

UNIVERSITÀ DEGLI STUDI DI NAPOLI FEDERICO II

SCUOLA POLITECNICA E DELLE SCIENZE DI BASE

DIPARTIMENTO DI INGEGNERIA ELETTRICA E TECNOLOGIE DELL'INFORMAZIONE

Corso di Laurea Magistrale in Ingegneria Informatica

DOCUMENTAZIONE SHARE-HUB UNINA

Progetto del corso Software Architecture Design

a.a. 2023-24

Studente:

Antonio Sposito M63001285

Sommario

Capitolo 1: Introduzione	3
Capitolo 2: Specifica dei requisiti.....	6
2.1 Requisiti funzionali	6
2.2 Requisiti non funzionali.....	7
2.3 Requisiti sui dati	7
Capitolo 3: Analisi dei requisiti.....	8
3.1 Attori	8
3.2 Analisi dei casi d'uso	8
3.3 Analisi degli scenari dei casi d'uso	9
Capitolo 4: Modellazione del sistema.....	15
4.1 Modello di dominio	15
4.2 Diagrammi di sequenza.....	15
Capitolo 5: Progettazione.....	27
5.1 Scelta dell'architettura.....	27
5.1.1 Scelta delle tecnologie	28
5.2 Diagramma dei componenti	28
5.3 Componenti frontend	29
5.4 Componenti backend	29
5.4.1 API endpoints	31
5.5 Componenti livello dati	43
5.5.1 Diagramma delle classi	43
Capitolo 6: Implementazione	45
6.1 Descrizione delle tecnologie utilizzate	45
6.2 Composite structure diagram backend	46
6.3 Composite structure diagram frontend	50
6.4 Misure di autenticazione e autorizzazione	54
6.5 Struttura del codice	57
Capitolo 7: Testing	58
7.1 Functional testing con Postman	58
7.2 Testing endpoints con ruolo Studente	60
7.2.1 Testing endpoints con ruolo Professore	63
7.2.2 Testing endpoints con ruolo Admin	66
Capitolo 8: Rilascio	69
8.1 Deployment diagram fase di sviluppo.....	69
8.2 Deployment diagram fase di produzione	69
8.3 Installazione.....	70

Capitolo 1: Introduzione

Share-Hub Unina è un'applicazione web sviluppata per il corso di Software Architecture Design. La webapp mette a disposizione degli utenti una piattaforma dove i professori possono caricare del materiale didattico per i loro corsi mentre gli studenti possono, dopo essersi iscritti ai corsi, scaricarne il materiale didattico e recensire i singoli file per poter condividere con altri studenti le loro opinioni sui file.

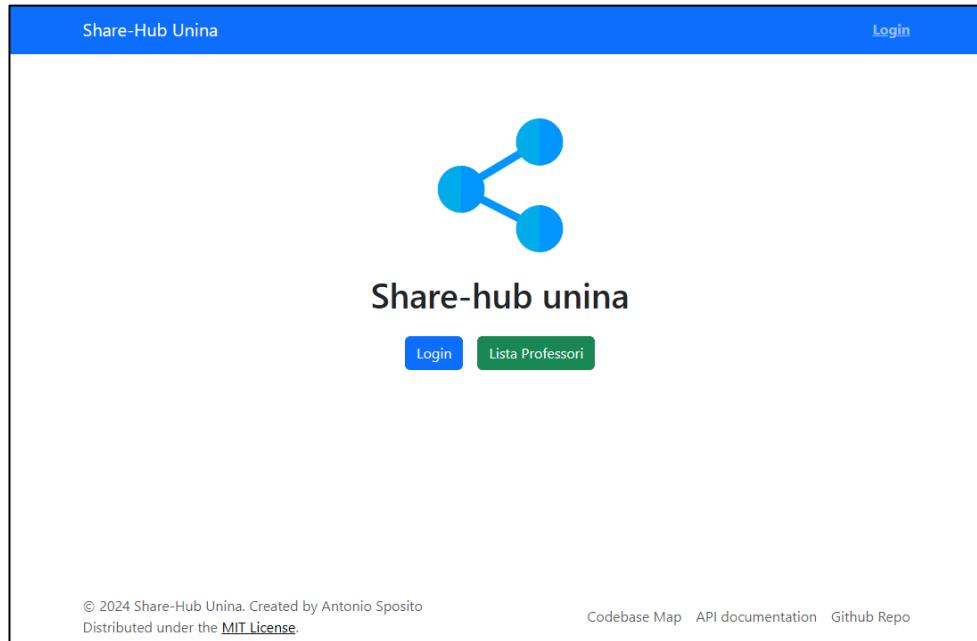


Figura 1-1: Homepage di Share-Hub Unina

The screenshot shows the "Elenco professori" (List of professors) page. At the top, there is a blue header bar with the text "Share-Hub Unina" on the left and "Professori - Corsi - antonio.sposito@studenti.unina.it" on the right. Below the header is a section titled "Elenco professori:" with a blue icon of a person at a computer monitor. There is a search bar labeled "Ricerca professori:" with the placeholder "Cerca per cognome, email o id". Below the search bar is a table with the following data:

ID	Nome	Cognome	Email	Ruolo	
4	Anna Rita	Fasolino	annarita.fasolino@unina.it	Professor	Corsi
5	Marcello	Cinque	macinque@unina.it	Professor	Corsi
6	Roberto	Natella	roberto.natella@unina.it	Professor	Corsi
7	Simon Pietro	Romano	spromano@unina.it	Professor	Corsi
1004	pippo	baudo	prova@unina.it	Professor	Corsi

© 2024 Share-Hub Unina. Created by Antonio Sposito
Distributed under the [MIT License](#). [Codebase Map](#) [API documentation](#) [Github Repo](#)

Figura 1-2: Pagina dei professori

Share-Hub Unina

Professori Corsi antonio.sposito@studenti.unina.it

Elenco corsi:

Ricerca corsi:

Id corso	Titolo corso	Descrizione	Docente
4	IS	Ingegneria del Software	Anna Rita Fasolino
5	RTIS	Real Time Industrial Systems	Marcello Cinque
6	SO	Sistemi Operativi	Marcello Cinque
48	SS	Software security	Roberto Natella
50	NS	Network Security	Simon Pietro Romano
57	SAD	Software Architecture Design	Anna Rita Fasolino
58	WebRTC	Web and Real Time Communication Systems	Simon Pietro Romano
59	CE	Calcolatori Elettronici	Roberto Natella

© 2024 Share-Hub Unina. Created by Antonio Sposito
Distributed under the [MIT License](#).

Codebase Map API documentation Github Repo

Figura 1-3: Pagina dei corsi

Share-Hub Unina

Professori Corsi antonio.sposito@studenti.unina.it

Elenco file per corso:

Id file	Corso	Nome file	Descrizione	Valutazione media
1	5	Slide 1	Intro RTIS	★★★★★(5)
2	5	Slide 2	Real time scheduling	★★★★☆(4)
5	5	Slide 3	Aperiodic servers	Nessuna recensione :(

© 2024 Share-Hub Unina. Created by Antonio Sposito
Distributed under the [MIT License](#).

Codebase Map API documentation Github Repo

Figura 1-4: Pagina dei file di un corso

The screenshot shows a web application interface for 'Share-Hub Unina'. At the top, there's a blue header bar with the text 'Share-Hub Unina' on the left and 'Professori Corsi antonio.sposito@studenti.unina.it' on the right. Below the header, a section titled 'Elenco recensioni:' displays two reviews. Each review has a small icon of three stars in a speech bubble. The first review (ID 16) has the text 'Mi è piaciuto moltoooo' and a rating of 3 stars. The second review (ID 17) has the text 'molto utile' and a rating of 5 stars. A button at the bottom of this section says 'Hai già recensito'. At the bottom of the page, there's a footer with copyright information: '© 2024 Share-Hub Unina. Created by Antonio Sposito' and 'Distributed under the [MIT License](#)'. To the right of the footer, there are links to 'Codebase Map', 'API documentation', and 'Github Repo'.

Id	Testo	Rating	User Id	File Id
16	Mi è piaciuto moltoooo	★★★☆☆	2	2
17	molto utile	★★★★★	1	2

Figura 1-5: Pagina delle recensioni di un file

Questa documentazione raccoglie e descrive le varie fasi del processo di progettazione, in particolare: nel capitolo 2 verranno illustrati i requisiti funzionali, non funzionali e sui dati del progetto che sono alla base del progetto; nel capitolo 3 verranno delineati i casi d'uso e gli attori derivanti dai requisiti, quindi le entità che interagiranno con il sistema; nel capitolo 4, con la modellazione, si andranno ad individuare ad alto livello le possibili interazioni tra le entità del sistema; nel capitolo 5 si affronterà prima la descrizione dell'architettura ad alto livello del sistema, per poi scendere più in dettaglio sulla progettazione dei vari componenti che lo compongono; nel capitolo 6 si descriveranno in dettaglio le tecnologie utilizzate per implementare il progetto; nel capitolo 7 verranno illustrati i test funzionali effettuati per verificare il corretto funzionamento del sistema durante lo sviluppo; infine, nel capitolo 8, si parlerà del rilascio dell'applicazione con gli appositi diagrammi di deployment.

Il progetto è opensource e, con tutta la documentazione prodotta, è accessibile liberamente su [GitHub](#).

Capitolo 2: Specifica dei requisiti

2.1 Requisiti funzionali

I requisiti funzionali descrivono cosa deve fare il sistema, ossia descrivono il suo comportamento in termini di funzionalità e verranno espressi più in dettaglio mediante i casi d'uso. Per Share-Hub Unina sono stati individuati i seguenti requisiti funzionali:

Id Requisito	Descrizione
RF1.1	Il sistema deve permettere agli utenti di registrare un account fornendo nome, cognome, email e password
RF1.2	Il sistema deve, dopo un'autenticazione terminata con successo, fornire un token di autenticazione all'utente
RF1.3	Il sistema deve poter distinguere gli utenti correttamente autenticati in admin, professori e studenti
RF1.4	Il sistema deve permettere agli utenti di autenticarsi tramite email e password
RF2.1	Il sistema deve permettere agli utenti correttamente autenticati la ricerca e la visualizzazione dei profili dei professori
RF2.2	Il sistema deve permettere agli utenti correttamente autenticati di poter modificare il loro profilo
RF2.3	Il sistema deve permettere agli admin di poter modificare ed eliminare profili di altri utenti
RF3.1	Il sistema deve permettere agli utenti correttamente autenticati la ricerca e la visualizzazione dei corsi inseriti nel sistema
RF3.2	Il sistema deve permettere ai professori di creare, modificare i dettagli ed eliminare corsi associati al loro utente
RF3.3	Il sistema deve permettere agli admin di creare, modificare i dettagli ed eliminare corsi di qualsiasi corso
RF4.1	Il sistema deve permettere agli studenti di iscriversi ai corsi e di poter cancellare la loro iscrizione
RF4.2	Il sistema deve permettere ai professori di poter visualizzare le iscrizioni degli studenti relative ai propri corsi
RF4.3	Il sistema deve permettere agli admin di iscriversi ai corsi e di visualizzare ed eliminare qualsiasi iscrizione
RF5.1	Il sistema deve permettere agli studenti di visualizzare i file dei corsi ai quali sono iscritti
RF5.2	Il sistema deve permettere agli studenti di scaricare i file dei corsi ai quali sono iscritti
RF5.3	Il sistema deve permettere ai professori di visualizzare, caricare, modificare ed eliminare file relativi ai propri corsi
RF5.4	Il sistema deve permettere agli admin di visualizzare, caricare, modificare ed eliminare file relativi a tutti i corsi
RF6.1	Il sistema deve permettere agli studenti di aggiungere, visualizzare, modificare ed eliminare recensioni ai file dei corsi ai quali sono iscritti
RF6.2	Il sistema deve permettere ai professori di visualizzare le recensioni dei file dei loro corsi
RF6.3	Il sistema deve permettere agli admin di aggiungere, visualizzare, modificare ed eliminare recensioni ai file dei corsi

Tabella 2-1: Requisiti funzionali del sistema

L'unico requisito non soddisfatto al momento della stesura di questa documentazione è il RF5.3, quello relativo al download dei file dall'applicazione.

2.2 Requisiti non funzionali

I requisiti non funzionali non riguardano le capacità del sistema ma descrivono come il sistema deve comportarsi, includono quindi attributi di qualità del sistema stesso. Per Share-Hub Unina sono stati individuati i seguenti requisiti non funzionali:

Id Requisito	Descrizione
RNF1.1	Il sistema deve proteggere le password memorizzate degli utenti attraverso un apposito algoritmo di hashing
RNF1.2	Il sistema deve avere un'interfaccia utente deve essere intuitiva e accessibile, con un design responsive che funzioni su dispositivi desktop e mobili

Tabella 2-2: Requisiti non funzionali del sistema

2.3 Requisiti sui dati

I requisiti sui dati sono specifiche che definiscono come i dati devono essere gestiti all'interno del sistema. Per Share-Hub unina sono stati individuati i seguenti requisiti sui dati:

Id Requisito	Descrizione
RD1 (validazione)	Il sistema deve validare i dati inseriti dagli utenti, per esempio le email devono terminare in “@unina.it” oppure in “@studenti.unina.it”
RD2 (persistenza)	Il sistema deve memorizzare in maniera persistente, mediante un database, i dati relativi agli utenti, corsi, file, iscrizioni e recensioni
RD3 (integrità)	Il sistema deve verificare l'integrità dei dati, per esempio: <ul style="list-style-type: none"> - Non possono esistere due utenti con la stessa email - Non deve essere possibile per uno studente iscriversi più volte allo stesso corso - Non devono esistere più corsi con lo stesso nome per lo stesso professore - Non deve essere possibile per uno studente recensire più volte lo stesso file
RD4 (accessibilità)	Il sistema deve permettere agli utenti di poter facilmente cercare e filtrare i professori e i corsi presenti nel sistema

Tabella 2-3: Requisiti sui dati del sistema

Capitolo 3: Analisi dei requisiti

3.1 Attori

Dai requisiti presentati nel capitolo precedente, risulta chiaro come gli utenti che usufruiranno dei servizi messi a disposizione dall'applicazione, gli **attori primari**, sono 3:

1. **Studenti (Students):** utilizzano l'applicazione per iscriversi ai corsi, visualizzare il materiale di ogni corso e condividere con altri studenti la propria opinione, attraverso le recensioni, sui file messi a disposizione dai professori
2. **Professori (Professors):** utilizzano l'applicazione per poter creare corsi dove poter condividere il materiale didattico con gli studenti e leggono le recensioni dei materiali didattico scritte dagli studenti
3. **Amministratori (Admin):** sono utenti privilegiati in grado di avere accesso completo a tutti i dati inseriti dagli altri utenti, potendo anche modificarne ed eliminarne i dati inseriti. Sono una specializzazione

Quando non c'è necessità di distinguere tra i vari attori, si parlerà di "Utente registrato" oppure più semplicemente di "Utente" per evitare la ridondanza dei casi d'uso (per esempio nel caso d'uso "Login" l'attore primario può essere Studente, Professore o Admin).

3.2 Analisi dei casi d'uso

La seguente tabella riassume i principali casi d'uso e gli attori primari coinvolti in essi. Da notare che siccome l'attore Admin ha il controllo sui vari elementi del sistema, si sarebbero dovuti illustrare altri casi d'uso simili a quelli già esistenti (come per esempio "Gestisci corsi di un altro utente" e "Gestisci corso" che differiscono solo per la possibilità dell'Admin di modificare corsi anche di altri utenti). Per sinteticità questi casi d'uso sono stati riassunti in un unico caso d'uso, in cui si riassume tutte le possibili operazioni che l'Admin può svolgere sui dati degli altri utenti.

ID	CASO D'USO	ATTORE PRIMARIO
UC01	Registra utente	Utente non registrato
UC02	Login	Utente
UC03	Modifica profilo	Utente
UC04	Modifica profilo di un altro utente	Admin
UC05	Visualizza professore	Utente
UC06	Gestisci corso	Professore
UC07	Visualizza corso	Studente
UC08	Gestisci file	Professore
UC09	Visualizza file	Studente
UC10	Gestisci iscrizione	Studente
UC11	Visualizza iscrizioni	Professore
UC12	Gestisci recensione	Studente
UC13	Visualizza recensione	Professore
UC14	Gestisci corsi, file, iscrizioni e recensioni	Admin

Tabella 3-1: Casi d'uso del sistema e relativi attori primari

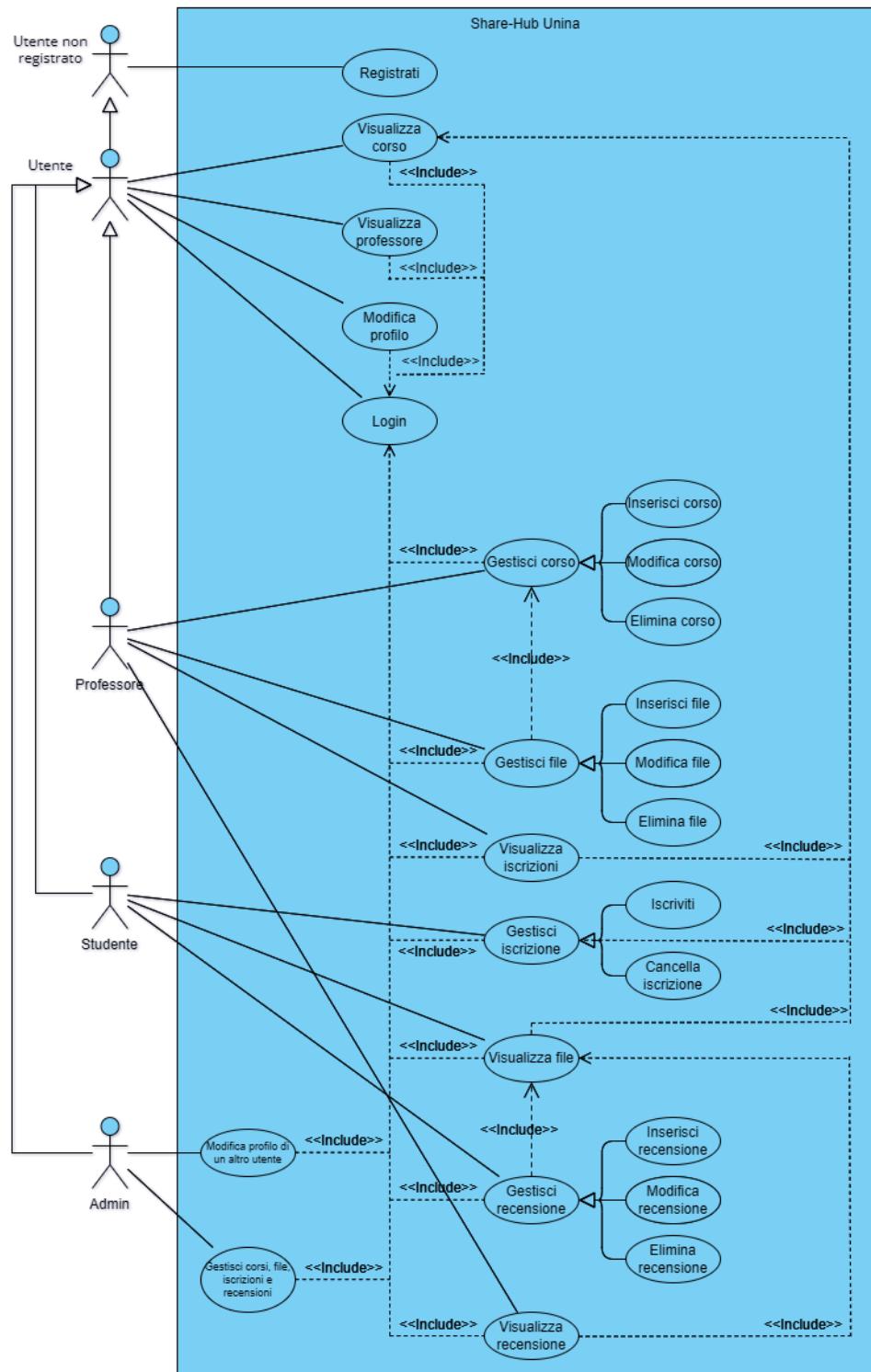


Figura 3-1: Diagramma dei casi d'uso del sistema

3.3 Analisi degli scenari dei casi d'uso

In questa sezione verranno descritti in dettaglio i casi d'uso individuati nella precedente sezione, presentando uno scenario per ognuno di esso

UC01 - REGISTRATI	
Attore primario	Utente non registrato
Descrizione	L'utente fornisce nome, cognome, email e password per creare un account.
Pre-condizioni	L'utente deve essere in possesso di una mail studenti unina o docenti unina
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente non registrato accede alla schermata di registrazione 2. L'utente non registrato inserisce i suoi dati personali, come nome, cognome, email e password nel form apposito e clicca sul pulsante “Registrati” 3. Il sistema registra l'utente nel database 4. L'utente viene reindirizzato alla pagina di login
Post-condizioni	Nel database del sistema è presente un nuovo utente che può autenticarsi
Casi d'uso correlati	-
Sequenza di eventi alternativa	<ol style="list-style-type: none"> 3a. Se l'email inserita è già registrata, il sistema genera un messaggio di errore 3b. Se l'email inserita non è una mail Unina (cioè, che termina in @unina.it oppure in @studenti.unina.it) il sistema genera un messaggio di errore

UC02 - LOGIN	
Attore primario	Utente
Descrizione	L'utente fornisce email e password per autenticarsi nel sistema.
Pre-condizioni	L'utente deve essere registrato nel sistema
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente accede alla schermata di login 2. L'utente inserisce email istituzionale e password 3. Il sistema verifica le credenziali 4. Se le credenziali sono corrette il sistema genera e fornisce un token di autenticazione all'utente 5. L'utente viene reindirizzato alla sua pagina del profilo
Post-condizioni	L'utente è autenticato e ha un token di autenticazione valido.
Casi d'uso correlati	-
Sequenza di eventi alternativa	<ol style="list-style-type: none"> 4a. Se le credenziali sono errate, il sistema mostra un messaggio di errore

UC03 - MODIFICA PROFILO	
Attore primario	Utente
Descrizione	L'utente modifica il proprio profilo (nome, cognome)
Pre-condizioni	L'utente deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente accede alla pagina del proprio profilo. 2. L'utente modifica i campi desiderati. 3. Il sistema salva le modifiche. 4. Il sistema conferma l'avvenuta modifica.
Post-condizioni	Il profilo dell'utente è aggiornato.
Casi d'uso correlati	Il caso d'uso Login è incluso
Sequenza di eventi alternativa	-

UC04 - MODIFICA PROFILO DI UN ALTRO UTENTE	
Attore primario	Admin
Descrizione	L'admin modifica o elimina il profilo di un altro utente.
Pre-condizioni	L'amministratore deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'admin accede alla lista degli utenti. 2. L'admin seleziona un utente da modificare o eliminare. 3. L'admin modifica i dettagli del profilo (nome e cognome) o elimina l'utente. 4. Il sistema salva le modifiche o elimina l'utente.
Post-condizioni	Il profilo dell'utente è aggiornato o eliminato
Casi d'uso correlati	Il caso d'uso Login è incluso
Sequenza di eventi alternativa	-

