**I.Introduction**

**I.1. Rezumat**

Aplicatia AsiguraEZ este conceputa pentru a facilita diverse aspecte ale domeniului asigurarilor printr-o arhitectura de microservicii. Aplicatia este compusa din patru microservicii distincte, fiecare fiind specializat in gestionarea unor functionalitati specifice legate de politele de asigurare, plati, despagubiri si informatii despre clienti.

**I.2. Features**

* I.2.1 Arhitectura

Aplicatia este structurata ca un set de microservicii, permitand dezvoltarea, implementarea si scalarea independenta a fiecarui serviciu. Aceasta arhitectura sporeste performanta si usurinta de intretinere.

* I.2.2 Gestionarea politelor

Microserviciul serviciu-polite se concentreaza pe gestionarea politelor. Acesta ofera API-uri pentru crearea, actualizarea, modificarea si stergerea politei de asigurare. Serviciul asigura coerenta si validitatea datelor pe tot parcursul ciclului de viata al politei.

* I.2.3 Gestionarea platilor

Microserviciul serviciu-plati este responsabil pentru procesarea platilor de asigurare. Serviciul se integreaza cu serviciul de gestionare a politei pentru a asigura o procesare precisa si sigura a platilor.

* I.2.4 Tratarea cererilor de despagubire

Serviciul-daune se ocupa de cererile de despagubire pentru asigurari. Acesta permite utilizatorilor sa depuna cereri de despagubire, sa verifice statutul cererilor de despagubire existente si actualizeaza statutul cererilor de despagubire pe baza rezultatelor procesarii. Serviciul asigura un proces simplificat si transparent de solutionare a cererilor de despagubire.

* I.2.5 Gestionarea informatiilor despre clienti

Microserviciul serviciu-clienti gestioneaza informatiile despre clienti. Acesta se ocupa de inregistrarea utilizatorilor si ofera detalii despre clientii asigurati.

**I.3. Technologies Used**

Aplicatia Microservicii de Asigurari este construita pe o tehnologie moderna si robusta pentru a asigura fiabilitate, securitate si performanta.

I.3.1 Limbaj de programare

• Python

I.3.2 Framework-uri si Biblioteci

• Flask: Un framework web usor pentru construirea de API-uri

• SQLAlchemy: Un toolkit SQL pentru Python, folosit pentru interactiuni cu baza de date

• Flask-SQLAlchemy: Extensie Flask pentru integrarea cu SQLAlchemy

• Flask-CORS: Activeaza partajarea resurselor intre origini diferite in aplicatiile Flask

• Prometheus Flask Exporter: Integreaza metrici Prometheus in aplicatiile Flask

• GKE Logging: Faciliteaza jurnalizarea structurata pentru Google Kubernetes Engine

I.3.3 Baza de date

• PostgreSQL: Un sistem de baza de date relational, open-source, folosit pentru stocarea datelor aplicatiei

I.3.4 Containerizare si Orchestare

• Docker: Folosit pentru containerizarea microserviciilor

• Kubernetes: Folosit pentru orchestrare si implementare de containere

I.3.5 Platforma Cloud

• Google Kubernetes Engine (GKE): Serviciu Kubernetes gestionat de Google Cloud pentru implementarea, gestionarea si scalarea aplicatiilor containerizate

Combinarea acestor tehnologii formeaza o baza solida pentru Aplicatia de Asigurari, asigurand scalabilitate, mentenabilitate si functionare eficienta.

**II. Microservices Architecture**

Arhitectura microserviciilor este o abordare prin care o aplicatie complexa este descompusa intr-un set de servicii mici si independente. Fiecare serviciu, cunoscut sub numele de microserviciu, este conceput pentru a indeplini o functie de afaceri specifica si comunica cu alte microservicii prin intermediul unor API bine definite. Acest stil ofera mai multe avantaje, printre care

* Modularitate: Microserviciile sunt independente, permitand dezvoltarea, testarea si implementarea izolata. Modificarile aduse unui microserviciu nu le afecteaza pe celelalte, promovand usurinta intretinerii.
* Scalabilitate: Microserviciile pot fi scalate individual in functie de cerere. Aceasta flexibilitate asigura utilizarea eficienta a resurselor.
* Rezilienta: Izolarea serviciilor inseamna ca o defectiune intr-un microserviciu nu afecteaza neaparat intreaga aplicatie. Sistemul poate ramane operational chiar daca anumite servicii intampina probleme.
* Diversitate tehnologica: Microserviciile pot fi dezvoltate folosind diferite limbaje de programare si cadre de lucru, permitand echipelor sa aleaga cea mai potrivita tehnologie pentru fiecare serviciu.

**III. Detalii Microservicii**

**III.1 serviciu-polite (Managementul Polițelor)**

Descriere: serviciu-polite este responsabil pentru gestionarea polițelor de asigurare. Se ocupă de crearea, actualizarea și ștergerea polițelor. Acest microserviciu se asigură că informațiile despre polițe sunt precise și actualizate.

**Endpoint-uri**

• GET /policy/:id: Recuperează detaliile poliței după ID-ul poliței.

• GET /policy/expiring: Recuperează polițe care expiră într-un interval de date specificat.

• POST /policy: Creează o nouă poliță de asigurare.

• PUT /policy/:id: Actualizează o poliță existentă.

• DELETE /policy/:id: Șterge o poliță de asigurare.

**Schema Bazei de Date**

Tabela principală a bazei de date este numită insurance\_policies și are următoarea schemă:

• id (Cheie Primară): Integer

• policy\_number: String

• holder\_name: String

• coverage: String

• premium\_amount: Numeric(10, 2)

• start\_date: Timestamp

• end\_date: Timestamp

**III.2 serviciu-plati (Procesarea Plăților)**

Descriere: serviciu-plati se concentrează pe procesarea plăților de asigurare. Validează informațiile despre poliță, înregistrează tranzacțiile de plată și menține un istoric al plăților.

**Endpoint-uri**

• GET /payment/:id: Recuperează detaliile plății după ID-ul plății.

• GET /payment/policy/:policy\_id: Recuperează toate plățile pentru o poliță specifică.

• POST /payment: Adaugă o nouă înregistrare de plată.

• PUT /payment/:id: Actualizează o plată existentă.

**Schema Bazei de Date**

Tabela principală a bazei de date este numită insurance\_payments și are următoarea schemă:

• id (Cheie Primară): Integer

• id\_policy (Cheie Străină): Integer

• amount: Numeric(10, 2)

• payment\_date: Timestamp

**III.3 serviciu-daune (Gestionarea Cererilor de Despăgubire)**

Descriere: serviciu-daune este responsabil pentru gestionarea cererilor de despăgubire în asigurări. Permite utilizatorilor să depună cereri, să verifice starea cererilor și să actualizeze starea acestora.

