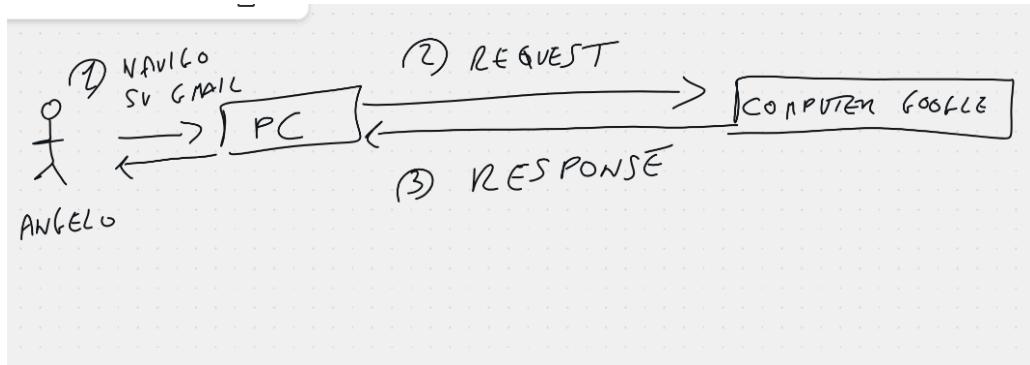
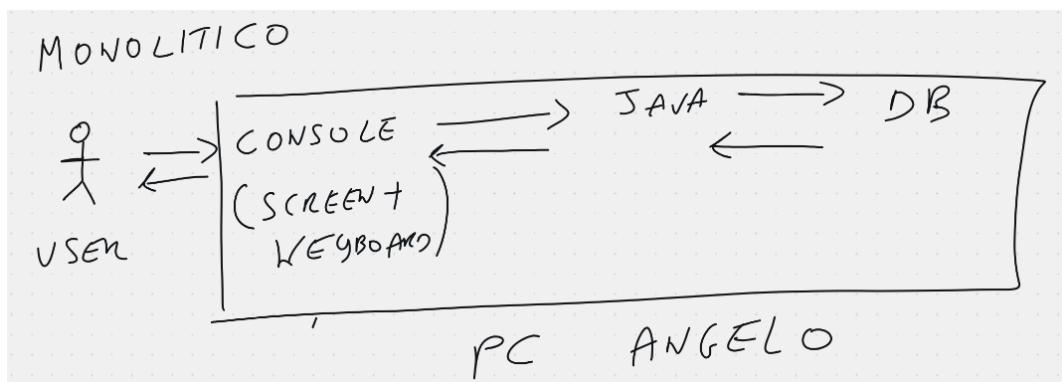


Appunti 2/02/26 - BRUTTA



I programmi per come li abbiamo visti fino ad adesso erano CLIENT ONLY - monolitico, tutto quanto in locale (perché lo hai fisicamente davanti a te - memoria, i byte, il programma) Se do fuoco al computer, perdo il database.



Alcuni computer forniscono un servizio e altri lo consumano → nell'era del web

Client, Server, Request e Response

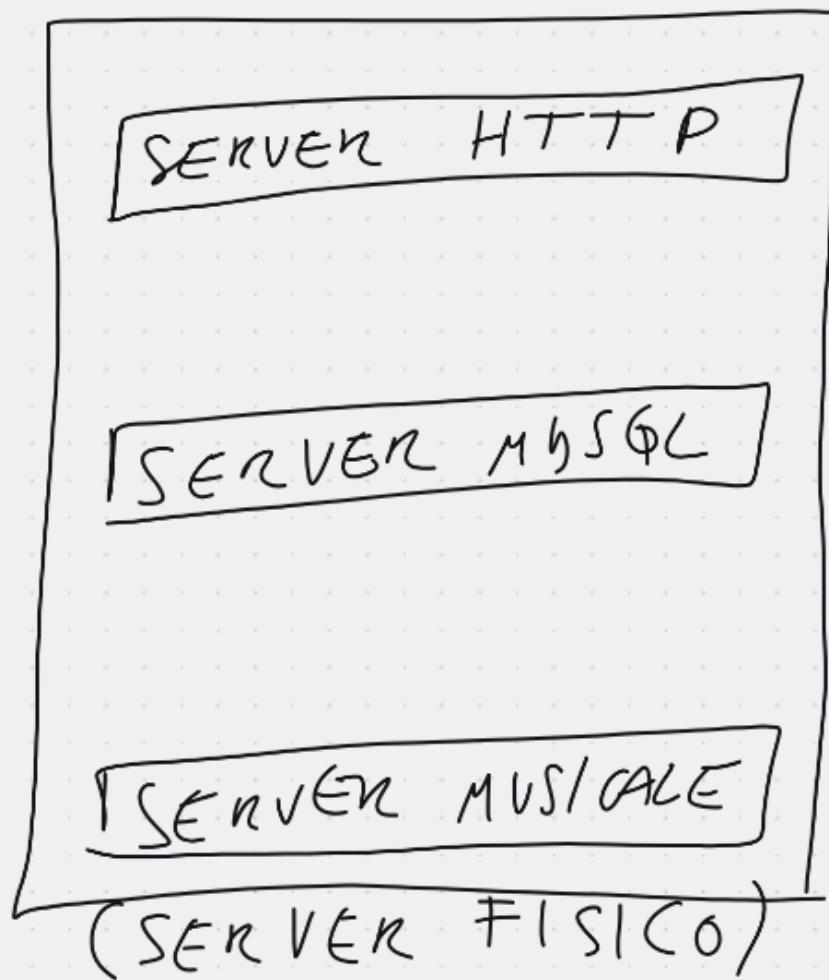
1. **Client** → Il client è un programma o dispositivo che richiede servizi o risorse da un altro computer. Nello schema, il PC è il client. Può essere un browser web, un'app mobile, un programma da terminale - qualsiasi cosa che inizia una comunicazione chiedendo qualcosa. Il client è "chi domanda"
2. **Server** → Il server è un computer o programma che fornisce servizi o risorse ad altri computer. Nel disegno, i "Computer Google" sono i server. Il

server aspetta richieste, le processa, e risponde. È "chi risponde e fornisce".

3. **Request (Richiesta)** → La request è un messaggio inviato dal client al server per chiedere un'azione o dei dati. Contiene informazioni su cosa vuoi (URL/endpoint), come lo vuoi (metodo HTTP: GET, POST, PUT, DELETE), e eventuali dati aggiuntivi (parametri, body, headers). **Response (Risposta)** → La response è il messaggio che il server rimanda al client dopo aver elaborato la richiesta.

