter for risk selection, pricing, and profitability.

Exploding costs and moving deadlines

All of this complexity has a direct impact on cost and timelines. Custom insurance projects bring together senior engineers, domain experts such as underwriters and actuaries, and long integration chains spanning rating engines, document generation, e-signature tools, and back-office systems. Initial estimates are typically built on optimistic assumptions: integrations will be straightforward, non-functional requirements can be handled later, and certain "nice-to-haves" will wait for a second phase. Reality tends to be less kind. Legacy integration turns out to be more involved than expected. Performance, auditability, security, and multi-tenancy emerge as hard, non-negotiable requirements that demand architectural work. Items earmarked for a future phase quietly creep into the current one. Budgets stretch well beyond the original plan, and go-live dates move steadily to the right. At some point, executives start wondering whether they committed to a project or to a permanent transformation program with no clear finish line.

The hidden cost: long-term maintenance

Even when a custom platform finally goes live and appears successful, a different kind of cost begins to surface: ongoing maintenance.

From that moment on, the organization is responsible for keeping the technology stack current, patching security vulnerabilities, scaling for higher transaction volumes or new territories, and adapting to regulatory changes. New developers need to be onboarded into the architecture and domain model, and their productivity depends heavily on how clean and well-documented the system is. Without a strong architectural backbone, clear conventions, and disciplined governance, the shiny new custom platform can degrade over time into a fragile, tightly coupled system that only a handful of people truly understand. It becomes a new kind of legacy—not because of the programming language used, but because the accumulated complexity was never tamed.

The Other Side of the Coin: Customizability and Vendor Lock-In

However, the story doesn't end there. Over time, insurers discover that configurable platforms come with their own constraints.

1. **Configuration ≠ full flexibility**
   - You can configure within what the platform designers imagined. When your product or workflow needs to break those assumptions, you're back to custom code, extensions, and complex workarounds.
   - Deep customizations can make upgrades painful and expensive, which slowly erodes the benefit of being on a standard platform.
2. **Vendor lock-in shows up in several forms**
   - Technology lock-in: You're tied to the vendor's stack, patterns, and release cycles.
   - Economic lock-in: Per-user or per-module licensing means that scaling usage or adding new teams can significantly grow costs.
   - Roadmap lock-in: Your strategic priorities compete with other clients' requests; critical features may take years to appear, if ever.
3. **Change still isn't as fast as the business needs**
   - Even with configuration tools, meaningful changes often require vendor-certified partners, long testing cycles, and careful alignment with the platform's upgrade path.
   - For MGAs and innovative insurers trying to launch new products quickly or experiment with embedded models, this can feel almost as rigid as the old legacy systems.

So we end up with two imperfect extremes: on one side, fully custom builds that give insurers maximum control over their products, workflows, and integrations, but are notoriously expensive, slow to deliver, and inherently risky from a project and maintenance perspective;

on the other side, massive configurable platforms that provide strong, battle-tested foundations and industry-standard capabilities, yet often limit real flexibility and gradually lock organizations into a specific vendor's technology stack, pricing model, and roadmap.

The natural question becomes: *Is there a middle ground?*

The Middle Ground: Platform + Full Ownership with Openkoda

This is where platforms like **Openkoda** come in — sitting deliberately between "build everything from scratch" and "hand your core to a black-box vendor".

Openkoda is an open-source insurance application development platform that gives you a ready-made enterprise foundation (authentication, roles, multi-tenancy, admin panel, logging), plus insurance-specific templates for things like claims, policy management, and embedded insurance. Instead of starting from a blank page, your team begins with working modules and focuses directly on your business logic.

