
0.1 Introduzione

La web application è strutturata secondo un'**architettura three-tier**.

Il Client non comunica direttamente con il server del database, ma con il server dell'applicazione. Il Client svolge solo il compito di interfaccia utente e la logica dell'applicazione è inserita nel server applicativo.

Server applicativo e server di database possono risiedere nella stessa macchina o su macchine diverse collegate in rete: nel nostro caso risiedono entrambi su **Azure Cloud**, ma su due locazioni differenti.

Il Client è un semplice browser web, mentre il server applicativo è un server web: come server applicativo è stato utilizzato **Apache Tomcat 9.0**.

Per quanto riguarda il server del database, invece, è stato adoperato come DBMS **MySQL 8.0**.

Il web browser del client invia le proprie richieste, tramite il protocollo HTTP/HTTPS, al livello intermedio ovvero al web server. Quest'ultimo interpreta e serve tali richieste, interagendo anche con il DBMS, per poi generare una risposta in formato XML da inviare allo stesso browser, che la interpreterà e la presenterà all'utente sotto forma di Web.

D'altra parte l'applicazione fa ricorso ad un servizio esterno, **Janus**, per implementare la funzionalità di videochiamata, il quale comunicherà direttamente con il Client.

0.2 Component Diagram

Per illustrare l'architettura del sistema, si è fatto ricorso al Component Diagram, il quale è un diagramma che ha lo scopo di rappresentare la struttura interna del sistema software modellato in termini dei suoi componenti principali e delle relazioni fra di essi. Per componente si intende una unità software dotata di una precisa identità, nonché responsabilità e interfacce ben definite.