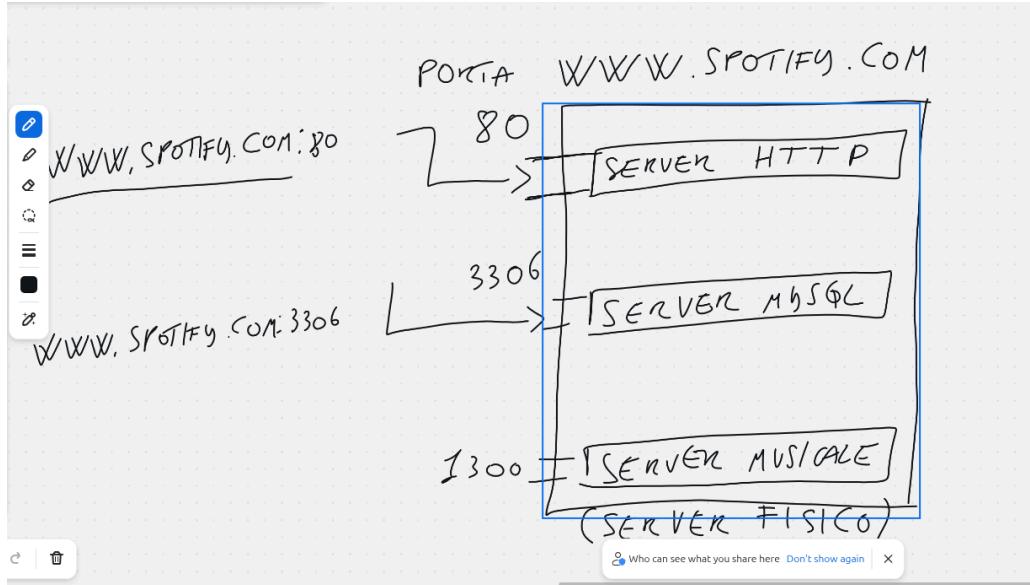
Un **server fisico** è semplicemente una macchina - un computer con CPU, RAM, disco, scheda di rete. È l'hardware tangibile che sta in un datacenter. Questo singolo computer può ospitare **multipli server software** contemporaneamente, ognuno che fa cose diverse.

WWW.SPOTIFY.COM



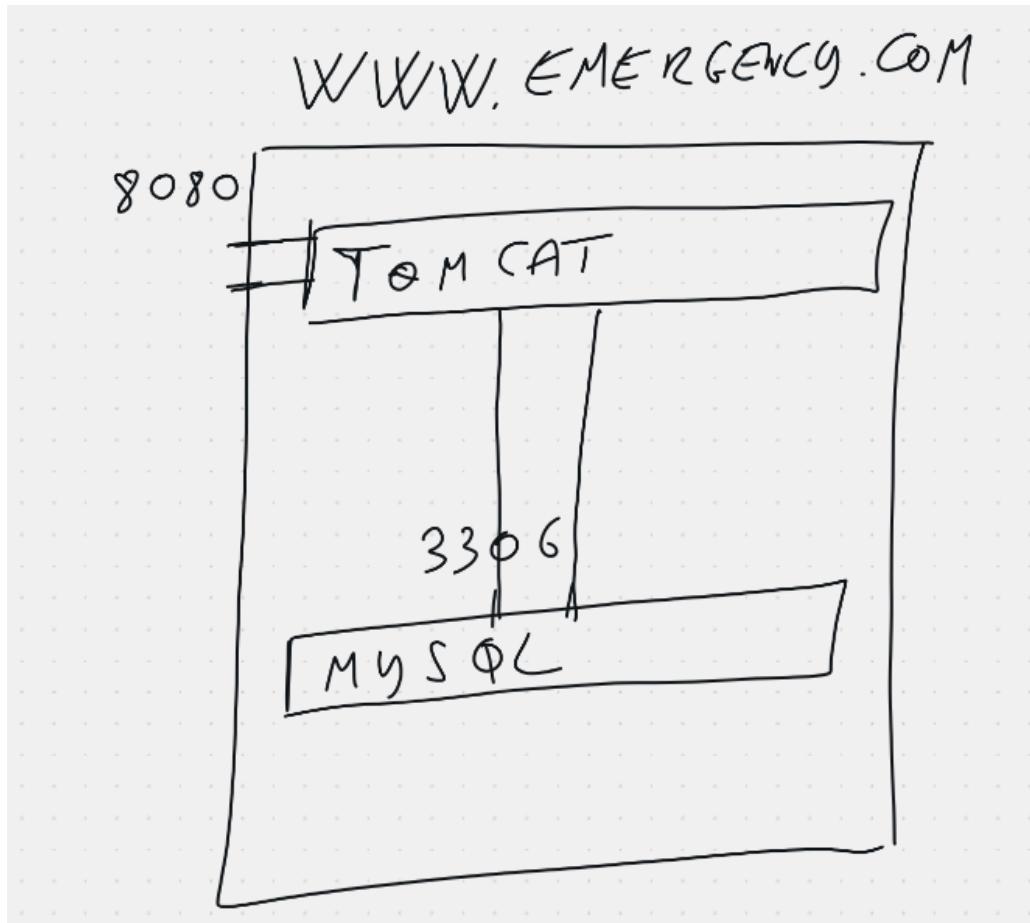
Who can see what you share here [Don't show again](#)



