Tor Vergata Safe Study Service

1 Introduzione

In questo anno così particolare, per colpa della pandemia globale, sono cambiate molte cose nel quotidiano modo di vivere. Nel contesto universitario, studenti e docenti hanno dovuto compiere sacrifici ed enormi adattamenti pur di continuare al meglio l'attività didattica.
Da studenti possiamo dire che uno dei sacrifici più grandi è stato quello di rinunciare a un confronto diretto, faccia a faccia, con compagni di studio e docenti. Questo sacrificio non è roba da poco, in quanto il confronto tra le menti stimola molto la curiosità, la voglia di imparare nuove cose ed *infiamma* la passione per le materie.
Nonostante informatici, più a nostro agio nell'interfacciarci con macchine piuttosto che con persone, siamo pur sempre uomini, e quindi, animali sociali 😋.

Adesso che la situazione pandemica è in apparente miglioramento, si iniziano lentamente a riprendere le varie attività. Alcuni corsi si svolgono in modalità mista, si stanno programmando esami da svolgere in presenza ed è di nuovo consentito l'accesso alle strutture universitarie, con le dovute precauzioni.
Proprio su quest'ultimo punto ci siamo voluti soffermare. Nonostante alcune aule siano state adibite allo studio, abbiamo notato che:

1. Non c'è un sistema che cerchi di monitorare la densità di persone presenti.
2. Con una corretta organizzazione si potrebbero aprire più aule, garantendo l'accesso a più studenti, il tutto rispettando i vincoli di sicurezza vigenti.

1.1 Esigenza

Dopo il periodo di quarantena, ora che stanno riaprendo le strutture, in quanto studenti sentiamo la necessità di un sistema che ci permetta di ritornare in tranquillità a studiare in università. Abbiamo notato che sono disponibili poche aule studio aperte, e che ci vuole del tempo per trovarne una non troppo affollata. Abbiamo anche notato che ci sono parecchie aule libere che non vengono usate, e che si potrebbero adibire per gli studenti.
Chiaramente sorge il problema che più aule sono aperte, più è difficile controllare la densità di gente presente, per evitare conseguenti assembramenti.
Perciò sarebbe utile un sistema che semplifichi e automatizzi questa ricerca di aule, e che nello stesso momento possa aiutare l'università a gestire la densità di studenti, cercando di garantire al meglio l'incolumità di tutti.

1.2 Obiettivi

Si vuole realizzare un servizio universitario che consenta agli studenti di poter prenotare l'accesso alle aule studio predisposte dalla **Macroarea di Scienze MM.FF.NN**., cercando inoltre di semplificare all'università il problema dell'organizzazione delle entrate degli studenti a fronte della situazione pandemica attuale.

È possibile guardare a questo servizio come a uno strumento utile per tenere sotto controllo al meglio la densità di persone presenti negli edifici universitari, minimizzando la possibilità che si creino affollamenti nelle aule.

La prenotazione delle aule **NON** è un requisito necessario per accedervi, in quanto si necessiterebbe di un terzo attore (umano o dispositivo elettronico) che controlli, studente per studente, chi è effettivamente in possesso o meno di una prenotazione valida; questo per ogni aula ad ogni ora. Perciò ci si affida alla *collaborazione* e al *buon senso* dello studente che, per correttezza verso il prossimo, prenoti un'aula prima di farne uso.

Il servizio dovrà offrire:

1. Un'interfaccia web **SEMPLICE** e **INTUITIVA** agli studenti interessati ad usufruire delle aule al di fuori dell'orario di lezione.
2. Un sistema di prenotazione semplice e veloce, che non comporti fasi troppo articolate (registrazioni, login, ecc...).
3. Un sistema di gestione delle disponibilità delle aule.

2 Fase di analisi

2.1 Analisi e definizione dei requisiti

La fase di analisi del software è il processo di comprensione e definizione dell'ambiente di esecuzione del SW, dei servizi richiesti dal sistema, dell'identificazione dei vincoli sul funzionamento e sullo sviluppo dello stesso.
L'analisi e la definizione dei requisiti verranno discussi nei seguenti paragrafi.