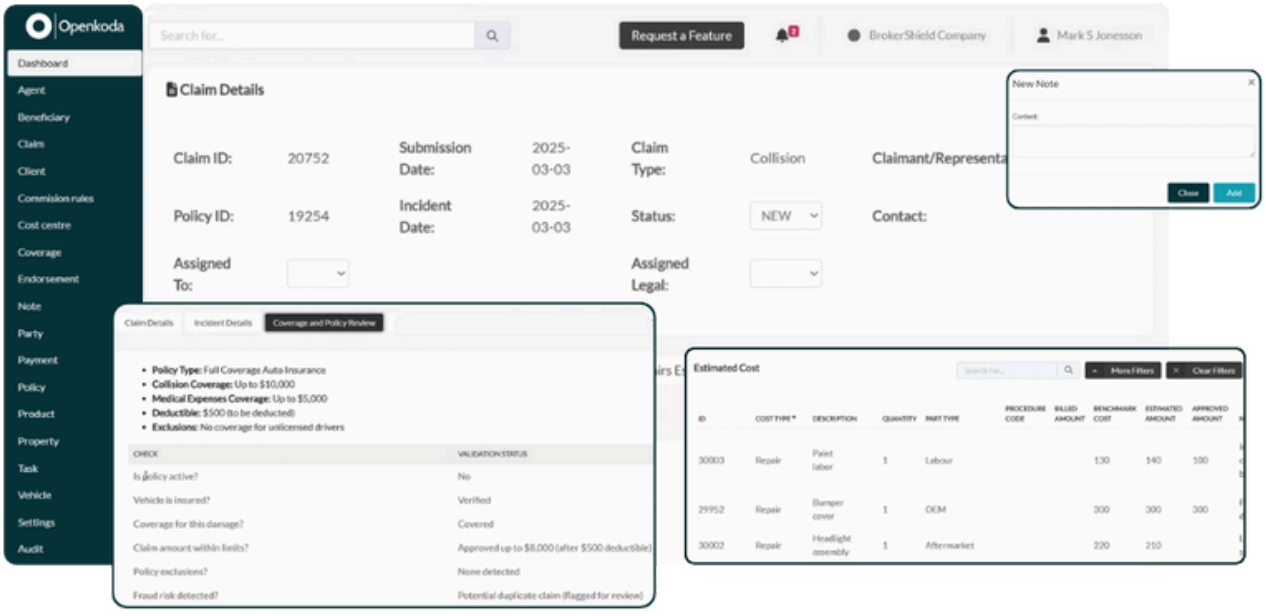
It was created as a flexible alternative to proprietary solutions (like Salesforce) for insurance companies, giving developers full ownership of the code with no vendor lock-in or per-user licensing fees.

Core Features of Openkoda

Openkoda includes a comprehensive suite of features out-of-the-box to accelerate development of insurance and enterprise applications:

- **Enterprise Application Foundation:** Common backend features are built in – authentication and user management (with multiple auth methods like LDAP, Google, SAML SSO, etc.), organization and multi-tenant support, and an admin panel for logs/health monitoring. For example, you can invite and manage users across organizations, handle password resets, and monitor system health without extra coding
- **Pre-Built Insurance Templates:** Openkoda provides domain-specific templates (for claims management, policy administration, embedded insurance, etc.) as a starting point. These come with predefined data models and insurance logic, so you can begin with a working claims engine or policy lifecycle module and then customize it to your needs rather than starting from scratch

- **Visual Data Model & Form Builders:** The platform includes a Development Kit UI that lets you define data entities and forms through a graphical interface. You can easily add or modify database tables and fields (with various types like text, number, dropdown, reference, etc.), and changes apply instantly to both the database and the UI forms. This means you can configure forms for policies or claims (e.g. add fields like Policy Type, Start Date, Claim Description) in seconds, with built-in validation rules and relationships to ensure data consistency
- **Customizable Dashboards and UI**: Openkoda comes with a drag-and-drop dashboard builder and pre-built widgets to create rich dashboards for different user roles. You can design custom views (for example, an underwriter's dashboard or an executive overview) by arranging charts, tables, and widgets. The look and feel of the application is fully customizable via standard HTML/CSS, allowing you to white-label or brand the interface to match your company's style
- **Business Logic & Automation**: Beyond configuration, developers can extend Openkoda with custom code. You can write server-side JavaScript or Java to implement custom business rules and logic within the platform
- **Integration & API-First Design:** Every Openkoda application automatically generates a secure REST API for all standard operations on your data model. his makes it easy to integrate with external systems or mobile apps.
- **Security and Access Control:** Role-based access control (RBAC) is a core feature. You can define roles and fine-grained privileges at various levels – global, per-organization, per-module, down to specific tables or even fields.
- **Multitenancy & Scalability:** The platform is built to support multiple organizations or clients on one instance – ideal for scenarios like a SaaS solution or a third-party administrator serving many insurers. You can run multiple organizations in a single database or opt for isolated schemas/databases per tenant for higher data separation.