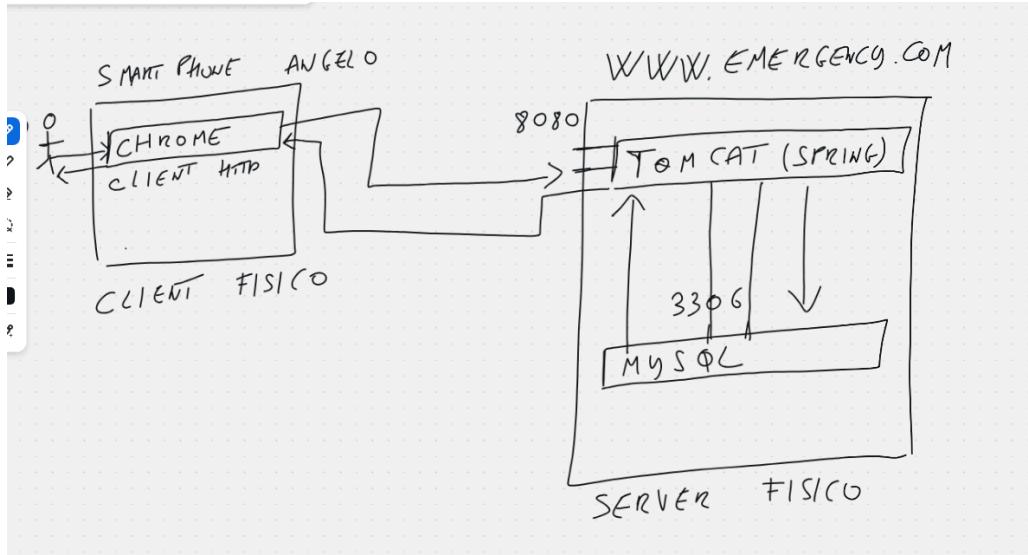


Una porta è un numero che identifica quale server software deve ricevere la richiesta su un server fisico.

(Serveri fisico —)

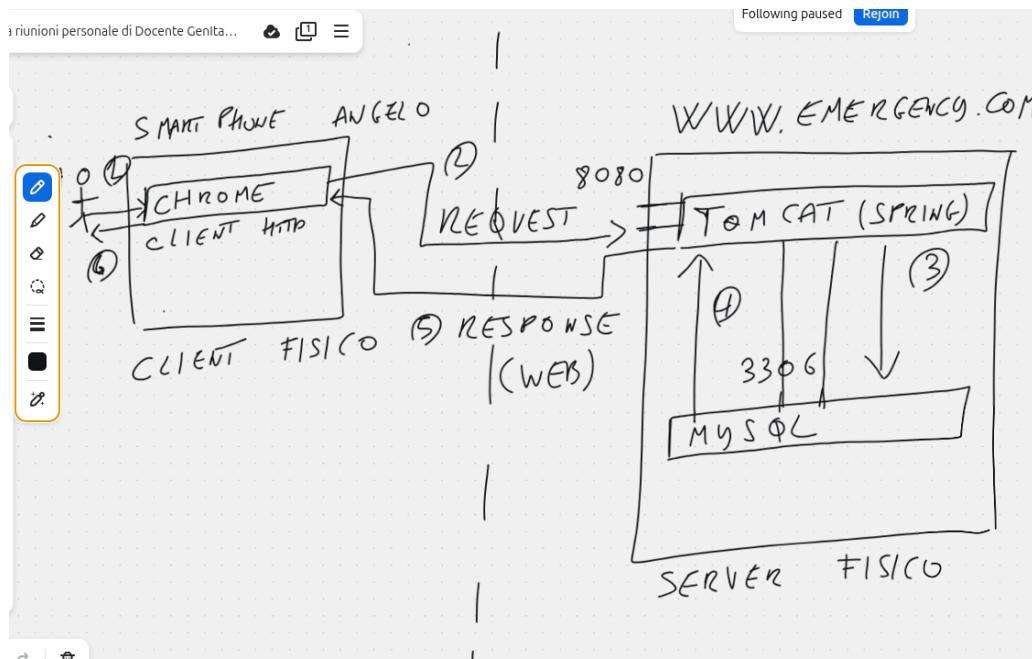


www.emergency.com, solo la porta 8080 è accessibile da Internet, mentre la porta 3306 è chiusa al traffico esterno. Questo è un principio di sicurezza chiamato network segmentation o firewall rules. Se la porta MySQL fosse pubblica, chiunque su Internet potrebbe tentare di connettersi al database



Il Ciclo Client-Server Completo

(utente umano) interagisce con lo Smartphone (client fisico), che esegue Chrome (client HTTP software). Chrome è responsabile di tradurre le azioni dell'Utente in richieste HTTP e di visualizzare le risposte in modo comprensibile. Le frecce che collegano Chrome a Tomcat rappresentano il traffico HTTP bidirezionale attraverso Internet: Freccia destra (→): Request da Chrome a Tomcat sulla porta 8080, Freccia sinistra (←): Response da Tomcat a Chrome. Lato Server → Il server fisico di www.emergency.com ospita due server software: Tomcat (Spring) sulla porta 8080: Riceve le request HTTP da Chrome, Esegue la business logic (validazione, autenticazione, calcoli), Comunica con MySQL per operazioni sui dati, Costruisce e invia le response a Chrome.



Il Ciclo Request-Response Completo → Angelo è seduto con il suo smartphone e decide di accedere all'applicazione Emergency.com. Tocca il display per eseguire un'azione specifica, ad esempio visualizzare la lista delle emergenze attive nella sua zona. Questo tocco sullo schermo è il passaggio 1 dello schema - l'input dell'utente che avvia l'intero processo di comunicazione client-server.

Lo smartphone è il client fisico, ma il vero protagonista della comunicazione è Chrome, il browser web installato sul dispositivo. Chrome è un client HTTP software che ha la responsabilità di gestire tutta la comunicazione con il server remoto. Nel passaggio 2, Chrome costruisce una richiesta HTTP. Questa richiesta contiene il metodo HTTP appropriato (GET se Angelo vuole recuperare dati, POST se deve inviarne di nuovi), l'URL completo che specifica l'indirizzo del server e la risorsa richiesta, una serie di headers che forniscono metadati sulla richiesta, e eventualmente un body con dati aggiuntivi. Chrome invia questa richiesta attraverso la rete Internet verso il server remoto.

La richiesta attraversa la rete e raggiunge il server fisico che ospita www.emergency.com. Il server riceve il pacchetto sulla porta 8080, che è la porta su cui Tomcat è configurato per ascoltare le connessioni in entrata. Tomcat è un application server che esegue codice Java con il framework Spring. Nel passaggio 3, Tomcat riceve la richiesta HTTP e inizia a processarla. Esegue diverse operazioni: verifica l'autenticazione dell'utente controllando i

token o le credenziali presenti negli headers, valida i parametri della richiesta per assicurarsi che siano nel formato corretto, e applica la logica di business specifica dell'applicazione Emergency.

Durante il processamento, Tomcat determina che ha bisogno di accedere ai dati persistenti memorizzati nel database. Nel passaggio 4, Tomcat apre una connessione con MySQL sulla porta 3306. Questa connessione avviene internamente al server fisico, utilizzando l'indirizzo localhost o l'IP privato della macchina. Tomcat invia una query SQL a MySQL per recuperare i dati necessari. MySQL esegue la query, cerca nei suoi file di database, applica eventuali indici per ottimizzare la ricerca, e restituisce il risultato a Tomcat. Questa comunicazione tra Tomcat e MySQL è bidirezionale e non è accessibile dall'esterno perché la porta 3306 non è esposta pubblicamente.

