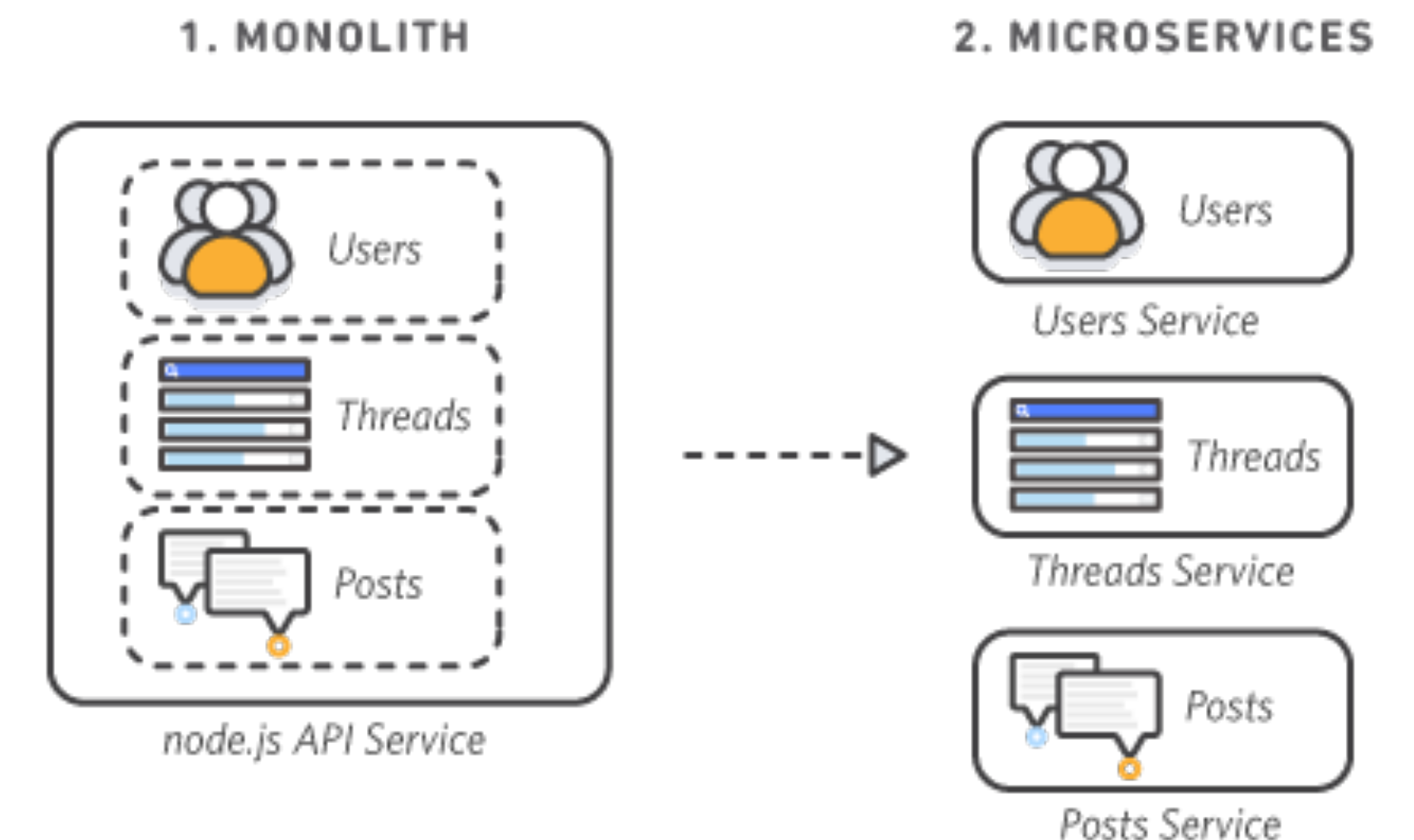


Driving School Distributed Management Software

