Corso di Piattaforme Cloud e Mobile

Migrazione della piattaforma Vitalink su AWS

di Andrea Roggeri, matricola 1079033

Introduzione

Vitalink è una piattaforma sviluppata nel contesto della tesi della triennali in Ingegneria Informatica all'Università di Bergamo. Essa si pone l'obiettivo di funzionare da interfaccia comune tra paziente e utente utilizzatore, permettendo al personale sanitario di svolgere le operazioni di follow-up clinico seguendo gli andamenti dei parametri vitali relativi al paziente comodamente da browser.

La piattaforma prevede la possibilità di integrare molteplici categorie di dispositivi, anche da diversi vendor (l'implementazione effettiva è stata effettuata solo per dispositivi *Fitbit* ai fini della dimostrazione nel concreto) e di andare a fare le letture da remoto dei parametri vitaili ottenuti da questi tramite il meccanismo *dell'OAuth 2.0*. Vengono inoltre forniti vari servizi accessori alla gestione e al monitoraggio dei pazienti e della piattaforma stessa (uno tra questi è il log di audit).

Il repository può essere consultato al seguente link, branch main, sul quale è possibile trovare ulteriori dettagli riguardanti il progetto di tesi.

Soluzione precedente ad AWS

Per la pubblicazione del container l'idea è stata quella di scegliere un servizio quanto più semplice e diretto possibile: la scelta è ricaduta su *Koyeb*. Esso rappresenza una piattaforma di *deployment serverless* in larga misura semplificato. I vantaggi di questo Approccio sono:

- Un deployment estremamente rapido
- Nessuna infrastruttura da gestire
- Auto-scaling automatico (anche se limitato nella versione utilizzata per la demo).
- Costi molto prevedibili

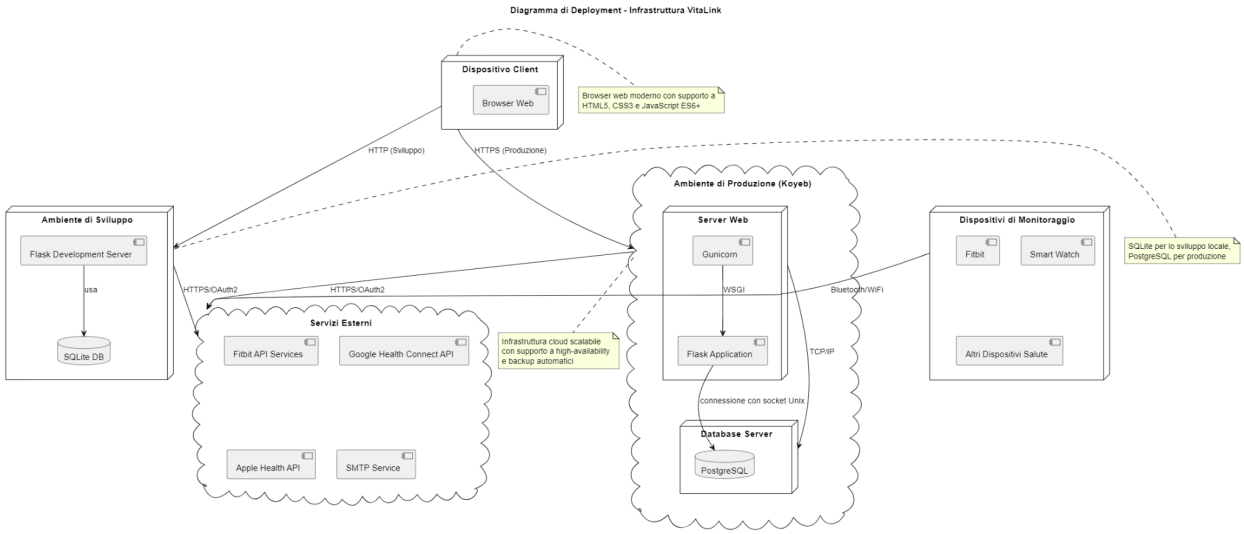
Questa semplicità però dall'altro lato porta anche degli svantaggi non indifferenti, tra cui:

- Dipendenza dalle feature messe a disposizione da *Koyeb* stesso (limitate in numero e in estensività)
- Controllo ridotto sull'infrastruttura
- Scarsa possibilità di organizzare composizioni di servizi

Risulta chiaro che per un semplice test/demo, questa soluzione può andare anche bene.