La Porta MySQL Non È Visibile dall'Esterno

Il primo concetto è che la porta 3306 di MySQL non è esposta a Internet. Questo significa che un attaccante esterno non può nemmeno tentare di connettersi direttamente al database. La configurazione del firewall sul server fisico blocca tutte le connessioni in entrata sulla porta 3306 che provengono da indirizzi IP esterni. Solo le connessioni che originano dall'interno del server stesso, tipicamente da localhost (127.0.0.1), possono raggiungere MySQL. Quindi quando Tomcat vuole comunicare con MySQL, lo fa attraverso una connessione locale che non esce mai dalla macchina fisica.

MySQL È Chiamabile Internamente

Il secondo punto è che MySQL rimane completamente funzionale e accessibile per i processi che girano sullo stesso server. Tomcat, che è in esecuzione sullo stesso server fisico, può aprire connessioni TCP verso localhost:3306 senza alcun problema. La porta 3306 è in ascolto, MySQL sta ricevendo connessioni, ma solo da processi locali. Questa è la differenza tra una porta chiusa (che non accetta connessioni da nessuno) e una porta con accesso limitato (che accetta solo connessioni specifiche).

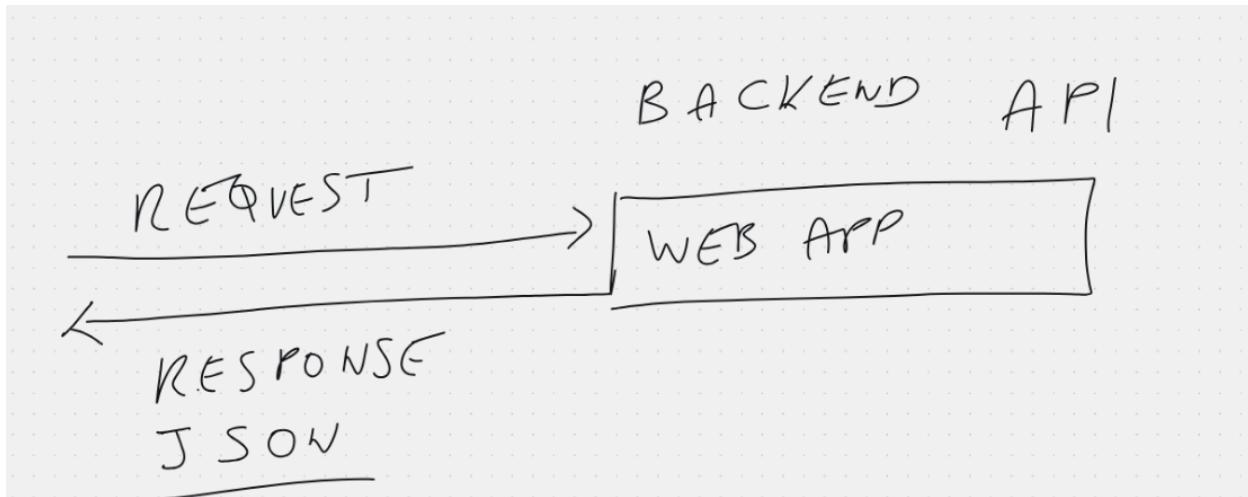
Non Esiste Sicurezza Lato Client

Il terzo concetto è quello più critico per lo sviluppo di applicazioni sicure. Qualsiasi dato o codice che viene inviato al client è completamente accessibile e modificabile dall'utente. Se Chrome riceve del codice JavaScript, l'utente può aprire gli strumenti di sviluppo del browser, leggere tutto il codice, modificarlo, e eseguirlo alterato. Se l'applicazione web invia dati sensibili al client pensando che il JavaScript li proteggerà, un utente malintenzionato può semplicemente ignorare quella protezione. Può modificare le richieste HTTP prima che vengano inviate al server, può falsificare token, può manipolare qualsiasi dato che passa attraverso il browser. Il client è completamente sotto il controllo dell'utente finale.

Per questo motivo, tutta la vera sicurezza deve essere implementata sul server. Quando Tomcat riceve una richiesta, deve sempre validare tutto: l'autenticazione dell'utente deve essere verificata sul server, le autorizzazioni devono essere controllate sul server, i dati in input devono essere validati e sanitizzati sul server. Anche se l'applicazione JavaScript nel browser sembra permettere solo certe operazioni, il server deve comportarsi come se qualsiasi richiesta potesse arrivare, perché un attaccante può bypassare completamente il client e inviare richieste HTTP dirette usando strumenti come curl o Postman.