**Endpoint-uri**

• POST /claim: Depune o nouă cerere de despăgubire.

• GET /claim/:id: Recuperează detaliile cererii după ID-ul cererii.

• GET /claim/policy/:policy\_id: Recuperează toate cererile pentru o poliță specifică.

• PUT /claim/:id/status: Actualizează starea unei cereri existente.

**Schema Bazei de Date**

Tabela principală a bazei de date este numită insurance\_claim și are următoarea schemă:

• id (Cheie Primară): Integer

• id\_policy (Cheie Străină): Integer

• description: String

• date\_reported: Timestamp

• status: String

**III.4 serviciu-clienti (Managementul Informațiilor despre Clienți)**

Descriere: serviciu-clienti gestionează informațiile despre clienți, inclusiv înregistrarea utilizatorilor și detaliile despre clienții asigurați. Facilitează recuperarea și actualizarea datelor despre clienți.

**Endpoint-uri**

• GET /insured: Recuperează detalii despre toți clienții asigurați.

• GET /insured/:id: Recuperează detalii despre un client asigurat specific.

• POST /insured: Adaugă un nou client asigurat.

• PUT /insured/:id: Actualizează detaliile unui client asigurat existent.

• DELETE /insured/:id: Șterge un client asigurat.

• GET /insured/name/:name: Recuperează clienții asigurați pe baza unui nume parțial sau complet.

• GET /insured/expiring: Recuperează clienții asigurați cu polițe care expiră într-un interval de date specificat.

**Schema Bazei de Date**

Tabela principală a bazei de date este numită insured și are următoarea schemă:

• id (Cheie Primară): Integer

• first\_name: String

• surname: String

• phone: String

• address: String

• email: String

**III.5 Metrics endpoint (valabil pentru toate micro-serviciile)**

Endpoint-ul **/metrics**, ca parte a integrarii Prometheus (PrometheusMetrics), expune informatii despre performanta, sanatate si utilizare a microserviciului. Prometheus, un tool de instrumente de monitorizare si alertare, poate recupera aceste metrici la intervale regulate pentru a obtine perspectiva asupra comportamentului serviciului.

**III.6 Comunicarea intre servicii**

Comunicarea serviciilor este cruciala pentru ca acestea sa functioneze impreuna si sa realizeze procese de afaceri end-to-end. In aplicatia de microservicii de asigurari, comunicarea intre servicii se realizeaza prin intermediul API-urilor RESTful bazate pe HTTP. Fiecare microserviciu expune un set de endpoint-uri bine definite, permitand altor servicii sa interactioneze cu functionalitatile sale.

De asemenea, aceasta arhitectura faciliteaza integrarea de noi servicii in viitor**.**

**IV. Deployment**

**IV.1 Prezentarea a procesului de deployment**

Procesul de deployment implica containerizarea microserviciilor folosind Docker, urmata de orchestrarea si gestionarea containerelor cu ajutorul Kubernetes. Google Kubernetes Engine (GKE) este folosit ca serviciu Kubernetes pentru implementarea si scalarea aplicatiei in cloud.

**IV.2 Docker Images**

Imaginile Docker joaca un rol crucial in procesul de implementare, incapsuland fiecare microserviciu si dependintele sale. Imaginile sunt create pentru fiecare microserviciu folosind Dockerfiles, asigurand consistenta intre mediile de lucru. Imaginile Docker sunt incarcate pe Docker Hub pentru accesibilitate in timpul implementarii cu Kubernetes.

• **Imaginea serviciu-polite**: alexpraf/serviciu\_polite:latest

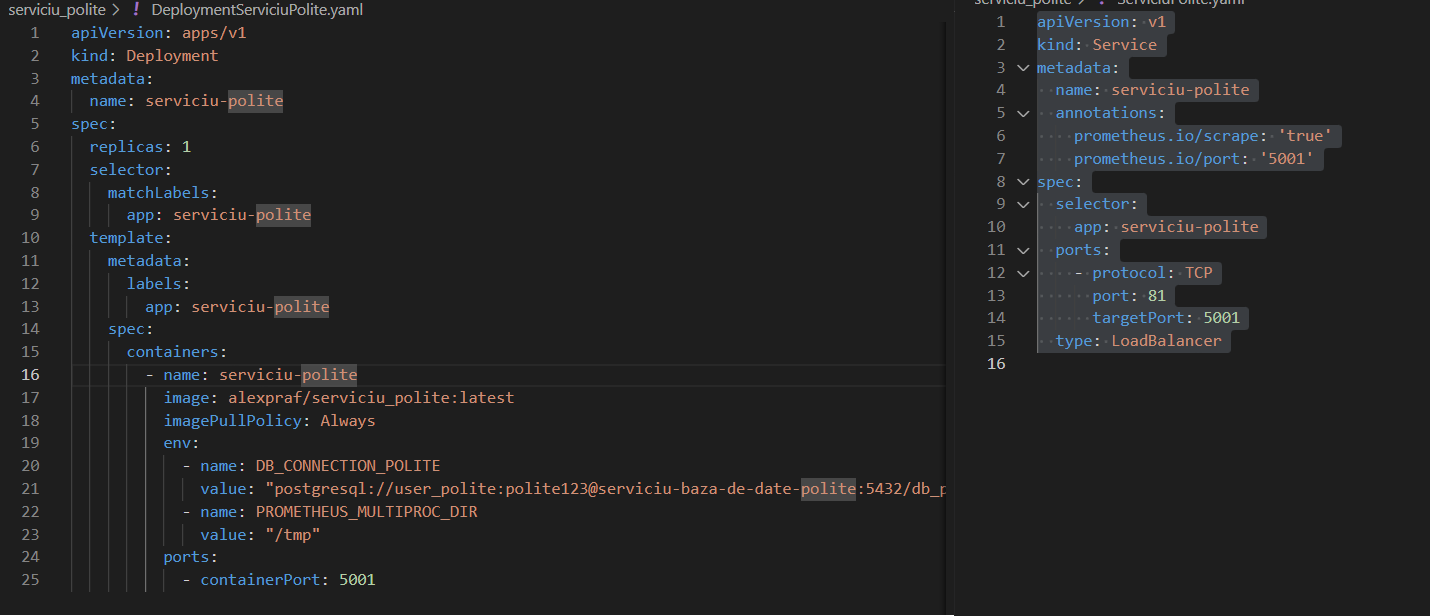
• **Imaginea serviciu-plat**i: alexpraf/serviciu\_plati:latest