UC05 - VISUALIZZA PROFESSORE	
Attore primario	Utente
Descrizione	L'utente visualizza e filtra l'elenco dei professori
Pre-condizioni	L'utente deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente accede alla pagina dei professori 2. Opzionalmente l'utente filtra i professori in base a cognome o mail 3. L'utente accede alla pagina del singolo professore
Post-condizioni	Viene restituita la lista eventualmente filtrata dei professori
Casi d'uso correlati	Il caso d'uso Login è incluso
Sequenza di eventi alternativa	-

UC06 - GESTISCI CORSO	
Attore primario	Professore
Descrizione	Il professore crea, modifica o elimina un corso associato al proprio utente.
Pre-condizioni	Il professore deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il professore accede alla pagina di gestione dei corsi. 2. Il professore inserisce/modifica/elimina i dettagli del corso. 3. Il sistema salva le modifiche o elimina il corso. 4. Il sistema conferma l'operazione.
Post-condizioni	Il corso è creato, modificato o eliminato
Casi d'uso correlati	Il caso d'uso Login è incluso
Sequenza di eventi alternativa	-

UC07 - VISUALIZZA CORSO	
Attore primario	Utente
Descrizione	L'utente visualizza e filtra l'elenco dei corsi
Pre-condizioni	L'utente deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'utente accede alla pagina dei corsi 2. Opzionalmente l'utente filtra i corsi in base al nome 3. L'utente accede alla pagina del singolo corso
Post-condizioni	Viene restituita la lista eventualmente filtrata dei corsi
Casi d'uso correlati	Il caso d'uso Login è incluso
Sequenza di eventi alternativa	-

UC08 - GESTISCI FILE	
Attore primario	Professore
Descrizione	Il professore crea, modifica o elimina un file associato ad uno dei suoi corsi
Pre-condizioni	Il professore deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il professore accede alla pagina di gestione dei corsi. 2. Il professore seleziona un corso 3. Il professore accede alla pagina di gestione dei file del corso 4. Il professore inserisce/modifica/elimina un file per il corso selezionato 5. Il sistema conferma l'operazione.
Post-condizioni	Il file è creato, modificato o eliminato
Casi d'uso correlati	Il caso d'uso Login e il caso Gestione corso sono inclusi
Sequenza di eventi alternativa	-

UC09 - VISUALIZZA FILE	
Attore primario	Studente
Descrizione	Lo studente visualizza i file di un corso al quale è iscritto
Pre-condizioni	Lo studente deve essere autenticato e iscritto al corso a cui il file appartiene
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Lo studente accede alla pagina dei corsi 2. Lo studente seleziona un corso 3. Lo studente accede alla pagina con l'elenco dei file del corso 4. Lo studente visualizza ed eventualmente fa il download dei file del file
Post-condizioni	Il file viene visualizzato e/o scaricato
Casi d'uso correlati	Il caso d'uso Login e il caso Visualizza corso sono inclusi.
Sequenza di eventi alternativa	3a. Se l'utente non è iscritto al corso il sistema genera un errore

UC10 - GESTISCI ISCRIZIONE	
Attore primario	Studente
Descrizione	Lo studente si iscrive o cancella l'iscrizione ad un corso
Pre-condizioni	Lo studente deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Lo studente accede alla pagina dei corsi 2. Lo studente seleziona un corso a cui iscriversi o da cui cancellarsi 3. Il sistema salva l'iscrizione o la cancellazione 4. Il sistema conferma l'operazione
Post-condizioni	Lo studente è iscritto o cancellato dal corso
Casi d'uso correlati	Il caso d'uso Login e il caso Visualizza corso sono inclusi
Sequenza di eventi alternativa	-

UC11 – VISUALIZZA ISCRIZIONI	
Attore primario	Professore
Descrizione	Il professore visualizza le iscrizioni di un corso di cui è il proprietario
Pre-condizioni	Il professore deve essere autenticato e proprietario del corso
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il professore accede alla pagina di gestione dei suoi corsi 2. IL professore seleziona un corso 3. Il professore seleziona le iscrizioni del corso 4. Il professore visualizza le iscrizioni degli studenti al suo corso
Post-condizioni	Le iscrizioni vengono visualizzate
Casi d'uso correlati	Il caso d'uso Login e il caso Visualizza corso sono inclusi.
Sequenza di eventi alternativa	3a. Se l'utente non è iscritto al corso il sistema genera un errore

UC12 - GESTISCI RECENSIONE	
Attore primario	Studente
Descrizione	Lo studente inserisce/modifica/elimina una recensione per un file
Pre-condizioni	Lo studente deve essere autenticato e deve essere iscritto al corso a cui il file da recensire appartiene
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Lo studente accede alla pagina dei file di un corso 2. Lo studente seleziona un file per visualizzarne i dettagli 3. Lo studente inserisce/modifica/elimina una sua recensione per il file 4. Il sistema conferma l'operazione
Post-condizioni	La recensione è inserita/modifica/eliminata
Casi d'uso correlati	Il caso d'uso Login e il caso Visualizza file sono inclusi
Sequenza di eventi alternativa	3a. Se l'utente tenta di recensire un file che ha già recensito il sistema genera un errore

UC13 - VISUALIZZA RECENSIONE	
Attore primario	Professore
Descrizione	Il professore visualizza le recensioni per i file appartenenti ai suoi corsi
Pre-condizioni	Il professore deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. Il professore accede alla pagina di visualizzazione dei file di un proprio corso 2. Il professore seleziona un file 3. Vengono restituite le recensioni scritte dagli utenti per quel file
Post-condizioni	Le recensioni del file vengono visualizzate
Casi d'uso correlati	Il caso d'uso Login e il caso Visualizza file sono inclusi.
Sequenza di eventi alternativa	-

UC14 - GESTISCI CORSI/FILE/ISCRIZIONI/RECENSIONI	
Attore primario	Admin
Descrizione	L'admin gestisce gli elementi inseriti nel sistema dagli altri utenti
Pre-condizioni	L'admin deve essere autenticato
Sequenza di eventi principale	<ol style="list-style-type: none"> 1. L'admin accede alla pagina di gestione del sistema 2. L'admin seleziona e visualizza la pagina dei corsi/file/iscrizioni/recensioni presenti nel sistema 3. L'admin aggiunge/modifica/elimina gli elementi della pagina
Post-condizioni	Corsi, file, iscrizioni e recensioni sono gestiti secondo l'azione richiesta.
Casi d'uso correlati	Il caso d'uso Login e il caso Visualizza file sono inclusi.
Sequenza di eventi alternativa	-

Capitolo 4: Modellazione del sistema

Di questo capitolo fanno parte due artefatti:

1. Il modello di dominio, che metterà in evidenza le entità del sistema Share-Hub Unina e le relazioni che intercorrono tra di loro
2. I diagrammi di sequenza, che metteranno in evidenza invece gli eventi e le operazioni possibili nel sistema

4.1 Modello di dominio

Il modello di dominio è una rappresentazione concettuale del dominio del problema che il sistema software Share-Hub Unina intende risolvere. È una descrizione ad alto livello delle entità e delle relazioni che esistono tra di loro nel contesto del sistema, che può essere utile per aiutare ad identificare le entità principali che popolano il sistema e le relazioni che intercorrono tra di esse.

Per il sistema Share-Hub Unina si sono individuate le seguenti entità: Utente, Corso, File, Iscrizione, Recensione. Le relazioni messe in evidenza del modello di dominio possono aiutare a carpire più informazioni sul sistema e sulle informazioni scambiate. In particolare, viene messa in evidenza la molteplicità delle relazioni, che non sono esplicate nei requisiti e/o negli scenari dei casi d'uso.

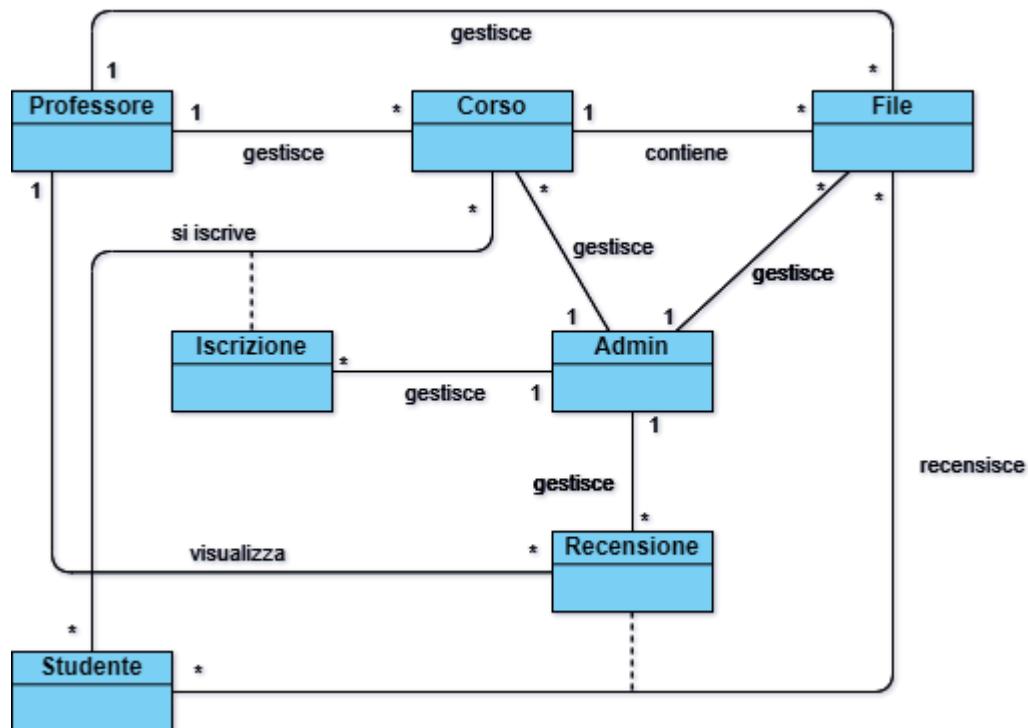


Figura 4-1: Modello di dominio del sistema

4.2 Diagrammi di sequenza

Un diagramma di sequenza può essere utile per definire gli input e gli output del sistema in relazione ad un caso d'uso, di cui ne rappresenta uno scenario. Analizzando il diagramma si possono dedurre le operazioni che il sistema deve offrire.

Da notare che già in questa fase è stata proposta una suddivisione architetturale a livelli (Web Browser, Backend e Database). Questa scelta è volta a facilitare le fasi successive di progettazione in quanto, rappresentando il comportamento dinamico e le interazioni tra i vari livelli, vengono messe in evidenza le responsabilità di ognuno e le operazioni che devono offrire agli altri livelli.

Di seguito, verrà proposto un diagramma di sequenza per ognuno degli scenari dei casi d'uso descritti nella sezione 3.3