Main Use Cases for Openkoda

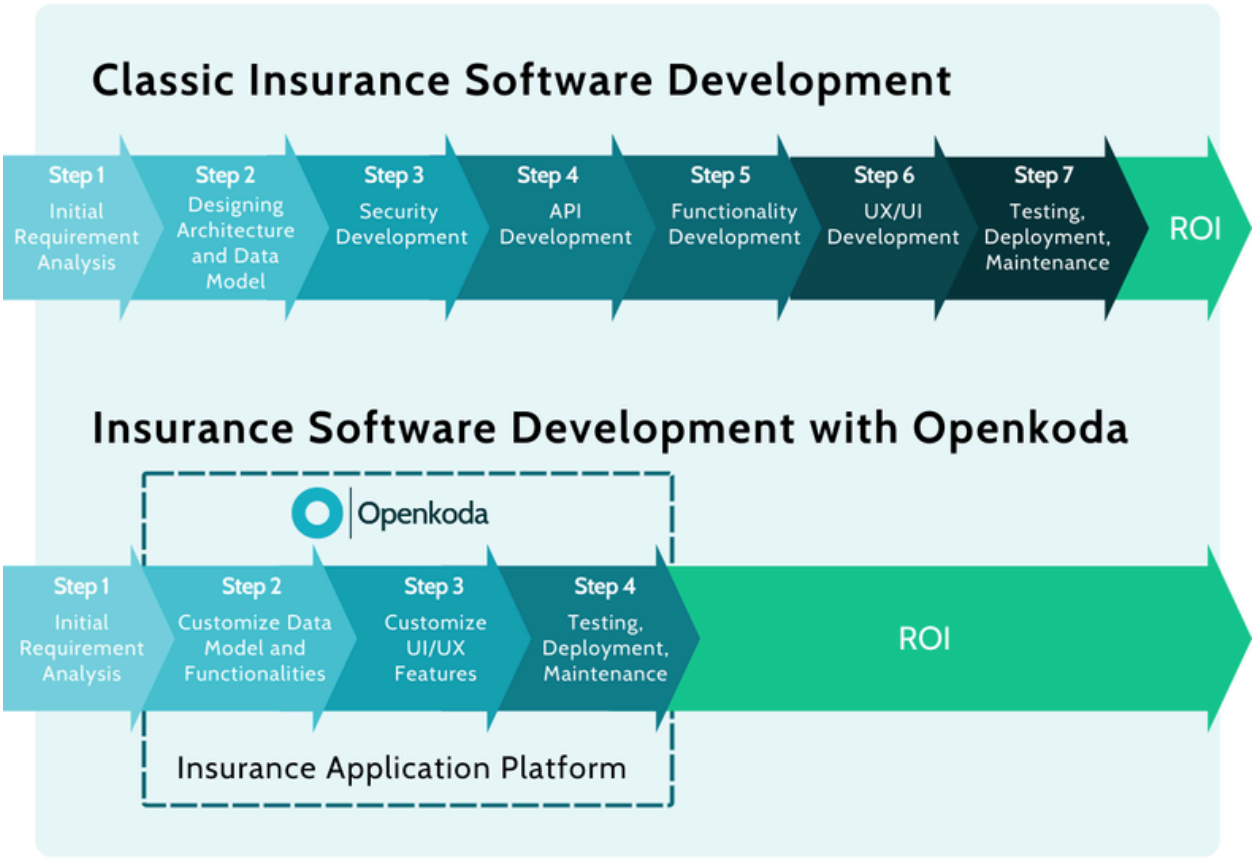
Openkoda is designed for building a variety of insurance and enterprise applications. Some of the main use cases include:

- **Claims Management Systems:** Developing custom claims handling software for insurers or TPAs (Third-Party Administrators). Openkoda's claims template provides a starting point with a claims workflow engine, and you can add features like automated claim processing, fraud detection AI, claim tracking dashboards, and payment tracking
- **Policy Administration Systems:** Building internal systems to manage the entire insurance policy lifecycle – from issuance through endorsements to renewal/cancellation. With Openkoda's policy management foundation, insurers can handle policy data, documents, endorsements, and renewals in one place.
- **Embedded Insurance Solutions**: Creating embedded insurance offerings, where insurance products are sold via third-party digital channels. Openkoda supports building embeddable insurance forms (for quotes, sign-ups, claims) that can be dropped into external websites or apps with a simple snippet. For example, an insurtech might use Openkoda to quickly set up a backend for offering travel insurance on an airline's booking site.
- **Customer and Agent Portals:** Implementing self-service portals for insurance customers, agents, or brokers. Openkoda's client portal builder and role management allow creation of secure web portals where users can log in to view their policies, submit claims, download documents, or get quotes.
- **Underwriting Dashboards:** Developing internal tools for underwriters and analysts. Openkoda can be used to build an underwriting workbench where underwriters review submissions, assess risk, and collaborate. The platform's dashboard and reporting features (including AI-driven queries) enable creation of risk dashboards, portfolio analysis screens, and workflow for quote approvals.
- **Launching New Insurance Products:** Accelerating product development for new insurance offerings or startups. Openkoda is often used by innovative insurers and insurtech startups to go from idea to market quickly. Instead of coding an entire stack, teams can use Openkoda's modules to set up a minimum viable product – for instance, a new niche insurance product (pet insurance, cyber insurance, etc.) – in a fraction of the time. This includes configuring the product's data model, building quote and policy screens, and integrating with payment or third-party services as needed. The result is a functional application ready to launch, which can then be continually extended with new features as the business scales.
- **Legacy System Modernization:** Replacing or augmenting legacy insurance systems gradually. Enterprises with old COBOL-based or monolithic systems can use Openkoda to modernize components step by step. For example, an insurer might first build a new claims module in Openkoda and integrate it with the old policy system, then progressively migrate other modules. Openkoda's flexibility and integration capabilities help in incremental legacy replacement, allowing co-existence and data integration between new and old systems during a transition period.

Building a Custom Insurance Application with Openkoda (Example)

To illustrate how one might use Openkoda in practice, consider an insurer **building a customer portal with policy management and claims submission features**. Using traditional development, this would be a greenfield project taking many months.

With Openkoda, a great deal of functionality is ready-made, and the development process might involve the following steps:



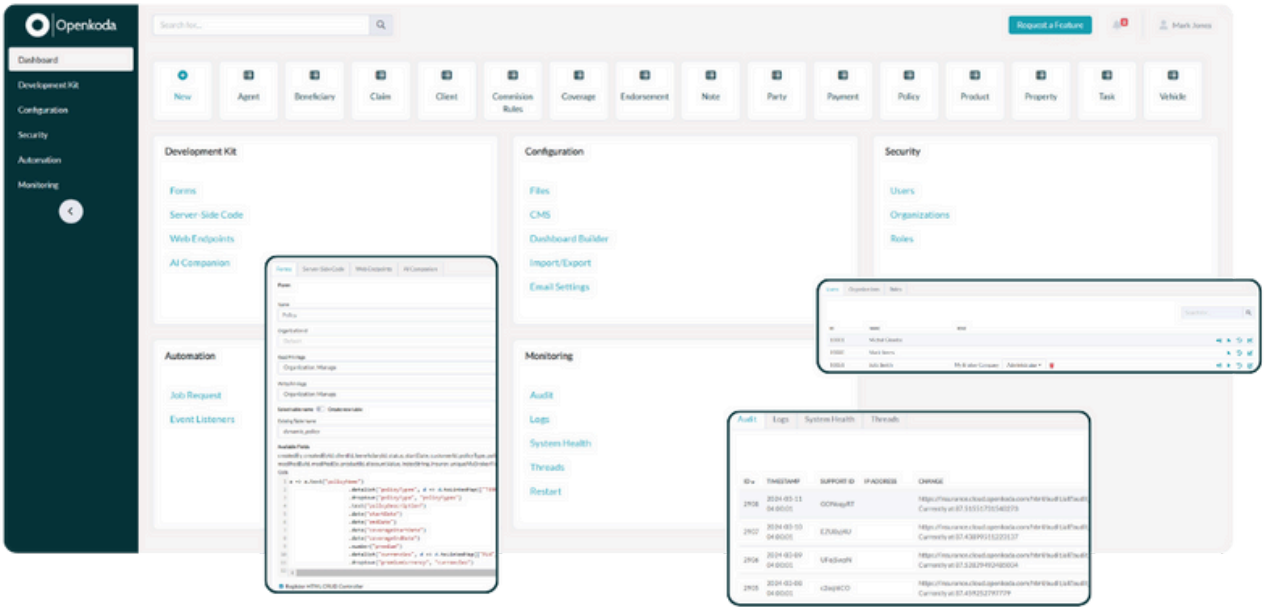
Step 1: Start with a Prebuilt Template

Rather than beginning from a blank slate, the team selects one of Openkoda's insurance templates as a foundation. For a portal that includes policies and claims, you might initialize the project with the Policy Management template (for policy lifecycle features) or a Claims Management template – both come with built-in data models and workflows relevant to insurance operations.

This provides a running application (with login, user roles, basic policy/claim entities, etc.) within minutes, saving months of initial development time.

Step 2: Customize the Data Model and Forms

Using Openkoda's visual Development Kit, developers (or even business analysts) configure the specific fields and forms needed for the insurer's products. For example, they create a Policy form with fields like Policy Number, Coverage Type, Effective Date, Premium, etc., and a Claim form with fields like Claim Description, Loss Date, Claim Amount, and file attachments for supporting documents.



Field types are chosen from drop-downs (dates, numbers, references to other entities like Customer or Agent), and validation rules are set (e.g. policy number must be unique, claim amount cannot exceed a certain value). These forms define what data is captured and ensure data quality (using required fields and consistent options) without writing low-level code.

Step 3: Customize the Data Model and Forms

Next, the team adds custom logic that reflects their business processes. Openkoda allows writing server-side scripts (in JavaScript or Java) directly in the platform to implement business rules.

For instance, you might create a rule that automatically approves claims under $1,000 or routes certain claims to a special team if the Claim Type is "Fraud Alert". Similarly, one can set up event listeners and schedulers – for example, configuring an event that triggers an email notification to a customer when their claim status changes, or a scheduled job that scans for policies nearing renewal and sends reminders.

These automations streamline operations (no need for manual tracking) and can be configured largely through Openkoda's UI with minimal custom code.

Step 4: Set Up Roles and a Client Portal UI

The portal will be used by different personas (customers, insurance staff, admins), so proper access control is configured. The team defines roles such as Customer, Claims Adjuster, Underwriter, and Admin, each with permissions to view or edit specific data.