• **Imaginea serviciu-daune**: alexpraf/serviciu\_daune:latest

• **Imaginea serviciu-clienti**: alexpraf/serviciu\_clienti:latest

**IV.3 Kubernetes Deployment**

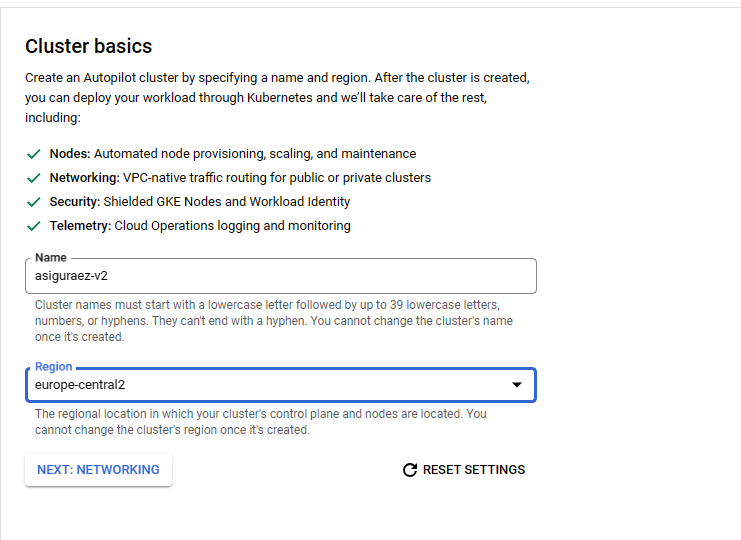
Kubernetes este folosit pentru orchestrarea containerelor Docker, asigurand scalabilitate si usurinta in gestionare. Aplicatia este implementata ca microservicii, fiecare reprezentat printr-un obiect Deployment. Serviciile sunt create pentru a expune aceste microservicii in cadrul clusterului Kubernetes.