Diverso il caso in cui si pensi ad un'adozione della piattaforma su scala globale e con molteplici interazioni con servizi esterni (quali ad esempio i server con i quali si fanno chiamate API per recuperare i dati dai dispositivi collegati dal paziente).

Questo progetto va ad analizzare una possibile soluzione in ambiente AWS per deployare la piattaforma Vitalink e rendere la sua infrastruttura quanto più modulare e scalabile possibile. Il codice sorgente e tutti i file relativi sono resi disponibili in un branch CLOUD_MOBILE_PLATFORM.



1 - Soluzione Koyeb

Introduzione ad AWS: I servizi implementati

Nell'ambito del corso di *Piattaforme Cloud e Mobile* abbiamo avuto modo di ricevere nozioni riguardanti alcuni dei maggiori servizi offerti da AWS, come ad esempio ***Glue*** o ***EC2***.

Per la migrazione dalla soluzione *Koyeb* ad AWS la scelta di adottare alcuni dei servizi offerti risultava ovvia (come ad esempio l'uso di ***RDS*** per il salvataggio di dati su un database relazionale), mentre per altri l'adozione è stata subordinata ad altri servizi.

Alcuni servizi sono stati volutamente omessi dalla soluzione finale in quanto:

- Eccessivamente onerosi dal punto di vista implementativo (es. ***API Gateway*** avrebbe richiesto la rivisitazione di una buona parte del codice).
- Non compatibili con i **Role** offerti da *AWS Academy* o che richiedevano la creazione di **Role** specifici (es. ***CodeBuild*** nell'ambito di ***Code Pipeline*** per permettere di soddisfare le pratiche CI/CD ad ogni nuovo commit sul repository).

Di seguito sono riportati i servizi effettivi inclusi:

1. Rete e connettività

1.1 Amazon VPC

Per creare una Virtual Private Cloud, ovvero la rete virtuale isolata in cui risiedono tutti gli altri componenti (subnet, istanze, load-balancer, database).

1.2 Subnet

Pubbliche: ospitano risorse accessibili da Internet (es. ALB, NAT Gateway se presente).

Private: ospitano risorse interne (ECS Fargate tasks, RDS) non direttamente esposte all'esterno.

1.3 Internet Gateway & Route Table pubblica

Consentono alle subnet pubbliche di raggiungere (e farsi raggiungere da) Internet.

1.4 Route Table privata

Definisce il routing interno della VPC per le subnet private.

2. Sicurezza di rete (Security Groups)

2.1 ALB Security Group

Firewall virtuale per l'Application Load Balancer. Definisce quali porte sono aperte verso l'esterno e verso i target.

2.2 ECS Security Group

Protegge i task ECS (i container Fargate). Può permette traffico in ingresso solo dal load-balancer (ALB) e traffico in uscita verso il database RDS.

2.3 RDS Security Group

Protegge l'istanza PostgreSQL RDS. Permette connessioni in ingresso solo dai task ECS (tramite la regola ECSToRDSEgressRule) e blocca tutto il resto.

2.4 Regola di egress ECS -> RDS

Specifica che i container in esecuzione in ECS (gruppo di sicurezza ECS) possono aprire connessioni in uscita alla porta PostgreSQL dell'RDS.

3. Bilanciamento del carico (Load Balancing)

3.1 Application Load Balancer

Distribuisce le richieste HTTP(S) in ingresso ai container ECS attivi, bilanciando il carico su più task per alta disponibilità e scalabilità.

3.2 Target Group

Definisce il gruppo di target a cui l'ALB invia il traffico, compresa configurazione di health check.

3.3 Listener

Ascolta su una porta sul load-balancer e instrada il traffico al TargetGroup associato.

4. Computazione containerizzata (ECS + Fargate)

4.1 ECS Cluster

Risulta il contenitore logico dei service e dei task. Su Fargate non bisogna preoccuparsi delle istanze EC2; il cluster definisce il contesto di esecuzione.