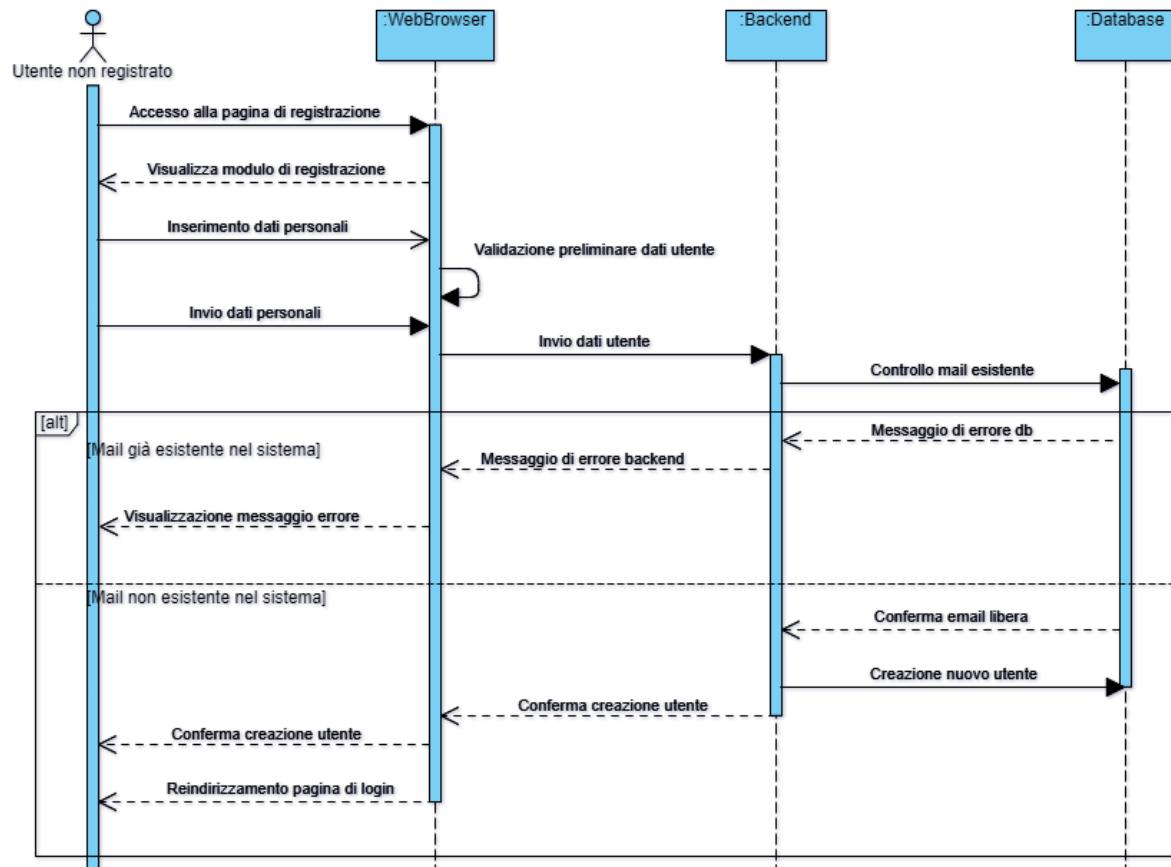


Figura 4-2: Diagramma di sequenza UC01 - Registrati

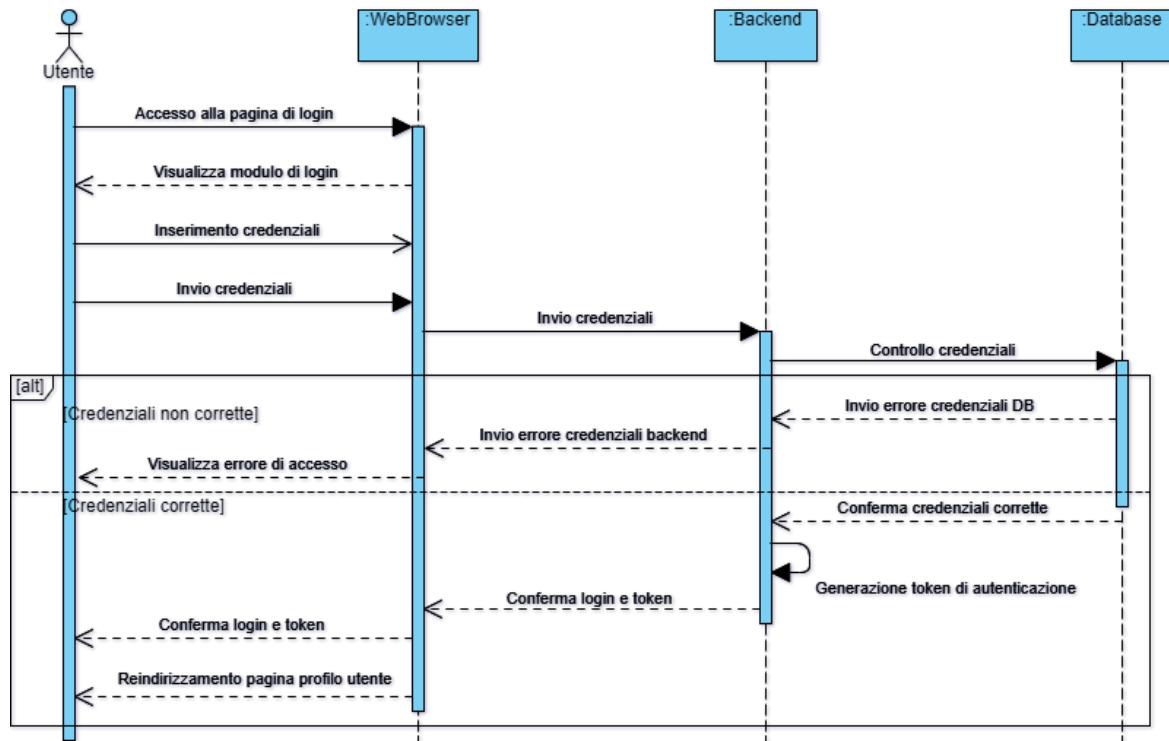


Figura 4-3: Diagramma di sequenza UC02 – Login

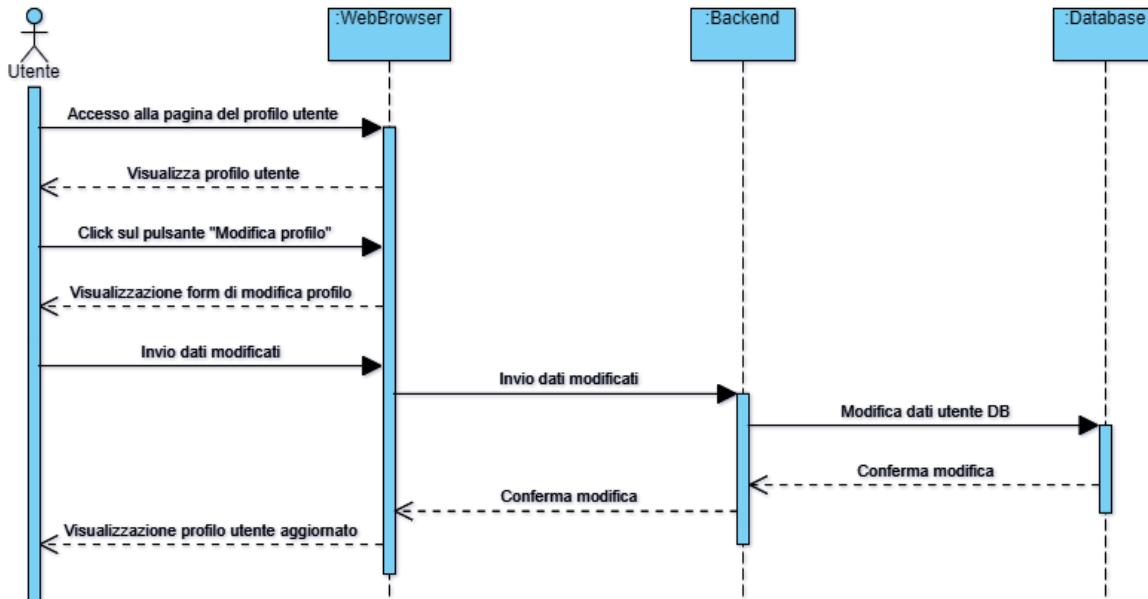


Figura 4-4: Diagramma di sequenza UC03 – Modifica profilo

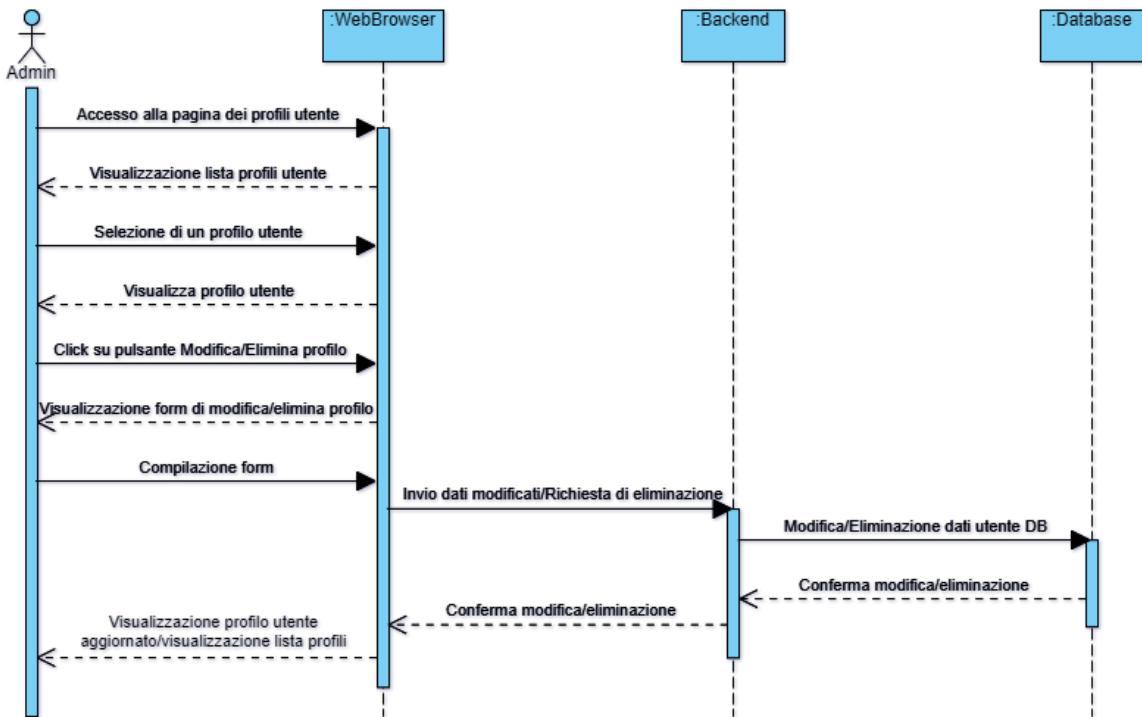


Figura 4-5: Diagramma di sequenza UC04 – Modifica profilo di un altro utente

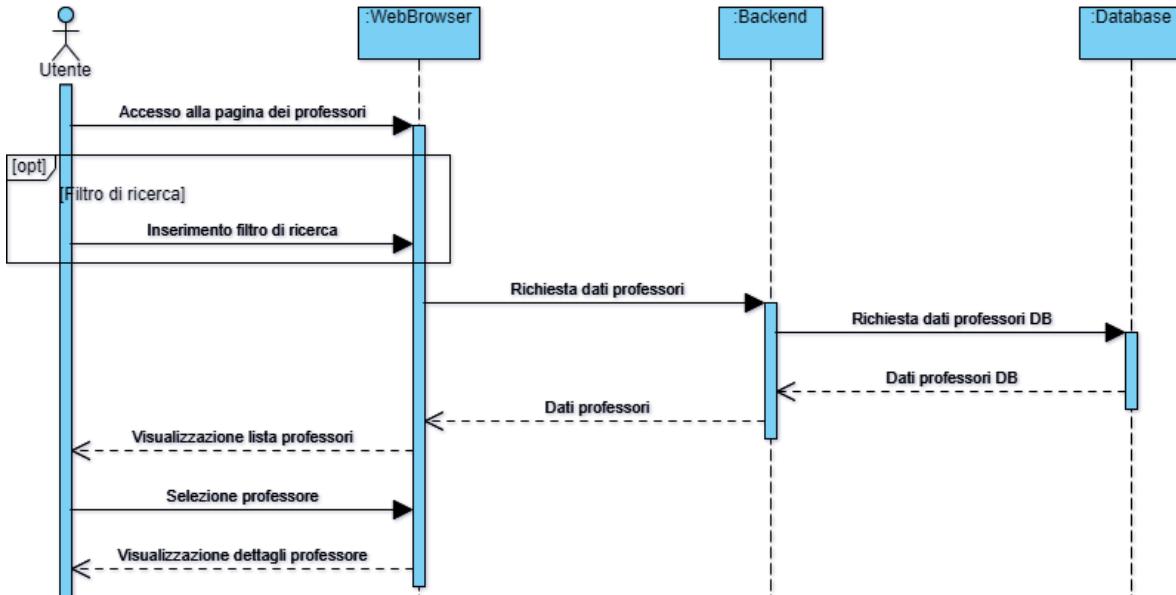


Figura 4-6: Diagramma di sequenza UC05 – Visualizza professore

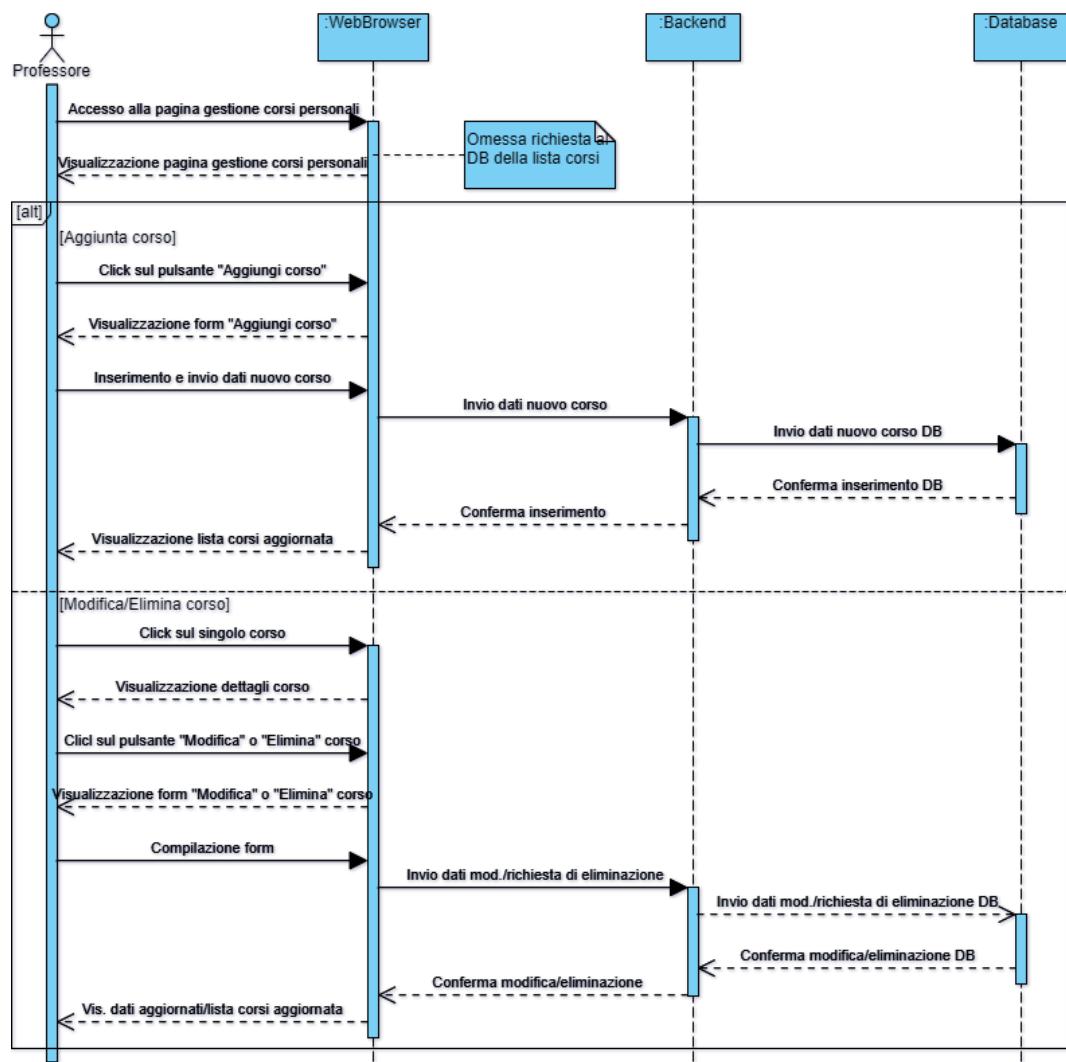


Figura 4-7: Diagramma di sequenza UC06 – Gestisci corso

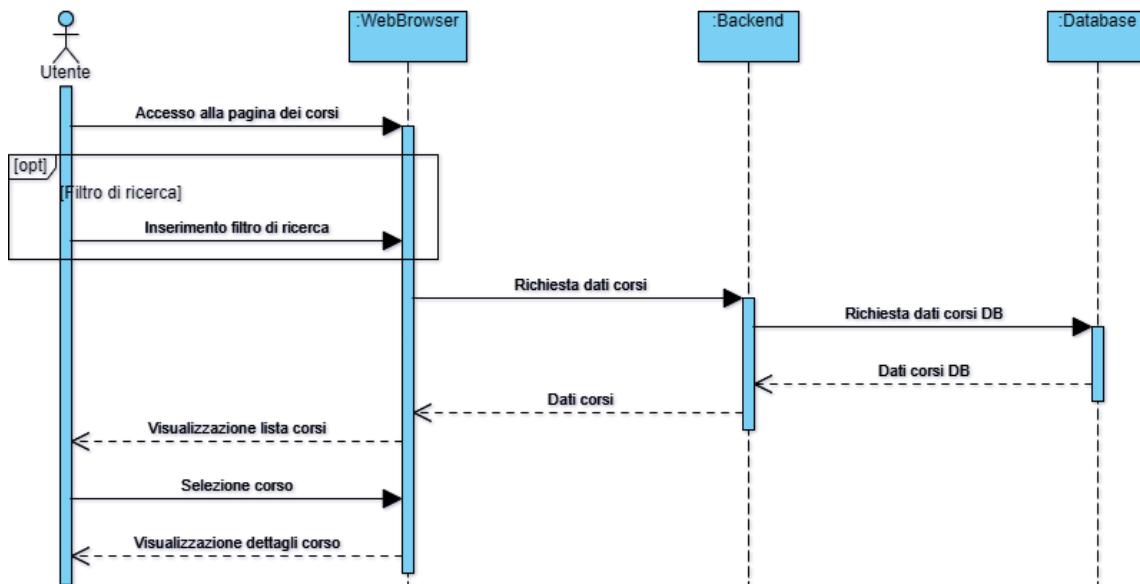


Figura 4-8: Diagramma di sequenza UC07 – Visualizza corso

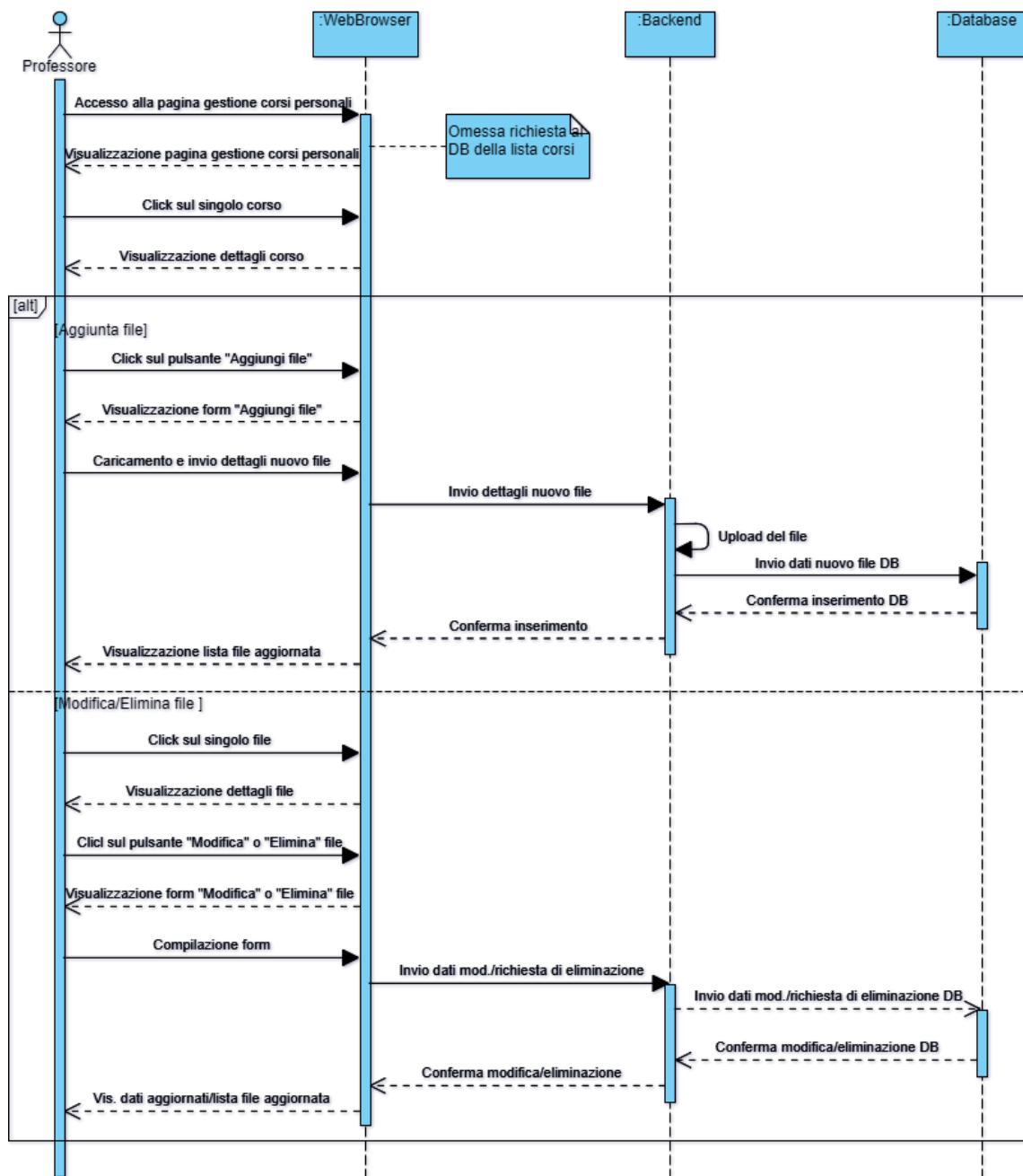


Figura 4-9: Diagramma di sequenza UC08 – Gestisci file

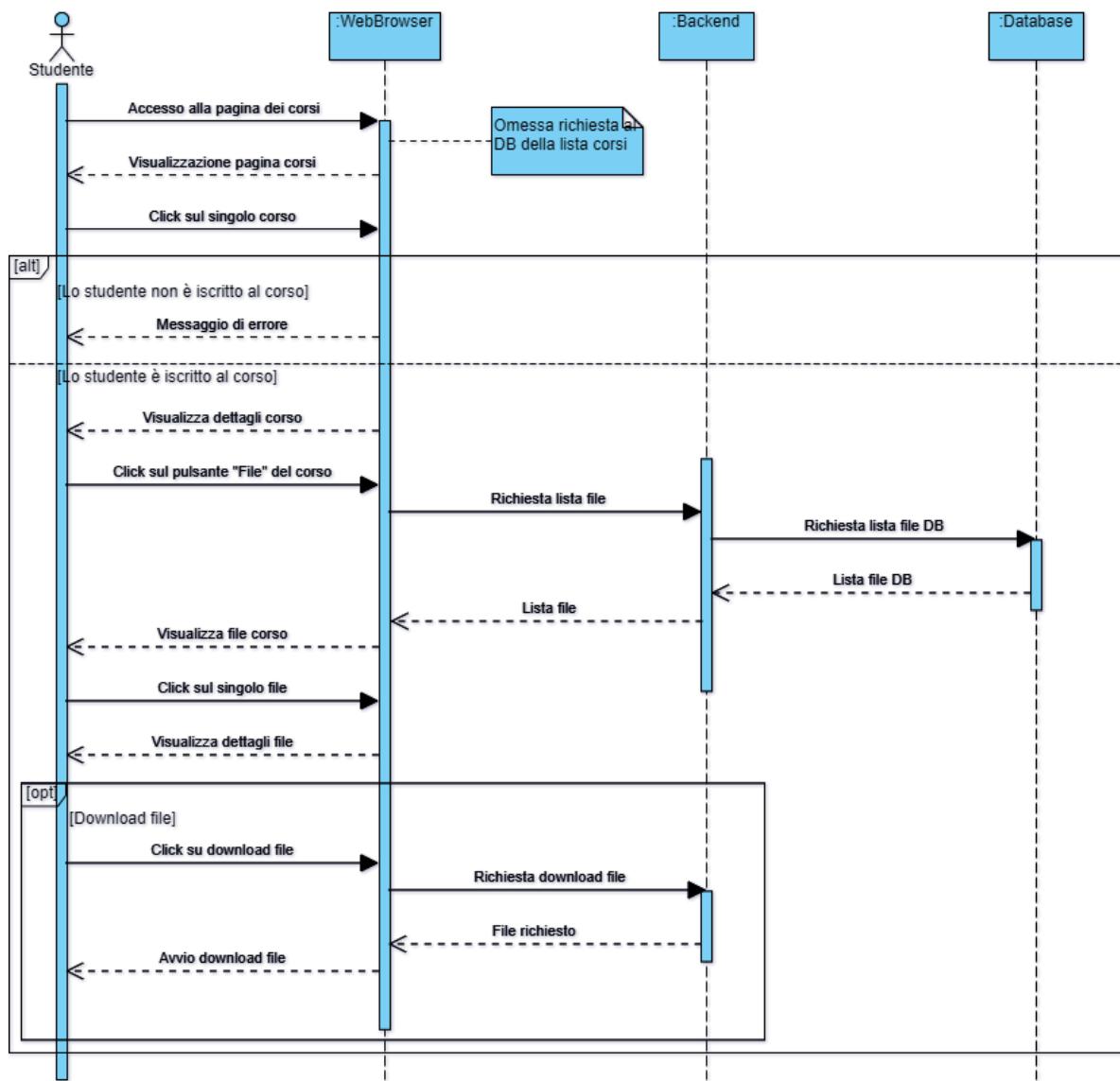


Figura 4-10: Diagramma di sequenza UC09 – Visualizza file

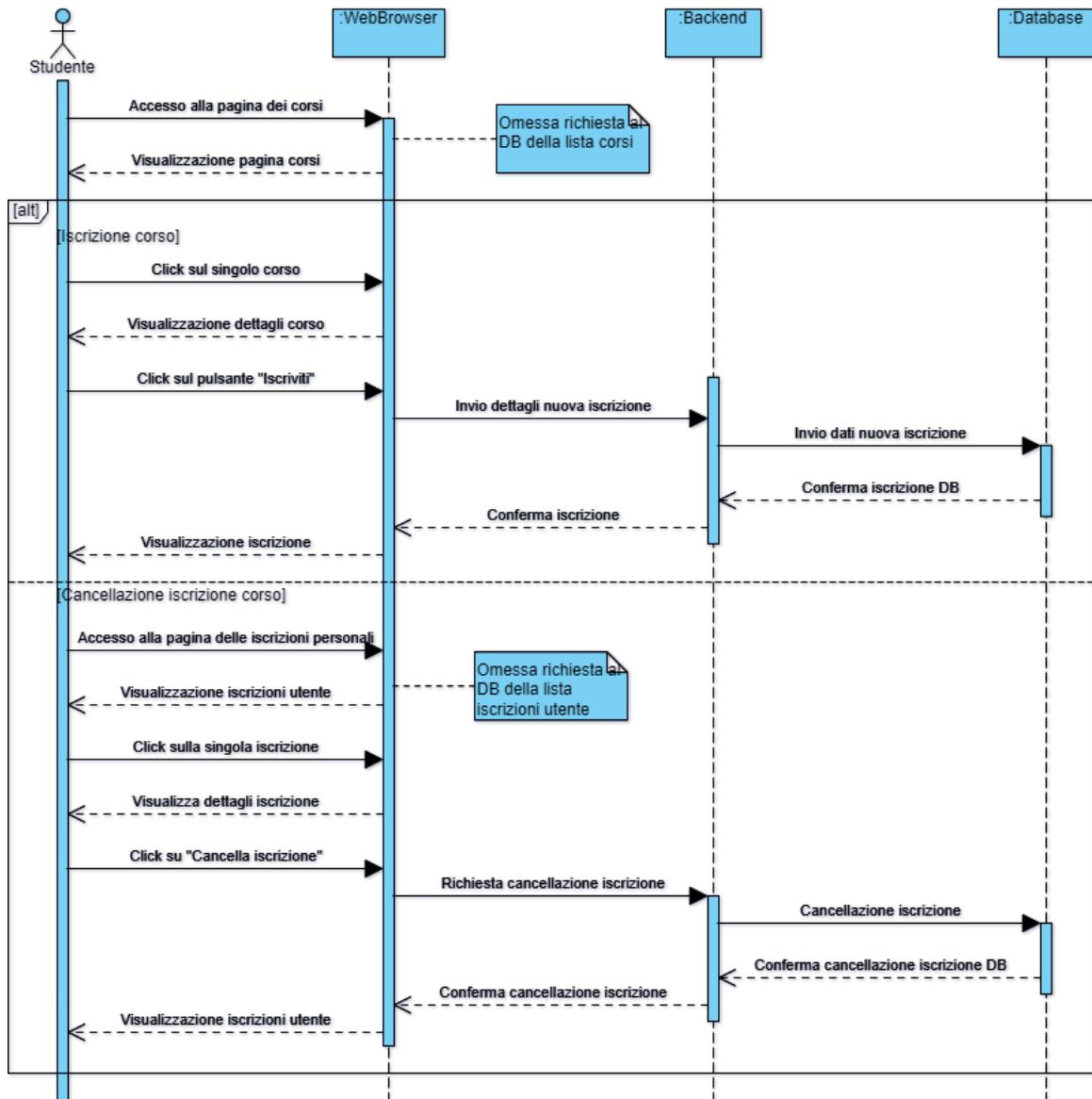


Figura 4-11: Diagramma di sequenza UC10 – Gestisci iscrizione

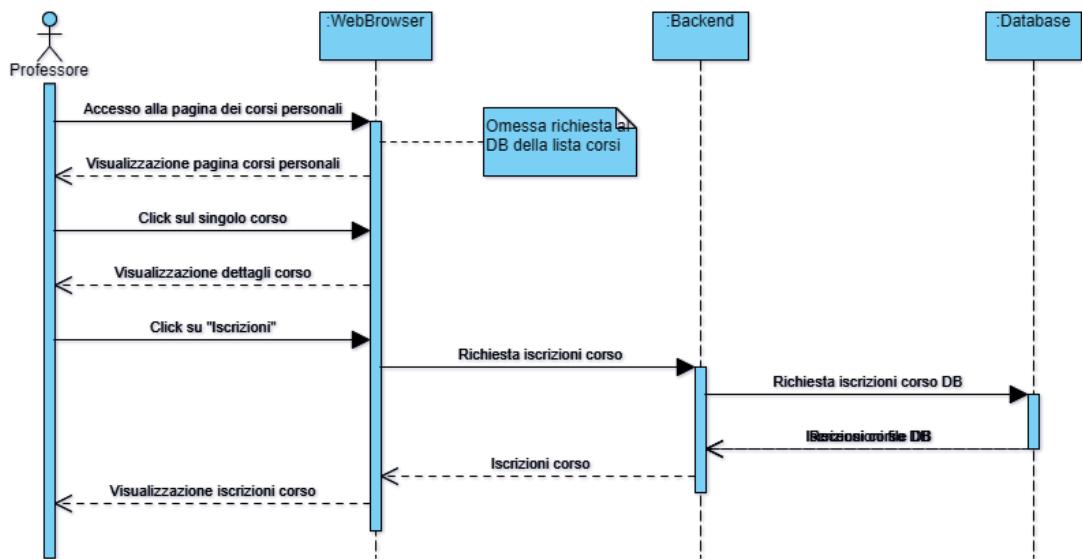


Figura 4-12: Diagramma di sequenza UC11 – Visualizza iscrizioni

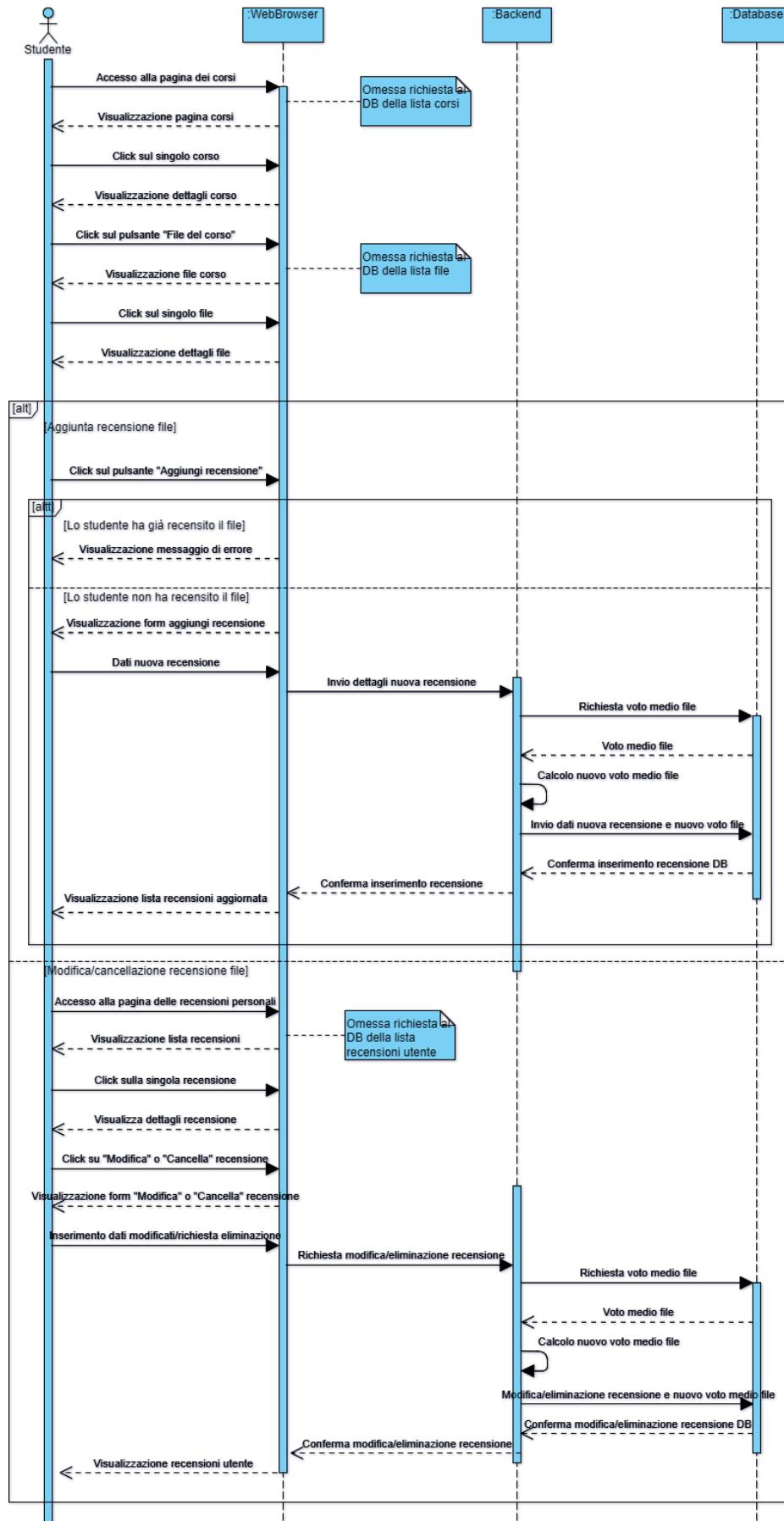


Figura 4-13: Diagramma di sequenza UC12 – Gestisci recensione

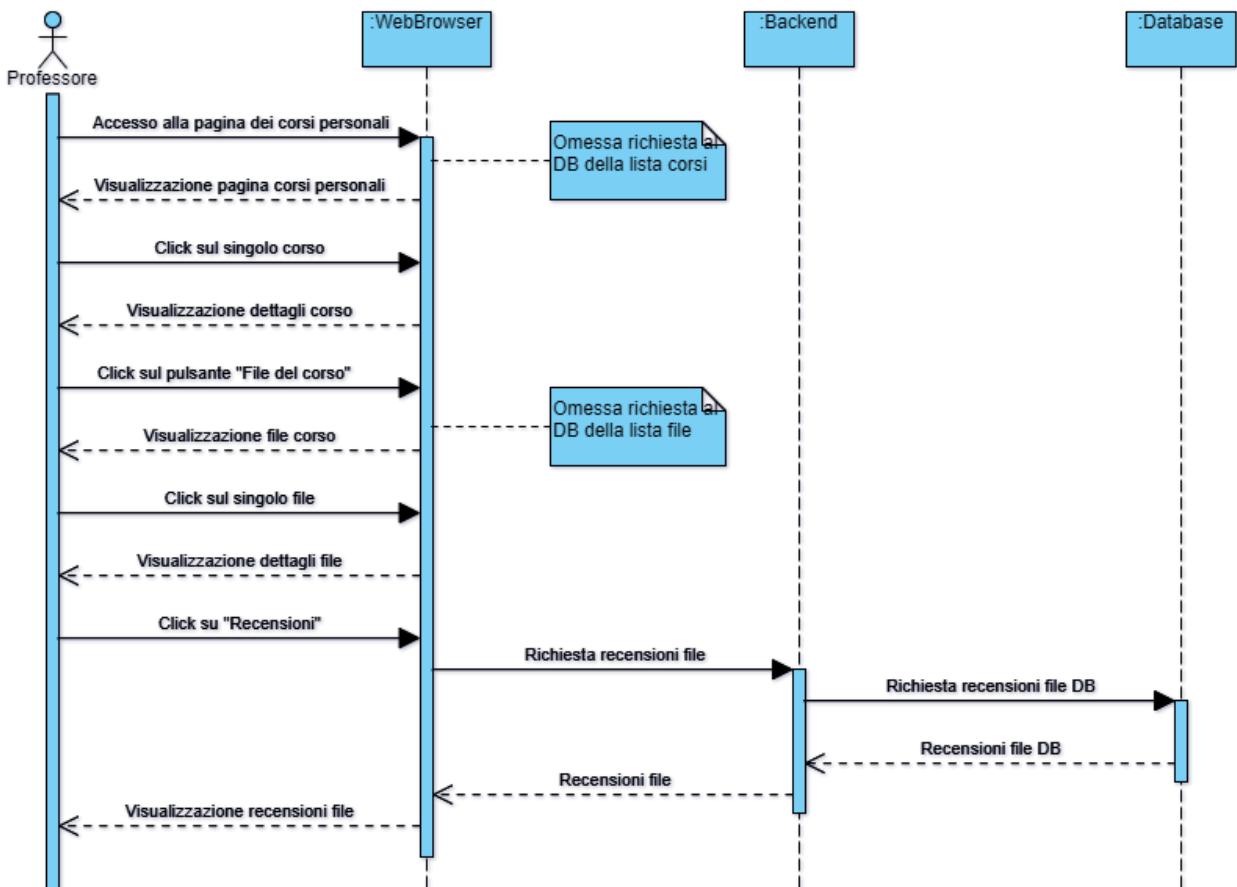


Figura 4-14: Diagramma di sequenza UC13 – Visualizza recensione

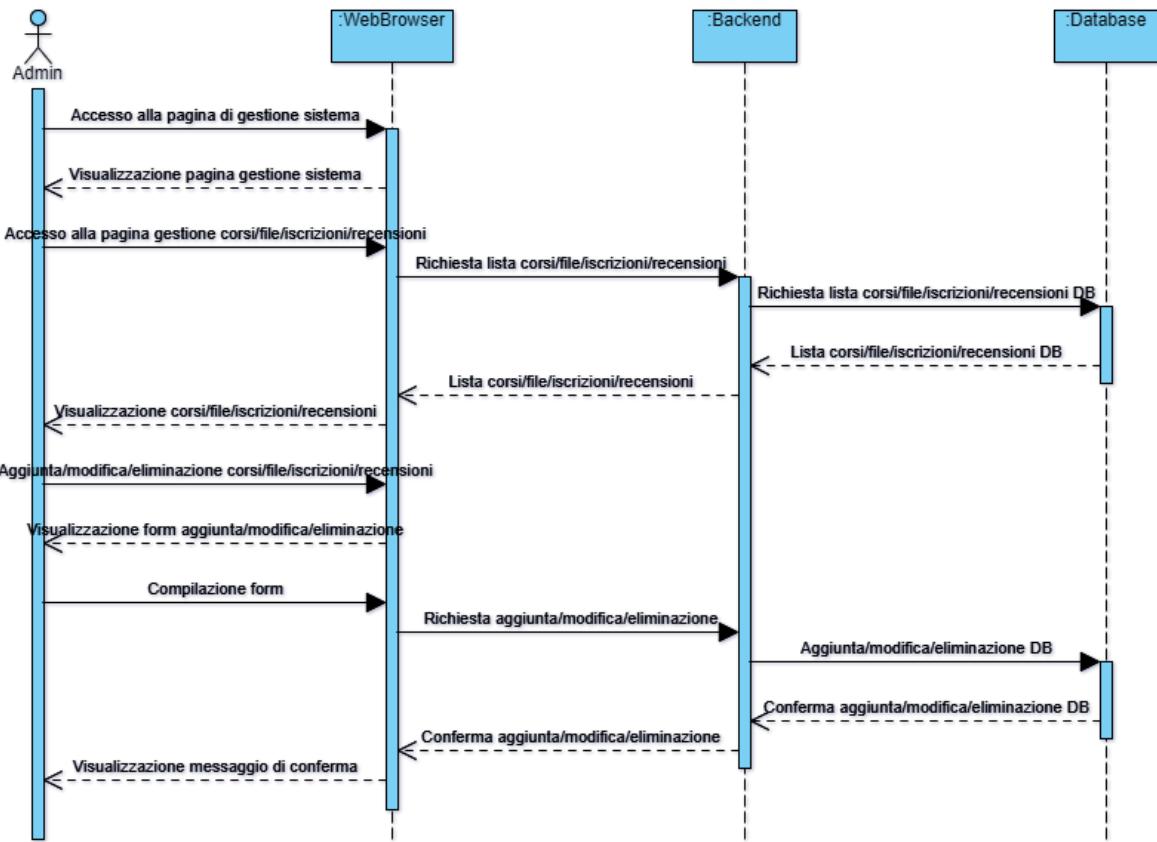


Figura 4-15: Diagramma di sequenza UC14 – Gestisci corsi, file, iscrizioni e recensioni

Capitolo 5: Progettazione

In questo capitolo verrà descritta l'architettura del sistema software Share-Hub Unina. Verranno illustrati i motivi principali e le scelte di progettazione che hanno portato all'individuazione dell'architettura di riferimento, per poi descrivere i moduli che compongono suddetta architettura, infine verranno descritte e analizzate le interfacce e le funzionalità dei moduli stessi.

5.1 Scelta dell'architettura

La scelta dell'architettura è stata fatta considerando i requisiti specifici del progetto e le responsabilità individuate mediante i diagrammi di sequenza, in particolare, volendo tenere separate la logica di presentazione, dalla logica di business e di persistenza dati, si è optato per un'**architettura a tre livelli modulare**.

Questa architettura, derivata dalle classiche architetture Client-Server e conosciuta anche come **Three-tier architecture**, divide un'applicazione in tre livelli distinti e consente lo sviluppo separato di ogni livello, facilitando il lavoro di diversi gruppi di programmatore (o di un singolo programmatore, come in questo caso!) che si possono concentrare su un solo livello alla volta.

I tre livelli sono:

1. **Livello presentazione:** punto di incontro tra utente e sistema, consente l'interazione diretta tra i due. Le funzioni principali sono la visualizzazione dei dati e la raccolta degli input dell'utente che poi devono essere indirizzati verso il livello successivo.
2. **Livello applicazione:** costituisce il nucleo vitale dell'applicazione. A questo livello avviene l'elaborazione della logica, gestita da un insieme di regole che permettono di aggiungere, modificare o rimuovere dati dal livello successivo.
3. **Livello dati:** qui vengono archiviate e gestite le informazioni elaborate dall'applicazione, di solito è costituito da un database relazionale.

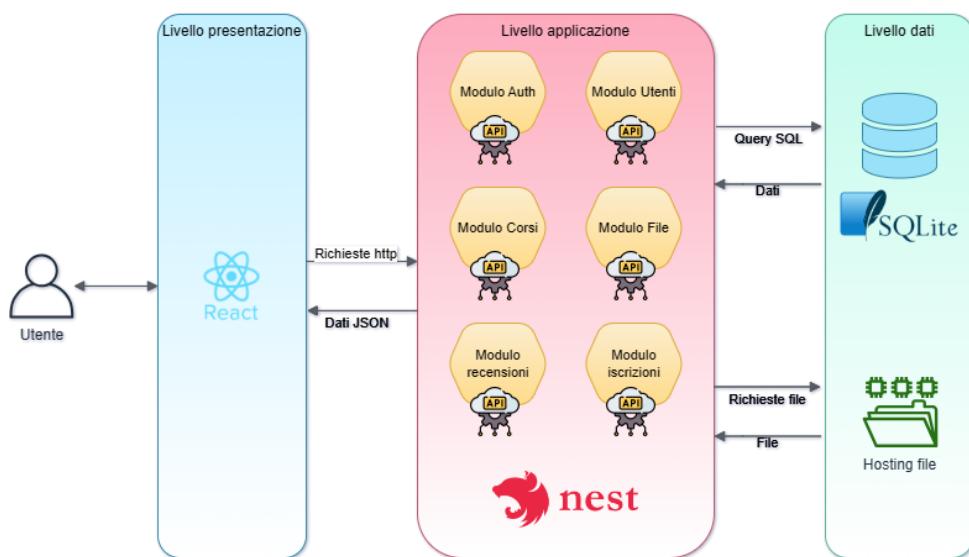


Figura 5-1: Architettura a livelli del sistema

Questa separazione rende lo **sviluppo più rapido** permettendo la scelta di tecnologie e linguaggi di programmazione adeguati diversi per ogni livello, introduce una migliore **scalabilità** in quanto ogni livello può essere scalato indipendentemente dagli altri secondo necessità, aumentando inoltre l'**affidabilità** e la **sicurezza** delineando i compiti di ognuno di essi. Inoltre, la **modularità**

introdotta nel livello applicazione offre: (1) **separazione delle responsabilità**, in quanto ogni modulo gestisce funzioni specifiche dell'applicazione; (2) **comprendione maggiore** e gestione del codice facilitata; (3) **indipendenza**, in quanto ogni modulo è progettato per essere più indipendente possibile dagli altri; (4) **testabilità**, essendo infatti ogni modulo relativamente isolato dagli altri è più semplice l'esecuzione di test; (5) **riutilizzo del codice**, infatti se un modulo è ben strutturato, il suo codice può essere riutilizzato in altri punti del sistema che richiedono funzionalità simili.

5.1.1 Scelta delle tecnologie

Il livello di presentazione, ossia il frontend della nostra applicazione, sarà realizzato con **React**, e comunicherà con il livello applicazione (backend) basato su API RESTful scritte in **NestJS**. Il backend, organizzato in moduli, si occuperà della gestione delle operazioni CRUD su diverse entità del database SQLite (il livello dati), fornendo un'interfaccia chiara e ben definita sia per il client frontend che per altri potenziali consumatori delle API. Il database SQLite è stato scelto in sostituzione di un database tradizionale per la fase di sviluppo, per la sua leggerezza, la sua velocità e per la semplicità con cui è configurabile e gestibile (è composto da un unico file), a fronte di un rilascio effettivo dell'applicazione andrebbe sostituito con un database diverso, che possa offrire gestione della concorrenza, gestione dei permessi di accesso, protocolli di rete e altro ancora. Al livello dati, oltre al database **SQLite** sarà presente un servizio di hosting file che permetterà l'upload e il download del materiale didattico (al momento non è implementato). React e NestJS sono stati scelti per la relativa inesperienza del team di sviluppo con queste due tecnologie (ancora e solo io, che sto scrivendo questa documentazione), alla ricerca di nuove tecnologie ben documentate e con una grande community alle spalle da poter aggiungere a quelle già conosciute. Per l'autenticazione si è scelto di utilizzare appositi **token JWT** che dovranno essere allegati alle richieste, questi token permetteranno di discernere gli utenti autenticati da quelli non autenticati oltre al ruolo e quindi le possibili interazioni con il sistema di ogni utente. Nella sezione 6.1 è presente una descrizione più dettagliata di tutte le tecnologie utilizzate per il progetto.

5.2 Diagramma dei componenti