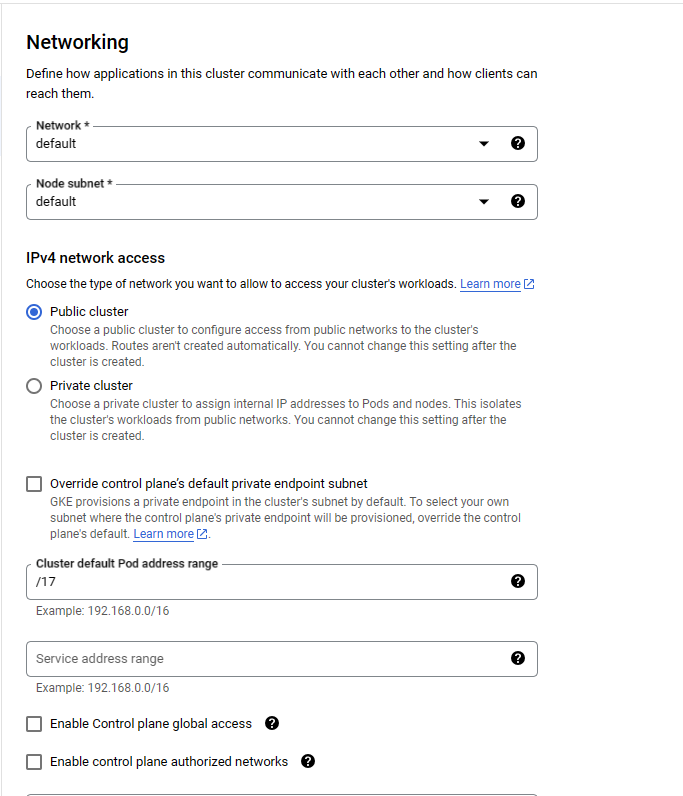


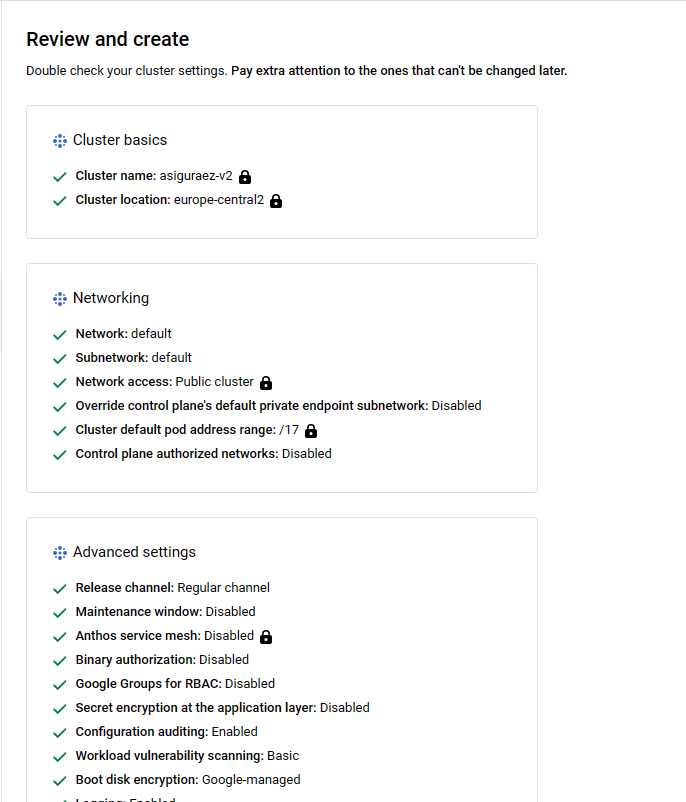
**IV.4 GKE Setup**

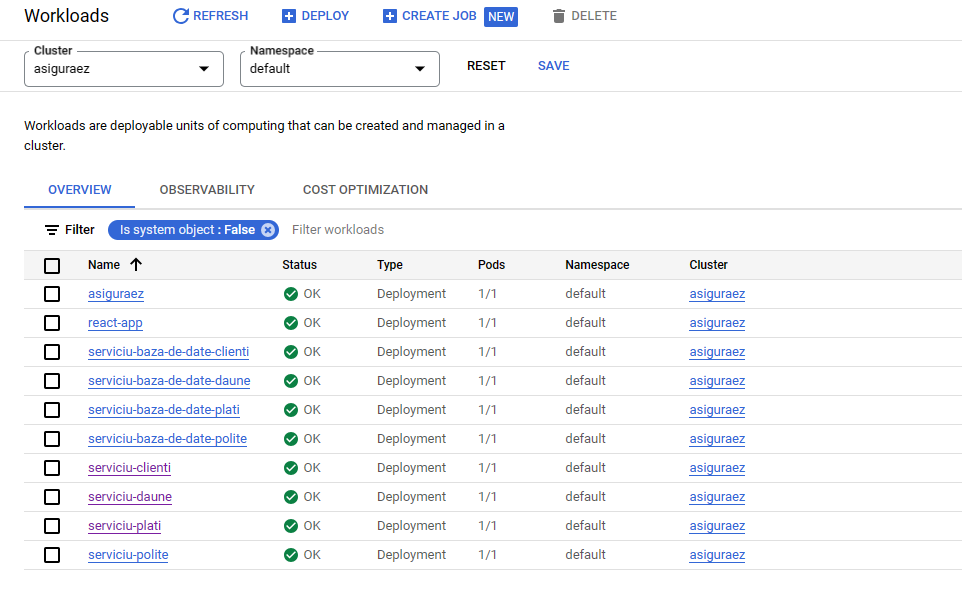
Google Kubernetes Engine (GKE) ofera un mediu Kubernetes gestionat. Configurarea GKE presupune crearea unui cluster Kubernetes pe Google Cloud Platform (GCP). Acest cluster serveste drept infrastructura pentru implementarea, gestionarea, si scalarea aplicatiei.

Google Kubernetes Engine (GKE) provides a managed Kubernetes environment. The GKE setup involves creating a Kubernetes cluster on Google Cloud Platform (GCP). This cluster serves as the infrastructure for deploying, managing, and scaling the application.









**IV.5 Comenzi kubectl utilizate**

kubectl este un instrument de linie de comanda utilizat pentru a interactiona cu clusterul Kubernetes. Mai multe comenzi sunt executate pentru a gestiona implementarile, serviciile si pentru a monitoriza clusterul. Mai jos sunt prezentate cateva comenzi cheie:

**- kubectl apply -f <fisier>:** Aplica configuratiile din fișierul YAML specificat.

**- kubectl get pods: kubectl get pods:** Recupereaza informații despre pod-urile care ruleaza.

**- kubectl get services:** Listeaza toate serviciile din cadrul clusterului.

**- kubectl get deployments:** Listeaza toate deployments din cadrul clusterului.

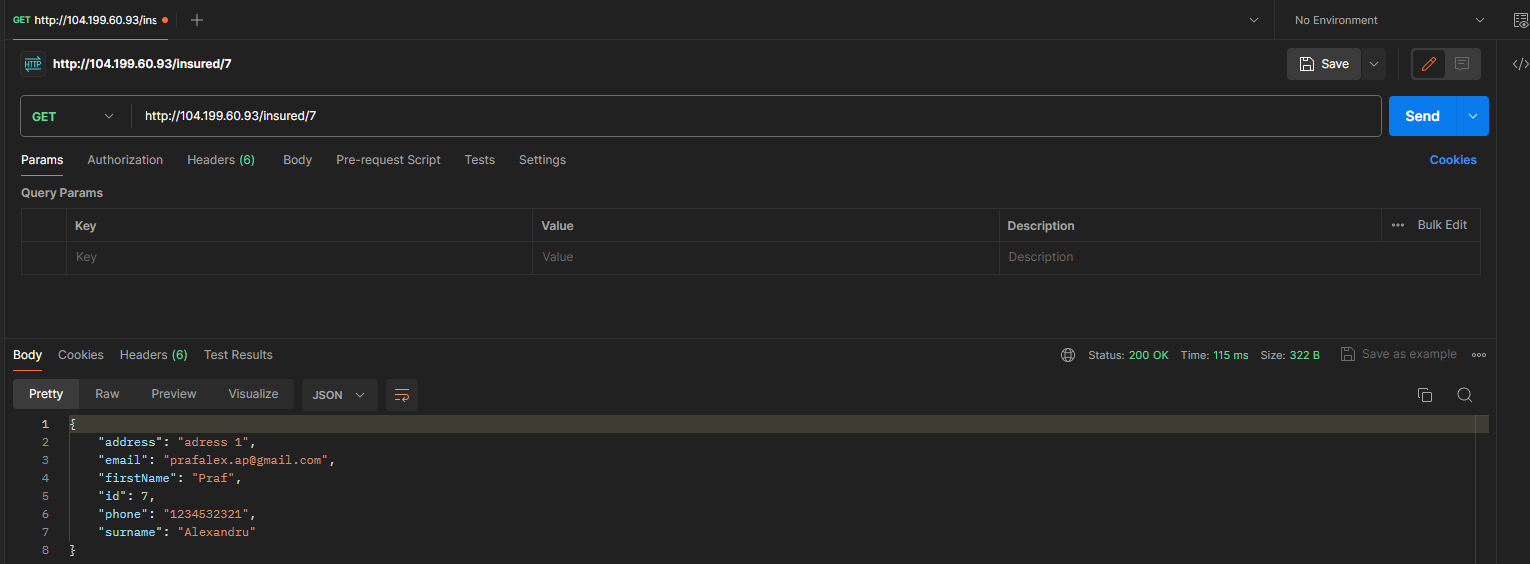
**- kubectl describe pod <pod\_name>:** Obtine informatii detaliate despre un anumit pod.