4.2 Log Group

Gruppo CloudWatch Logs dove i container mandano stdout/stderr, per permettere monitoraggio e debug centralizzati.

4.3 Task Definition

Descrive l'immagine Docker, le risorse, variabili d'ambiente, mount point e configurazione dei log per ogni container.

4.4 ECS Service

Mantiene in esecuzione un numero desiderato (configurabile) di replica dei task definiti nella TaskDefinition, si integra automaticamente con l'ALB (registrazione/deregistrazione nei target group) e gestisce l'auto-scaling se configurato.

5. Database relazionale (RDS)

5.1 DB Subnet Group

Specifica le subnet private in cui l'istanza RDS verrà effettivamente lanciata, garantendo che rimanga non esposta direttamente a Internet.

5.2 PostgreSQL Database

Istanza gestita di Amazon RDS con motore PostgreSQL. Fornisce storage resiliente, backup automatici, patching del motore DB, crittografia (non abilitata nel nostro caso) e replica fra zone, se configurata(nemmeno questa caratteristica è stata configurata).

Per completezza è opportuno citare alcuni dei servizi che invece non sono stati implementati, tra cui:

CloudFront (CDN)

Vantaggio: Distribuzione globale dei contenuti statici, miglioramento delle performance.

Motivo della non implementazione: Complessità aggiuntiva non giustificata per un'applicazione dimostrativa.

Amazon ElastiCache

Vantaggio: Cache in-memory per ridurre il carico sulle richieste alle API di fornitura dati e migliorare i tempi di risposta.

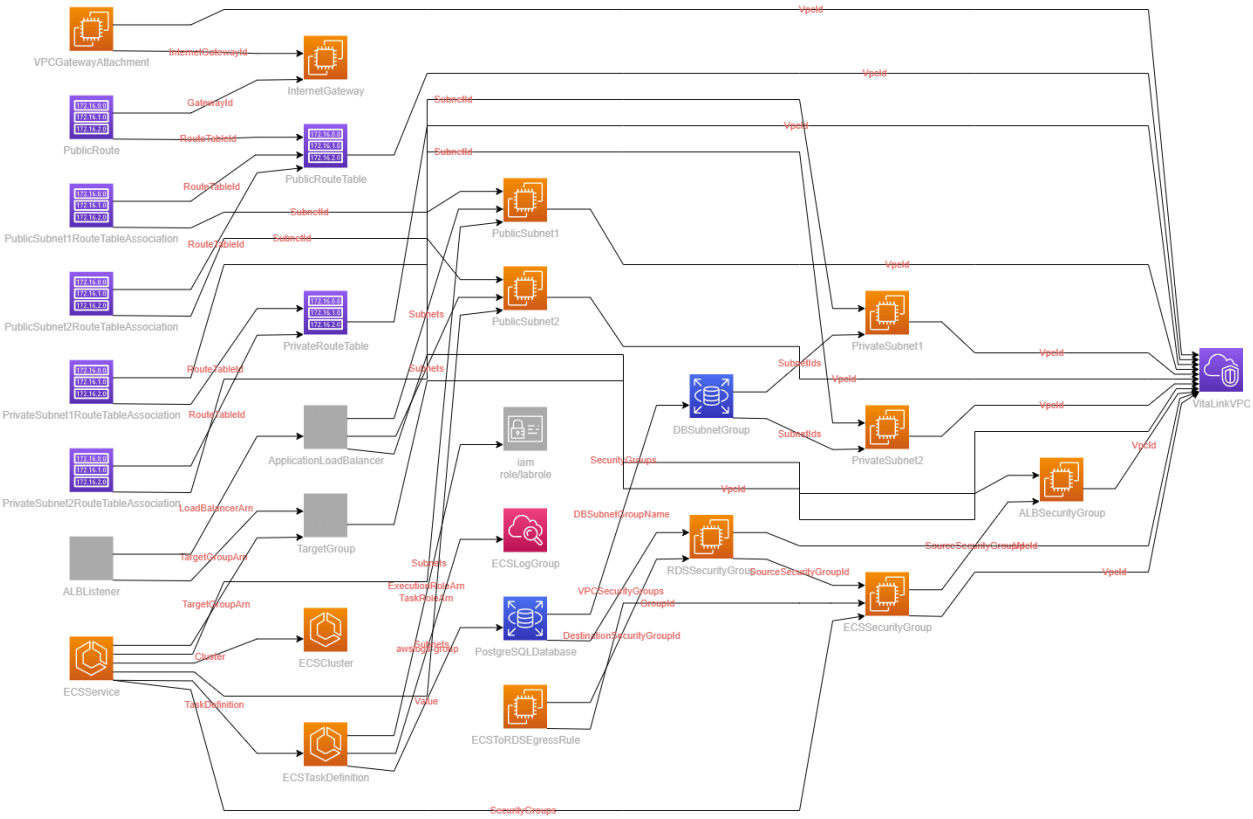
Limitazione di implementazione: Avrebbe richiesto modifiche al codice Flask per gestire la cache, oltre alla necessità di prevedere adeguamenti per motivi normativi.