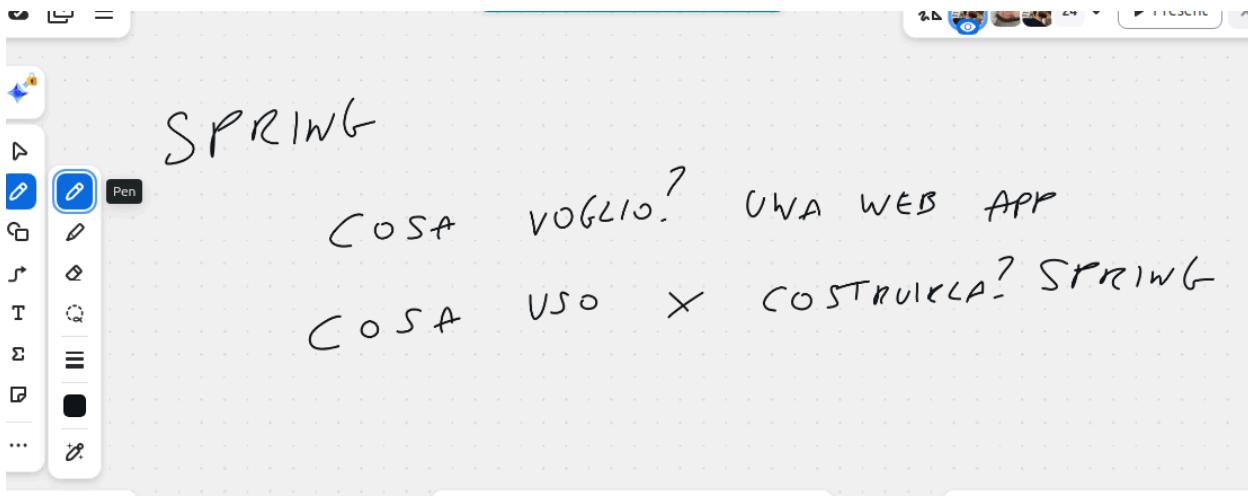
La Sicurezza Fisica del Datacenter

Il professore menziona anche che il server fisico si trova in un datacenter protetto fisicamente. Questo aggiunge un ulteriore livello di sicurezza che il client non avrà mai. Il server è in una struttura con controllo degli accessi fisici, personale di sicurezza, telecamere, sistemi antincendio, alimentazione ridondante, e connettività di rete controllata. Il client invece è uno smartphone o un laptop che potrebbe essere ovunque, connesso a qualsiasi rete WiFi pubblica insicura, potenzialmente già compromesso da malware. Questa asimmetria fondamentale tra la sicurezza del server e quella del client è il motivo per cui l'architettura delle applicazioni web è costruita sul principio "never trust the client".



Backend API e JSON

Backend API → Il termine "Backend API" si riferisce alla parte server dell'applicazione che espone delle interfacce programmatiche. L'API (Application Programming Interface) è un insieme di endpoint che il client può chiamare per eseguire operazioni specifiche. In questo caso, la "WEB APP" rappresenta il server backend che riceve richieste HTTP e risponde con dati strutturati. La freccia di ritorno indica che la RESPONSE viene inviata in formato JSON. JSON (JavaScript Object Notation) è un formato di scambio dati testuale, leggibile sia dalle macchine che dagli umani. È diventato lo standard de facto per le API web moderne perché è leggero, facile da parsare, e indipendente dal linguaggio di programmazione. Un framework è un sistema preconfezionato che fornisce una struttura di base già pronta. Invece di scrivere tutto il codice da zero, tu lavori dentro una cornice già esistente che ha già risolto i problemi comuni dello sviluppo web. Il framework fornisce l'architettura, le librerie, e i pattern già implementati, e tu ti concentri sulla logica specifica della tua applicazione.



Framework Spring

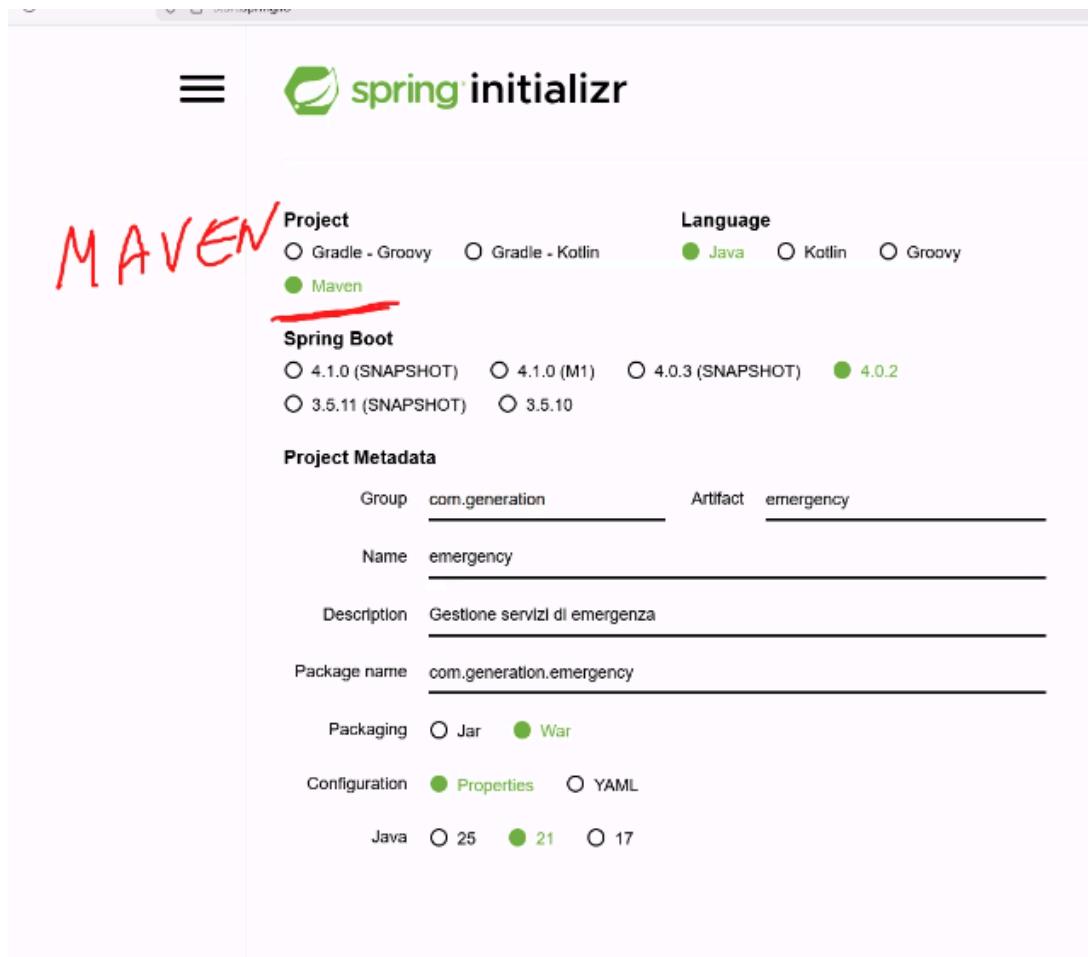
1. Come Funziona il Lavoro con un Framework Quando usi un framework come Spring, tu scrivi il tuo codice dentro la struttura che il framework ti fornisce. Spring ha già implementato la gestione delle richieste HTTP, il routing degli URL verso i metodi corretti, la connessione ai database, la gestione delle transazioni, la sicurezza, e molte altre funzionalità complesse. Tu devi solo personalizzare questa base scrivendo il codice specifico per la tua applicazione Emergency.com. Spring per le Web App Se vuoi creare una web app, Spring è una delle scelte più popolari nell'ecosistema Java. Spring Boot, in particolare, semplifica ulteriormente la configurazione fornendo convenzioni predefinite che funzionano nella maggior parte dei casi. Puoi creare un'applicazione web completa scrivendo relativamente poco codice perché Spring gestisce automaticamente tutta l'infrastruttura sottostante. Il Vantaggio Pratico Invece di dover implementare manualmente come Tomcat riceve le richieste HTTP, come parsare il JSON, come gestire le connessioni al database MySQL, come gestire la sicurezza delle password, Spring fa tutto questo per te. Tu ti concentri su scrivere i controller che definiscono gli endpoint della tua API, i service che contengono la logica di business, e le entità che rappresentano i tuoi dati. Il framework collega automaticamente tutti questi pezzi e li fa funzionare insieme.