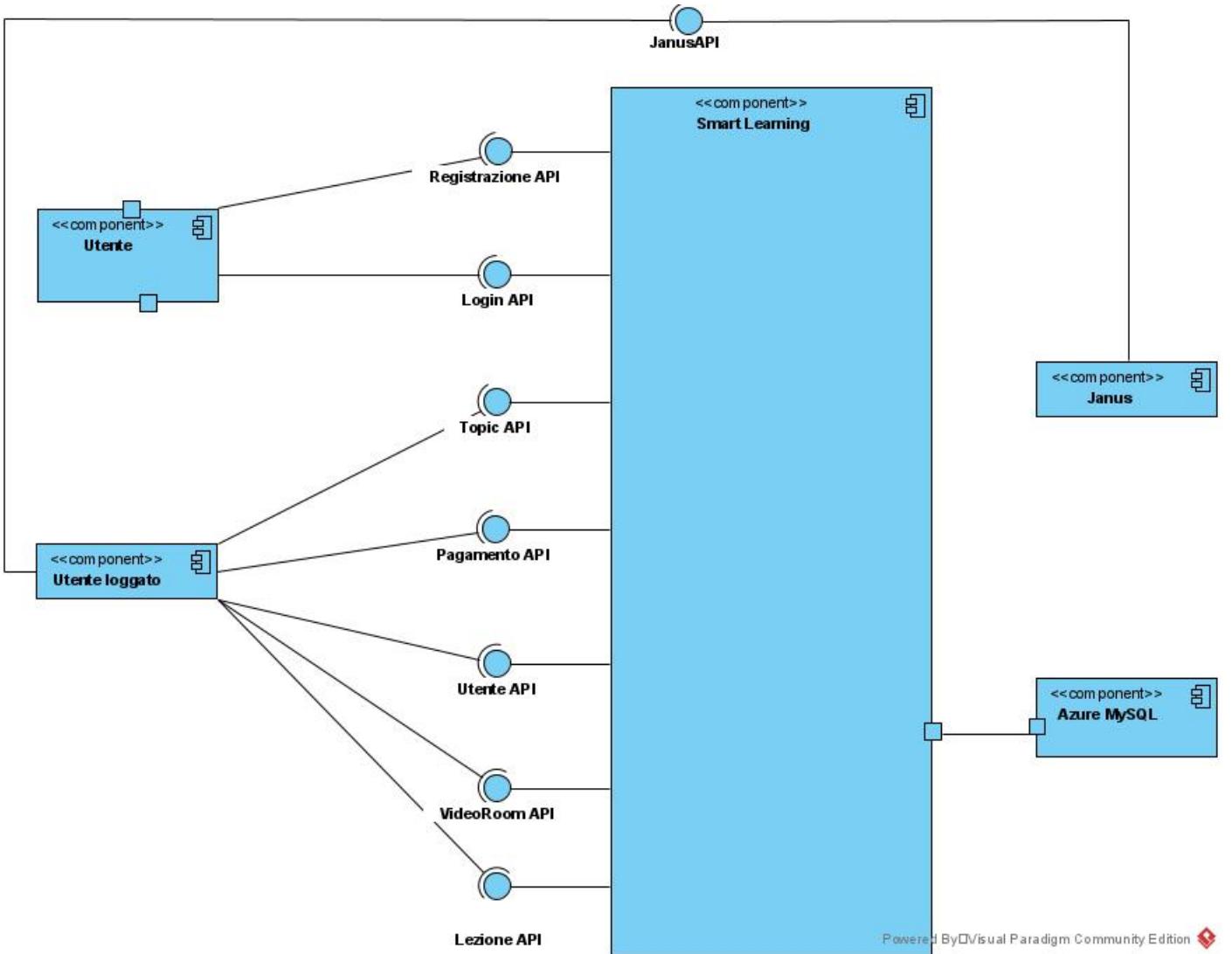


Figura 1: Component Diagram

Com'è possibile evincere dalla figura, è possibile scomporre il sistema nei seguenti componenti:

- Una Web Application: Smart Learning;
- Due User Interface: Utente ed Utente loggato;
- Un Service (servizi esterni adoperati dal sistema): Janus;
- Un Database (necessario per gestire la persistenza dei dati): MySQL;
- Sette API interposte tra Client e Web Application (alcune ulteriormente scomponibili in REST API):
 - Registrazione API (REST API);
 - Login API (REST API);
 - Utente API:
 - * Utente API (REST API);
 - * UpgradeToDocente API (REST API);
 - Lezione API:
 - * LezioniPrenotate API (REST API);

- * Programmazioni API (REST API);
- * LezioniUtente API (REST API);
- * ProgrammazioneDocente API (REST API);
- * LezioneDocente API (REST API);
- * LezioniDocente API (REST API);
- Pagamento API:
 - * Prenotazione API (REST API);
 - * Prenotazioni API (REST API);
- Topics API (REST API);
- Videoroom API:
 - * VideoCallUtente API (REST API);
 - * VideoCallDocenteAPI (REST API).

0.3 Application Server

Le funzionalità dell'applicazione Smart Learning, precedentemente definite nella specifica dei requisiti, possono essere raggruppate nei seguenti moduli: Registrazione, Login, Utente, Lezione, Pagamento, Topic e Videoroom.

Dato il numero dei servizi offerti, sono state valutate due possibili soluzioni architetturali: un'architettura a microservizi o un'architettura three-layer.

In particolare, l'architettura a microservizi sembra adattarsi perfettamente al sistema. D'altra parte essa, grazie alla separazione dei componenti in microservizi indipendenti (ognuno con la propria base di dati), attribuirebbe al software flessibilità, scalabilità ed estensibilità (nuovi servizi), requisiti ottimali per un'applicazione sviluppata ex-novo.

Inoltre, i servizi sarebbero sviluppati e distribuiti in maniera indipendente, risultando più facili da manutenere, correggere e aggiornare, rispetto ad una classica applicazione monolitica.

Di contro, lo sviluppo di un'architettura a microservizi richiederebbe un grosso sforzo implementativo, dovuto alla complessità di una tale tipologia di architettura. Di conseguenza, in seguito ad un'analisi di fattibilità, si è constatato che non si sarebbero potuti rispettare i tempi di consegna del progetto.

In particolare la problematica principale è rappresentata dal mantenimento della consistenza dei dati, distribuiti su basi di dati indipendenti; essa prevederebbe una comunicazione asincrona tra i microservizi (tramite broker), evitando l'overhead della comunicazione sincrona (*two-phase-commit*), e degli orchestratori delegati a sfruttare questa comunicazione asincrona per implementare la logica di business e dei meccanismi affidabili di rollback.

Per tale motivo, si è optato verso un'architettura più semplice da implementare, ovvero un'**architettura three-layer** (monolitica), in cui saranno presenti servizi esterni.