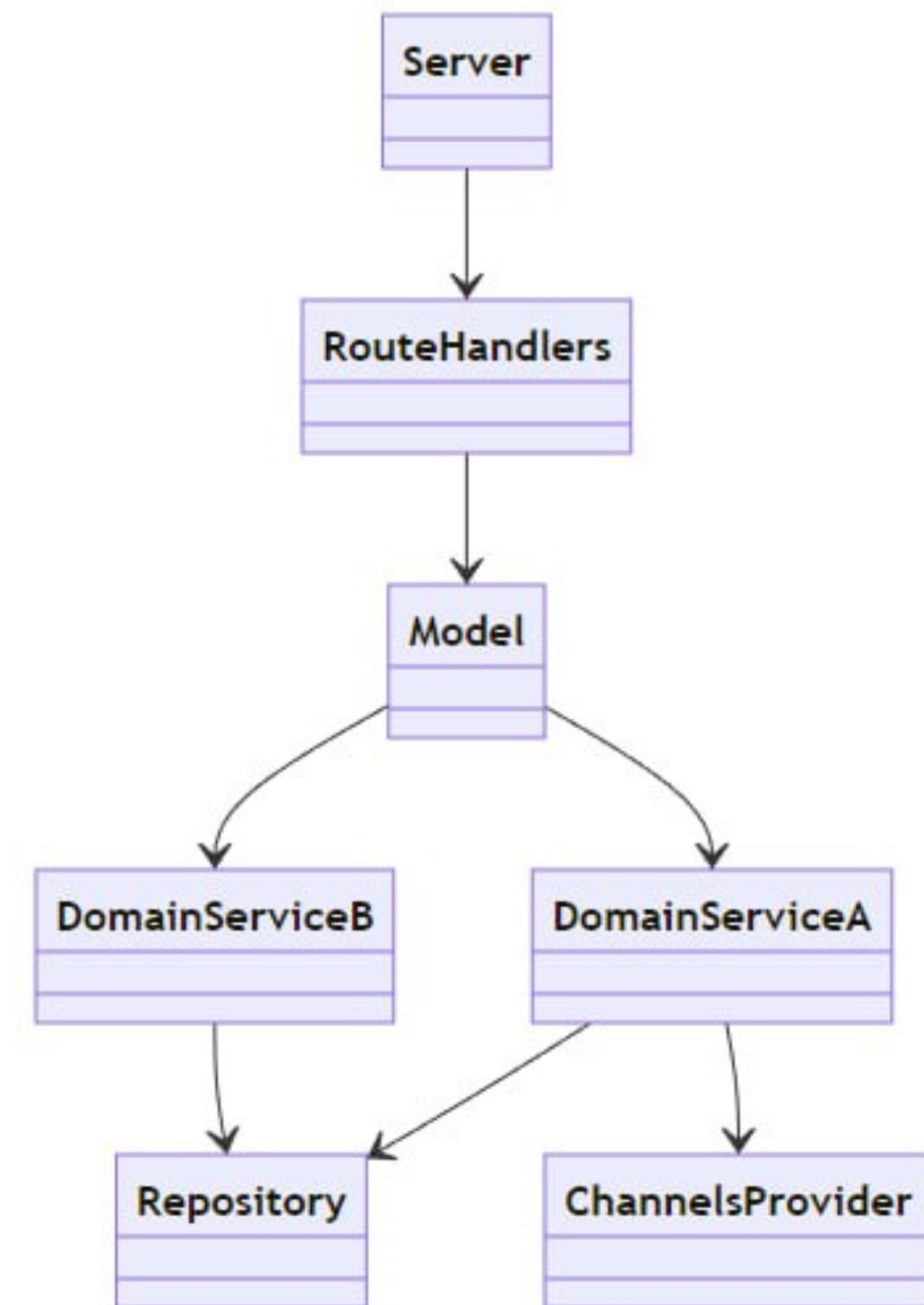
Software Distribuito basato su Microservizi