**- kubectl logs <pod\_name>:** Vizualizeaza logurile unui anumit pod.

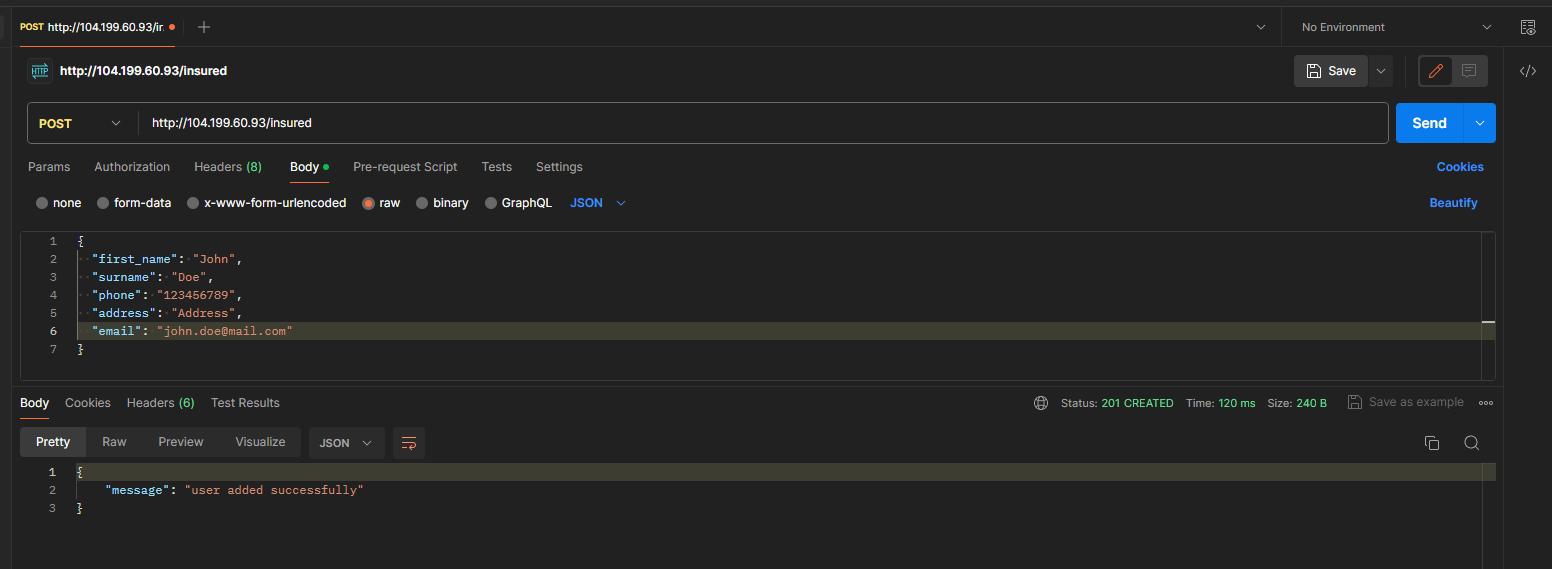
**V.Testing**

API requests cu Postman

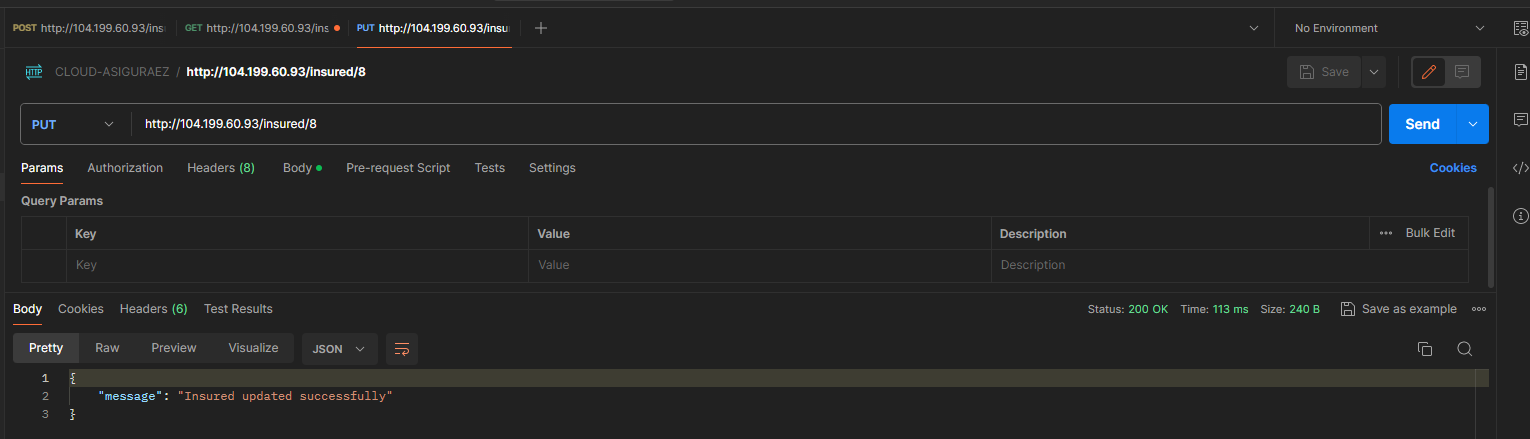
* GET insured by ID



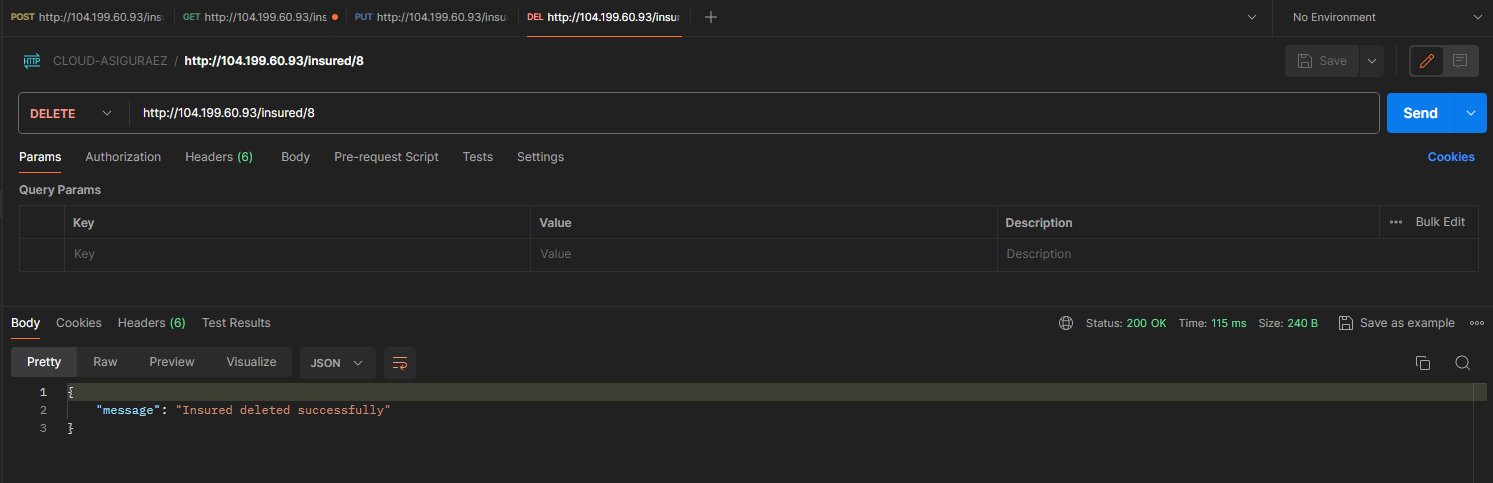
* POST INSURED



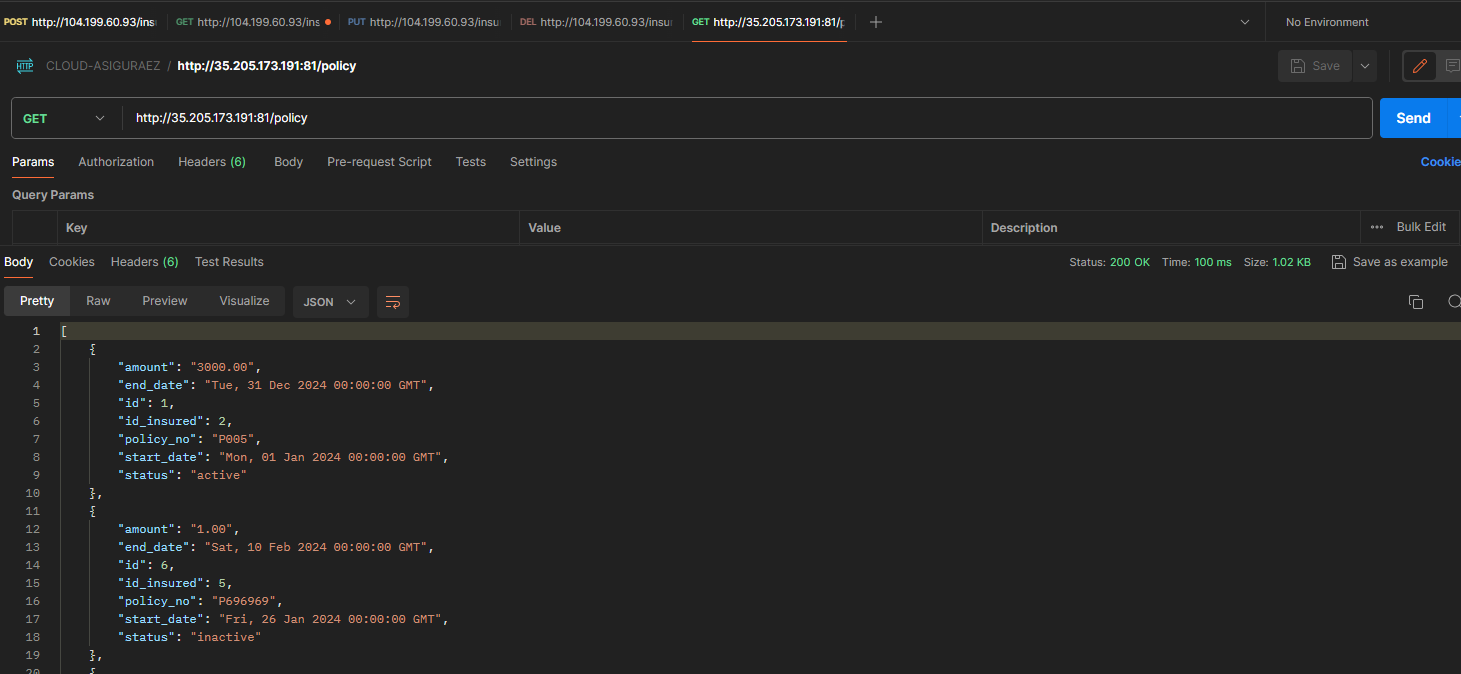
* PUT INSURED



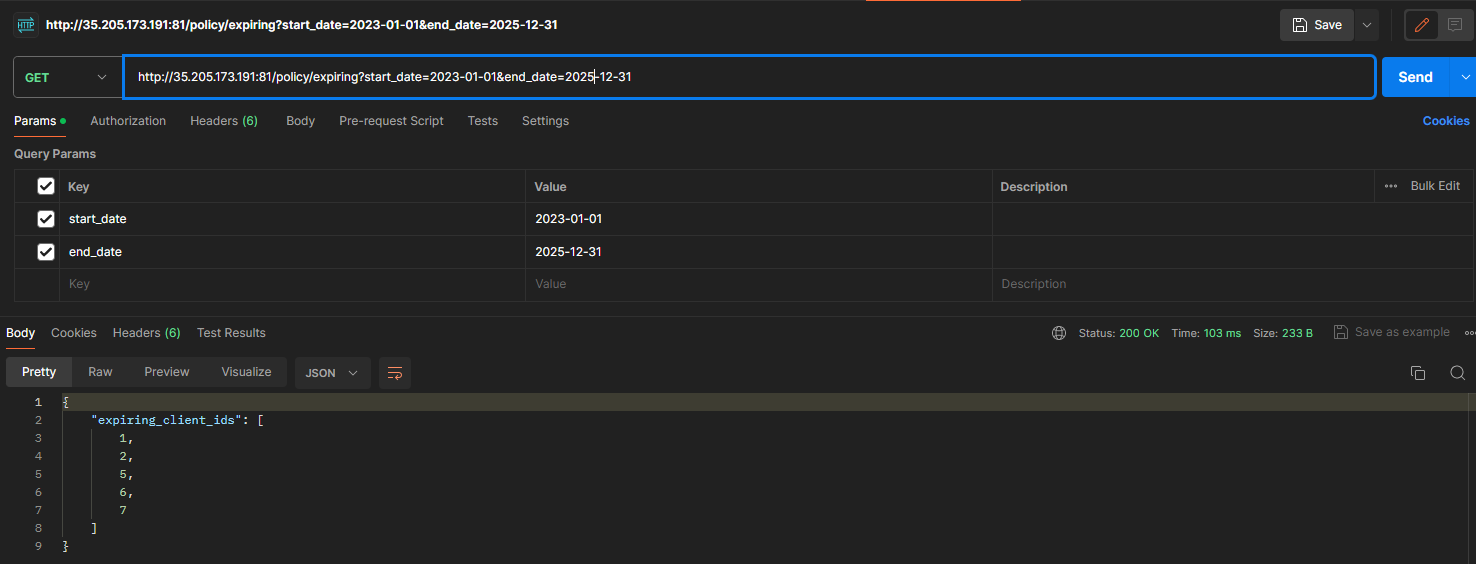
* DELETE INSURED



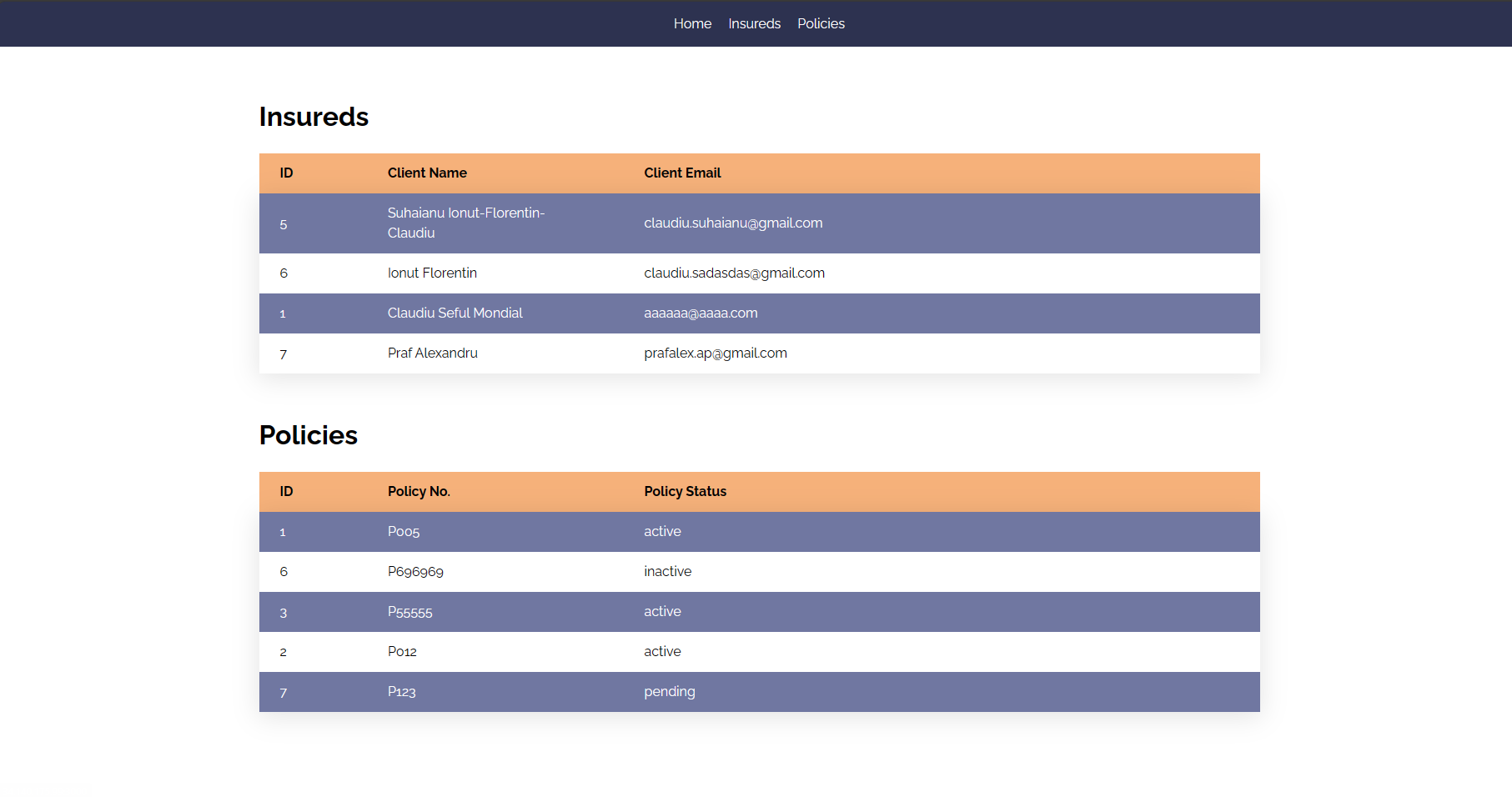
* GET POLICIES