Essa prevede la suddivisione dell'applicazione in tre diversi moduli: Web Layer, Service Layer e Data Layer.

Nell'architettura a tre livelli, ciascuna delle funzionalità principali è isolata dalle altre, in modo che il Web Layer sia indipendente dal Service Layer, che a sua volta è separato dai dati (Data Layer).

D'altra parte quando l'applicazione è relativamente "piccola", l'architettura monolitica può avere numerosi vantaggi; essa risulta infatti semplice da sviluppare, testare, deployare e scalare, inoltre l'applicazione di cambiamenti radicali è facilitata.

Tuttavia, al crescere delle dimensioni, lo sviluppo, i test, il deploy e lo scaling diventano sempre più difficili.

0.3.1 Package Diagrams

I Package Diagrams sono utilizzati per modellare la struttura modulare del sistema, d'altra parte essi evidenziano i differenti strati dell'architettura che si rifletteranno nel codice sorgente.

Nel caso in esame sono stati elaborati due differenti diagrammi:

- Un Package Diagram generale, il quale mostra la struttura generale del sistema;
- Un Package Diagram specifico, il quale entra nel dettaglio dei vari moduli del sistema.

Tale distinzione è stata fatta per fornire una versione più "leggibile" dell'architettura ed una più dettagliata.

Package Diagram generale

Come è possibile evincere dalla figura sottostante, l'applicazione lato server è strutturata secondo un'**architettura three-layer** (a tre livelli), soluzione ampiamente utilizzata per le applicazioni web.

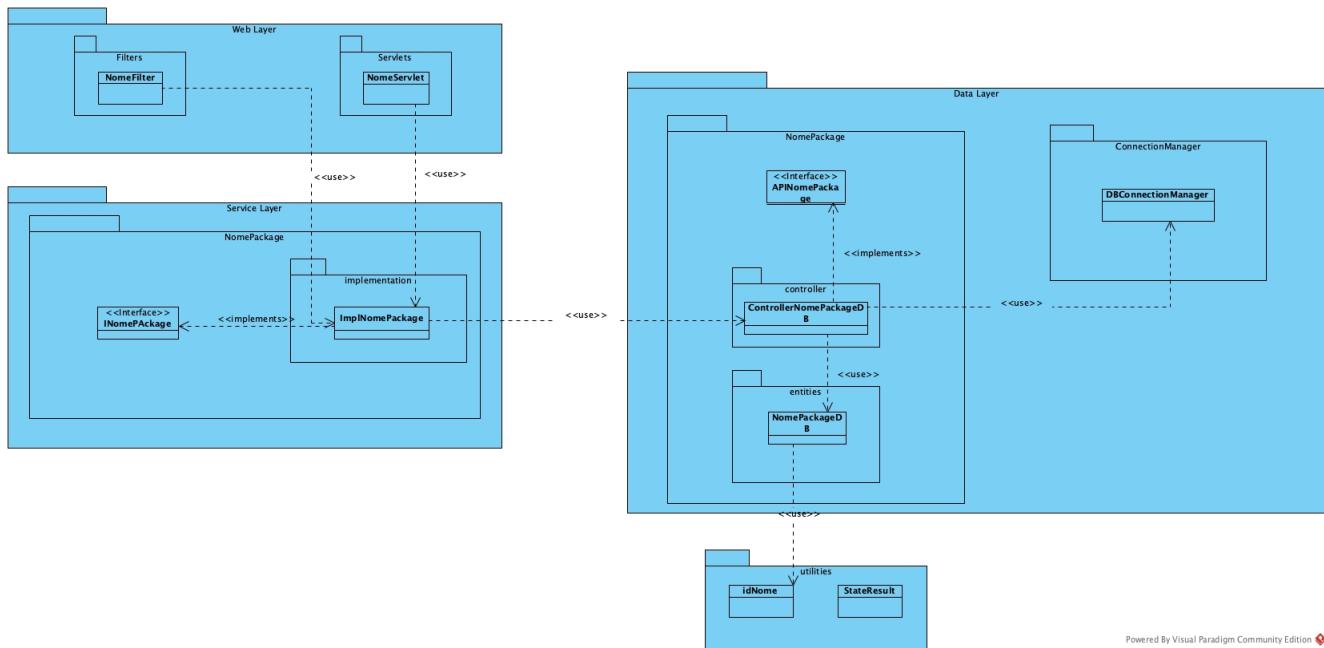


Figura 2: Package Diagram generale

Package Diagram specifico

Tale diagramma specifica puntualmente la struttura ed i contenuti dei vari package del sistema, tuttavia risulta poco leggibile a causa delle dimensioni dello stesso.