- Perchè questo progetto:
 - Software per scuola guida: esami, dossier, dottore
 - Microservizi
 - Principi DevOps, Build Automation, Continuous Integration
- Architettura dei Microservizi:
 - Servizi autonomi e specializzati
 - SoC (Separation of Concerns), resilienza
 - No condivisione di tecnologie o codice, ma riutilizzo
 - Scalabilità, utilizzi futuri (manutenibilità ed espansione), flessibilità



Quali Microservizi abbiamo utilizzato

- Partendo dalle informazioni di base: **Dossier Service**
- Arrivando alle funzioni svolte nella pratica:
 - **Dossier Service**: gestione dei dati riguardanti gli iscritti (registrazione, visualizzazione, aggiornamento)
 - **Doctor Service**: gestione delle visite necessarie al conseguimento delle patenti
 - **Driving Service**: gestione delle guide pratiche in strada e dei veicoli
 - Constraints: auto, istruttori, foglio rosa
 - **Exam Service**: gestione degli esami teorici e pratici
 - Maggioranza della complessità
 - Constraints: numero di guide, tentativi, data di validità
- Composizione a livelli



Comunicazione tra i Servizi

- **Api-REST:**

- API - interfacce esposte separatamente da ciascun microservizio
- **REST**: vincoli architetturali sulle api
 - Consegna delle risorse in formato Http/Json
 - Disaccoppiamento client-server: unica interazione mediante Uri
 - Identificatore delle risorse messe a disposizione
 - Stateless: non richiedono nessuna sessione lato server. Ogni richiesta contiene tutte le info necessarie