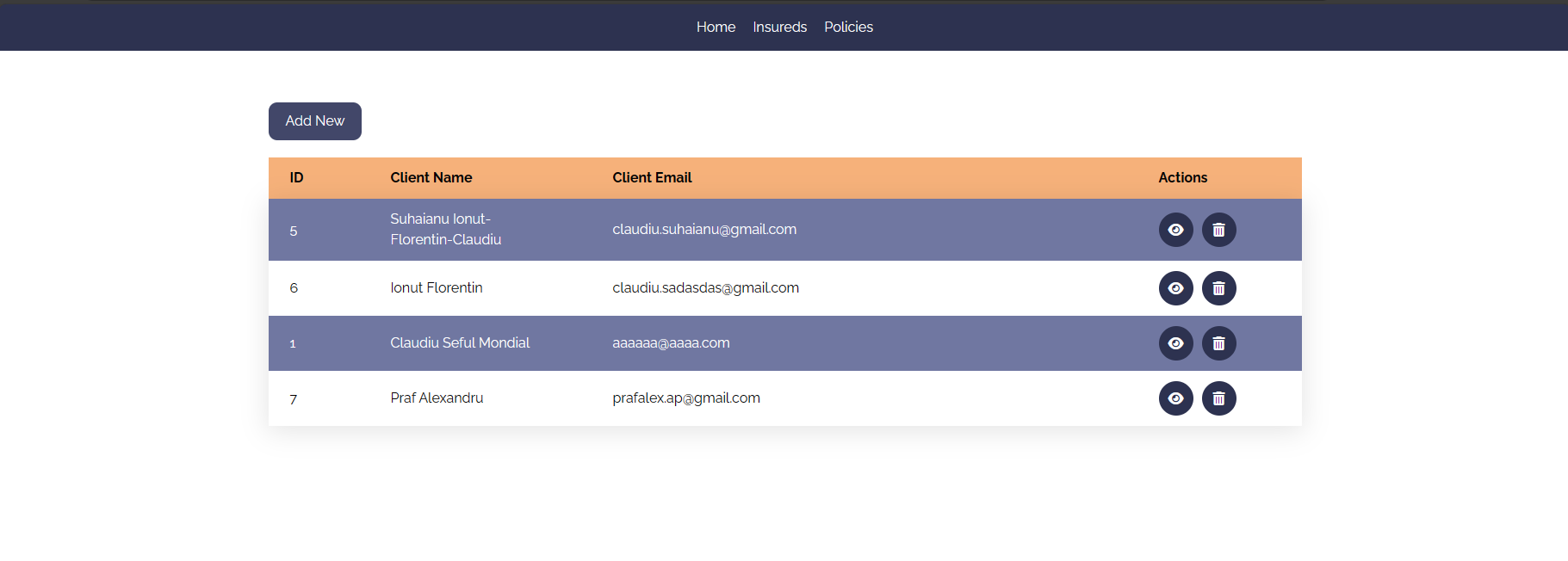


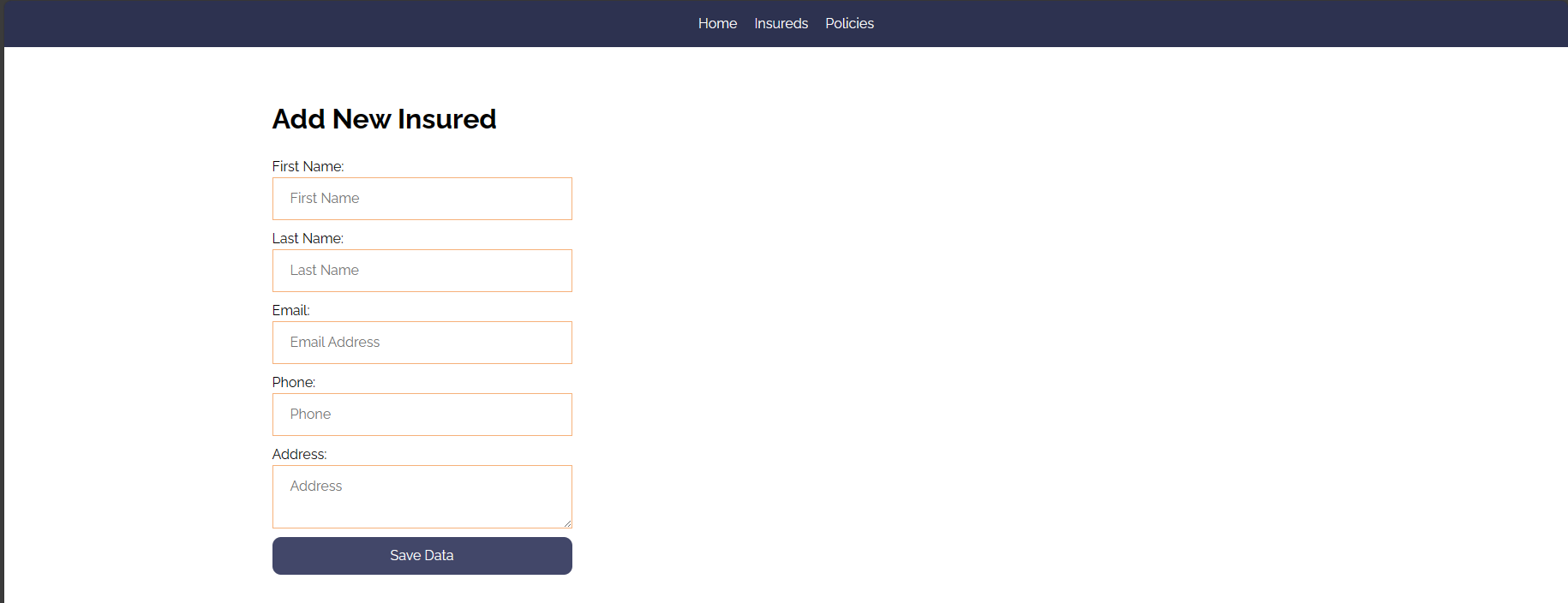
* GET EXPIRING POLICIES CLIENT IDs

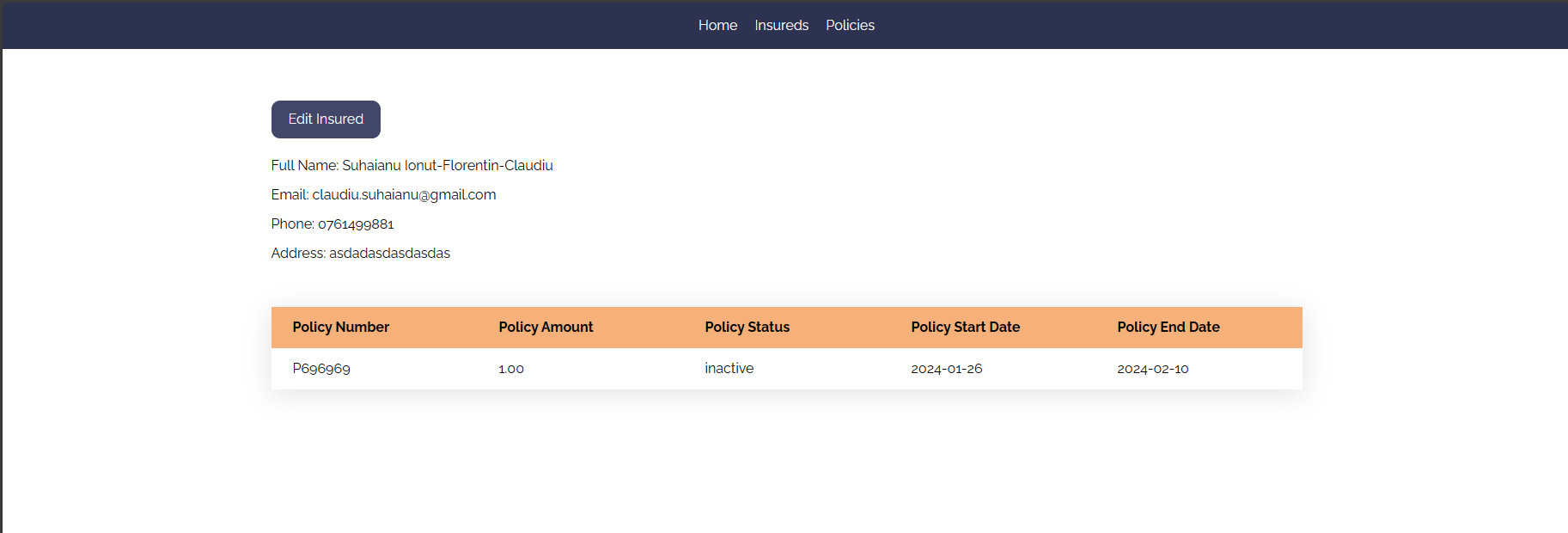


**VI. FrontEnd Application**









**VII. Concluzie**

In concluzie, aceasta aplicatie de asigurari bazata pe microservicii ofera o solutie robusta si scalabila pentru gestionarea politelor de asigurare, a platilor, a cererilor de despagubire si a informatiilor despre clienti. Utilizand o arhitectura de microservicii, aplicatia este conceputa pentru a fi modulara, permitand dezvoltarea, implementarea si scalarea independenta a fiecarui serviciu.

**Next steps:**

In timp ce implementarea actuala indeplineste cerintele de baza pentru gestionarea proceselor legate de asigurari, exista mai multe cai pentru imbunatatiri:

* **Masuri de securitate imbunatatite**: Implementarea unor masuri de securitate suplimentare, cum ar fi OAuth2 pentru autentificare si autorizare, asigurand confidentialitatea și integritatea datelor sensibile.
* **Monitorizare si logging**: Consolidarea capacitatilor de monitorizare si de logging in cadrul microserviciilor pentru o mai buna depanare, optimizare a performantei si observabilitate generala a sistemului.
* **Mecanisme de cache**: Introducerea mecanismelor de caching pentru a imbunatati timpii de raspuns pentru datele accesate frecvent, imbunatatind performanta generala a sistemului.