2.2 Analisi di dominio

Bisogna innanzitutto raccogliere più informazioni possibili sul dominio in cui sarà eseguita l'applicazione. Una buona analisi di dominio porta molteplici vantaggi:

- Scelte più oculate nelle fasi successive dello sviluppo del SW.
- Uno sviluppo più veloce. Infatti, conoscendo meglio il dominio, la comunicazione con gli stakeholder e la comprensione delle loro esigenze sarà facilitata.
- Una migliore comprensione dell'ambiente porta ad un'astrazione migliore. Ne consegue una progettazione di sistemi più efficaci ed efficienti.
- Anticipazione delle estensioni del software. Una migliore conoscenza di dominio ci permette di prevedere possibili sviluppi del SW e quindi già dalle prime fasi, si sviluppa un sistema adattabile ed estendibile.

Detto questo, l'ambiente della nostra applicazione non è particolarmente complesso, come potrebbe esserlo quello di un'applicazione tecnica. Per esempio, un'applicazione gestionale di esami medici, necessiterebbe di una conoscenza delle pratiche e dell'organizzazione sanitarie, cosa che non accade nel nostro caso.

Task e procedure correntemente eseguite: Non essendoci un sistema preposto alla prenotazione delle aule studio, per raggiungere il medesimo obiettivo del nostro software, esiste il tradizionale metodo del:

1. Apri la porta dell'aula
2. "Scusate, c'è lezione?" - "No"

3. Scegliere il posto a sedere

Nonostante la robustezza di questo metodo, scambiare informazioni a parole è antiquato per la frenesia e apatia della modernità, dove l'unico modo per scambiare informazioni **deve** essere digitale/virtuale.

Software Concorrente/Avversario: il nostro non è un software commerciale, quindi non si può parlare di concorrenza vera e propria. A nostra conoscenza, non ci sono applicazioni che abbiano lo stesso obiettivo del nostro SW. Sul portale Delphi è possibile prenotare aule, non per lo studio, ma per seguire le lezioni.

2.2.1 Glossario entità

Il glossario dei termini che utilizzeremo (prenotazione, conferma, etc..) è abbastanza diretto e semplice.

Il nostro dominio di interesse riguarda due entità:

Entità	Descrizione
Aule	Identificate dal loro nome (e.g. "3A"), rappresentano le aule universitarie (nel nostro caso **Sogene**) che gli studenti potranno prenotare.
Prenotazioni	Identificate da un id (stringa lunga un numero fissato di caratteri alfanumerici), rappresentano le prenotazioni che gli studenti faranno. Le informazioni contenute saranno la mail universitaria del prenotante, l'aula prenotata e il giorno/ora in cui farà accesso.
Utenti	Uno **studente** di Tor Vergata che desidera prenotare un'aula.

2.3 Studio di fattibilità

In questa fase, viene stimato se le esigenze dell'utente possono essere soddisfatte utilizzando le attuali tecnologie software e hardware.
Le tecnologie richieste per questo sistema sono:

- Un browser per l'accesso al servizio di prenotazione.
- Un web server che restituisca le pagine web utili al sistema nelle varie fasi.
- Un applicativo per la gestione delle richieste di prenotazione.
- Un database per la memorizzazione delle prenotazioni e per la gestione della loro disponibilità.
- Un mail server per la notifica dell'avvenuta prenotazione e delle informazioni a lei associate.

2.4 Requisiti funzionali

I requisiti funzionali definiscono a livello *astratto* le funzioni di base che il sistema deve fornire.
I requisiti funzionali richiesti all'applicazione sono i seguenti:

- Quando richiesto, l'applicazione deve mostrare in tempo reale il calendario in cui sono contenute le disponibilità orarie delle aule studio.
- Vanno mantenute tutte le informazioni sulle aule, la loro disponibilità e le prenotazioni che le riguardano.
- L'applicazione deve permettere, allo studente che la richiede, la prenotazione di un'aula studio nella fascia oraria indicata.
- Il sistema deve permettere l'annullamento di una prenotazione precedentemente effettuata.