For example, a Customer role might only access their own policies and can submit new claims, whereas an Adjuster can view all claims in their assigned region and update claim statuses. Openkoda's Client Portal Builder is then used to craft the front-end experience for customers

Step 5: Integrate with External Services

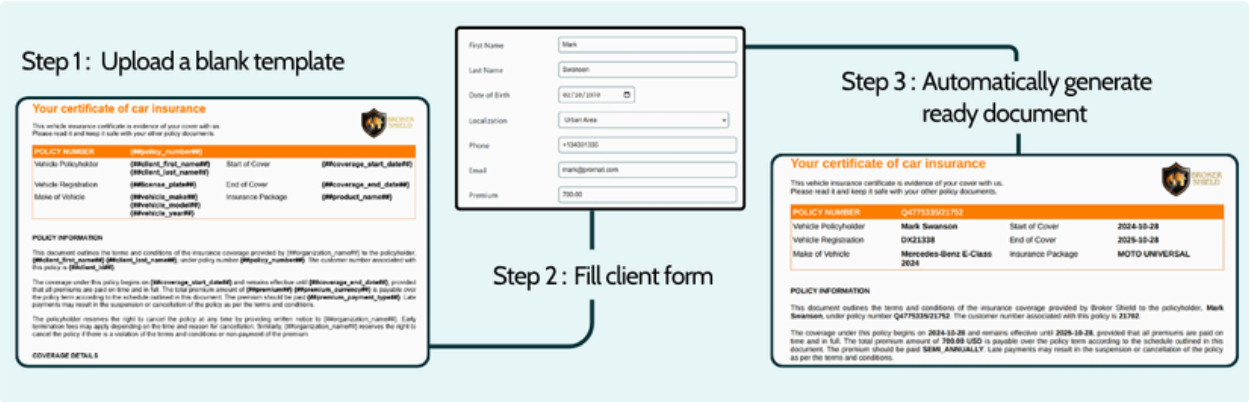
Most insurance apps need to connect with other systems. With Openkoda's API-first approach, the team can easily integrate external services. For instance, they might connect to a payment gateway to handle online premium payments, or hook into a third-party underwriting service for real-time risk scoring. Openkoda provides a REST API and webhooks out-of-the-box for the core data, so the team might build a small frontend widget for payments that calls Openkoda's API to update policy payment status upon success.

Likewise, if the insurer uses a CRM or wants to push data to an analytics platform, they can utilize Openkoda's integration hooks or add custom Java code to sync data.

Step 6: Automate Communications and Document Generation

A key part of insurance processes is documentation and communication. With minimal effort, the team sets up automated emails and in-app notifications for various events. For example, when a customer submits a claim, the system can instantly email them a confirmation and also notify an adjuster in-app about the new assignment.

Openkoda's document generation feature is used to create templates for policy schedules and claim acknowledgment letters, pulling in data fields to produce PDF documents on the fly. Now, when a policy is issued or a claim is settled, the system can generate the paperwork (with the company's branding) automatically and even send it to the customer's portal or email – saving significant manual effort.