Per visualizzare più in dettaglio l'architettura, senza scendere nei dettagli implementativi, viene presentato il component diagram del sistema. Questo ci è utile per individuare le parti indipendenti del sistema (moduli, servizi o librerie) che svolgono una funzione specifica in autonomia. Possiamo notare come sia visibile anche qui la divisione a livelli del sistema, ma più in dettaglio possiamo vedere come nel livello di presentazione, il front end React, verranno realizzate diverse pagine front end per ognuna delle entità individuate del sistema nella sezione 4.1; mentre nel livello applicazione, il back end NestJS, verrà creato un modulo per ognuna di esse. Questi moduli metteranno a disposizione delle Rest API che potranno essere sfruttate dalle pagine front end per recuperare i dati e file dal livello dati. I moduli del back end avranno sia un componente controller dove verrà gestita la logica di routing e di gestione delle autorizzazioni delle richieste alle API, sia un componente servizio che avrà il compito vero e proprio di elaborare le richieste e interfacciarsi con il livello dati.

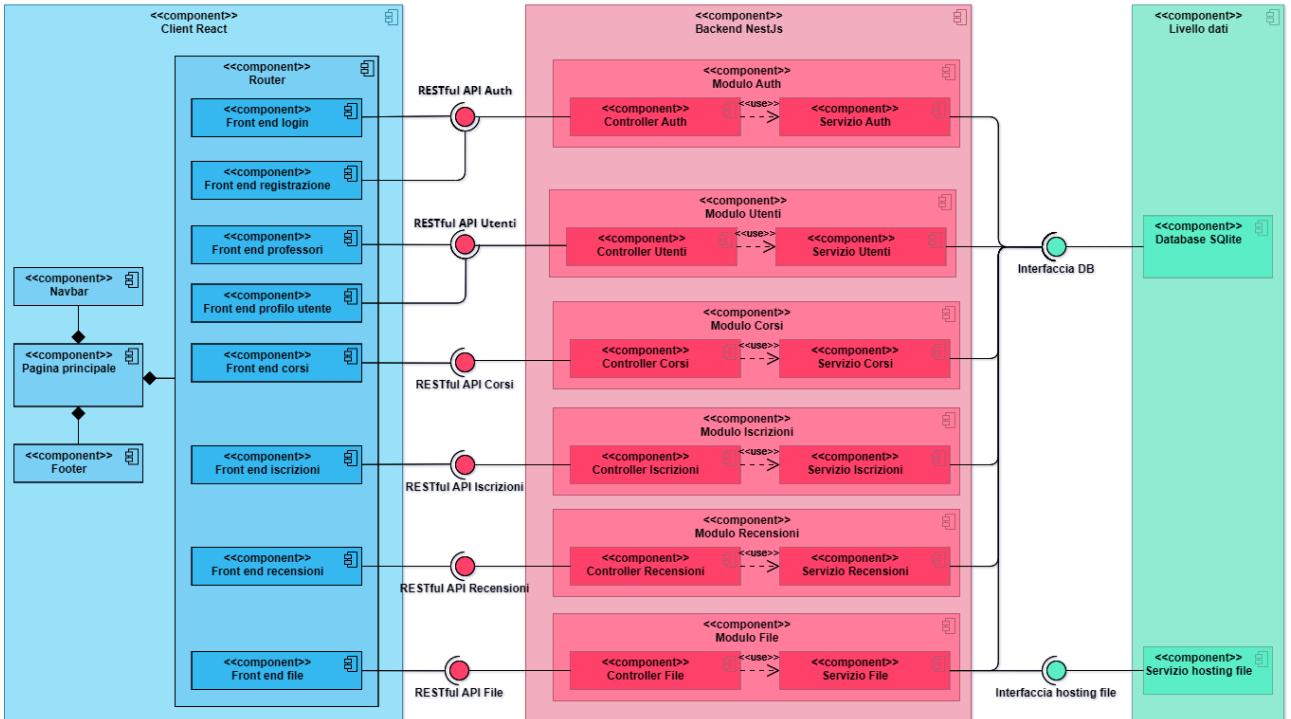


Figura 5-2: Component Diagram del sistema

5.3 Componenti frontend

Ogni componente del frontend offre un'interfaccia grafica all'utente, con la quale è possibile interagire con i componenti backend. Le diverse interfacce grafiche mandano richieste HTTP con metodi diversi per recuperare, modificare oppure eliminare i dati.

5.4 Componenti backend

In questa sezione verranno descritti i diversi moduli che compongono il back end, nucleo vero e proprio del sistema. Ognuno dei moduli offre delle funzionalità che possono soddisfare più casi d'uso del sistema (il modulo File è capace di soddisfare UC08 – Gestisci file e UC09 – Visualizza file), in particolare ogni modulo metterà a disposizione una **RESTful API** capace di offrire una parte o tutte le operazioni **CRUD (Create, Read, Update, Delete)** su una determinata risorsa del sistema. Le API (Application Programming Interface) sono un insieme di protocolli di comunicazioni tra sistemi software, per essere considerate RESTful devono rispettare diversi criteri, come: (1) comunicazione stateless, ossia senza memorizzazioni delle informazioni del client tra le richieste, dove quindi ogni richiesta è distinta e non connessa; (2) dati memorizzabili nella cache che ottimizzano le interazioni client-server; (3) le richieste devono identificare le risorse utilizzando un identificatore di risorse uniforme. L'interazione tra il client e i moduli avverrà tramite richieste HTTP, i cui metodi andranno ad identificare l'azione CRUD da compiere sulle risorse (POST – create, GET – read, PATCH – update, DELETE – delete), ciascun modulo si preoccuperà di realizzare l'azione sulla risorsa, i cui risultati saranno restituiti al client sottoforma di dati JSON. A seguire la lista dei diversi moduli con relativa descrizione:

ID	DESCRIZIONE SINTETICA	DESCRIZIONE DETTAGLIATA	CASI D'USO INTERESSATI
M1	Modulo autenticazione	Il servizio permette: - agli user non registrati di: (a) registrarsi inserendo nome, cognome, e-mail e password; (b) effettuare la procedura di login; (c) effettuare la procedura di logout	UC01 – Registrati UC02 – Login
M2	Modulo gestione degli utenti	Il servizio permette: - agli user non registrati di: (a) poter vedere la lista dei professori - agli studenti e ai professori di: (a) modificare il proprio profilo. - agli admin di: (a) modificare ed eliminare il profilo di altri utenti e il proprio	UC03 – Modifica profilo UC04 – Modifica profilo di un altro utente UC05 – Visualizza professore
M3	Modulo gestione dei corsi	Il servizio permette - agli studenti e ai professori di: (a) visualizzare l'elenco completo dei corsi sulla piattaforma, (b) visualizzare i dettagli di ogni corso. - ai professori di: (a) creare nuovi corsi, (b) modificare ed eliminare i propri corsi - agli admin di: (a) creare nuovi corsi, (b) modificare ed eliminare corsi esistenti	UC06 – Gestisci corso UC07 – Visualizza corso UC14 – Gestisci corsi, file, iscrizioni e recensioni
M4	Modulo gestione dei file	Il servizio permette - agli studenti di: (a) visualizzare i file dei corsi ai quali sono iscritti e di farne il download - ai professori di: (a) visualizzare, caricare, modificare ed eliminare file relativi ai propri corsi agli admin di: (a) visualizzare, caricare, modificare ed eliminare file relativi a tutti i corsi	UC08 – Gestisci file UC09 – Visualizza file UC14 – Gestisci corsi, file, iscrizioni e recensioni
M5	Modulo gestione delle iscrizioni	Il servizio permette - agli studenti di: (a) iscriversi ai corsi, (b) visualizzare ed eliminare le proprie iscrizioni - ai professori di: (a) visualizzare le iscrizioni degli studenti relative ai propri corsi - agli admin di: (a) iscriversi ai corsi, (b) di visualizzare ed eliminare le iscrizioni esistenti	UC10 – Gestisci iscrizione UC11 – Visualizza iscrizioni UC14 – Gestisci corsi, file, iscrizioni e recensioni
M6	Modulo gestione delle recensioni	Il servizio permette - agli studenti di: (a) aggiungere recensioni a file che fanno parte di corsi ai quali sono iscritti, (b) visualizzare, modificare ed eliminare le proprie recensioni - ai professori di: (a) visualizzare recensioni relative a file dei propri corsi - agli admin di: (a) aggiungere recensioni a file, (b) visualizzare, modificare ed eliminare le recensioni	UC12 – Gestisci recensione UC13 – Visualizza recensione UC14 – Gestisci corsi, file, iscrizioni e recensioni

5.4.1 API endpoints

Verranno descritti in questa sezione gli endpoint esposti dalle diverse API del backend, questi endpoint sono delle specifiche locazioni all'interno dell'API che accettano richieste HTTP e restituiscono dati in formato JSON. La colonna “Permessi” indica quale ruolo deve avere l'utente per poter mandare il tipo di richiesta; nella colonna “Parametri richiesta” sono riportati tra parentesi graffe i parametri obbligatori, mentre sono prefissati con un punto interrogativo i parametri di query, che sono opzionali.

Per evitare ripetizioni, è sottinteso che ogni richiesta ha anche le possibili risposte:

- 400: Bad request (se la richiesta non è formattata correttamente)
- 401: Unauthorized (se il ruolo dell'utente non corrisponde con i permessi)

Mentre andrebbe implementata la risposta 404: Not Found quando un elemento acceduto tramite url non viene trovato

Una documentazione simile dei vari API endpoints, prodotta in automatico tramite **Swagger**, è consultabile all'URL “/apidocs” dell'applicazione. Alcune funzioni sono testabili, mentre altre, richiedendo particolare permessi, non lo sono. In generale si rimanda a sviluppi futuri del sistema il miglioramento della documentazione ottenuta tramite Swagger.