AWS Lambda con API Gateway

Vantaggio: Architettura serverless, pay-per-use, scalabilità automatica.

Complessità: Molto alta, avrebbe richiesto la decomposizione dell'applicazione in funzioni.

Soluzione attuale: Approccio containerizzato più familiare e vicino all'implementazione originale.



2. Architettura estrapolata dal template di CloudFormation

Per seguire il principio dell'Infrastructure-as-a-code, è stato scelto di utilizzare ***CloudFormation.***

L'intera infrastruttura viene descritta da un file .yaml che può essere versionato e di cui si può tenere traccia come con un sorgente.

L'infrastruttura descritta da ***CloudFormation*** verrà riprodotta in maniera identica ad ogni deployment, azzerando la necessità di intervento manuale.

Va fatto notare anche che ***CloudFormation*** riolve automaticamente l'ordine di creazione delle risorse e garantisce il rollback automatico in caso di errori.

Settaggio dell'ambiente di sviluppo e test

Per l'interfacciamento con AWS, la decisione è stata quella di utilizzare il tool **CLI** per Powershell messo a disposizione per ambiente Windows in maniera tale da andare a velocizzare le attività di configurazione.

Ogni **Lab** quando viene avviato genera 3 parametri caratteristici:

- *aws_access_key_id*
- *aws_secret_access_key*
- *aws_session_token*

Questi vengono utilizzati per andare a connettersi al **Lab** dalla Shell.

Nell'ambito del seguente progetto sono stati preparati 3 script *Powershell* appositi per andare ad automatizzare le operazioni di:

Connessione con il Lab

Settaggio delle variabili d'ambiente

Deploy tramite CloudFormation

Per eseguirli basta posizionarsi nella directory radice del progetto da Shell *Powershell* ed andare ad eseguire il comando:

.\<*nome-file*>

Durante l'esecuzione degli script, verranno richiesti dei parametri da inserire (alcuni tralasciabili, altri obbligatori ai fini della riuscita del deploy). I valori standard di default possono essere modificati andando ad editare il codice dei singoli file .ps1.

Occorre specificare che l'ordine di esecuzione corretto degli script è:

1. *set-credentials-aws*
2. *set-params-aws*
3. *set-cloudformation-aws*

Requisiti per una corretta riuscita del deploy

- Assicurarsi di avere **Docker** installato ed in esecuzione sul proprio PC al momento del lancio dello script *set-cloudformation-aws*.
- Nell'esecuzione di *set-cloudformation-aws*, assicurarsi di fornire i dati corretti relativi ai percorsi.
- Risulta necessario avere una connessione attiva valida al **Lab** per poter effettuare il deploy, assicurarsi quindi di ricevere la conferma durante l'esecuzione di *set-credentials-params*.
- Alcune *variabili d'ambiente* possono essere generate in automatico se non vengono compilate durante l'esecuzione di *set-params-aws,* mentre altre no. In qualsiasi caso la compilazione con dati reali o fittizi non inficia il deploy effettivo della piattaforma, ma solo determinate funzioni (come ad esempio l'invio del report generato automaticamente dal medico curante per email al paziente).
- Se esiste uno stack già creato precedentemente, lo script *set-cloudformation-aws* domanderà se si vuole ricreare un nuovo stack (con conseguente eliminazione di quello già esistente) o se non si vuole fare nulla (lasciando lo stack inalterato).