Figura 3: Package Diagram specifico

0.3.2 Architettura Three-Layer

Come detto in precedenza il Server applicativo è a sua volta suddiviso secondo un'architettura a tre livelli:

- **Web Layer:** si occupa dell'interfacciamento con il Client.
- **Service Layer:** si occupa delle elaborazioni dei dati in base alla cosiddetta *business logic*. Esso crea un ponte tra il Web Layer e il Data Layer.
- **Data Layer:** il livello dati garantisce il meccanismo di archiviazione persistente dei dati. Esso supporta la connessione con il database e l'esecuzione delle operazioni di inserimento, aggiornamento, eliminazione e recupero dei dati.

Web Layer

Il Web Layer è composto da **filtri** e **servlet**.

Le richieste effettuate dal Client sono in formato HTTP, esse sono ricevute dai filtri, che dopo averle processate le inoltreranno alle servlet, che si occupano di accedere alla risorsa richiesta; ogni risorsa ha una servlet associata.

Viene così a crearsi una catena costituita, per l'appunto, da filtri e servlet.

Quest'ultime si occupano di richiamare le interfacce esposte dai servizi, che implementano le operazioni per servire le richieste, e di formattare la risposta da inviare al Client in formato XML.

Le interfacce esposte dalle servlet sono state realizzate adottando lo stile architettonico **REST**.

L'architettura REST si basa sul protocollo HTTP. Il funzionamento prevede una struttura degli URL ben definita che identifica univocamente una risorsa o un insieme di risorse e l'utilizzo dei metodi HTTP specifici per il recupero di informazioni (GET), per la modifica (POST, PUT, PATCH, DELETE) e per altri scopi (OPTIONS, ecc.).

Le interfacce REST sono state interamente documentate secondo lo standard **OpenAPI 3.0** in un file allegato, come mostrato nella seguente figura:

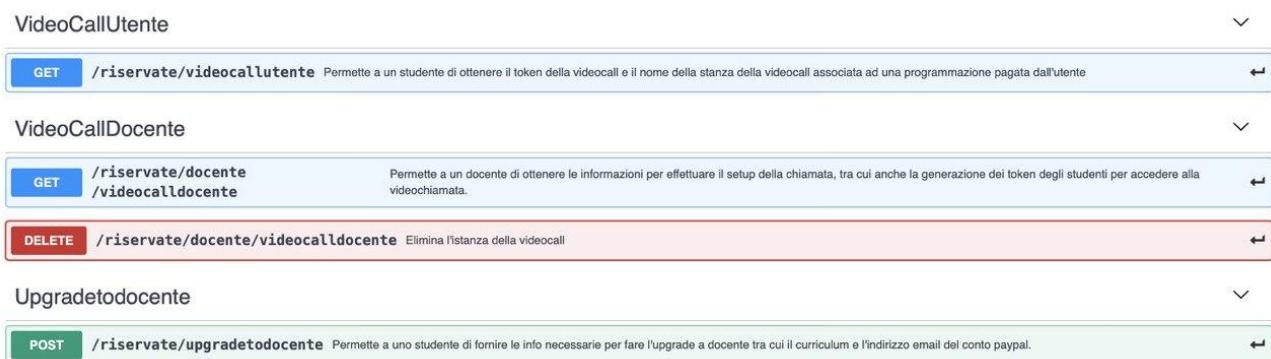


Figura 4: REST API

Service Layer

Il Service Layer si occupa delle elaborazioni dei dati in base alla cosiddetta *business logic*; le elaborazioni del livello intermedio generano i risultati richiesti dall'utente.