2.5 Requisiti non funzionali (di sistema)

I requisiti di sistema, anche noti come proprietà non funzionali, riguardano il comportamento del sistema nel suo ambiente operativo. Possiamo alternativamente definirle come delle proprietà **qualitative** dei servizi offerti dal sistema. Al fine di raggiungere gli obiettivi precedentemente discussi, il sistema dovrà avere le seguenti caratteristiche:

- **Usability**: Il servizio deve essere il più user-friendly possibile. L'interfaccia deve riuscire a trasmettere, senza spiegazioni, i passi da seguire per potersi prenotare.
  L'intero processo è scomponibile in 5 step elementari:
  1. Scelta della data con un menu drop-down (popup nei dispositivi mobile).
  2. Scelta dell'orario da una tabella delle aule, attraverso un semplice click (o tocco).
  3. Inserimento dell'indirizzo e-mail.
  4. Inserimento del codice conferma ricevuto via mail.
  5. Notifica via mail dell'avvenuta prenotazione.
  6. *Facoltativo:* Eventuale click sul link di annullamento della prenotazione, ricevuto nel punto (5).

- **Reliability**: il sistema dovrà garantire la consistenza nelle prenotazioni, ovvero:
  - Soltanto gli studenti dell'università di Tor Vergata possono prenotare le aule studio. Ciò è facilmente implementabile permettendo la prenotazione solo con l'uso della mail istituzionale.
  - Uno studente non deve poter effettuare più prenotazioni che si sovrappongano. Non deve essere possibile prenotare aule diverse nello stesso giorno-orario né duplicare una stessa prenotazione.
  - Soltanto lo studente che ha effettuato la prenotazione può annullarla (infatti il link di annullamento è inviato esclusivamente a chi fa la prenotazione). Solo in casi straordinari la prenotazione sarà annullata dal sistema. Per esempio, in caso di prenotazione di un'aula da parte di un professore che deve sostenervi un esame. In tal caso verrà notificato in breve tempo a tutti gli studenti che avevano prenotato.
  - Non è possibile prenotare aule al di fuori degli orari universitari.

- Le prenotazioni devono essere **consistenti** (e.g. non posso prenotare più posti di quanti ne dispone un'aula).

- **Availability**:
  - Il servizio deve essere attivo h24, 7 giorni su 7. Lo studente può prenotare anche la domenica per un qualunque giorno futuro, fino ad un massimo di 15 giorni dalla data corrente.

- **Safety**:
  - Nelle aule non posso prenotarsi più persone di quelle consentite dalle norme vigenti contro la pandemia.

- **Security**:
  - Non deve essere possibile forgiare email contenenti link malevoli, spacciandole come inviate dal nostro sistema.

- **Privacy:**
  - L'email inserita dallo studente dovrà essere usata solamente dai server dell'applicazione e non da terze parti.
  - I dati inerenti al flusso delle prenotazioni non possono essere usati per altri scopi che non siano utili al funzionamento del sistema.
  - Non si deve poter risalire all'identità dello studente tramite l'ID della prenotazione.

- **Performance:**
  - **Responsiveness:** Il sistema deve essere in grado di inviare velocemente l'email di conferma prenotazione. Un ritardo troppo marcato, potrebbe essere percepito come un malfunzionamento del sistema. Un tempo di risposta ideale sarebbe di massimo 10 secondi. Inoltre, anche l'interfaccia web dovrebbe essere reattiva e dinamica.

Nota: l'*Usability* e la *Responsiveness* è una caratteristica critica del sistema. Questo perché gli alunni, non essendo obbligati a usare questo servizio, dovranno essere incentivati a farlo tramite una interfaccia semplice, intuitiva e reattiva.