Maven

Maven è un sistema di gestione e automazione dei progetti Java che risolve problemi fondamentali dello sviluppo software. La funzione principale di Maven è

gestire le dipendenze del progetto. Quando sviluppi un'applicazione Spring, hai bisogno di decine di librerie esterne: Spring Framework stesso, librerie per connettersi a MySQL, librerie per gestire JSON, librerie per la sicurezza, e molte altre. Senza Maven, dovresti scaricare manualmente ogni file JAR, verificare che le versioni siano compatibili tra loro, e aggiungerle tutte al classpath del progetto. Maven automatizza completamente questo processo. Packaging del Progetto
Maven gestisce anche il processo di build e packaging. Quando hai finito di sviluppare la tua applicazione Emergency, Maven compila tutto il codice sorgente Java, esegue i test automatici, e crea un file WAR (Web Application Archive) che contiene l'intera applicazione pronta per essere distribuita su Tomcat. Questo pacchetto può essere condiviso con altri sviluppatori o deployato sui server di produzione. Standardizzazione della Struttura

Maven impone una struttura di cartelle standardizzata per i progetti Java. Il codice sorgente va in src/main/java, i file di configurazione in src/main/resources, i test in src/test/java. Questa convenzione significa che qualsiasi sviluppatore Java che apre un progetto Maven sa immediatamente dove trovare ogni cosa, indipendentemente da chi ha creato il progetto. L'Origine del Nome
Il nome Maven viene effettivamente dall'ebraico e significa "accumulatore di conoscenza". Questo riflette la filosofia dello strumento: accumula tutte le librerie e le conoscenze necessarie per costruire il progetto, centralizzando la gestione delle dipendenze e delle configurazioni in un unico posto



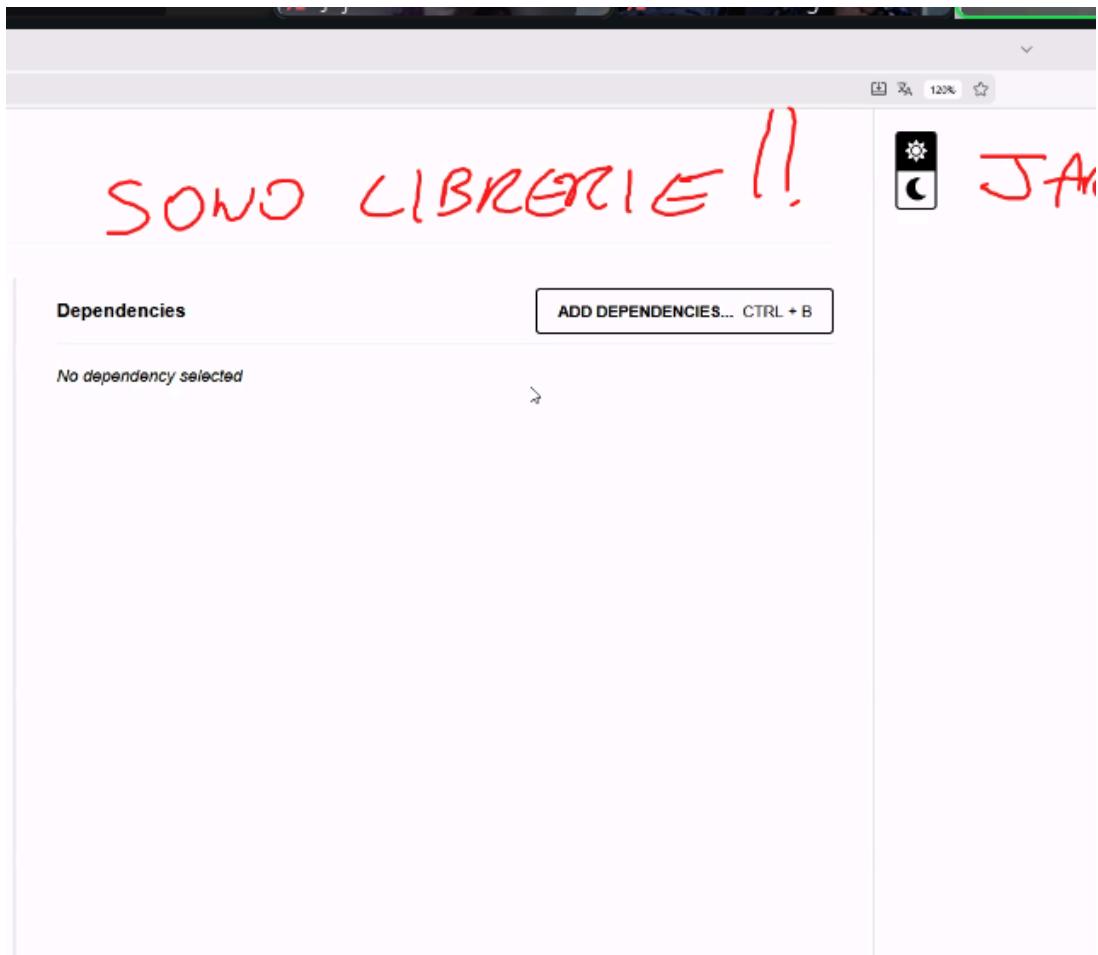
Dependencies = Librerie

Dependencies = Librerie

Quando parliamo di "dependencies" nel contesto di Maven e Spring, stiamo parlando di librerie esterne - file JAR che contengono codice già scritto da altri sviluppatori che puoi usare nel tuo progetto. Non hanno nulla a che fare con il concetto di dipendenza che hai visto in altri contesti come JDBC.

Nel contesto di Maven, una dependency è semplicemente una libreria di terze parti che il tuo progetto utilizza. È un termine tecnico specifico dell'ecosistema Java e dei build tools. Quando aggiungi una "dependency" al tuo progetto Maven, stai dicendo: "Il mio progetto ha bisogno di questa libreria per funzionare". Differenza da JDBC → In JDBC, quando parlavamo di dipendenze, ci riferivamo a dipendenze funzionali o a relazioni tra componenti software. Qui invece è puramente una questione di packaging e distribuzione del codice. Le

dependencies di Maven sono pacchetti di codice compilato che vengono scaricati e inclusi nel tuo progetto per fornire funzionalità specifiche.