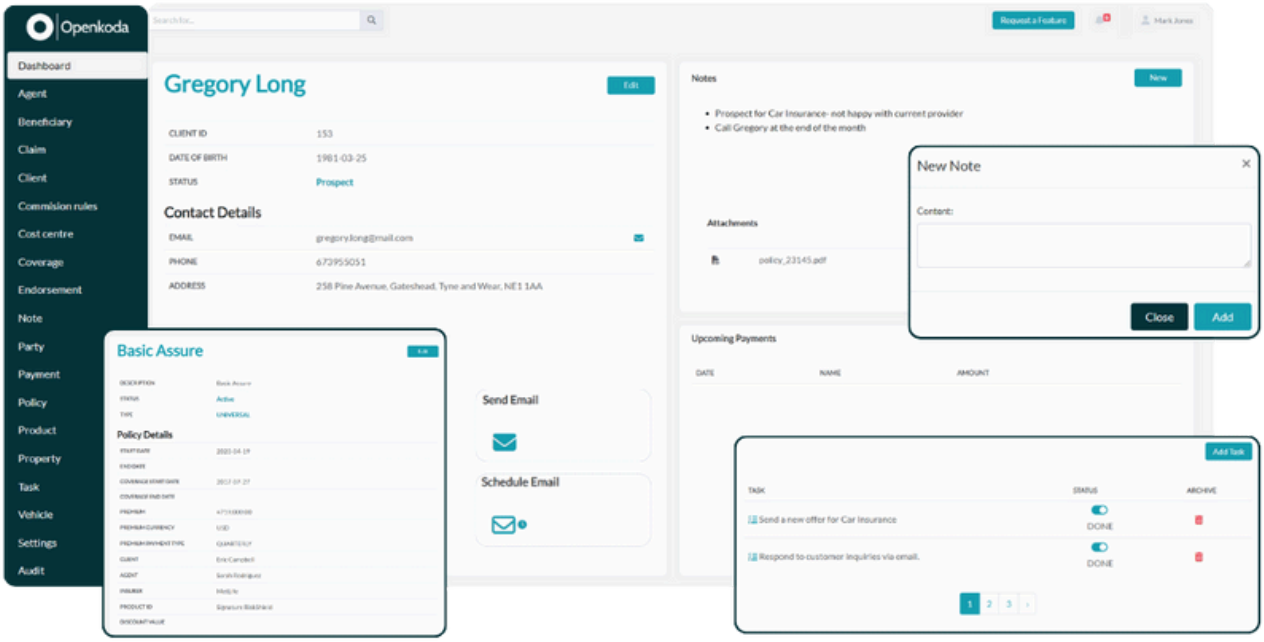


Step 7: Deploy, Test, and Scale

After configuring the portal, the application is deployed. Since Openkoda is open-source, the insurer can choose to host it on their own cloud or on-premises infrastructure, or use Openkoda's managed cloud service.

The team thoroughly tests the application's functionality and security (benefiting from the fact that much of the underlying framework is already proven and stable). As usage grows, they can scale the system by running multiple instances in a cluster and enabling multi-tenancy if the portal is extended to partner agencies or additional lines of business.

Importantly, the insurer retains full control: they have the entire codebase, and can continue to enhance the application – for example, adding a mobile app front-end that talks to the same Openkoda backend, or introducing new modules like an Underwriting Dashboard next. With the core platform in place, future enhancements are much faster to implement than if they had coded a solution from the ground up.



By following steps like the above, an organization can build a fully custom insurance application in a fraction of the time it would normally take. Openkoda's mix of pre-built features and extensibility provides a balance between speed and customization: the insurer gets a tailored solution (not an off-the-shelf generic product) but avoids months of boilerplate development work.

This empowers insurance teams to launch new digital capabilities quickly, respond to market changes, and continuously improve their systems without the usual heavy IT backlog.

Key Benefits of Using Openkoda

Choosing Openkoda as the foundation for insurance software is not just a technology decision, but a way to de-risk and accelerate digital initiatives. Instead of investing time and budget in rebuilding standard components, teams start from a proven enterprise core and focus on the parts of the system that truly differentiate their business.

Adopting Openkoda for insurance software development yields several key benefits:

- **Faster Time-to-Market:** By reusing Openkoda's ready-made modules (authentication, user management, forms, reporting, etc.), teams can reduce development effort by up to around 60% and deliver new applications in weeks rather than months. This allows insurers to respond quickly to new product ideas, regulatory changes, or partnership opportunities.
- **Scalability and Performance:** Openkoda is designed to handle large datasets and concurrent users, supporting high-volume operations such as policy renewals, claims processing, or reporting. Built-in multi-organization support and clustering enable a smooth transition from a small pilot to a nationwide or multi-tenant SaaS solution without re-architecting the system.
- **High Degree of Customization:** With full access to the source code and data model, Openkoda can be tailored to any insurance product, workflow, or integration scenario. Its modular architecture and lack of vendor lock-in mean you can introduce new coverages, rating engines, or bespoke processes on your own terms, without waiting for a vendor roadmap.
- **Ease of Maintenance:** A standardized, well-tested core built on mainstream technologies (Java, Spring, PostgreSQL) reduces technical debt and makes it easier to find and onboard developers. Built-in admin tools, audit logs, health monitoring and backup routines simplify day-to-day operations.