POST api/auth/signup Non protetto				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Permette di registrare un nuovo utente		JSON dettagli utente, esempio: { "email" : "antonio.sposito@studenti.unina.it", "password" : "*****", "name": "Antonio", "lastname": "Sposito" }	201: Created	{ "message": "signup was successfull" }
			400: Bad request	{ "message": "Email already exists", "error": "Bad Request", "statusCode": 400 }
POST api/auth/signin Non protetto				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Permette di fare il login, restituisce all'utente il token di autenticazione JWT		JSON credenziali utente: { "email" : "antonio.sposito@studenti.unina.it", "password" : "*****" }	201: Created	JSON dettagli utente loggato: { "message": "logged in successfully", "user": { "id": 2, "email": "antonio.sposito@studenti.unina.it", "name": "Antonio", "lastname": "Sposito", "role": "Student" } }
			400: Bad request	{ "message": "Wrong credentials", "error": "Bad Request", "statusCode": 400 }
GET /api/auth/signout Utenti loggati				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta

Permette di fare il logout, rimuove token di autenticazione JWT			200: Ok	{ "message": "logged out successfully" }
---	--	--	---------	--

Tabella 5-1: Descrizione degli API endpoints del servizio Auth

GET		/api/users	Admin	
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'elenco di tutti gli utenti			200: OK	JSON dettagli utenti (tranne password): [{ "id": 1, "name": "admin", "lastname": "admin", "email": "admin@unina.it", "role": "Admin" }, {"id": 2, ... },]
GET		/api/users/professors	Utenti loggati	
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'elenco di tutti i professori			200: OK	JSON dettagli professori (tranne password): [{ "id": 4, "name": "Anna Rita", "lastname": "Fasolino", "email": "annarita.fasolino@unina.it", "role": "Professor" }, {"id": 5, ... },]
GET		/api/users/{id}	Utenti loggati	
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce i dettagli dell'utente a cui appartiene l'id specificato. Gli admin possono visualizzare tutte le pagine utenti, i professori e gli studenti solo le pagine relative a tutti i professori più la propria pagina	{id}: id dell'utente da visualizzare		200:OK	JSON dettagli utente (tranne password) { "id": 2, "name": "Antonio", "lastname": "Sposito", "email": "antonio.sposito@studenti.unina.it", "role": "Student" }
			403: Forbidden	{ "message": "Access denied. Students can view only professors pages and their own", "error": "Forbidden", "statusCode": 403 }
PATCH		/api/users/{id}	Admin o Self	
Descrizione	Parametri	Body richiesta	Risposta	Body risposta

Aggiorna le informazioni riguardanti l'user a cui appartiene l'id specificato	{id}: id dell'utente da modificare	<u>JSON dettagli utente da modificare:</u> { "name": "Antonio", "lastname": "Sposito", "email": "antonio.sposito2@studenti.unina.it" }	200: OK	<u>JSON dettagli utente modificato (tranne password), esempio:</u> { "id": 2, "name": "Antonio", "lastname": "Sposito", "email": "antonio.sposito2@studenti.unina.it", "role": "Student" }
DELETE		/api/users/{id}		Admin
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Elimina l'user a cui appartiene l'id specificato	{id}: id dell'utente da eliminare		200: OK	<u>JSON dettagli utente eliminato (tranne password), esempio:</u> { "id": 2, "name": "Antonio", "lastname": "Sposito", "email": "antonio.sposito2@studenti.unina.it", "role": "Student" }

Tabella 5-2: Descrizione degli API endpoints del servizio Users

POST	/api/courses			Admin e Professori
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Creazione di un corso. Dopo la creazione, il campo userId del corso corrisponderà all'id del professore che lo sta creando. Se invece lo sta creando un Admin, può scegliere lo userId che vuole, a patto che sia presente un user con quell'id		<p>JSON dettagli corso, esempio:</p> <pre>{ "userId": 1, "title" : "titolo corso", "description": "descrizione" }</pre>	201: Created	<p>JSON dettagli corso creato, esempio:</p> <pre>{ "id": 56, "title": "titolo corso", "description": "descrizione", "createdAt": "...", "modifiedAt": "...", "userId": 1 }</pre>
GET	/api/courses			Utenti loggati
Descrizione	Parametri	Body richiesta	Risposta	Body risposta

Restituisce l'elenco di tutti i corsi			200: OK	<u>JSON dettagli tutti corsi, esempio:</u> <pre>[{ "id": 4, "title": "IS", "description": "Ingegneria del Software", "createdAt": "...", "modifiedAt": "...", "userId": 4 }, { "id": 5, "title": "RTIS", "description": [...] }, ...]</pre>
GET <u>/api/courses?userId=X</u> Utenti loggati				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'elenco dei corsi che sono tenuti dall'utente a cui appartiene userId				
	?userId = id del professore di cui si vogliono visualizzare i corsi		200: OK	<u>JSON dettagli di tutti corsi del professore, esempio:</u> <pre>[{ "id": 5, "title": "RTIS", "description": "Real Time Industrial Systems", "createdAt": "...", "modifiedAt": "...", "userId": 5 }, { "id": 6, "title": "RTS", "description": "Real time systems", "createdAt": "", "modifiedAt": "", "userId": 5 }]</pre>
GET <u>/api/courses/{id}</u> Utenti loggati				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce i dettagli del corso a cui appartiene l'id specificato	{id}: id del corso da visualizzare		200: OK	<u>JSON dettagli corso, esempio:</u> <pre>{ "id": 4, "title": "IS", "description": "Ingegneria del Software", "createdAt": "", "modifiedAt": "", "userId": 4 }</pre>
PATCH <u>/api/courses/{id}</u> Admin e Professori				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta

Aggiorna le informazioni riguardante il corso a cui appartiene l'id specificato. Il campo userId del corso deve corrispondere all'id del professore che lo sta modificando	{id}: id del corso da modificare	<u>JSON dettagli corso da modificare, esempio:</u> { "userId": 4, "title" : "IS2", "description": "Ingegneria del Software 2" }	200: OK	<u>JSON dettagli corso modificato, esempio:</u> { "id": 4, "title" : "IS2", "description": "Ingegneria del Software 2", "createdAt": "...", "modifiedAt": "...", "userId": 4 }
			403: Forbidden	{ "message": "Only admins and the Professor owner of this course can edit it.", "error": "Forbidden", "statusCode": 403 }
DELETE /api/courses/{id}		Admin e Professori		
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Elimina il corso a cui appartiene l'id specificato. Il campo userId del corso deve corrispondere all'id del professore che lo sta eliminando	{id}: id del corso da eliminare		200: OK	<u>JSON dettagli corso eliminato, esempio:</u> { "id": 4, "title" : "IS2", "description": "Ingegneria del Software 2", "createdAt": "...", "modifiedAt": "...", "userId": 4 }
			403: Forbidden	{ "message": "Only admins and the Professor owner of this course can delete it.", "error": "Forbidden", "statusCode": 403 }

Tabella 5-3: Descrizione degli API endpoints del servizio Courses

POST		/api/files		Admin e Professori
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Creazione di un file. Un professore può creare un file solamente se il courseId del file corrisponde ad un corso da lui tenuto.		JSON dettagli file, esempio: <pre>{ "name": "Slide teoria 1", "description": "Prima slide di teoria", "courseId": 47 }</pre>	201: File created 403: Forbidden	JSON dettagli file creato, esempio: <pre>{ "id": 27, "name": "Slide teoria 1", "description": "Prima slide di teoria", "path": "Dammi/un/path!", "avgRating": null, "createdAt": "...", "modifiedAt": "...", "courseId": 47 }</pre> <pre>{ "message": "Access denied. You cannot upload files with a courseId of a course you are not the owner", "error": "Forbidden", "statusCode": 403 }</pre>

GET /api/files Admin				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'elenco di tutti i file			200: OK	<p>JSON dettagli di tutti i file, esempio:</p> <pre>[{ "id": 1, "name": "Slide 1", "description": "Dammi una descrizione!", "path": "Dammi/un/path!", "avgRating": 2, "createdAt": "...", "modifiedAt": "...", "courseId": 5 }, { "id": 2, "name": "Slide 2", "description": "Dammi una descrizione!", "path": "Dammi/un/path!", "avgRating": 4, "createdAt": "...", "modifiedAt": "...", "courseId": 5 }, ...]</pre>
GET /api/files?courseId=X Utenti loggati				
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'elenco di tutti i file che appartengono al corso X. L'elenco viene restituito se l'user è admin; oppure se l'user è professore e i file fanno parte di un suo corso; oppure se l'user è studente e i file fanno parte di un corso a cui è iscritto	?courseId: id del corso di cui si vogliono visualizzare i file		200:OK	<p>JSON dettagli di tutti i file di un corso, esempio:</p> <pre>[{ "id": 18, "name": "Slide molto interessante", "description": "s", "path": "Dammi/un/path!", "avgRating": 0, "createdAt": "...", "modifiedAt": "...", "courseId": 4 }, { "id": 20, "name": "Pdf lunghissimo", "description": "ciao", "path": "Dammi/un/path!", "avgRating": null, "createdAt": "...", "modifiedAt": "...", "courseId": 4 }, ...]</pre>
			403: Forbidden	<pre>{ "message": "Access denied. You are not the professor of this course so you cannot see its files.", "error": "Forbidden", "statusCode": 403 }</pre>

				<pre>{ "message": "Access denied. You are not enrolled in this course so you cannot see its files.", "error": "Forbidden", "statusCode": 403 }</pre>
--	--	--	--	--

GET		/api/files/{id}		Utenti loggati
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce i dettagli del file a cui appartiene l'id specificato. Il file viene restituito se l'user è admin; oppure se l'user è professore e il file fa parte di un suo corso; oppure se l'user è studente e il file fa parte di un corso a cui è iscritto	{id}: id del file da visualizzare		200: OK	<u>JSON dettagli file, esempio:</u> <pre>{ "id": 1, "name": "Slide 1", "description": "Dammi una descrizione!", "path": "Dammi/un/path!", "avgRating": 2, "createdAt": "...", "modifiedAt": "...", "courseId": 5 }</pre>
			403: Forbidden	<pre>{ "message": "Access denied. You are not the professor that owns this file", "error": "Forbidden", "statusCode": 403 }</pre>
				<pre>{ "message": "Access denied. You are not enrolled in the course this file is from", "error": "Forbidden", "statusCode": 403 }</pre>

PATCH		/api/files/{id}		Admin e Professori
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Aggiorna le informazioni riguardante il file a cui appartiene l'id specificato. Il file viene modificato solo se l'user è admin; oppure se l'user è professore e il campo courseId del file corrisponde ad un corso che lui	{id}: id del file da modificare	<u>JSON dettagli file da modificare, esempio:</u> <pre>{ "name": "Slide 3", "description": "Nuova descrizione", "courseId": 5 }</pre>	200: OK	<u>JSON dettagli file modificato, esempio:</u> <pre>{ "id": 5, "name": "Slide 3", "description": "Nuova descrizione", "path": "Dammi/un/path!", "avgRating": -1, "createdAt": "...", "modifiedAt": "...", "courseId": 5 }</pre>

tiene. Inoltre, non si può modificare il campo courseId del file.			403: Forbidden	<pre>{ "message": "Access denied. You are not the professor that owns this file, thus you cannot edit it", "error": "Forbidden", "statusCode": 403 }</pre>
				<pre>{ "message": "Access denied. You cannot modify the course of an existing file", "error": "Forbidden", "statusCode": 403 }</pre>
DELETE	/api/files/{id}			Admin e Professori
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Elimina il file a cui appartiene l'id specificato. Il file viene eliminato solo se l'user è admin; oppure se l'user è professore e il campo courseId del file corrisponde ad un corso che lui tiene	{id}: id del file da eliminare		200: OK	<u>JSON dettagli file eliminato, esempio:</u> <pre>{ "id": 5, "name": "Slide 3", "description": "Nuova descrizione", "path": "Dammi/un/path!", "avgRating": -1, "createdAt": "...", "modifiedAt": "...", "courseId": 5 }</pre>
			403: Forbidden	<pre>{ "message": "Access denied. You are not the professor that owns this file, thus you cannot delete it", "error": "Forbidden", "statusCode": 403 }</pre>

Tabella 5-4: Descrizione degli API endpoints del servizio Files

POST	/api/reviews/			Admin e Studenti
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Creazione di una recensione per il file con l'id specificato. La recensione può essere creata da uno studente solo se lo studente è iscritto al corso al cui associato oppure se l'utente è		<u>JSON dettagli recensione, esempio:</u> <pre>{ "text": "Mi è piaciuto molto", "rating" : 5, "userId": 1, "fileId": 7 }</pre>	201: Review created	<u>JSON dettagli recensione creata, esempio:</u> <pre>{ "id": 28, "text": "Mi è piaciuto molto", "rating": 5, "userId": 1, "fileId": 7, "createdAt": "...", "modifiedAt": "..." }</pre>

admin. Ogni utente può recensire una sola volta ogni file.			403: Forbidden	<pre>{ "message": "Access denied. You can only review a file once", "error": "Forbidden", "statusCode": 403 }</pre> <pre>{ "message": "Access denied. You can only review files if you are enrolled in the course the file is from.", "error": "Forbidden", "statusCode": 403 }</pre>
--	--	--	-------------------	--

GET	/api/reviews/			Admin
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'elenco di tutte le recensioni			200: OK	<p>JSON dettagli di tutte le recensioni, esempio:</p> <pre>[{ "id": 16, "text": "Mi è piaciuto molto", "rating": 3, "userId": 2, "fileId": 2, "createdAt": "...", "modifiedAt": "..." }, { "id": 17, "text": "molto utile", "rating": 5, "userId": 1, "fileId": 2, "createdAt": "...", "modifiedAt": "..." }, ...]</pre>

GET	/api/reviews?userId=X&fileId=Y			Utenti loggati
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce le recensioni scritte dall'utente che ha userId = X e/o le recensioni del file che ha fileId=Y. Le recensioni vengono restituite se l'user è admin; oppure se l'user è professore e le recensioni sono	?userId = id dello studente di cui si vogliono visualizzare le recensioni ?fileId = id del file di cui si vogliono visualizzare le recensioni		200: OK	<p>JSON dettagli recensioni dell'utente/relative al file:</p> <pre>{ "id": 16, "text": "Mi è piaciuto molto", "rating": 3, "userId": 2, "fileId": 2, "createdAt": "...", "modifiedAt": "..." }</pre>

di un file che appartiene ad un suo corso; oppure se l'user è studente e ha scritto lui le recensioni			403: Forbidden	<pre>{ "message": "Access denied. You are trying to access reviews of a file and you are not the owner of the course the file is from", "error": "Forbidden", "statusCode": 403 }</pre>
				<pre>{ "message": "Access denied. You are trying to access reviews of a file from a course you are not enrolled in", "error": "Forbidden", "statusCode": 403 }</pre>
				<pre>{ "message": "Access denied. You are not the owner of these reviews", "error": "Forbidden", "statusCode": 403 }</pre>

GET		/api/reviews/{id}	Admin e Studenti	
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce la recensione a cui appartiene l'id specificato. La recensione viene restituita se l'user è admin; oppure se l'user è studente e ha scritto lui la recensione	{id}: id della recensione da visualizzare		200: OK	<p>JSON dettagli recensione creata, esempio:</p> <pre>{ "id": 28, "text": "Mi è piaciuto molto", "rating": 5, "userId": 1, "fileId": 7, "createdAt": "...", "modifiedAt": "..."}</pre>
			403: Forbidden	<pre>{ "message": "Access denied. You cannot see a review you did not write", "error": "Forbidden", "statusCode": 403 }</pre>

PATCH		/api/reviews/{id}	Admin e Studenti	
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Aggiorna le informazioni riguardante la recensione a cui appartiene l'id specificato. La recensione viene modificata se l'user è admin; oppure se l'user è lo studente che ha scritto la recensione. L'utente non	{id}: id della recensione da modificare	<u>JSON dettagli recensione da modificare, esempio:</u> <pre>{ "text": "Poteva essere meglio", "rating" : 2, "userId": 2, "fileId": 2 }</pre>	200: OK	<u>JSON dettagli recensione modificata, esempio:</u> <pre>{ "id": 16, "text": "Mi è piaciuto moltooooo", "rating": 3, "userId": 2, "fileId": 2, "createdAt": "...", "modifiedAt": "..."}</pre>

può modificare il file o l'utente a cui la recensione si riferisce			403: Forbidden	{ "message": "Access denied. You cannot modify the fileId or the userId of a review", "error": "Forbidden", "statusCode": 403 }
DELETE	/api/reviews/{id}			Admin e Studenti
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Elimina la recensione a cui appartiene l'id specificato. La recensione viene eliminata se l'user è admin; oppure se l'user è lo studente che ha scritto la recensione	{id}: id della recensione da eliminare		200: OK 403: Forbidden	<u>JSON dettagli recensione eliminata, esempio:</u> { "id": 25, "text": "Molto utile, però mancano alcune cose", "rating": 4, "userId": 2, "fileId": 7, "createdAt": "...", "modifiedAt": ... } <u>message": "Access denied. You cannot delete a review you did not write",</u> "error": "Forbidden", "statusCode": 403

Tabella 5-5: Descrizione degli API endpoints del servizio Reviews

POST	/api/enrollments/			Admin e Studenti
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Creazione di un'iscrizione ad un corso che corrisponde al courseId specificato. Dopo la creazione, il campo userId dell'iscrizione corrisponderà all'id dello studente che la sta creando. Se invece la sta creando un Admin, può scegliere lo userId che vuole, a patto che sia presente un user con quell'id		<u>JSON dettagli iscrizione, esempio:</u> { "userId": 2, "courseId": 35 }	201: Enrollment created()	<u>JSON dettagli iscrizione creata, esempio:</u> { "id": 46, "userId": 2, "courseId": 35, "createdAt": "...", "modifiedAt": ... }
GET	/api/enrollments/			Admin
Descrizione	Parametri	Body richiesta	Risposta	Body risposta

Restituisce l'elenco di tutte le iscrizioni			200: OK	<u>JSON dettagli di tutte le iscrizioni, esempio:</u> [{ "id": 16, "userId": 1, "courseId": 4, "createdAt": "...", "modifiedAt": "..." }, { "id": 26, "userId": 2, "courseId": 4, "createdAt": "...", "modifiedAt": "..." }, ...]
---	--	--	---------	---

GET /api/enrollments?userId=X&courseId=Y		Utenti loggati		
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'elenco delle iscrizioni dell'utente che corrisponde a userId e/o l'elenco delle iscrizioni al corso che corrisponde a courseId. Le iscrizioni vengono restituite se l'utente è admin; oppure se l'user è professore e le iscrizioni sono per un corso che lui tiene; oppure se l'user è studente e le iscrizioni sono relative a lui	?userId = id dello studente di cui si vogliono visualizzare le iscrizioni ?courseId = id del corso di cui si vogliono visualizzare le iscrizioni		200: OK	<u>JSON dettagli iscrizioni dell'utente/al corso, esempio:</u> [{ "id": 16, "userId": 1, "courseId": 4, "createdAt": "...", "modifiedAt": "..." }, { "id": 26, "userId": 2, "courseId": 4, "createdAt": "...", "modifiedAt": "..." }, ...] 403: Forbidden { "message": "Access denied. You are not the owner of these enrollments", "error": "Forbidden", "statusCode": 403 } { "message": "Access denied. You are not the owner of this course, thus you cannot see its enrollments", "error": "Forbidden", "statusCode": 403 }
			403: Forbidden	

GET /api/enrollments/{id}		Admin e Studenti		
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Restituisce l'iscrizione a cui appartiene l'id specificato. L'iscrizione viene restituita se l'user è admin; oppure se l'user è colui	{id}: id dell'iscrizione da visualizzare		200: OK	<u>JSON dettagli iscrizione, esempio:</u> { "id": 35, "userId": 2, "courseId": 6, "createdAt": "2024-09-01T09:28:54.949Z", "modifiedAt": "2024-09-01T09:28:54.949Z" }

che ha scritto la recensione			403: Forbidden	{ "message": "Access denied. You cannot see the details of an enrollment if you're not the owner", "error": "Forbidden", "statusCode": 403 }
DELETE		/api/enrollments/{id}		Admin e Studenti
Descrizione	Parametri	Body richiesta	Risposta	Body risposta
Elimina l'iscrizione a cui appartiene l'id specificato. La recensione viene eliminata se l'user è admin; oppure se l'user è lo studente che ha scritto la recensione	{id}: id dell'iscrizione da eliminare		200: OK	JSON dettagli iscrizione eliminata, esempio: { "id": 35, "userId": 2, "courseId": 6, "createdAt": "...", "modifiedAt": "..." }
			403: Forbidden	{ "message": "Access denied. You cannot delete an enrollment that's not yours", "error": "Forbidden", "statusCode": 403 }

Tabella 5-6: Descrizione degli API endpoints del servizio Enrollments

5.5 Componenti livello dati

Il livello dati è composto da due componenti, il servizio di hosting file (che ancora deve essere implementato allo stato attuale del sistema) e il database SQLite che contiene le informazioni necessarie per poter utilizzare il sistema. Il backend comunica con il database tramite query SQL e questo restituisce i valori desiderati; da notare che SQLite non gestisce in alcun modo le autorizzazioni di accesso alle singole tabelle, tutta la gestione è rimandata al livello applicazione e ai token JWT.

In una fase più avanzata dello sviluppo si potrebbe optare per un altro database relazionale come MySQL o PostgreSQL, ma allo stato attuale del sistema, dove è continua la modifica e l'integrazione di nuove funzionalità, si preferisce utilizzare SQLite in quanto basato un singolo file portatile e leggero, non richiede configurazione ed è progettato per avere prestazioni elevate su database piccoli o medi, come nel nostro caso.

5.5.1 Diagramma delle classi

Nel diagramma sono omessi il campo “createdAt” e “modifiedAt” di ogni classe, utilizzati per tener traccia della creazione e dell'ultima modifica rispettivamente delle entità.

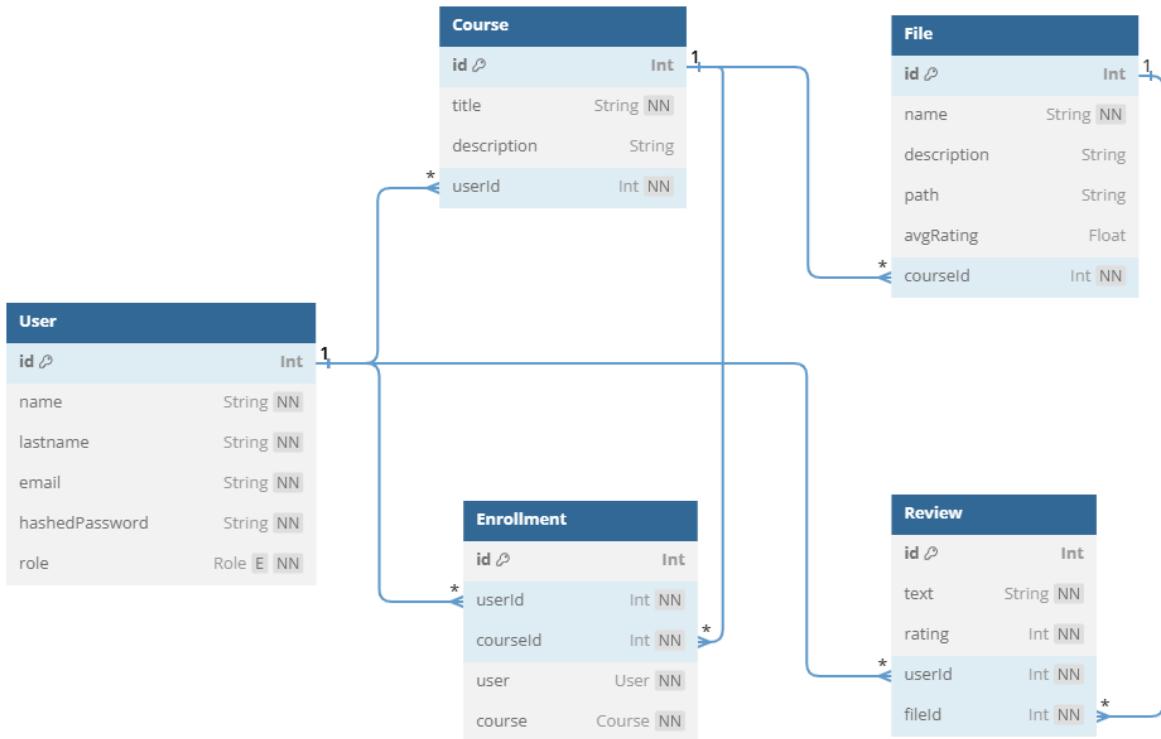


Figura 5-3: Diagramma delle classi del database. La notazione “NN” indica un parametro che non può essere nullo. Il campo “role” dell’entità User è un enumeration e può essere pari a Admin, Professor o Student

Capitolo 6: Implementazione

In questo capitolo viene descritta in dettaglio l'implementazione del sistema Share-Hub Unina, con un focus su come i vari componenti sono stati strutturati e integrati. Verrà presentata una descrizione dettagliata delle tecnologie utilizzate per sviluppare il sistema seguita dai diagrammi che illustrano la composizione del backend. Inoltre, sarà fornita una panoramica del codice, con un'analisi della suddivisione in package e dei principali elementi implementativi. L'obiettivo è fornire una visione completa e approfondita delle scelte tecniche e dell'organizzazione del progetto.