Il Service Layer è composto dai seguenti moduli:

- **Registrazione:** si occupa della logica di registrazione di nuovi utenti;
- **Login:** gestisce la logica sul quale si basa il Login;
- **Utente:** ha la responsabilità di gestire tuttò ciò che riguarda il profilo degli utenti (incluso l'Upgrade ad un profilo "Docente");
- **Lezione:** include la logica di gestione delle lezioni e del calendario a loro associato;
- **Pagamento:** include la gestione dei pagamenti e tiene traccia delle persone che hanno pagato una lezione;

-
- **Topic:** include la logica di gestione dei topic;
 - **Videoroom:** ha la responsabilità di gestire ciò che riguarda le videochiamate relative alle lezioni.

Si è scelto quindi di adottare uno stile architetturale Service-Oriented, di conseguenza i principali componenti di questo layer sono stati realizzati come servizi indipendenti, inoltre essi sono richiamati attraverso un’interfaccia indipendente dalla specifica implementazione; ciò garantisce modificabilità e riusabilità.

Data Layer

Il Data Layer ha il compito di gestire la persistenza dei dati, quindi di comunicare con il DBMS relativo al database utilizzato.

Tale livello è strutturato secondo il pattern architetturale **DAO (Data Access Object)**: è stata implementata una classe, con i relativi metodi, per ogni entità tabellare del DBMS.

Tali classi sono utili per stratificare e isolare l’accesso ad una tabella tramite query (poste all’interno dei metodi della classe) ovvero al data layer da parte della business logic, creando un maggiore livello di astrazione ed una più facile manutenibilità.

Infatti, il Data Layer espone ai componenti presenti nel Service Layer un insieme di interfacce (API) per accedere ai dati, ma è compito del DAO gestire la connessione al database e lo svolgimento delle operazioni CRUD; tutto ciò favorisce il disaccoppiamento tra il Service Layer con il Database persistente.

La connessione al database è gestita tramite una classe DBConnectionManager, dalla quale sono dipendenti tutte le altre classi del livello.

0.4 Client

Il Client è strutturato secondo un’architettura detta **Thin Client**, ovvero esso svolge solo il compito di interfaccia utente, mentre la logica dell’applicazione e la gestione dei dati è demandata ai server.

Nel caso in questione il Client è un browser che comunica con il server applicativo tramite il protocollo HTTP/HTTPS.

Il browser, attraverso il proprio motore di rendering, genera l’interfaccia utente, interpretando le pagine realizzate tramite HTML, CSS e tecnologia JavaScript. D’altra parte esso comunicherà direttamente con il server Janus, per la fruizione del servizio di videochiamata.

0.5 Database Server

Il Database Server è stato implementato tramite il DBMS MySQL, utilizzato per la gestione della persistenza dei dati e la loro interrogazione tramite JDBC, ricevendo e soddisfacendo le richieste di lettura/scrittura sul DB da parte della logica applicativa.

JDBC è un connettore per database, che consente l’accesso ad una base di dati da qualsiasi programma scritto in JAVA, indipendentemente dal tipo di DBMS utilizzato. Esso è costituito da un’API object oriented orientata ai database relazionali.

Il database è di tipo **relazionale** e le entità tabellari da cui è composto corrispondono alle classi del Data Layer, come definito dal pattern Data Access Object.

0.6 Servizi esterni: Janus

L’applicazione è basata sulla tecnologia open source WebRTC, disponibile su tutti i moderni browser, che consente di effettuare videochat P2P in tempo reale. A supporto di tale scelta tecnologica si è deciso di utilizzare il WebRTC server Janus. Questo infatti fornisce delle API in javascript che astraggono i dettagli implementativi della tecnologia e ne consentono un rapido e semplice utilizzo in svariati casi d’uso. In particolare sono messi a disposizione una serie di “plugin” da aggiungere al codice base così da personalizzarne l’utilizzo. In questo caso si è deciso di utilizzare il plugin “VideoRoom” che fornisce l’implementazione di una **SFU (Selective Forwarding Unit)**.

Maggiori dettagli sul Janus sono forniti nel capitolo 7.

0.7 Deployment Diagram

Il Deployment Diagram è utilizzato per descrivere un sistema in termini di risorse hardware, dette nodi, e di relazioni fra di esse. Se unito al Component Diagram, mostra come le componenti software siano distribuite rispetto alle risorse hardware disponibili sul sistema.