Descrizione del workflow Github

Visto e considerato che l'integrazione di **CodePipeline** per il CI/CD nell'ambito della soluzione proposta non può essere garantita per la restrizione riguardante la creazione di **Role IAM**, è stato seguito un approccio simile ma tramite l'ausilio dei *Workflow Github*.

Un file .yml è stato creato per gestire in sequenza, ad ogni commit sul branch CLOUD_MOBILE_PROJECT, i seguenti job:

1. *Checkout del codice sorgente*
2. *Setup dell'ambiente Python*
3. *Installazione delle dipendenze*
4. *Esecuzione dei test*
5. *Configurazione delle credenziali AWS*
6. *Login ad Amazon ECR*
7. *Creazione del repository ECR (se non esiste)*
8. *Build e push dell'immagine Docker*
9. *Validazione del template CloudFormation*
10. *Controllo dell'esistenza dello stack*
11. *Pulizia degli stack in stato fallito*
12. *Deployment su AWS CloudFormation*
13. *Monitoraggio del deployment*
14. *Recupero degli output del deployment*
15. *Verifica della salute dell'applicazione*
16. *Generazione del riepilogo del deployment*

Si noti che il seguente workflow necessita comunque dei 3 parametri per la connessione al **Lab AWS**. Verranno perciò passati sotto forma di *secrets* del repository stesso su **Github**. Il lato negativo di questo approccio è che, dopo breve tempo, o comunque dopo la terminazione del **Lab**, i parametri cambieranno in valore e perciò sarà necessario andare a modificare i valori dei *secrets* per adattarli ai nuovi valori del **Lab**, comunque ciò risulta un "problema" solo per l'ambiente *AWS Academy*.

Risulta chiaro che in ambiente non *Academy* potremmo andare ulteriormente a semplificare il processo di CI/CD tramite **CodePipeline** (previa opportuna configurazione) eliminando quasi completamente la necessità di intervento manuale per le fasi di ottenimento del codice sorgente, build e deploy.

Considerazioni finali

Ponendoci in un'ottica di adozione massiva della piattaforma **Vitalink**, risulta chiaro che entrano in gioco anche fattori legati alle economie di scala.

Utilizzando AWS possiamo permetterci di avere una gestione completa e granulare dei singoli costi, andando a modularli passivamente sulla base dei servizi scelti per l'integrazione nell'infrastruttura.

Tra i tanti elementi che concorrono al favorimento di AWS, risultano rilevanti tutti quei fattori come la presenza territoriale (AWS può contare su una copertura globale), il posizionamento commerciale, la stabilità finanziaria e i ridotti rischi legati alla fornitura.

Infine se consideriamo la rapidità dell'affermarsi di nuove tecnologie come ad esempio l'intelligenza artificiale negli ultimi anni, prediligere un colosso del settore può garantire di essere sempre al passo coi tempi riducendo al minimo il periodo di potenziale attesa dell'integrazione di quest'ultima nell'ecosistema adottato.