6.1 Descrizione delle tecnologie utilizzate

In questa sezione vengono illustrate le principali tecnologie utilizzate per lo sviluppo dell'applicazione. La scelta delle tecnologie è stata guidata dall'obiettivo di creare un'applicazione moderna, scalabile, e facilmente mantenibile. Ogni tecnologia è stata selezionata per le sue caratteristiche specifiche e per il contributo che offre all'implementazione e al ciclo di vita del progetto.



[React](#) è la libreria JavaScript utilizzata per sviluppare l'interfaccia utente del frontend. In questo progetto, la configurazione di [Vite](#) è stata utilizzata come tool di build per React. Vite offre un ambiente di sviluppo veloce e moderno, con tempi di avvio quasi istantanei e un rapido hot module replacement (HMR), migliorando così il flusso di lavoro e la produttività.



[React Bootstrap](#) è stato utilizzato per la costruzione dell'interfaccia grafica, una libreria che combina la potenza di Bootstrap con l'approccio component-based di React. Questa soluzione consente di utilizzare i componenti di Bootstrap (bottoni, navbar, form, modali, ecc.) in modo nativo all'interno di React, sfruttando la modularità e la sintassi JSX.



[NestJS](#) è un framework Node.js progettato per la creazione di applicazioni server-side scalabili e strutturate. Basato su TypeScript, offre un'architettura modulare che facilita la gestione e l'organizzazione del codice backend. La struttura a moduli di NestJS permette di suddividere le funzionalità in unità indipendenti e riutilizzabili, mantenendo il sistema facilmente manutenibile.



[Turbo](#) è uno strumento che permette di gestire progetti multipiattaforma in modo efficiente, facilitando lo sviluppo congiunto del frontend React e del backend NestJS all'interno dello stesso repository. Consente di eseguire build parallele e di ottimizzare i tempi di sviluppo.



[Prisma](#) è un ORM (Object-Relational Mapping) che facilita l'interazione con il database. Consente di scrivere query SQL in modo tipizzato e intuitivo direttamente da TypeScript, automatizzando la gestione dei dati e la sincronizzazione con lo schema del database. Prisma si integra perfettamente con NestJS, offrendo strumenti per la migrazione del database e la generazione automatica di modelli.



[Visual Studio Code \(VS Code\)](#) è l'IDE (Integrated Development Environment) scelto per lo sviluppo del progetto. È leggero, altamente personalizzabile, e supporta una vasta gamma di estensioni per la programmazione con React, NestJS, Prisma e TypeScript.



[Visual Paradigm](#) è uno strumento di modellazione visuale utilizzato per creare diagrammi UML, come diagrammi delle classi, dei componenti e di attività. È stato utilizzato per rappresentare graficamente l'architettura e il comportamento del sistema, facilitando la comprensione e la documentazione del progetto.



[Swagger](#) è un insieme di strumenti open source che semplifica la creazione, la documentazione e il test di API RESTful. È stato utilizzato per produrre in automatico la documentazione dei vari endpoint delle API.



[Postman](#) è uno strumento di sviluppo che consente di testare, documentare e automatizzare le API. È stato utilizzato per il testing dei vari endpoint delle API.



GitHub è stato utilizzato per il controllo di versione del codice; infatti, ogni parte di esso viene tracciata e archiviata in repository, con la possibilità di gestire branch, richieste di pull e revisione del codice.



Markmap è uno strumento che trasforma il testo markdown in una mappa interattiva. È stato utilizzato per creare la mappa della codebase di questo progetto, sfruttando la sua estensione di VS code.

6.2 Composite structure diagram backend

Si è scelto di sviluppare prima in dettaglio il composite structure diagram del livello applicativo, quindi del backend NestJS, per fornire una visione d'insieme della struttura modulare di questo livello e delle interazioni tra i vari moduli. Per evitare di dover visualizzare questo documento con un livello di zoom eccessivo, verranno riportate sezioni di questo diagramma di seguito con una rapida descrizione di ognuna.

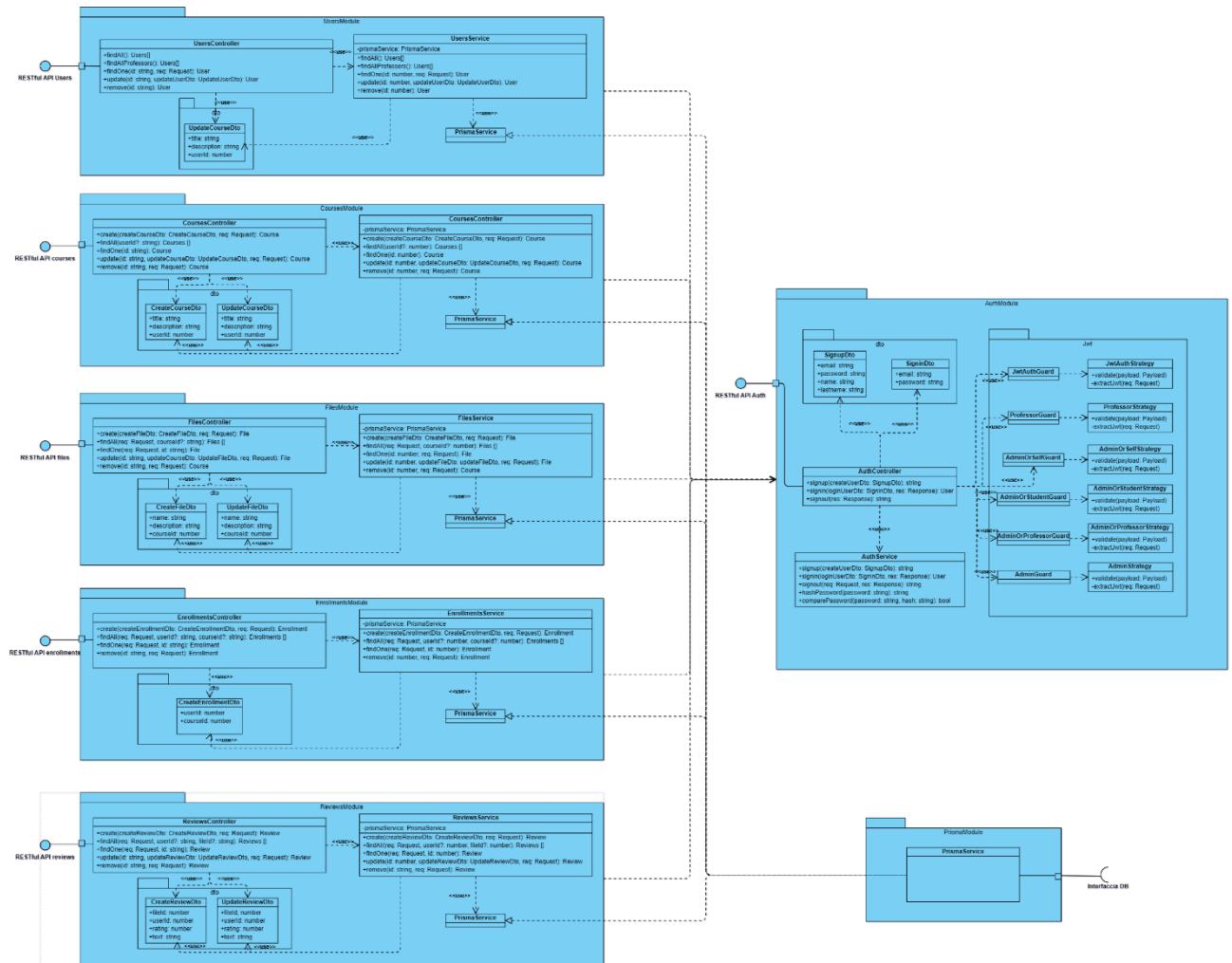


Figura 6-1: Composite Structure Diagram del backend

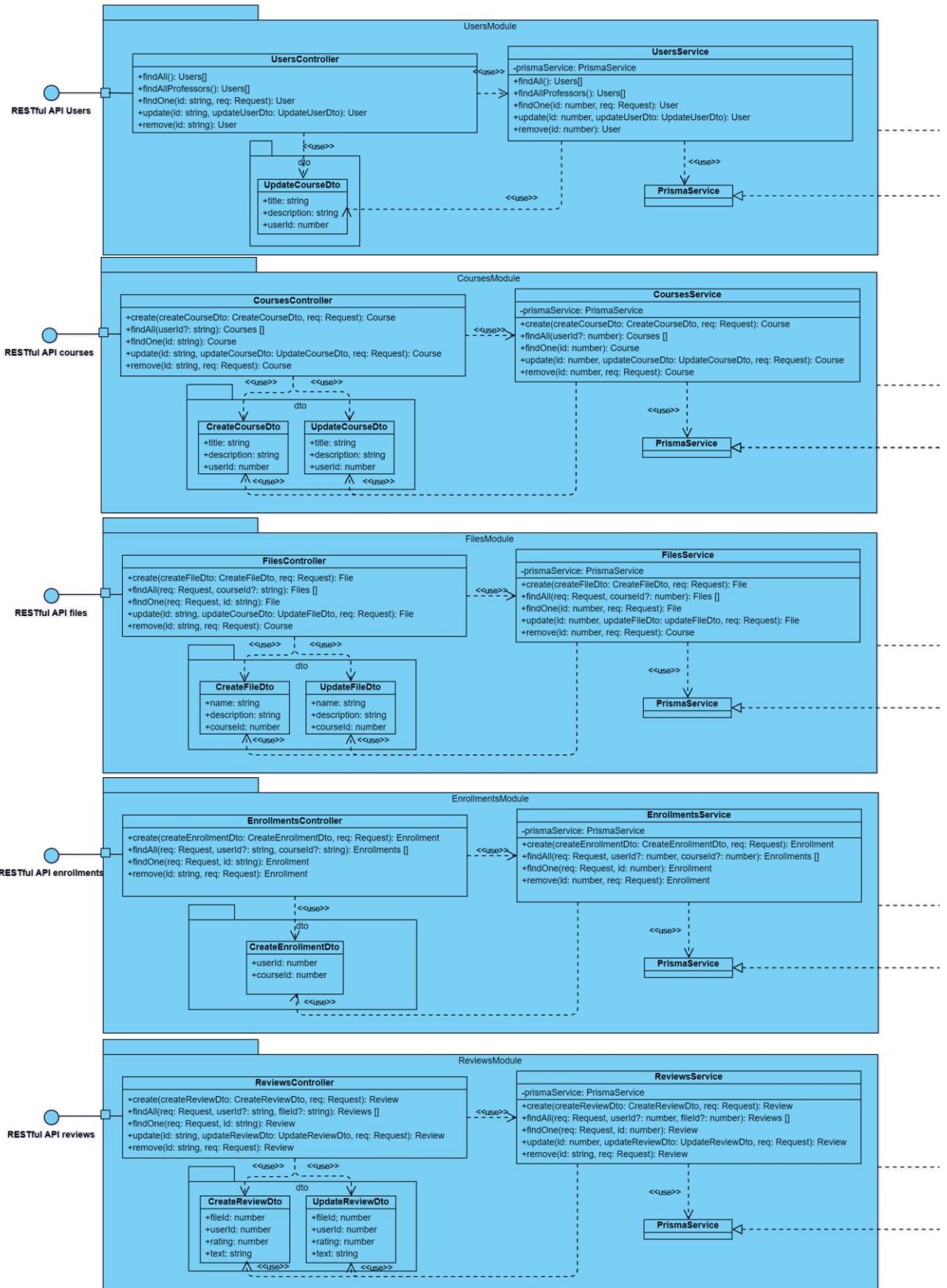


Figura 6-2: Dettagli dei moduli Utenti, Corsi, File, Iscrizioni e Recensioni del Composite Structure Diagram backend

Si può notare, come già illustrato nel component diagram nella sezione 5.2, come ogni modulo sia composto da diversi elementi: (1) un **controller**, che ha il compito di esporre gli API endpoints descritti nella sezione 5.4.1 verso l'esterno; (2) una cartella che contiene i **DTO (Data Transfer Object)**, utilizzati per modellare i dati ricevuti tramite le richieste HTTP ai vari API endpoints e i dati da mandare verso il database; (3) un **servizio**, che ha il compito di effettuare le operazioni preliminari sui DTO ricevuti dal controller prima di reindirizzare i dati verso il database; (4) un'istanza del **servizio Prisma**, servizio di ORM (Object-Relational Mapping) utilizzato per interagire con il database SQLite. I moduli **utente**, **corsi**, **file**, **iscrizioni** e **recensioni** hanno tutti questa struttura.

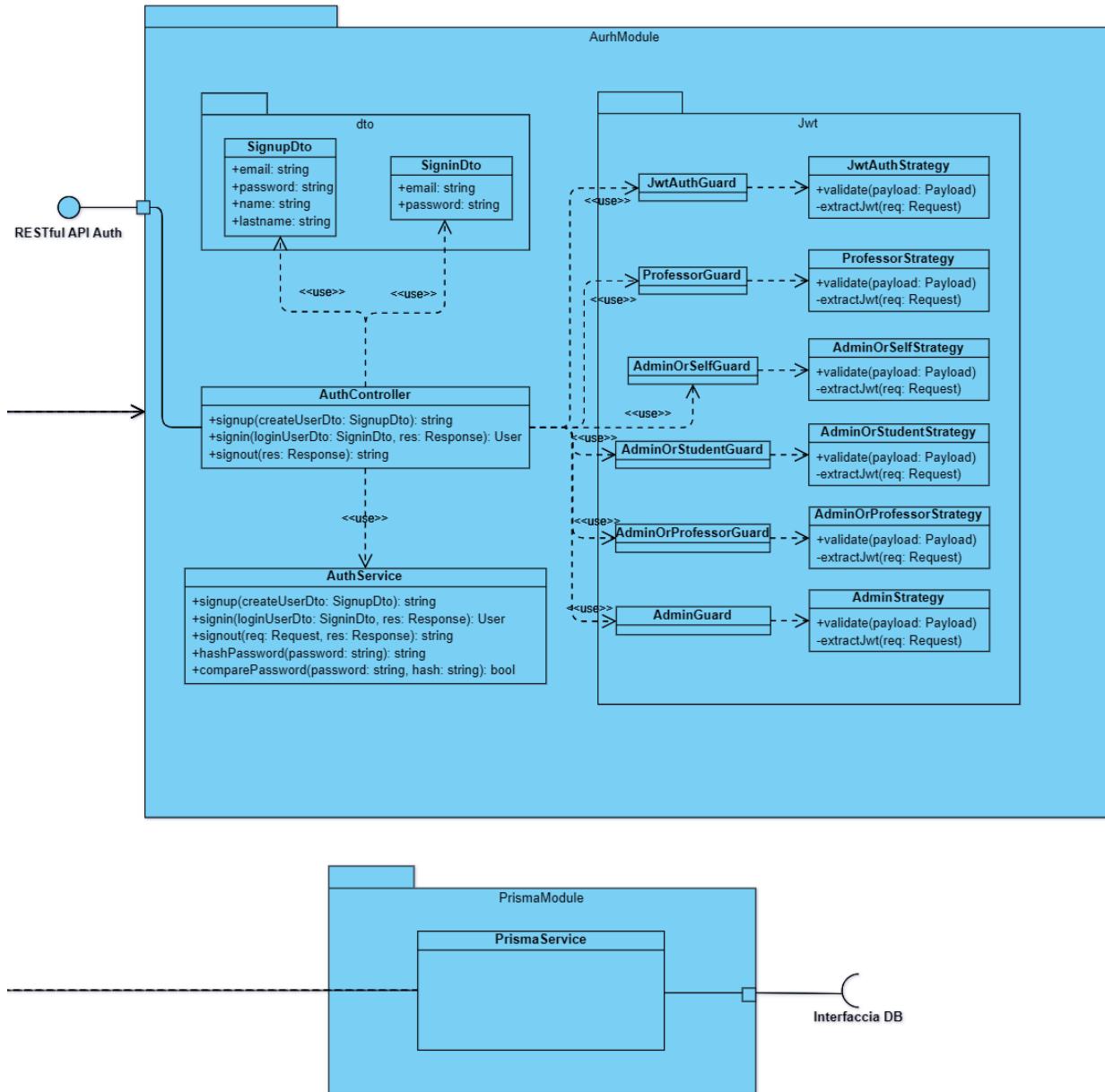


Figura 6-3: Dettagli dei moduli Auth e Prisma del Composite Structure Diagram backend

Il modulo di **autenticazione e autorizzazione**, oltre ai componenti controller, servizio e DTO già illustrati nel precedente paragrafo, contiene un pacchetto contenente le diverse strategie di autorizzazione per l'accesso ai diversi API endpoint. Il funzionamento è semplice: i controller degli altri moduli utilizzano le **JWT guard** per proteggere i vari metodi che corrispondono alle

richieste agli API endpoint, ogni guard fa riferimento ad una **strategy** per autorizzare/negare l'accesso all'utente. Per esempio, il metodo *findAll()* del controller utenti, è protetto dal guard “*AdminGuard*”, questa guard utilizza la strategia “*AdminStrategy*” che, andando a valutare il payload del token JWT allegato alla richiesta, valida la stessa solamente se l'utente che la sta facendo ha il ruolo di Admin. Per via di questa dipendenza così stretta con il modulo di autorizzazione, ogni altro modulo del diagramma è collegato ad esso da una relazione di dipendenza.

```
@Controller('users')
export class UsersController {
    [...]

    @UseGuards(AdminGuard)
    @Get()
    findAll() {
        return this.usersService.findAll();
    }

    [...]
}
```

6.3 Composite structure diagram frontend

Dopo aver mostrato come funziona il nucleo dell'applicazione, verrà spiegata in questa sezione, attraverso un diagramma, come è implementato il frontend React e quali sono le relazioni tra i diversi componenti che lo compongono.

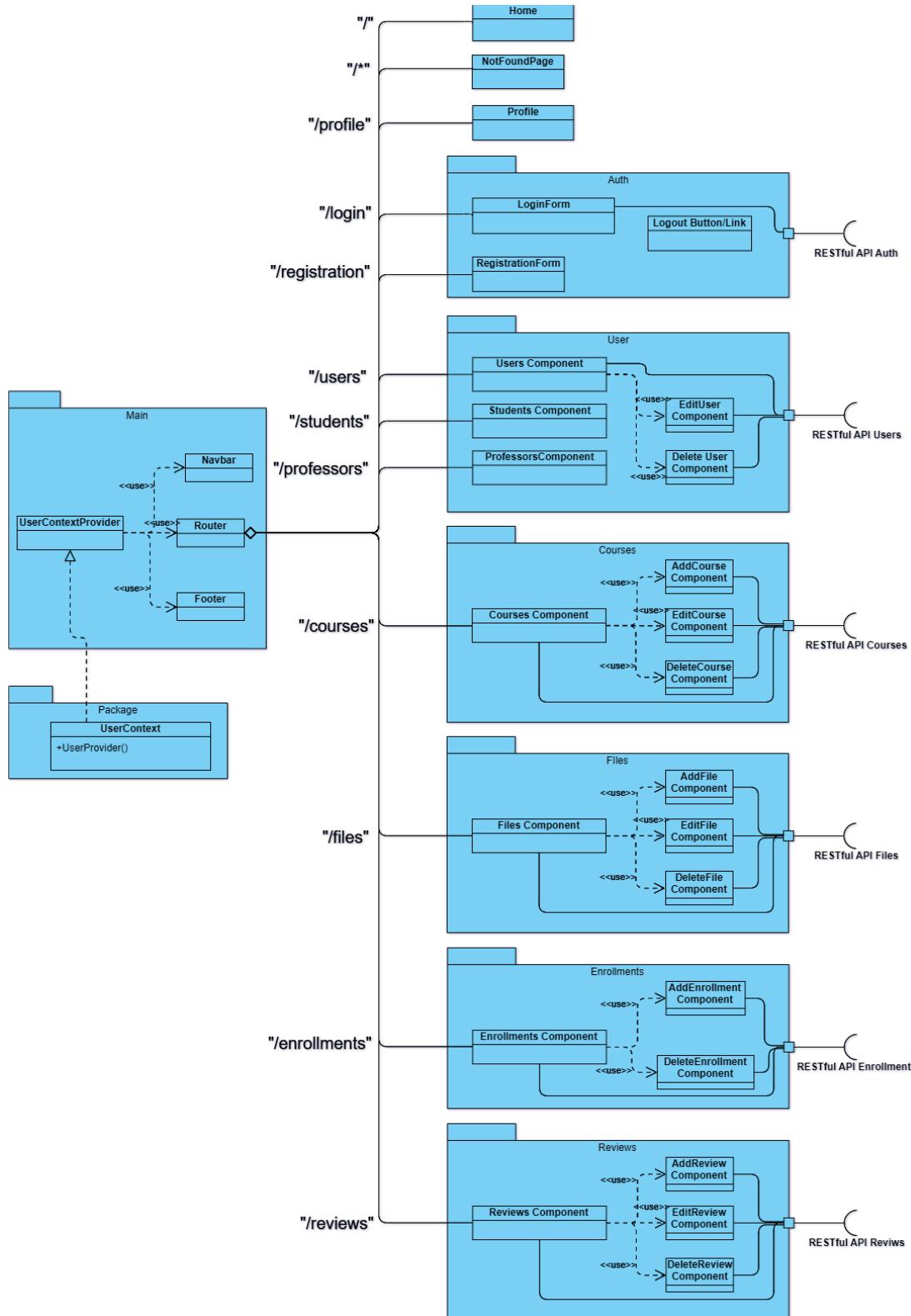


Figura 6-4: Composite Structure Diagram del frontend

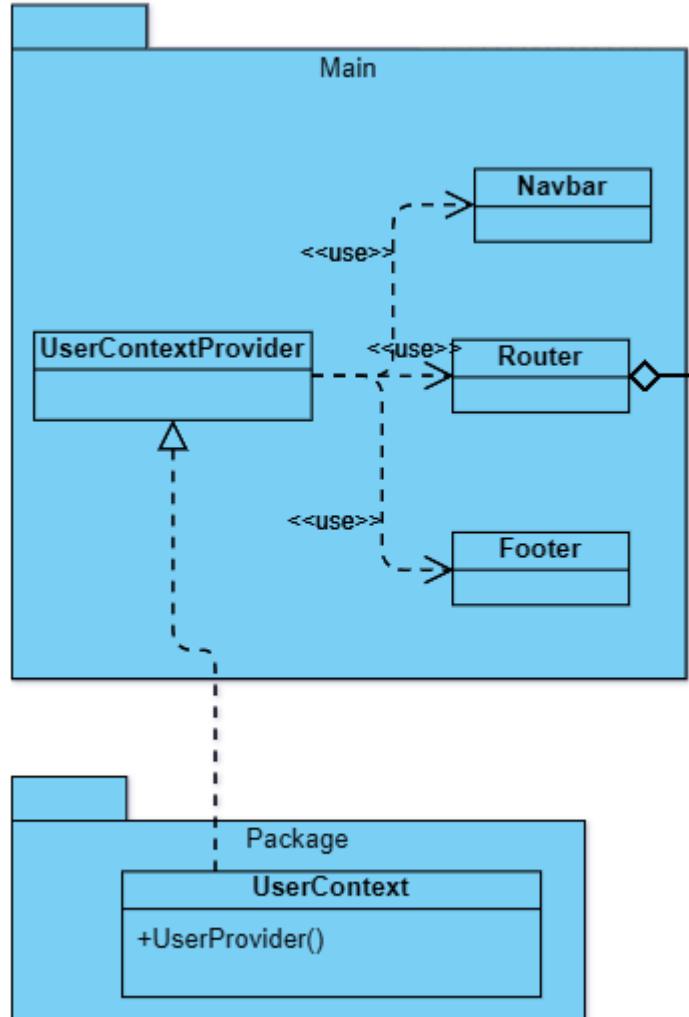


Figura 6-5: Dettagli del modulo main del Composite Structure Diagram frontend

Ogni elemento all'interno del composite structure diagram del frontend rappresenta un componente React con i quali si rappresenta l'interfaccia utente; ognuno di essi è indipendente, con il proprio scopo e, se necessario, una propria logica. La pagina principale è composta da una **navbar**, un **footer** e un router. I primi due altro non sono che la barra di navigazione superiore e inferiore, mentre il **router** ha il compito di gestire la navigazione tra le pagine dell'applicazione: quando l'utente naviga verso un certo URL, il router carica il componente associato, per esempio l'url “/login” carica il componente “LoginForm”.