Come è possibile evincere dalla figura sottostante, il sistema è “deployato” nella seguente modalità:

- **Nodo 1:** Azure Cloud contenente il server Apache Tomcat, sul quale viene eseguito “Smart Learning”;
- **Nodo 2:** Azure Cloud contenente il database MySQL;
- **Nodo 3:** Azure Cloud contenente il server Janus.

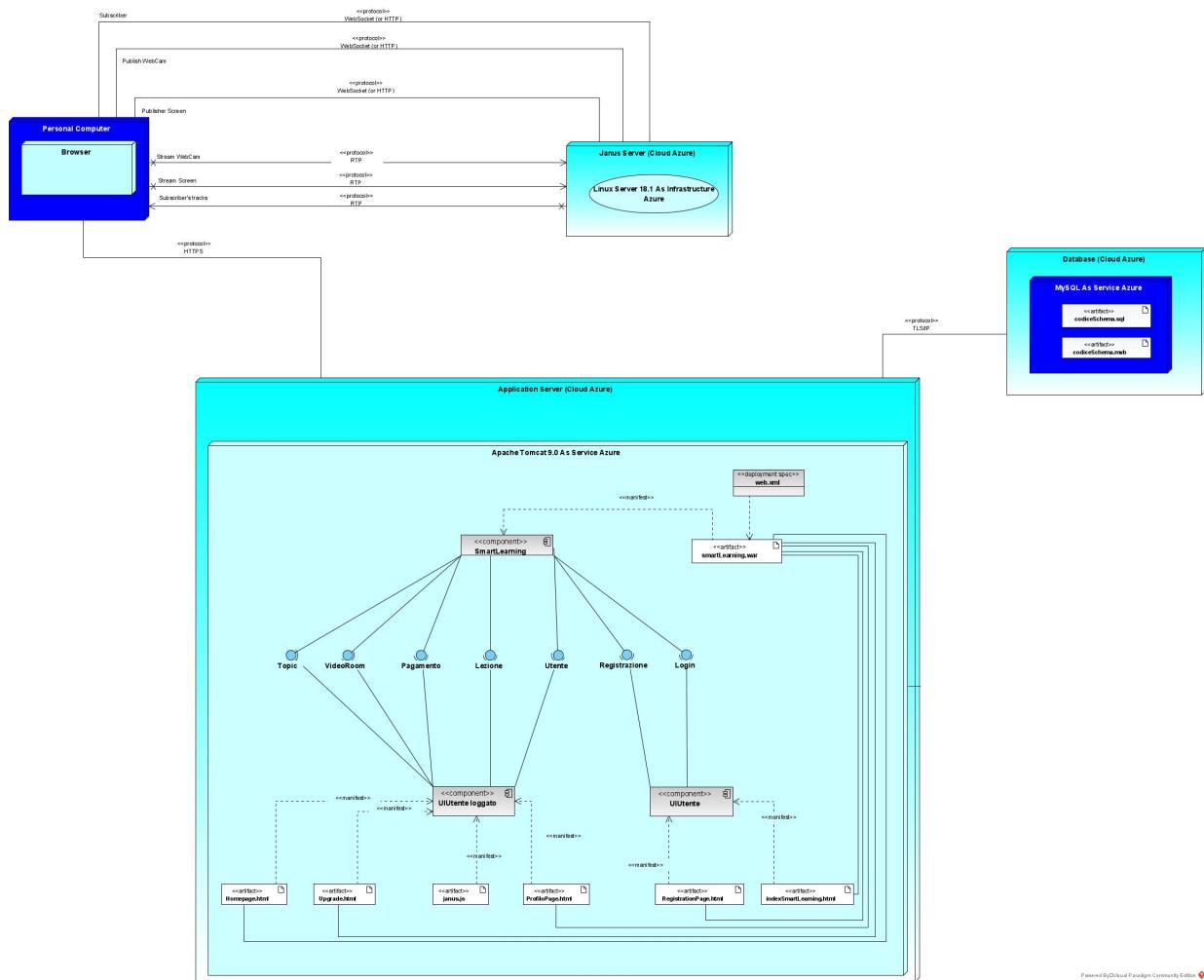


Figura 5: Deployment Diagram