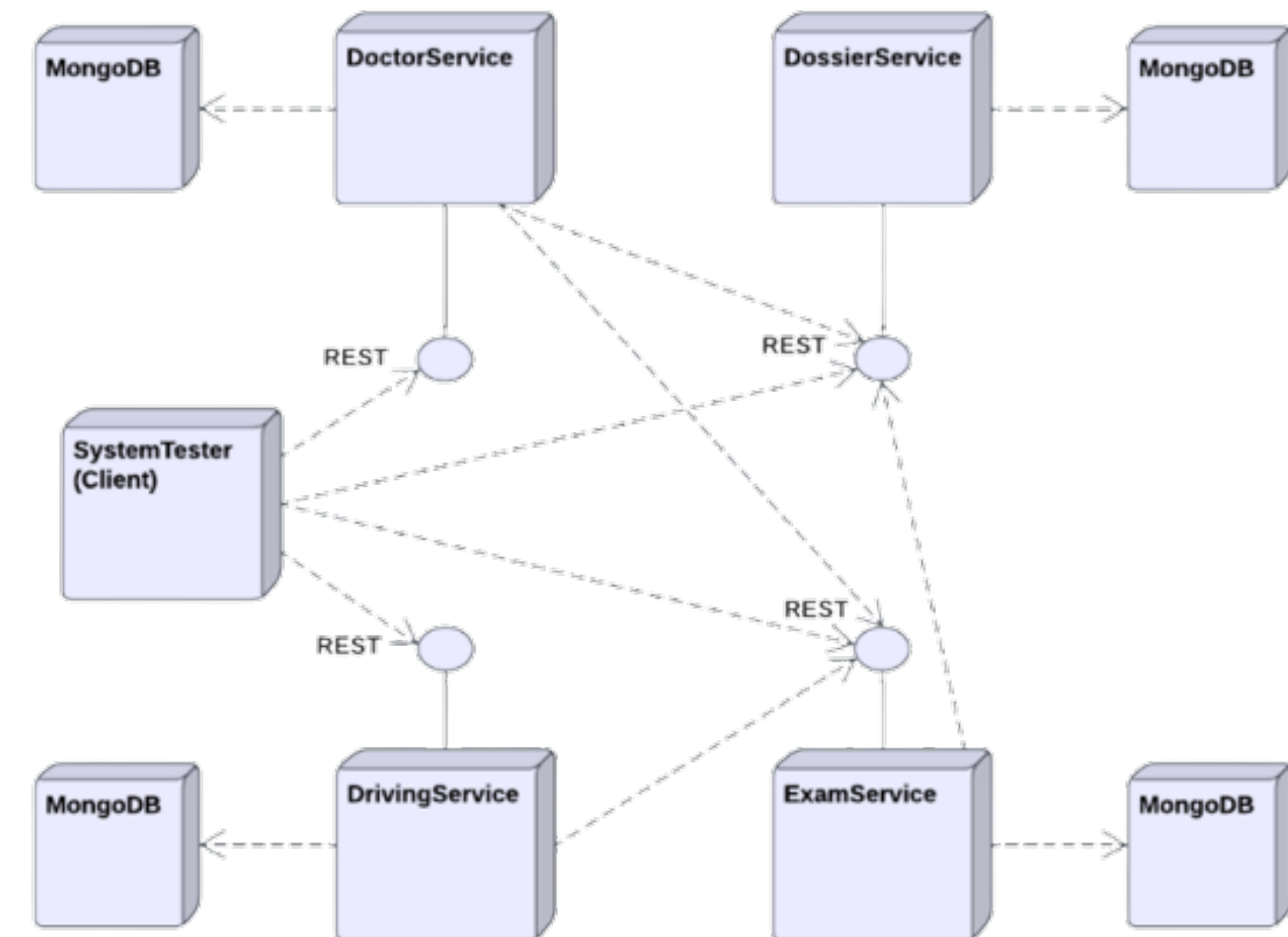
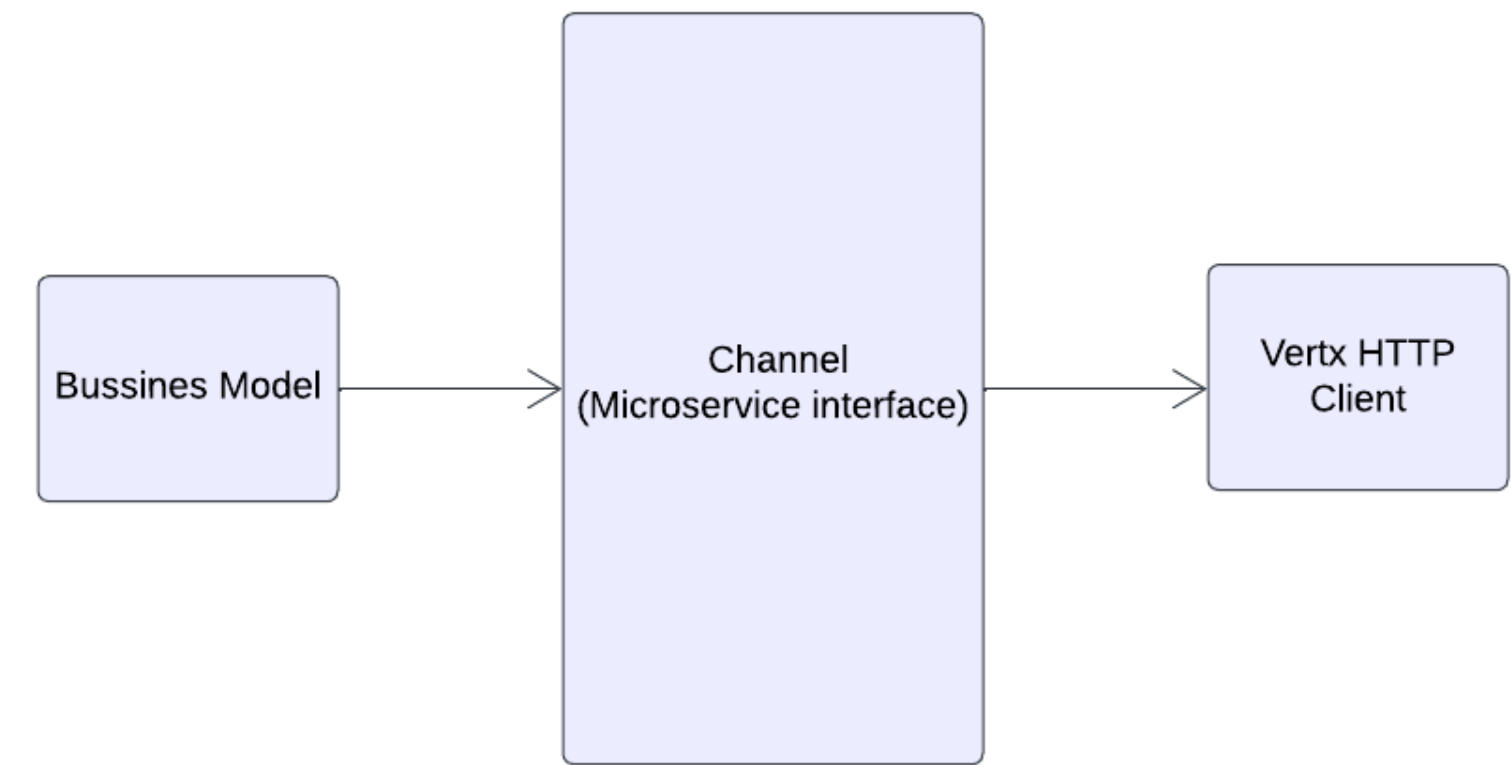
- **Channels:** comunicazione interna tra microservizi