App Flutter Mobile *Companion*

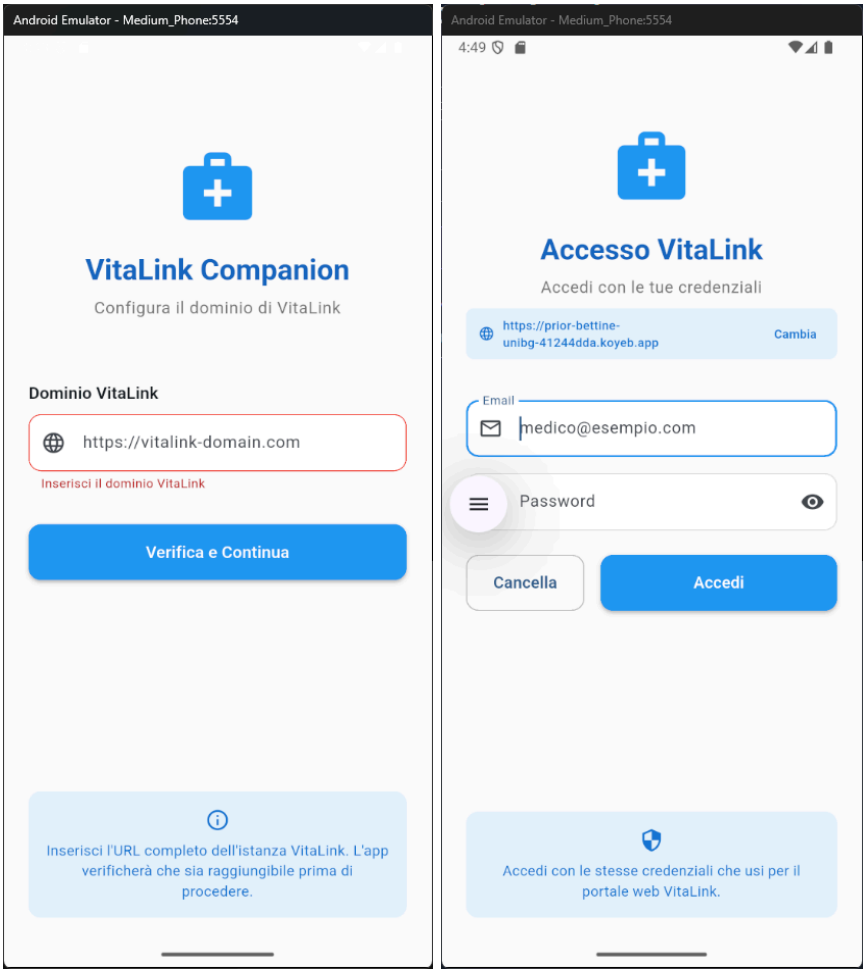
Sempre seguendo l'ottica generale del corso di *Piattaforme Cloud e Mobile*, è stato pensato di progettare un'app *Companion* per *Android* utilizzabile dagli specialisti per andare a consultare rapidamente i dati vitali dei propri pazienti e poter andare ad inserire delle note sul profilo del paziente. Per fare ciò si è reso necessario l'utilizzo delle **REST API** già implementate durante lo sviluppo della piattaforma **Vitalink** (seppure con un dettaglio molto essenziale nelle risposte).

L'utilizzo del framework **Flutter** ha permesso di semplificare in maniera modesta il lavoro di sviluppo dell'app Mobile.