Il componente **UserContextProvider** avvolge l'intera applicazione, esso utilizza la **Context API di React** per gestire lo stato globale dell'utente, come i dati di login o i ruoli (admin, studente, professore). In questo modo, i dati dell'utente sono disponibili in tutti i componenti dell'app senza dover passare manualmente informazioni attraverso le props. Il meccanismo è approfondito nella sezione 6.4.

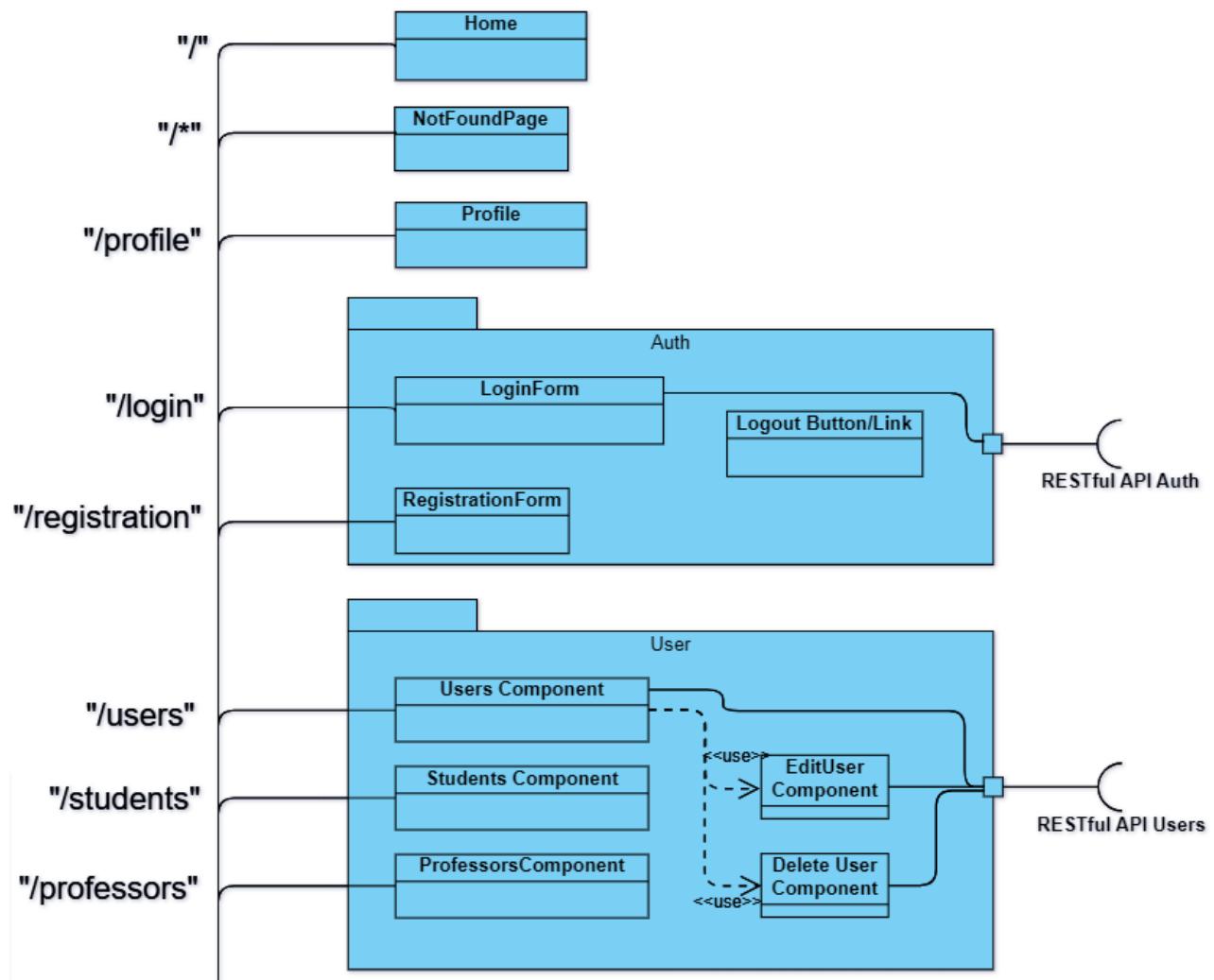


Figura 6-6: Dettagli dei moduli Auth e Users del Composite Structure Diagram frontend

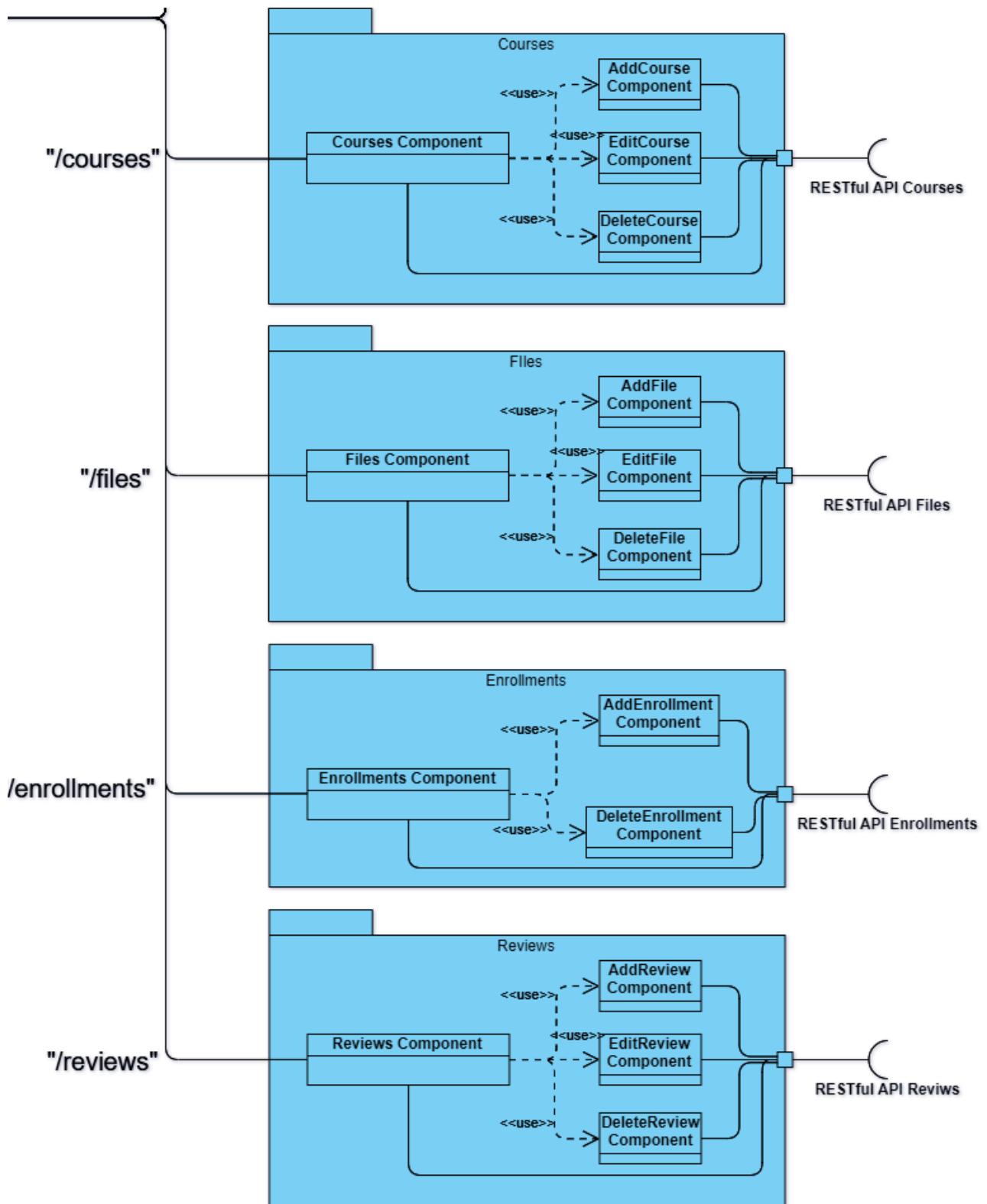


Figura 6-7: Dettagli dei moduli Courses, Files, Enrollments e Reviews del Composite Structure Diagram frontend

Come si può vedere dai diagrammi, ogni modulo è composto da diversi componenti che, interfacciandosi con le API messe a disposizione dal backend, hanno il compito di creare, visualizza, modificare ed eliminare le varie entità del sistema.

6.4 Misure di autenticazione e autorizzazione

Per garantire l'autenticazione e l'autorizzazione per i diversi endpoint del backend solamente agli utenti con il ruolo appropriato, si fa affidamento ai **JSON Web Token**, anche conosciuti come **token JWT**. Questi token sono uno standard aperto (RFC 7519) per la creazione di token che permettono di trasferire informazioni tra due parti in modo sicuro e compatto. Ogni token è composto da tre parti: (1) header, che contiene informazioni sull'algoritmo di firma digitale, nel nostro caso è stato utilizzato l'algoritmo HS256; (2) payload, che contiene le informazioni che si vogliono trasmettere con il token, nel nostro caso verranno trasferiti id dell'utente, email e ruolo; (3) signature, ovvero la firma criptografica che verifica che il token non è stato alterato, viene generato a partire dall'header e dal payload che vengono criptografati con l'algoritmo HS256, al quale viene aggiunto un segreto. Una volta firmato il token non può essere revocato, quindi è essenziale stabilire una scadenza per ognuno di essi, che in Share-Hub Unina è imposta a 30 minuti. Il vantaggio principale nell'utilizzare i token JWT è l'autenticazione **stateless**, ossia il server non deve tener traccia o memorizzare le informazioni degli utenti loggati al sistema, perché saranno loro stessi a presentare i loro permessi con il token aggiunto ad ogni richiesta.

Il funzionamento dei token è molto semplice:

- Autenticazione:** Quando un utente si autentica, il servizio di autenticazione genera un token JWT contenente le informazioni dell'utente (id, email e ruolo) e lo firma con il segreto (attualmente il segreto è “`secretjwt4565`”, non molto sicuro, da cambiare in fase di rilascio)
- Invio del token:** Il token JWT viene inviato al client e memorizzato, nel nostro caso nei cookie del browser
- Accesso alle risorse protette:** Ogni volta che il client fa una richiesta ad una risorsa protetta, il token JWT viene inviato insieme al resto della richiesta HTTP
- Verifica del token:** Il server decodifica il token e verifica la firma. Se il token è valido (cioè, la firma è corretta e il token non è scaduto), il backend esegue l'operazione richiesta.

Encoded

PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTIsImVtYWlsIjoiYWRtaW5AdW5pbmEuaXQiLCJpc1Byb2Zlc3NvciI6dHJ1ZSwicm9sZSI6IkFkbWluTiwiaWF0IjoxNzI10TYxOTM1LCJleHAiOjE3MjU5NjM3MzV9.0_z0yY5ea8at4sSkokf-GbKQsgn_M8Tz6mnJuqTgHTc
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE	
{ "alg": "HS256", "typ": "JWT" }	
PAYLOAD: DATA	
{ "id": 12, "email": "admin@unina.it", "isProfessor": true, "role": "Admin", "iat": 1725961935, "exp": 1725963735 }	
VERIFY SIGNATURE	
HMACSHA256(base64UrlEncode(header) + "." + base64UrlEncode(payload), secretjwt4565) □ secret base64 encoded	

Signature Verified

SHARE JWT

Tabella 6-1: Esempio di token JWT decodificato

Una volta chiaro come proteggere gli endpoint delle API messe a disposizione del backend, ci si pone il problema di salvare le informazioni riguardo l'utente attualmente collegato al frontend senza dover inondare di richieste il livello applicativo; a questo scopo è stato utilizzato il contesto **UserContext** implementato all'interno del componente **UserProvider**, questo permette a diverse parti dell'applicazione di accedere e modificare le informazioni dell'utente in maniera centralizzata. Il provider “avvolge” tutti gli altri componenti dell'applicazione e mette a disposizione globalmente il contesto, questo a sua volta mette a disposizione le due operazioni di prelievo e salvataggio dei dati dell'utente all'interno della memoria della sessione attuale (session storage). I dati, quindi, vengono memorizzati **solo per la durata della sessione di navigazione** e persistono finché la scheda del browser o la finestra in cui sono stati salvati rimane aperta.

Per capire meglio la gestione dei dettagli dell'utente loggato, viene proposto un activity diagram in cui:

1. L'utente visita la pagina di login dell'applicazione (in questo momento nel session storage non è salvato il contesto di nessun utente e nei cookie di sessione non c'è nessun token JWT)
2. Il frontend mostra il form di login
3. L'utente compila il form di login
4. Il frontend invia la richiesta di login al backend
5. Il backend verifica le credenziali e, se corrette, invia una risposta 200: OK al frontend e salva il token JWT all'interno dei cookie di sessione
6. Il componente che sta gestendo la richiesta del frontend, attraverso lo user provider, salva nel session storage lo user context con le informazioni dell'utente appena loggato.
7. Il frontend reindirizza l'utente alla pagina del profilo
8. In base al ruolo dell'utente loggato (si va a controllare lo user context) viene visualizzata la pagina adatta
9. L'utente è un professore, quindi nella pagina del profilo saranno presenti i corsi tenuti da lui; quindi, il frontend manda una richiesta al backend per i corsi del professore, alla richiesta aggiunge il cookie contenente il token JWT
10. Il backend recupera i corsi dell'utente e li invia al frontend
11. Il frontend visualizza i corsi del professore
12. Il professore clicca su logout
13. Il frontend resetta lo user context, che adesso riporterà i dati dell'utente di default e manda una richiesta al backend per fare il logout
14. Il backend effettua il logout ed elimina il token JWT dai cookie di sessione

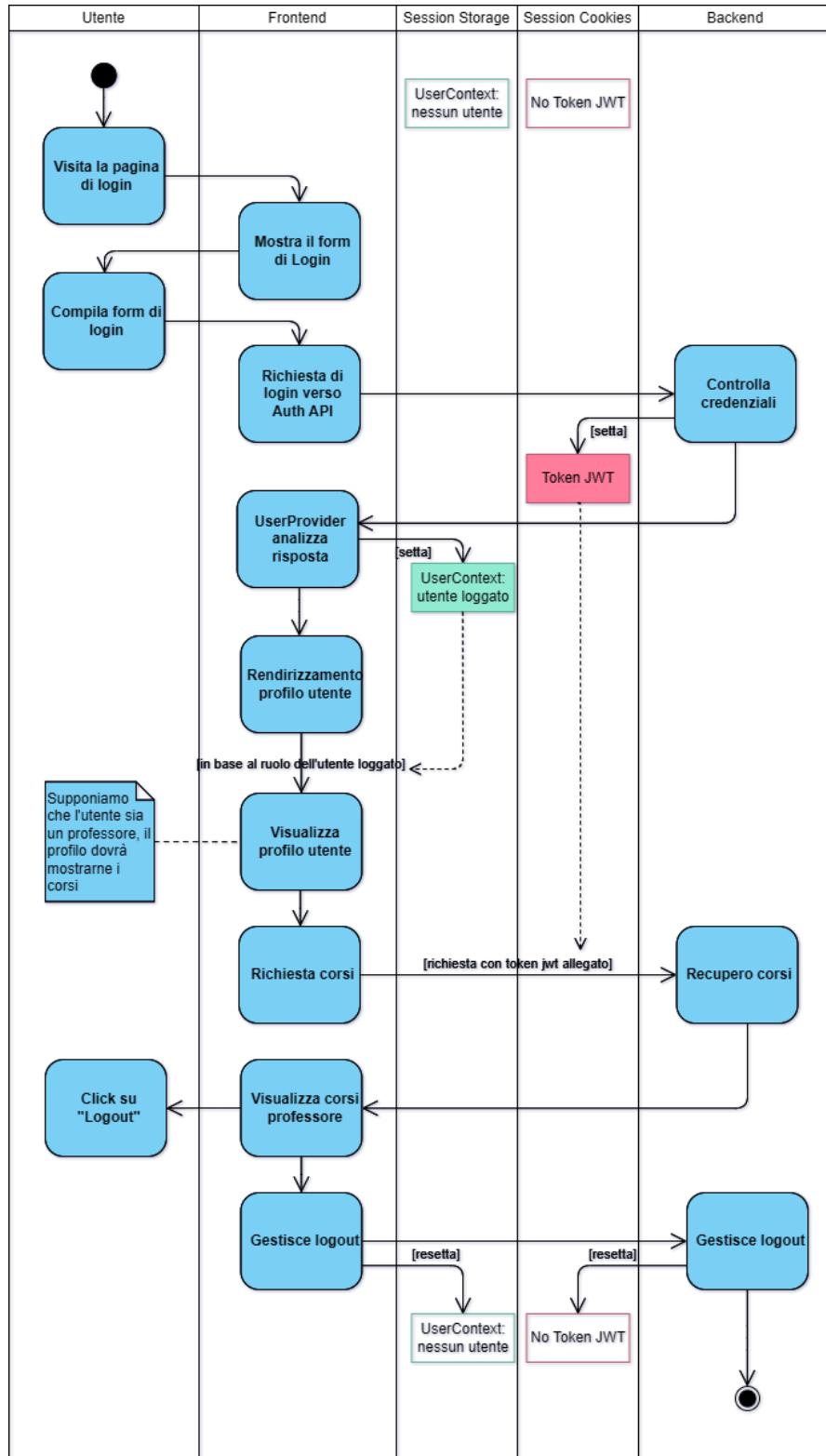


Figura 6-8: Activity diagram gestione dell'autenticazione/autorizzazione

Potrebbe sembrare ridondante questa doppia gestione delle autorizzazioni, però è indispensabile. Non sarebbe corretto, infatti, se un utente studente o professore potesse vedere il pulsante “Elimina utente” che, una volta cliccato, restituirebbe errore perché non dispone delle autorizzazioni per utilizzare quella funzione. Oppure non sarebbe corretto se un professore vedesse il pulsante “Aggiungi file” sulla pagina di un corso che non gli appartiene, pulsante non utilizzabile in quanto la richiesta verrebbe bloccata dal backend in quanto il token JWT inviato con

essa permette di identificare il professore correttamente come non proprietario del corso al quale si sta tentando di aggiungere un file.

Il componente **UserProvider**, quindi, è fondamentale per far coincidere le autorizzazioni sulle operazioni del backend con quello che visualizza ogni utente sul frontend. Facendo corrispondere le autorizzazioni del livello di presentazione con quelle del livello di applicazione, si ottiene un funzionamento più esatto dell'applicazione nonché una user experience più gradevole.

6.5 Struttura del codice

La seguente mappa, realizzata con MarkMap, vuole illustrare a grandi linee la struttura delle cartelle del codice sorgente, per aiutare l'eventuale navigazione dello stesso. Una versione interattiva e completa è presente all'interno dell'applicazione, visitando l'url “/map”.

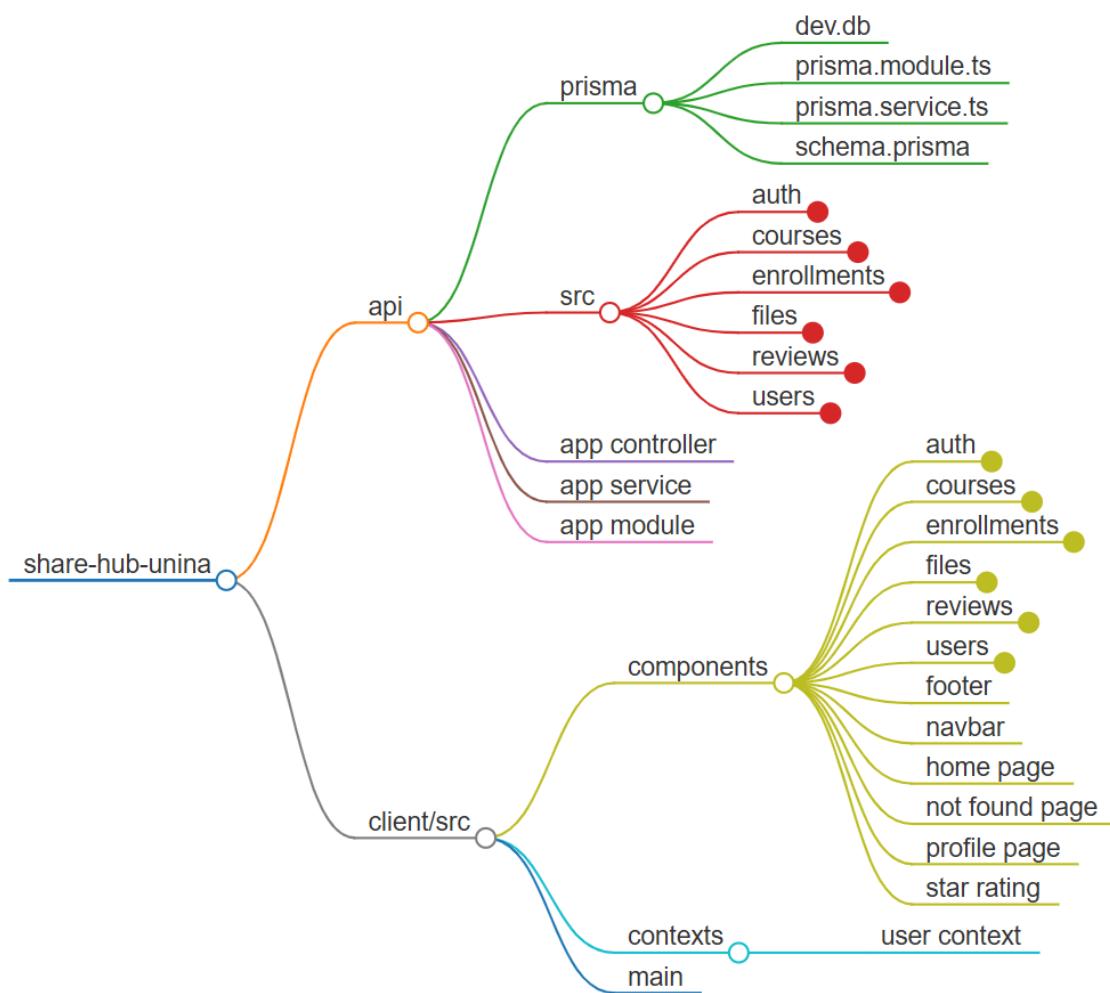


Figura 6-9: Struttura delle cartelle del codice sorgente

Capitolo 7: Testing

In questo capitolo verrà descritto il processo di testing utilizzato per garantire il corretto funzionamento di Share-Hub Unina; in particolare il testing dei vari API endpoints offerti dal backend dell'applicazione. Vista l'elevato numero di endpoints e la possibilità di essere visitati da utenti con tre ruoli diversi (Student, Professor, Admin), si è ricorso al testing automatico con Postman per velocizzare il processo.