Openkoda vs. Traditional Greenfield Development

When insurers or insurtechs decide to build a new core system, portal, or claims platform, the default mental model is often a "greenfield" project: start from a blank repository, define the architecture, and build every layer step by step. It sounds empowering, and in theory it offers unlimited freedom. In practice, however, teams quickly discover that a huge amount of time and budget goes into recreating the same foundations over and over again — authentication, user roles and permissions, admin panels, forms, logging, reporting, integrations, and so on. Openkoda sits in a different place on this spectrum: instead of replacing custom development, it accelerates it. You still build a solution tailored to your business, but you start from a battle-tested enterprise core rather than an empty project.

The most visible difference appears in time to market.

A greenfield build of an enterprise-grade insurance system requires constructing every layer — database schema, backend services, APIs, front-end UI, security and access control, plus domain logic for policies, claims, billing or commissions. Even with modern frameworks, teams spend months just getting to a stable foundation. Openkoda compresses this dramatically by providing that foundation up front, along with reusable modules for user management, role/permission matrices, multi-organization support, dashboards, reporting, and more. Instead of reinventing these standard components, teams immediately focus on what's unique: their products, workflows, and integrations. In practice, this means launches measured in weeks rather than months and development effort cut roughly in half. A new product or pilot portal can be assembled from "insurance LEGO bricks" already available in the platform, then extended where needed.

Time savings translate directly into cost differences.

With Openkoda, fewer developer hours are required to reach a working application, and a smaller portion of the budget is consumed by generic plumbing. Because the platform is open source, there are no per-user or per-module license fees; you can onboard unlimited internal users, partners, or agents without your software bill exploding.

A common argument for greenfield development is flexibility: if you build everything yourself, you can shape the system however you like. Openkoda's design intentionally preserves that level of freedom while eliminating the heavy lifting.

The platform is modular and extensible, so teams can toggle features, swap components, and customize behavior in a controlled way.

Every layer — data model, business logic, UI, and integrations — can be adapted to the specifics of a given product line or organizational structure. You have full access to the source code and data model, which means you can implement unusual coverages, bespoke rating algorithms, or non-standard workflows without waiting for a vendor roadmap. Because Openkoda is open source, you're not locked into a single supplier's vision of what an insurance system should look like; if you need a capability, you build or extend it on your timeline. In effect, Openkoda occupies a sweet spot between raw frameworks (maximum effort, maximum freedom) and closed platforms (limited freedom, lower initial effort), offering framework-level flexibility with a substantial head start.

Maintainability is another area where the two approaches diverge over the long term. Openkoda provides a standardized core maintained by both the Openkoda team and the broader community. Critical concerns like authentication, logging, data access and security are implemented consistently across modules, reducing the risk of subtle, hard-to-debug differences between features. Built-in audit trails, backup mechanisms, and health monitoring mean many operational tasks are covered from day one rather than bolted on later. The use of mainstream technologies (Java, Spring, PostgreSQL) makes it easier to hire and onboard developers, while the enforced structure for modules and events helps keep the codebase coherent as it grows.

Taken together, these characteristics position Openkoda as a productivity accelerant rather than a traditional product or framework. It blends the agility of custom development — you still own and control every line of application logic — with the speed of a ready-made platform where much of the heavy lifting has already been done.

Closing Thoughts

In the end, choosing the right technology foundation is less about chasing buzzwords and more about securing long-term control over your insurance business. Openkoda gives insurers and insurtechs a way to innovate quickly without sacrificing transparency, flexibility, or ownership of their systems. It sits comfortably between rigid off-the-shelf products and risky greenfield builds, offering a proven core that you can still shape around your own products, processes, and strategy.

Openkoda hands you a robust set of wheels and a steering wheel: you still define the route and drive the car, you just start the journey much faster — which, in competitive insurance markets, can make all the difference.