- **KotlinX**: serializzazione e deserializzazione dei dati

- Data class $\leftrightarrow$ Json

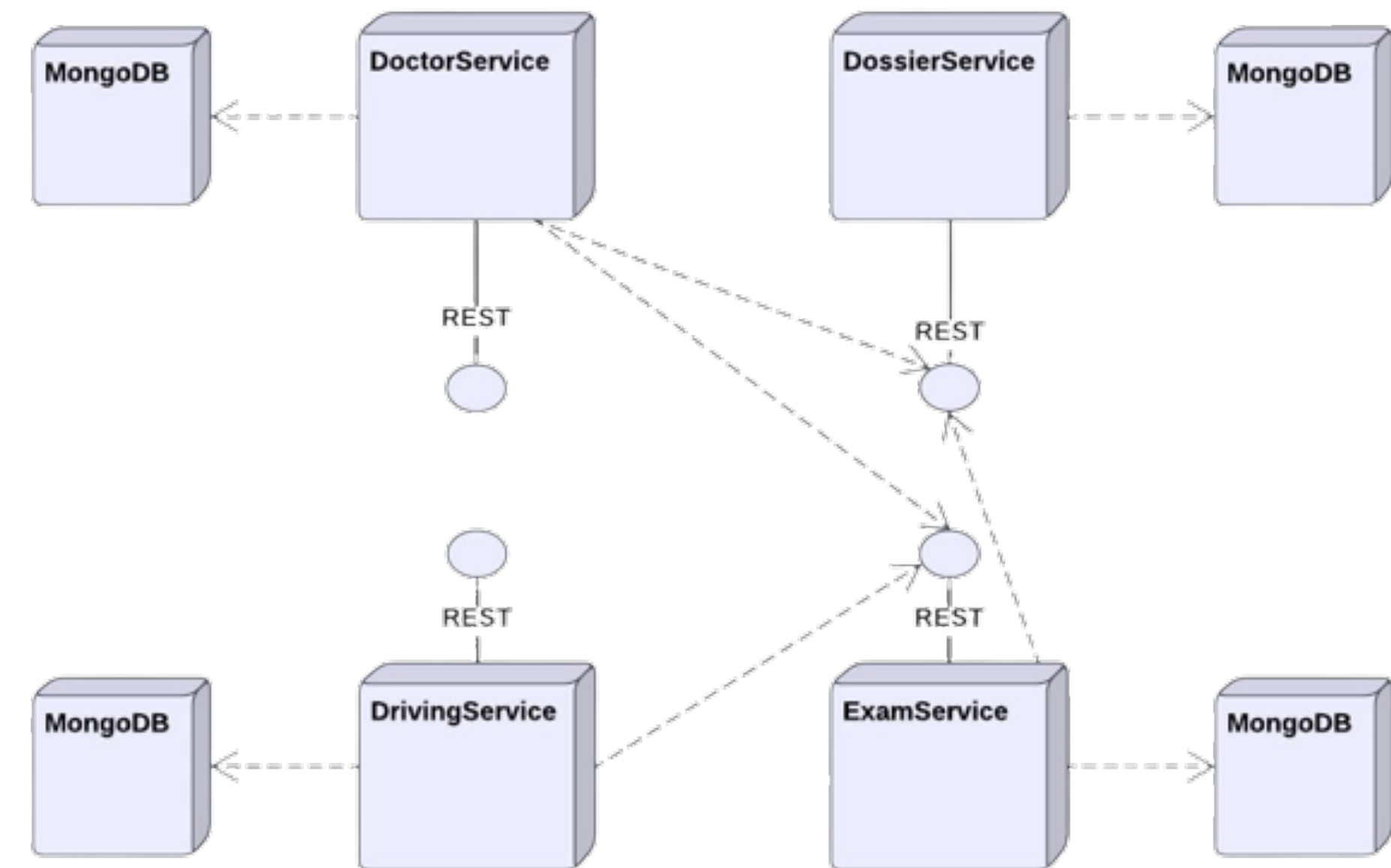
- Documentazione: **OpenApi**

- **Swagger**: toolset per lavorare con documentazione Api
- Hosting su **SwaggerHub**



MongoDB e KMongo