7.1 Functional testing con Postman

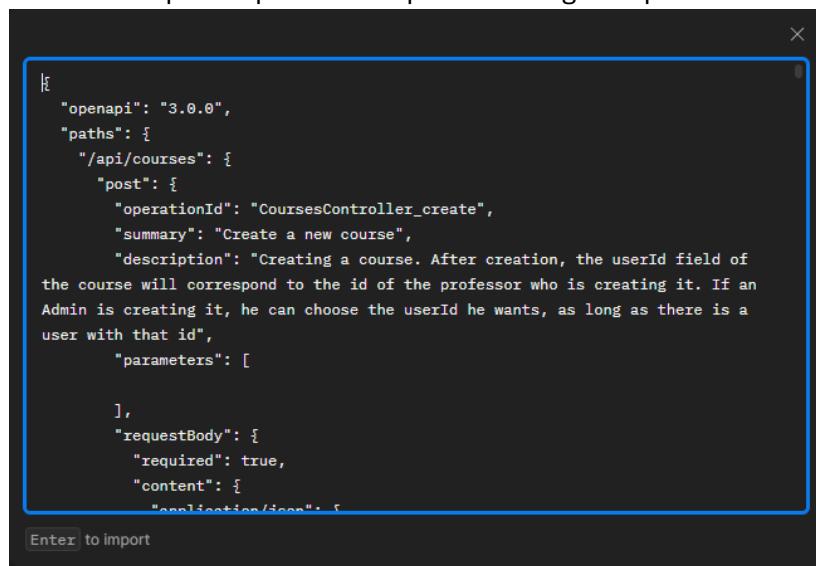
Per testare gli endpoint con Postman è stato fondamentale partire dalla documentazione prodotta tramite Swagger; infatti, da Swagger è possibile esportare una lista degli endpoint che può essere importata in Postman, a questa lista si possono poi aggiungere i test che verranno effettuati in automatico.

Per completezza si descrive il processo per passare dalla documentazione di Swagger al testing su postman:

1. All' url “/apidocs-json” è possibile avere le stesse informazioni riportate nella documentazione Swagger in formato JSON:

```
1 // 20240916111446
2 // http://localhost:3000/apiDocs-json
3
4 {
5   "openapi": "3.0.0",
6   "paths": {
7     "/api/courses": {
8       "post": {
9         "operationId": "CoursesController_create",
10        "summary": "Create a new course",
11        "description": "Creating a course. After creation, the userId field of the course will correspond to the id of the professor who is creating it. If an Admin is creating it, he can choose the userId he wants, as long as there is a user with that id",
12        "parameters": [
13          ],
14        },
15        "requestBody": {
16          "required": true,
17          "content": {
```

2. Sfruttando la funzione “Import” è possibile importare tutti gli endpoint all'interno di Postman:



3. Verrà creata una collezione dove sarà possibile visualizzare gli endpoint opportunamente divisi in varie categorie.

The screenshot shows the Postman interface with the title bar "Home Workspaces API Network". Below it is the "My Workspace" section with tabs for "New" and "Import". On the left, there's a sidebar with icons for "Collections" (selected), "Environments", "APIs", and "History". The main area displays a collection named "Share-Hub Unina API" which is expanded to show its contents. The collection includes categories for "courses", "files", "reviews", "enrollments", "users", and "auth", each containing various HTTP methods like POST, GET, PATCH, and DEL.

Una volta ottenuta la collezione su Postman, è sufficiente aggiungere per ogni richiesta dei test da effettuare sulla risposta, tramite i Post-response script, per verificare il corretto funzionamento dell'endpoint in base al ruolo dell'utente che sta effettuando la richiesta.

Per le richieste di login è stata verificata la presenza del token JWT nei cookie della risposta:

The screenshot shows a Postman request configuration for a "POST" method to the "Login" endpoint. The URL is set to "{{baseUrl}}/api/auth/signin". The "Scripts" tab is selected under the "Post-response" section. The script contains the following code:

```

1 pm.test("Successful POST request", function () {
2   | pm.expect(pm.response.code).to.be.oneOf([201, 202]);
3 });
4
5 pm.test("Verify Cookie 'token' is present in response", function(){
6   | pm.expect(pm.cookies.has('token')).to.be.true;
7 });

```

Per le richieste di logout è stato verificato il reset del token JWT:

```

HTTP Share-Hub Unina API / files / Logout

GET {{baseUrl}}/api/auth/signout

Params Authorization Headers (7) Body Scripts • Settings

Pre-request
1 pm.test("Successful GET request", function () {
2   | pm.response.to.have.status(200);
3 });
4
5 pm.test("Verify Cookie 'token' is not present in response", function(){
6   | pm.expect(pm.cookies.get('token')).to.be.undefined;
7 });

Post-response

```

Per le richieste di POST, oltre al codice di risposta atteso, è stato prelevato il valore dell'id dell'oggetto creato da poter poi essere utilizzato, tramite le variabili di collezione, nelle successiva richiesta di PATCH e DELETE

```

HTTP Admin / files / Create a new file

POST {{baseUrl}}/api/files

Params Authorization Headers (10) Body • Scripts • Settings

Pre-request
1 pm.test("Successful POST request", function () {
2   | pm.expect(pm.response.code).to.be.oneOf([201, 202]);
3 });
4
5 const response = pm.response.json();
6 pm.collectionVariables.set("fileId", response.id);
7
8 console.log("File ID salvato nella variabile: " + response.id);

Post-response

```

Per il resto delle richieste è stato controllato il codice di risposta atteso in base ai permessi dell'utente che stava svolgendo l'operazione

```

HTTP Admin / files / Get all files

GET {{baseUrl}}/api/files?courseId=<string>

Params • Authorization Headers (7) Body Scripts • Settings

Pre-request
1 pm.test("Status code is 200", function () {
2   | pm.response.to.have.status(200);
3 });

Post-response

```

In totale, per ogni run, sono stati effettuati **163 test** tra le varie combinazioni di richieste e ruoli. I file JSON contenenti le richieste e i risultati dei test effettuati sono disponibili nella repo Github del progetto

7.2 Testing endpoints con ruolo Studente

Testing degli endpoint servizio Courses:

▼ POST	Login student	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new course	1 0
	Pass Status code is 401	
▼ GET	Get all courses	1 0
	Pass Status code is 200	
▼ GET	Get all courses by userid	1 0
	Pass Status code is 200	
▼ GET	Get a course by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update a course by ID	1 0
	Pass Status code is 401	
▼ DELETE	Delete a course by ID	1 0
	Pass Status code is 401	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Files:

▼ POST	Login student	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new file	1 0
	Pass Status code is 401	
▼ GET	Get all files	1 0
	Pass Status code is 401	
▼ GET	Get all files of a course	1 0
	Pass Status code is 200	
▼ GET	Get all files of a course not enrolled	1 0
	Pass Status code is 403	
▼ GET	Get a file by id.	1 0
	Pass Status code is 200	
▼ GET	Get a file by id but not enrolled in the cour...	1 0
	Pass Status code is 403	
▼ PATCH	Update a file by ID	1 0
	Pass Status code is 401	
▼ DELETE	Delete a file by ID	1 0
	Pass Status code is 401	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Reviews:

▼ POST	Login student	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new review	1 0
	Pass Successful POST request	
▼ POST	Create the same review	1 0
	Pass Status code is 403	
▼ GET	Get all reviews	1 0
	Pass Status code is 401	
▼ GET	Get all reviews by fileId	1 0
	Pass Status code is 200	
▼ GET	Get all reviews by fileId but not enrolled in...	1 0
	Pass Status code is 403	
▼ GET	Get all reviews by own userId	1 0
	Pass Status code is 200	
▼ GET	Get all reviews by not own userId	1 0
	Pass Status code is 403	
▼ GET	Get a review by id.	1 0
	Pass Status code is 200	
▼ PATCH	Update a review by id.	1 0
	Pass Status code is 200	
▼ PATCH	Update a review by id.	1 0
	Pass Status code is 200	
▼ PATCH	Update a review by id with wrong userId o...	1 0
	Pass Status code is 403	
▼ PATCH	Update a review not owned by id	1 0
	Pass Status code is 403	
▼ DELETE	Delete a review by id.	1 0
	Pass Status code is 200	
▼ DELETE	Delete a review not owned by id	1 0
	Pass Status code is 403	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Enrollments:

▼ POST	Login student	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new enrollment	1 0
	Pass Successful POST request	
▼ GET	Get all the enrollments	1 0
	Pass Status code is 403	
▼ GET	Get all the enrollments by own userId	1 0
	Pass Status code is 200	
▼ GET	Get all the enrollments by not own userId	1 0
	Pass Status code is 403	
▼ GET	Get an enrollment by id.	1 0
	Pass Status code is 200	
▼ GET	Get an enrollment by id not owned	1 0
	Pass Status code is 403	
▼ DELETE	Delete an owned enrollment by id.	1 0
	Pass Status code is 200	
▼ DELETE	Delete a not owned enrollment by id	1 0
	Pass Status code is 403	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Users:

▼ POST	Login student	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ GET	Get all the users	1 0
	Pass Status code is 401	
▼ GET	Get all the professors	1 0
	Pass Status code is 200	
▼ GET	Get user by own ID	1 0
	Pass Status code is 200	
▼ PATCH	Update own user by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update not own user by ID	1 0
	Pass Status code is 401	
▼ DELETE	Delete user by ID	1 0
	Pass Status code is 401	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

7.2.1 Testing endpoints con ruolo Professore

Testing degli endpoint servizio Courses:

▼ POST	Login professor	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new course	1 0
	Pass Successful POST request	
▼ GET	Get all courses	1 0
	Pass Status code is 200	
▼ GET	Get a course by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update a course by ID	1 0
	Pass Status code is 200	
▼ DELETE	Delete a course by ID	1 0
	Pass Status code is 200	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Files:

▼ POST	Login professor	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new file	1 0
	Pass Successful POST request	
▼ POST	Create a new file with courseld not owned	1 0
	Pass Status code is 403	
▼ GET	Get all files	1 0
	Pass Status code is 401	
▼ GET	Get all files by courseld	1 0
	Pass Status code is 200	
▼ GET	Get all files by not owned courseld	1 0
	Pass Status code is 403	
▼ GET	Get a file by id.	1 0
	Pass Status code is 200	
▼ GET	Get a file not owned by id	1 0
	Pass Status code is 403	
▼ PATCH	Update a file by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update a file by ID with wrong courseld	1 0
	Pass Status code is 403	
▼ DELETE	Delete a file by ID	1 0
	Pass Status code is 200	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Reviews:

▼ POST	Login professor	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new review	1 0
	Pass Status code is 401	
▼ GET	Get all reviews	1 0
	Pass Status code is 401	
▼ GET	Get all reviews by userId	1 0
	Pass Status code is 403	
▼ GET	Get all reviews by fileId	1 0
	Pass Status code is 200	
▼ GET	Get all reviews of a file not owned	1 0
	Pass Status code is 403	
▼ GET	Get a review by id.	1 0
	Pass Status code is 401	
▼ PATCH	Update a review by id.	1 0
	Pass Status code is 401	
▼ DELETE	Delete a review by id.	1 0
	Pass Status code is 401	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Enrollments:

▼ POST	Login professor	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new enrollment	1 0
	Pass Status code is 401	
▼ GET	Get all the enrollments	1 0
	Pass Status code is 403	
▼ GET	Get all the enrollments by userId	1 0
	Pass Status code is 403	
▼ GET	Get all the enrollments by courseId	1 0
	Pass Status code is 200	
▼ GET	Get all the enrollments by wrong courseId	1 0
	Pass Status code is 403	
▼ GET	Get an enrollment by id.	1 0
	Pass Status code is 401	
▼ DELETE	Delete an enrollment by id.	1 0
	Pass Status code is 401	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Users:

▼ POST	Login professor	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ GET	Get all the users	1 0
	Pass Status code is 401	
▼ GET	Get all the professors	1 0
	Pass Status code is 200	
▼ GET	Get user by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update user by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update user by wrong ID	1 0
	Pass Status code is 401	
▼ DELETE	Delete user by ID	1 0
	Pass Status code is 401	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

7.2.2 Testing endpoints con ruolo Admin

Testing degli endpoint servizio Courses:

▼ POST	Login admin	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new course	1 0
	Pass Successful POST request	
▼ GET	Get all courses	1 0
	Pass Status code is 200	
▼ GET	Get a course by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update a course by ID	1 0
	Pass Status code is 200	
▼ DELETE	Delete a course by ID	1 0
	Pass Status code is 200	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Files:

▼ POST	Login admin	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new file	1 0
	Pass Successful POST request	
▼ GET	Get all files	1 0
	Pass Status code is 200	
▼ GET	Get a file by id.	1 0
	Pass Status code is 200	
▼ PATCH	Update a file by ID	1 0
	Pass Status code is 200	
▼ PATCH	Update a file by ID wrong course	1 0
	Pass Status code is 403	
▼ DELETE	Delete a file by ID	1 0
	Pass Status code is 200	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Reviews:

▼ POST	Login admin	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
▼ POST	Create a new review	1 0
	Pass Successful POST request	
▼ POST	Create the same review	1 0
	Pass Status code is 403	
▼ GET	Get all reviews	1 0
	Pass Status code is 200	
▼ GET	Get a review by id.	1 0
	Pass Status code is 200	
▼ PATCH	Update a review by id.	1 0
	Pass Status code is 200	
▼ PATCH	Update a review by id wrong userId or fileId	1 0
	Pass Status code is 403	
▼ DELETE	Delete a review by id.	1 0
	Pass Status code is 200	
▼ GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Enrollments:

POST	Login admin	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
POST	Create a new enrollment	1 0
	Pass Successful POST request	
GET	Get all the enrollments	1 0
	Pass Status code is 200	
GET	Get an enrollment by id.	1 0
	Pass Status code is 200	
DELETE	Delete an enrollment by id.	1 0
	Pass Status code is 200	
GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Testing degli endpoint servizio Users:

POST	Login admin	2 0
	Pass Successful POST request	
	Pass Verify Cookie 'token' is present in response	
GET	Get all the users	1 0
	Pass Status code is 200	
GET	Get all the professors	1 0
	Pass Status code is 200	
GET	Get user by ID	1 0
	Pass Status code is 200	
PATCH	Update user by ID	1 0
	Pass Status code is 200	
DELETE	Delete user by ID	0 0
GET	Logout	2 0
	Pass Successful GET request	
	Pass Verify Cookie 'token' is not present in resp...	

Capitolo 8: Rilascio

8.1 Deployment diagram fase di sviluppo

Durante la fase di sviluppo, l'applicazione è gestita da due server separati:

- **Frontend:** React espone il frontend su <http://localhost:5173>, servito da Vite
- **Backend:** NestJS che espone gli API endpoints su <http://localhost:3000>, servito da Webpack server. All'interno del backend è contenuto anche il database SQLite.

Sia Vite che Webpack monitorano il codice per dei cambiamenti e aggiornano automaticamente frontend/backend durante lo sviluppo

La comunicazione tra il frontend e il backend avviene tramite un **API proxy** configurato in Vite, che semplifica le richieste evitando problemi di CORS. Questo setup consente uno sviluppo separato e parallelo dei due ambienti, facilitando l'iterazione rapida. L'uso di **Turbo** permette di coordinare il progetto monorepo, gestendo efficacemente entrambe le applicazioni in modo sincrono, con build parallele e migliorando i tempi di sviluppo.

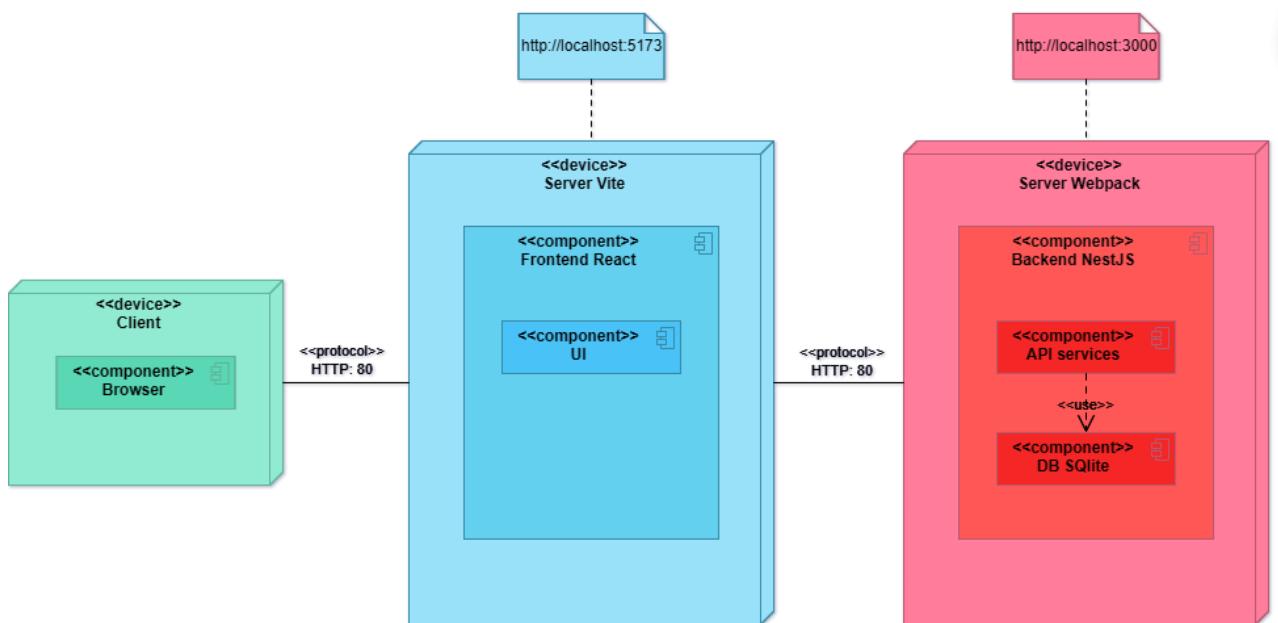


Figura 8-1: Deployment diagram fase di sviluppo

8.2 Deployment diagram fase di produzione

In fase di produzione, l'intero sistema viene consolidato su un singolo server NodeJS su <http://localhost:3000> senza Webpack. Il backend gestisce le API, mentre il frontend, compilato come risorse statiche da Vite, viene servito dallo stesso server NestJS. In questo modo, NestJS si occupa sia della logica server-side che della distribuzione del frontend, semplificando l'infrastruttura e migliorando l'efficienza.

Turbo svolge un ruolo cruciale anche in questa fase, gestendo il processo di build e deployment in modo centralizzato. Questo strumento permette di automatizzare la pipeline di rilascio, garantendo che il frontend e il backend siano sempre sincronizzati e pronti per la produzione.

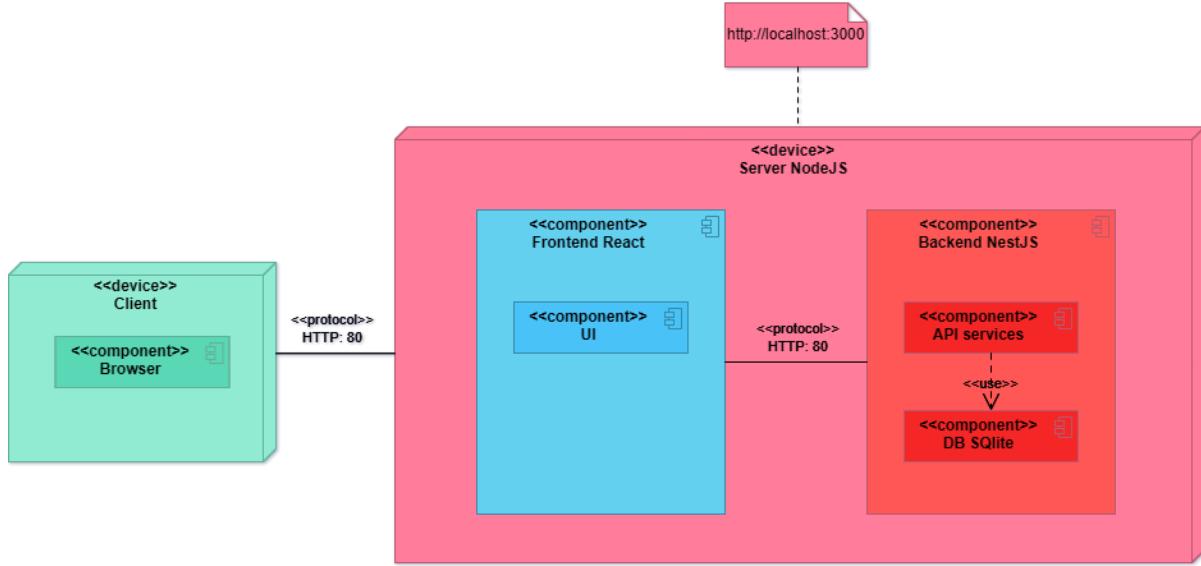


Figura 8-2: Deployment diagram fase di produzione

L'unico step rimanente sarebbe quello di implementare un database diverso in un componente separato dal backend, ma questo è rimandato a futuri sviluppi dell'applicazione.

8.3 Installazione

Per poter installare l'applicazione, testarla, o aggiungervi funzionalità è sufficiente seguire i seguenti passaggi.

Come prima cosa bisogna clonare la repo dell'applicazione, installarla e inizializzare Prisma:

```
$ git clone https://github.com/AntonioSposito/share-hub-unina.git
$ cd share-hub-unina
$ npm install
$ cd .\apps\api\
$ npx prisma generate
```

Per lanciare l'app in development mode (ogni cambiamento del codice si riflette immediatamente sul sistema):

```
$ npm run dev
```

Per lanciare l'app in production mode (ogni volta che si modifica l'app bisogna buildare e rilanciare il sistema):

```
$ npm run build
$ npm run start
```