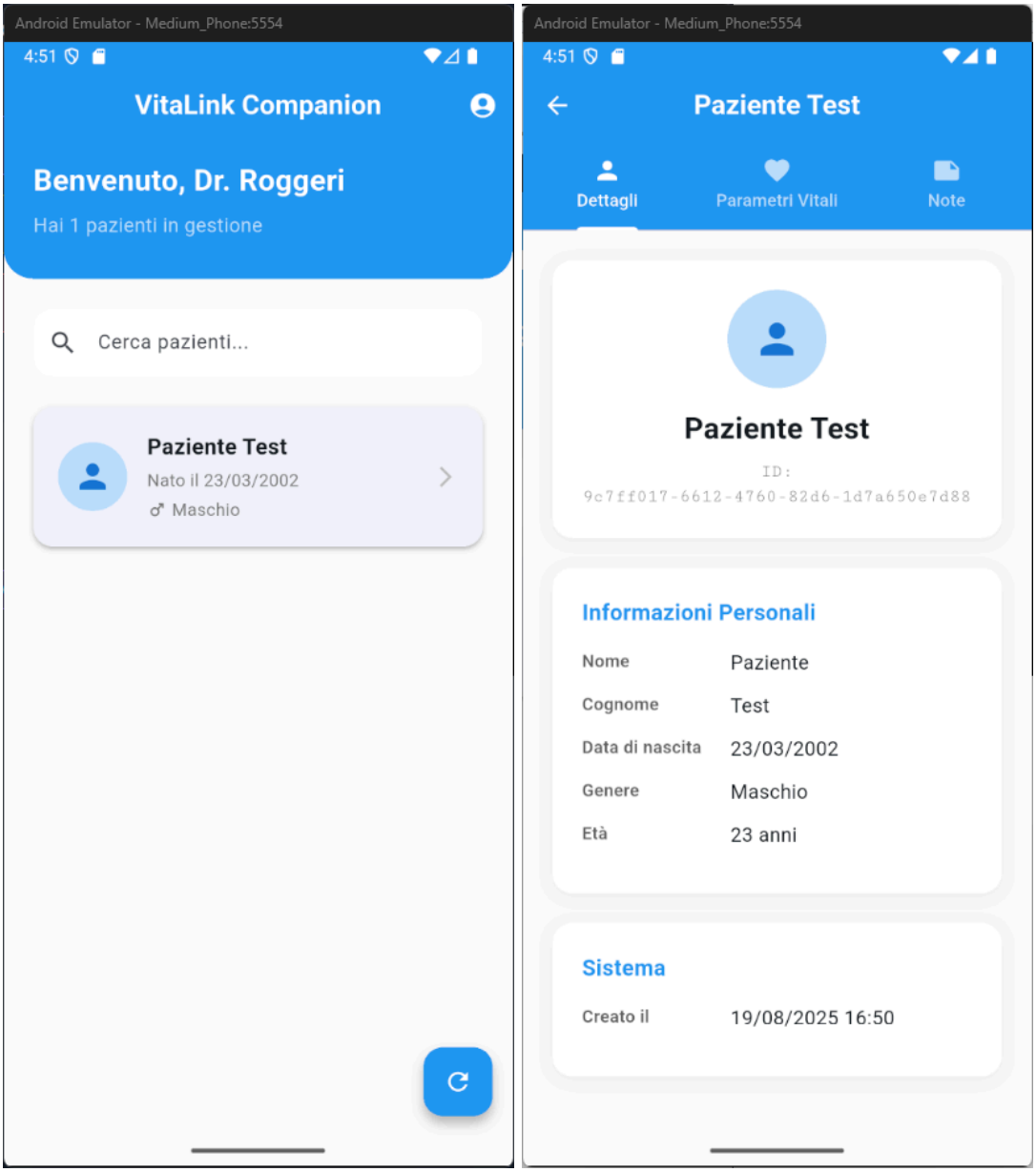
Nota: nonostante il contesto di questo progetto preveda l'utilizzo di *AWS Academy*, per la riuscita dell'ottenimento di una *Key* di accesso all'API di **Fitbit** per il recupero dei dati sanitari (da mostrare nell'app Mobile come esempio) risulta necessario avere un *URL* di callback in formato *HTTPS* (cosa non possibile con *AWS Academy*). Per questo è stato deciso creare un'istanza **Koyeb** esemplificativa in maniera tale da poter permettere il test delle funzionalità nel loro insieme dell'app *Companion* senza il bisogno di lunghe configurazioni. A questo link è possibile trovare un'istanza di Vitalink **valida per 7 giorni (fino a martedì 26/08/2025)** pronta per l'uso con una *Key* **Fitbit** già collegata (basta incollare tale link nel *dominio* dell'app *Companion*). Nel caso essa non fosse più disponibile è comunque possibile andare a creare un'istanza gratuita **Koyeb** seguendo i passaggi sul branch main di Vitalink per la configurazione delle variabili d'ambiente necessarie.

Dettagli di Vitalink *Companion*

Per iniziare con l'utilizzo dell'app *Companion*, visto e considerato che si tratta sempre di un contesto di testing, si è deciso di aggiungere un'opzione per selezionare il *dominio* sul quale si trova l'istanza di **Vitalink**, in maniera tale da andare a semplificare l'utilizzo in caso di cambiamenti continui del servizio sul quale si fa girare l'istanza. Una volta impostato il *dominio* (e verificata l'esistenza), il passo successivo è il *login* (con credenziali ottenute dalla registrazione di un account sul portale web). L'app *Companion* infatti non si pone come obiettivo di sostituire piattaforma web, ma di andare ad affiancarla.