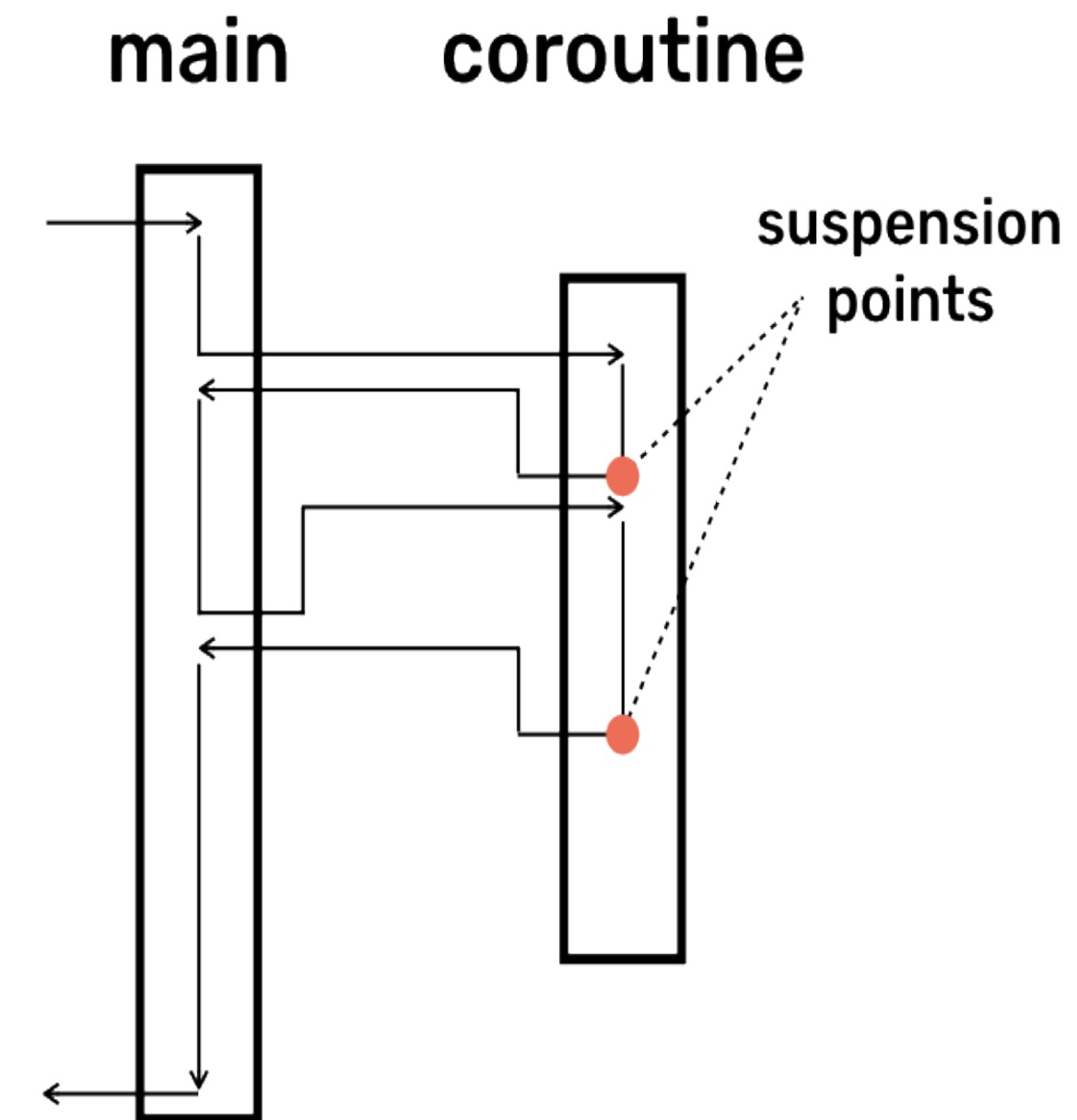
- **MongoDB:**
 - Un database NoSQL distribuito
 - Alta scalabilità
 - Configurabile per favorire Availability o Consistency
- **KMongo:**
 - Una libreria sviluppata dalla comunità
 - Pattern repository per nascondere i dettagli tecnologici
 - KMongo coroutine