CONFIGURAZIONE DI SPRING BOOT

```
spring.application.name=emergency
```

```
r1_0.author,
r1_0.content,
r1_0.score,
r1_0.title
from
review r1_0
where
r1_0.movie_id=?
```

LIB DBMS MIO STESSO PC

```
spring.application.name=emergency
# MySQL Database Configuration
spring.datasource.url=jdbc:mysql://localhost:3306/emergency?useSSL=false&serverTimezone=UTC
spring.datasource.username=root
spring.datasource.password=hf5q012!lk
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
# Hibernate settings for automatic schema creation
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
# Use MySQL dialect
spring.jpa.database-platform=org.hibernate.dialect.MySQLDialect
```

```
r1_0.author,
r1_0.content,
r1_0.score,
r1_0.title
from
review r1_0
where
r1_0.movie_id=?
```

Application Properties e Spring Boot

il file `application.properties` che vedi evidenziato nella struttura del progetto è il punto centrale di configurazione dell'applicazione Spring Boot. Questo file di testo contiene coppie chiave-valori che definiscono come l'applicazione deve comportarsi. Una volta compilato il file, ho la connection, creata da una factory ed è già nel context. Il Package Principale → Quando lavori con Spring Boot, devi sempre inserire le tue classi nel package principale o nei suoi sub-package. Spring Boot è essenzialmente Spring Framework con una configurazione predefinita che funziona out-of-the-box. Invece di dover configurare manualmente centinaia di opzioni come si faceva con Spring tradizionale, Spring Boot fornisce convenzioni sensate che coprono la maggior parte dei casi d'uso comuni.

Le Annotations Come Metacodice

Le annotations sono informazioni aggiuntive che scrivi nel codice Java precedute dal simbolo `@`. Non sono codice eseguibile vero e proprio, ma metadati che Spring legge per capire come comportarsi. Le annotations dicono a Spring cosa fare con le tue classi senza che tu debba scrivere configurazioni esplicite

Le Annotations Più Importanti in Spring

Le annotations fondamentali che userai costantemente sono `@Entity` per marcare una classe come entità del database mappata da JPA, `@Repository` per indicare che un'interfaccia gestisce l'accesso ai dati, `@Service` per le classi che contengono business logic, `@Controller` o `@RestController` per le classi che gestiscono le richieste HTTP, `@Autowired` per iniettare automaticamente le dipendenze, `@Id` per marcare il campo che rappresenta la chiave primaria, `@GeneratedValue` per indicare che il valore viene generato automaticamente, e `@Table` per specificare il nome della tabella nel database.

Estensione di JpaRepository

La dichiarazione `extends JpaRepository<Hospital, Integer>` significa che `HospitalRepository` eredita tutti i metodi da `JpaRepository`. Questa interfaccia fornita da Spring contiene già tutti i metodi standard per operazioni CRUD: `save()`, `findById()`, `findAll()`, `delete()`, `deleteById()`, e molti altri. Tu ottieni tutti questi metodi gratuitamente solo estendendo l'interfaccia

I Due Parametri Generics

I Due Parametri Generics JpaRepository<Hospital, Integer> richiede due parametri tra parentesi angolari. Il primo parametro Hospital è il tipo dell'entità che questo repository gestisce. Deve essere una classe annotata con @Entity che rappresenta una tabella nel database. Il secondo parametro Integer è il tipo della chiave primaria di quella entità. Se l'entità Hospital ha un campo id di tipo Integer annotato con @Id, allora specifichiamo Integer come secondo parametro.

il Controller Come Punto di Ingresso il controller è il componente che si trova al confine tra il mondo esterno e la tua applicazione. Quando un client, che sia un browser Chrome, un'app mobile, o qualsiasi altro programma, invia una richiesta HTTP al server, quella richiesta arriva direttamente al controller. Il controller è l'unico punto di contatto che il mondo esterno ha con la tua applicazione backend. Il controller è annotato con @RestController o @Controller e contiene metodi che sono mappati a specifici endpoint URL. Quando arriva una richiesta HTTP a un determinato path, ad esempio GET /api/hospitals, Spring routing dirige automaticamente quella richiesta al metodo corrispondente nel controller. Il controller riceve tutti i dati della richiesta: parametri nell'URL, dati nel body JSON, headers HTTP, e qualsiasi altra informazione che il client ha inviato.

Il Meccanismo di Autowiring

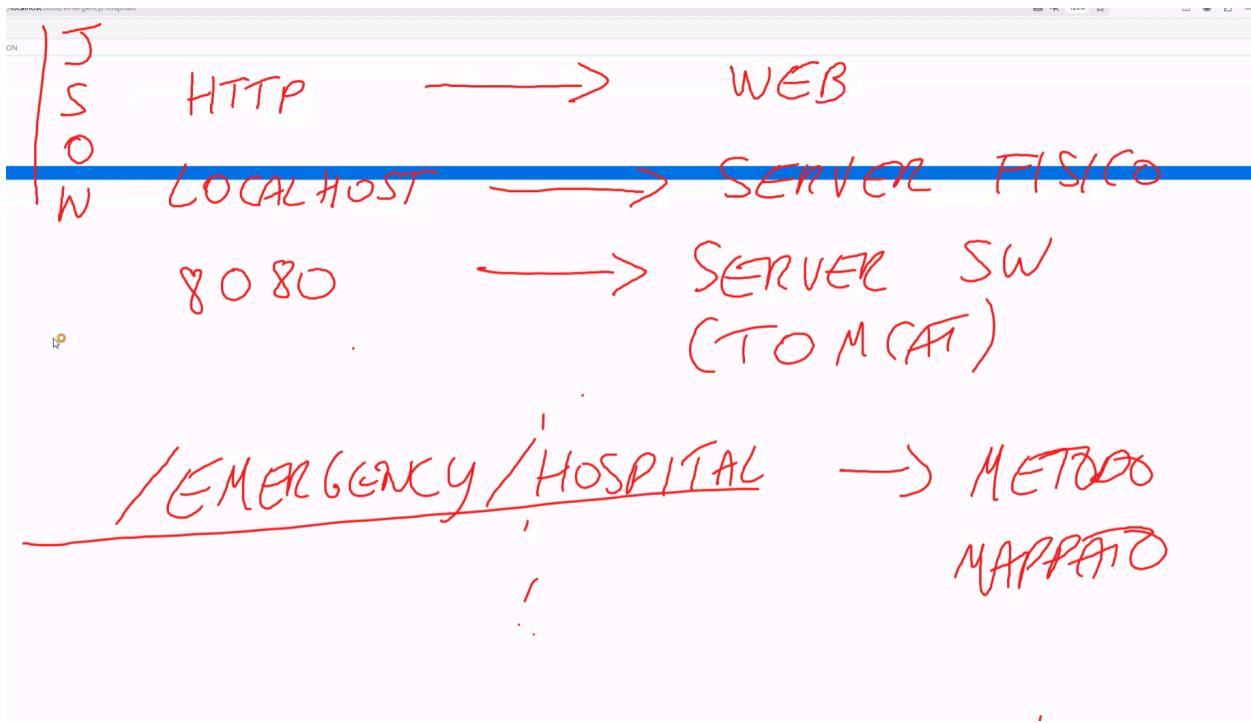
Quando Spring avvia l'applicazione, crea prima l'oggetto HospitalRepository generando la sua implementazione automatica, poi crea l'oggetto HospitalAPI del controller, e infine inietta il repository nel campo repo del controller. Questo avviene automaticamente senza che tu debba scrivere codice per istanziare o passare oggetti. Spring gestisce tutto il grafo delle dipendenze e assicura che ogni bean abbia accesso agli altri bean di cui ha bisogno.