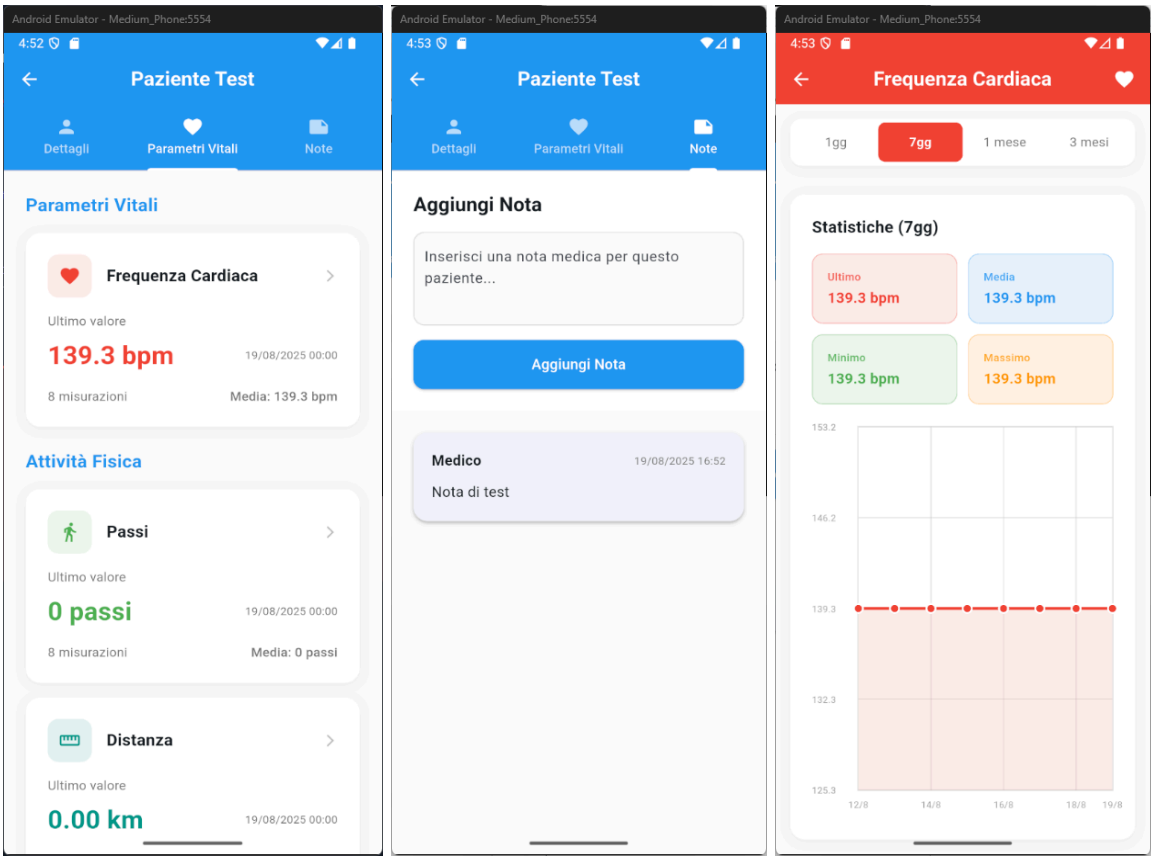


Una volta fatto l'accesso ci si troverà davanti alla *lista* dei pazienti seguiti (questa lista può essere aggiornata premendo sull'apposito pulsante) dove sarà anche possibile cercarli per nome e cognome. Cliccando su un paziente si aprirà la schermata dei dettagli del paziente, nella quale sono presenti i dati forniti dalle **API REST** già presenti e già documentate della piattaforma **Vitalink**.



Nelle schede in parte è poi possibile trovare i vari *parametri vitali*, i quali possono essere visualizzati in maniera sintetica in dei box o essere visualizzati nel dettaglio in un grafico (con la possibilità di andare a selezionare anche il livello di dettaglio) cliccando sul riquadro del parametro desiderato.

Infine nell'ultima scheda è possibile visionare le *note* già presenti sul profilo del paziente e andare ad aggiungerne di nuove.



Fonti

- Docs | Flutter Flutter
- docs.aws.amazon.com AWS
- docs.aws.amazon.com - AWS CLI
- Sviluppa per Android | Android Developers Android Studio
- Documentazione di PowerShell - PowerShell | Microsoft Learn Powershell Script
- MicrosoftDocs/PowerShell-Docs: The official PowerShell documentation sources Powershell Docs
- MicrosoftDocs/PowerShell-Docs: The official PowerShell documentation sources Draw.io (grafico architettura AWS)
- MicrosoftDocs/PowerShell-Docs: The official PowerShell documentation sources Koyeb