Comunicazione asincrona

Coroutine per la programmazione asincrona:

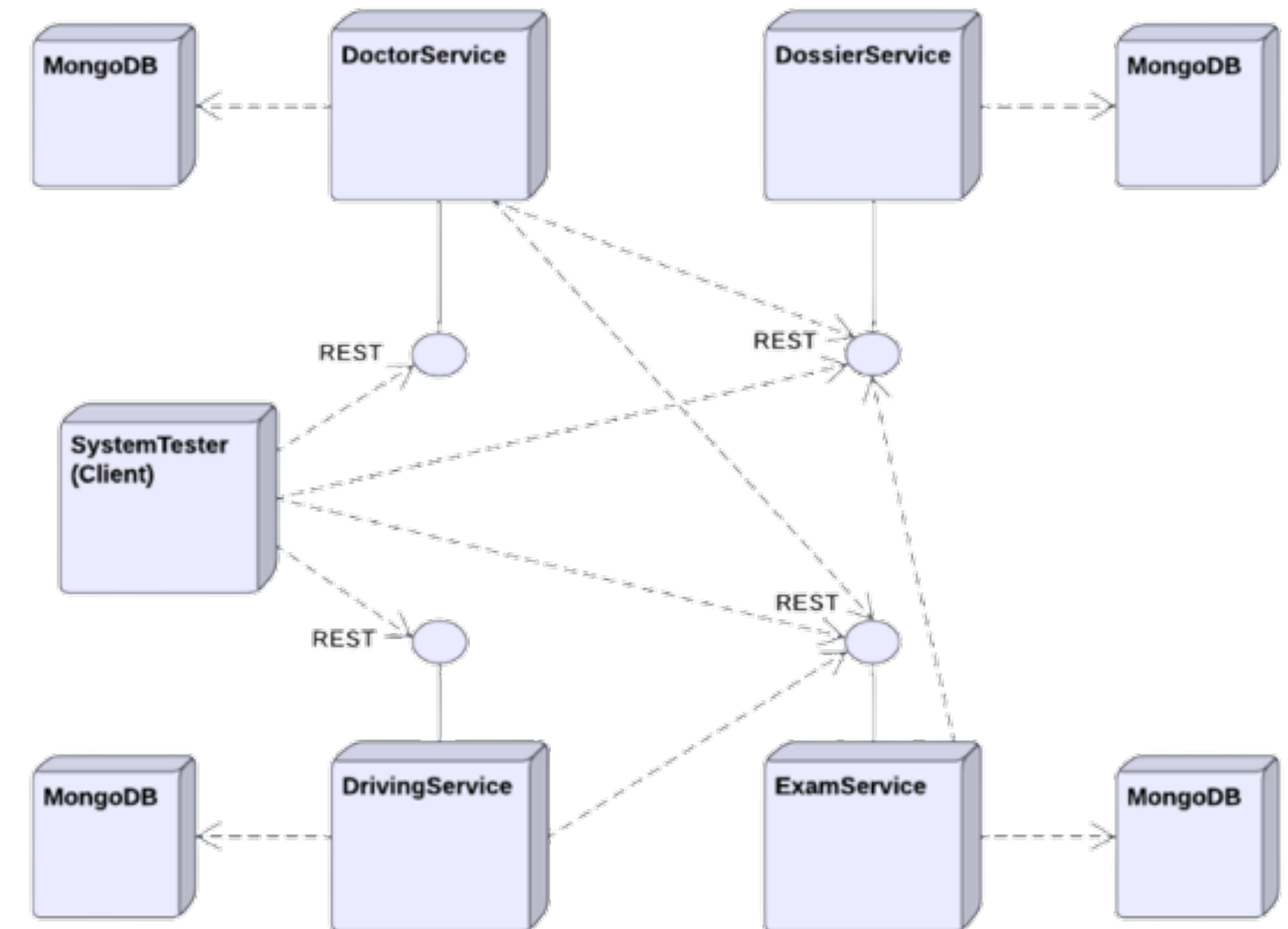
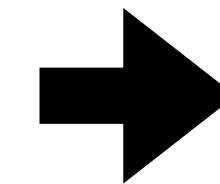
- Supporto nativo nel Kotlin
 - Semplificazione del codice
 - **Suspending** function
-
- Astrazioni utilizzate:
 - VertX -> **CoroutineVerticle**
 - KMongo -> **Coroutines client**



Testing

Acceptance Test Driven Development (ATDD)

- **Unit Testing:**
 - Testing dei componenti in maniera isolata
 - Testing di porzioni critiche del codice
- **Black-box testing:**
 - **System Tester** è il componente chiave che simula un web client reale.
 - ATDD con Gherkin: testing del sistema rispetto alle esigenze e alle aspettative degli utenti



Deployment

Docker, Continuous Integrations e Semantic Release

- **Docker compose**
 - Un container per ogni microservizio: Dossier, Driving, Exam, Doctor
 - Un container per SystemTester
 - Un container per una istanza di MongoDB che fa hosting per 4 database indipendenti.
- **GitHub Actions Workflow**
 - Build and Test: produzione degli artefatti, esecuzione e testing del sistema
 - Release: release automatico Semantic-Release
 - SwaggerHub Publish: unione, validazione e pubblicazione della documentazione



Conclusioni e Sviluppi futuri

- Utilizzo e manutenzione futura: pubblicazione OpenAPI e Swagger
- Aggiunta di un middleware tra Client e Servizi: Gateway
 - Gestire il traffico in entrata e uscita
 - Maggiore disaccoppiamento client-servizi:
 - Rotte private per la comunicazione tra microservizi interni
 - Rotte pubbliche e accessibili dai client, a discapito del singolo microservizio
- Espansione in supporto al dominio in esame: pazienti, quiz, vetture, corsi professionali...