API REST

concetto centrale delle API REST: associare operazioni del codice Java a indirizzi web accessibili dall'esterno. Quando hai un repository con il metodo findAll() che recupera tutti gli ospedali dal database, quel metodo vive all'interno dell'applicazione Java e non è direttamente accessibile da un browser o da un'app mobile. Devi creare un ponte tra il mondo esterno, che

parla HTTP, e il mondo interno dell'applicazione, che parla Java. Questo ponte viene costruito usando l'annotation @GetMapping("/emergency/hospitals") che vedi alla riga 26. Quando scrivi questa annotation sopra un metodo del controller, stai dicendo a Spring di creare una connessione precisa: ogni volta che arriva una richiesta HTTP GET all'indirizzo /emergency/hospitals, deve essere eseguito esattamente quel metodo. Il client non sa nulla del repository, delle entità, o di Hibernate che lavorano dietro le quinte, vede solo un URL che può chiamare

Il mapping degli endpoint con @GetMapping crea il ponte tra URL HTTP e metodi Java. Quando una richiesta arriva a /emergency/hospitals, Spring la instrada automaticamente al metodo corrispondente nel controller. Quel metodo chiama repo.findAll(), il repository comunica internamente con MySQL, Hibernate esegue la query SQL e mappa le righe in oggetti Java, il controller restituisce la lista, e Spring la serializza automaticamente in JSON usando Jackson. Il risultato finale che vedete nel browser è JSON puro, dati grezzi strutturati che qualsiasi frontend può consumare. Questa separazione tra backend e frontend è il cuore dell'architettura REST moderna. Il backend si concentra esclusivamente sulla logica di business e sulla gestione dei dati, fornendo API HTTP che restituiscono JSON. Il frontend, che sia un'applicazione React, Angular, Vue, o un'app mobile nativa, chiama queste API, riceve il JSON, e decide autonomamente come presentarlo visualmente all'utente.



Schema Completo dell'Architettura

Lo schema mostra tre livelli di comunicazione che rappresentano l'intero stack dell'applicazione Emergency. Nella parte superiore vedi "JSON HTTP → WEB", che rappresenta la comunicazione esterna tra il client e il server web attraverso il protocollo HTTP. Il JSON viaggia attraverso Internet raggiungendo il server web pubblicamente accessibile.

La scritta "LOCALHOST" indica che tutto ciò che sta sotto questa linea avviene internamente alla macchina server, senza uscire su Internet. La porta "8080" punta verso "SERVER SW (TOMCAT)", mostrando che Tomcat riceve le richieste HTTP esterne su quella porta specifica. Il Mapping degli Endpoint → L'URL /emergency/hospital viene associato a un metodo specifico nel controller Java. Quando arriva una richiesta HTTP a quell'URL, Spring instrada automaticamente la chiamata al metodo mappato che esegue repo.findAll() e restituisce i dati degli ospedali. Un client esterno invia una richiesta HTTP con JSON, questa attraversa Internet e arriva alla porta 8080 dove Tomcat è in ascolto. Tomcat passa la richiesta a Spring, che la instrada al controller corretto basandosi sul path /emergency/hospital. Il controller esegue il metodo mappato, che internamente comunica con il database MySQL, recupera i dati, e

li restituisce. Spring serializza la risposta in JSON e la invia indietro attraverso HTTP al client.

il Concetto di Mappatura

Un controller può collegare un metodo Java a un indirizzo web specifico attraverso la mappatura. Quando scrivi `@GetMapping("/emergency/hospitals")` sopra un metodo, stai creando un'associazione diretta: quel metodo verrà eseguito ogni volta che arriva una richiesta HTTP GET all'URL `/emergency/hospitals`.

Le Annotation per la Mappatura`@GetMapping` è una shortcut annotation che internamente equivale a `@RequestMapping(method = RequestMethod.GET)`. Spring fornisce diverse annotation specializzate: `@GetMapping` per le richieste GET che recuperano dati, `@PostMapping` per le richieste POST che creano nuove risorse, `@PutMapping` per aggiornamenti, e `@DeleteMapping` per cancellazioni. Queste annotation rendono il codice più leggibile e esplicito rispetto a usare sempre `@RequestMapping` con l'attributo method specificato manualmente.

Il Meccanismo di Spring

1. Il Meccanismo di Spring
Quando l'applicazione si avvia, Spring scansiona tutti i controller cercando le annotation di mappatura. Costruisce internamente una routing table che associa ogni pattern URL al metodo corrispondente. Quando arriva una richiesta HTTP, Spring esamina il metodo (GET, POST, etc.) e il path dell'URL, cerca nella routing table la corrispondenza esatta, e invoca il metodo Java associato. Il valore restituito dal metodo viene automaticamente serializzato in JSON se il controller è annotato con `@RestController`, e la risposta viene inviata al client.
La mappatura è quindi il ponte tra il mondo HTTP esterno e i metodi Java interni, permettendo al tuo codice di rispondere a richieste web specifiche in modo dichiarativo e semplice.