3 Progettazione

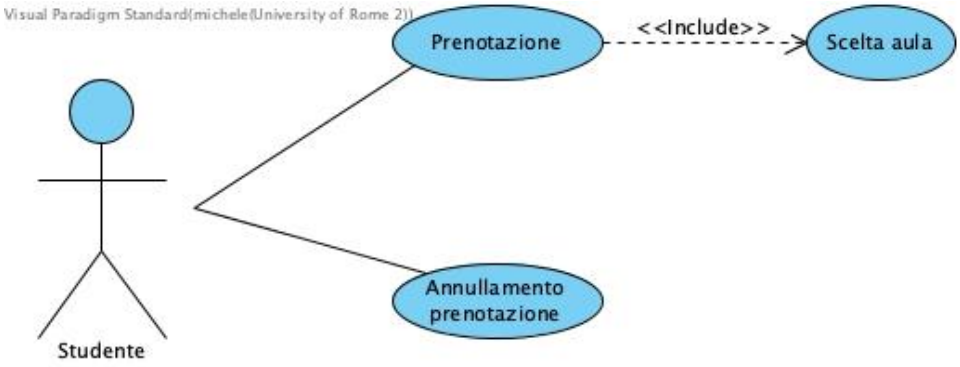
3.1 Funzionalità e casi d'uso

L'attore principale che userà il sistema è lo studente iscritto all'università di Tor Vergata e sarà l'Università a fornirgli le credenziali utili alla prenotazione. Una volta scelto il giorno in cui si vuole

prenotare, il sistema provvederà a fornire una tabella con le disponibilità delle aule suddivise in fasce orarie.

\	09:00	10:00	11:00	12:00
1	136	137	137	137
10	80	80	80	80
10A	189	189	189	189
10PP2	147	147	147	147
1A	179	179	179	179
1PP2	187	187	187	187
2	189	189	189	189
2A	191	191	191	191
2PP2	55	55	55	55

L'utente potrà quindi vedere la disponibilità dell'aula, prenotarne una ed eventualmente annullare una prenotazione.



3.1.1 Documentazione scelta aula

Use case Scelta Aula	
Nome	Scelta aula
Portata	Cliente
Attori	Studente
Input	Aula, data/ora
Output	Calendario con disponibilità aule
Pre-condizioni	
Scenario Principale	Una volta che lo studente riceve il calendario delle disponibilità giornaliere delle aule, divise per fasce orarie, effettua la sua scelta.
Scenario Alternativo	• Lo studente impiega troppo tempo nella scelta e nel frattempo le disponibilità cambiano.

	• Lo studente non gradisce le disponibilità che ha ricevuto dal sistema, e chiude l'applicativo.
Post-condizioni	Scelta l'aula, si avvia la fase di prenotazione.
Frequenza di ripetizione	A scelta dell'utente.

3.1.2 Documentazione prenotazione

Use case Prenotazione	
Nome	Prenotazione aula
Portata	Cliente
Attori	Studente
Input	Aula, data/ora, e-mail
Output	ID prenotazione
Pre-condizioni	L'email deve essere un'e-mail universitaria (di Tor Vergata).
Scenario Principale	Lo studente, regolarmente iscritto e quindi in possesso di un'e-mail universitaria, sceglie un'aula e l'ora tra quelle presenti nel calendario delle disponibilità, facendo partire una procedura di prenotazione.
Scenario Alternativo	• Lo studente, cambiando idea, non conferma la prenotazione in corso. • L'utente immette un'e-mail non riconosciuta e viene invitato ad inserirne una valida. • La prenotazione fallisce se l'utente non inserisce il codice di verifica entro un tempo prestabilito.
Post-condizioni	La prenotazione va a buon termine e viene registrata nel sistema.
Frequenza di ripetizione	Una sola prenotazione in corso.

3.1.3 Documentazione annullamento prenotazione

Use case Annullamento prenotazione	
Nome	Annullamento prenotazione aula
Portata	Cliente
Attori	Studente

Input	ID prenotazione
Output	Eventuale conferma
Pre-condizioni	Lo studente deve essere in possesso di una prenotazione in corso di validità.
Scenario Principale	Lo studente nella mail di conferma della prenotazione riceverà un ulteriore link da cliccare in caso voglia annullarla.
Scenario Alternativo	• Lo studente decide di non voler più annullare la prenotazione dopo aver ricevuto un pop-up di conferma. • Lo studente prova ad annullare una prenotazione già annullata. • Lo studente prova ad annullare una prenotazione "scaduta".
Post-condizioni	La prenotazione viene annullata e viene reso nuovamente disponibile il posto precedentemente occupato.
Frequenza di ripetizione	Una volta.

3.2 Sequence diagram

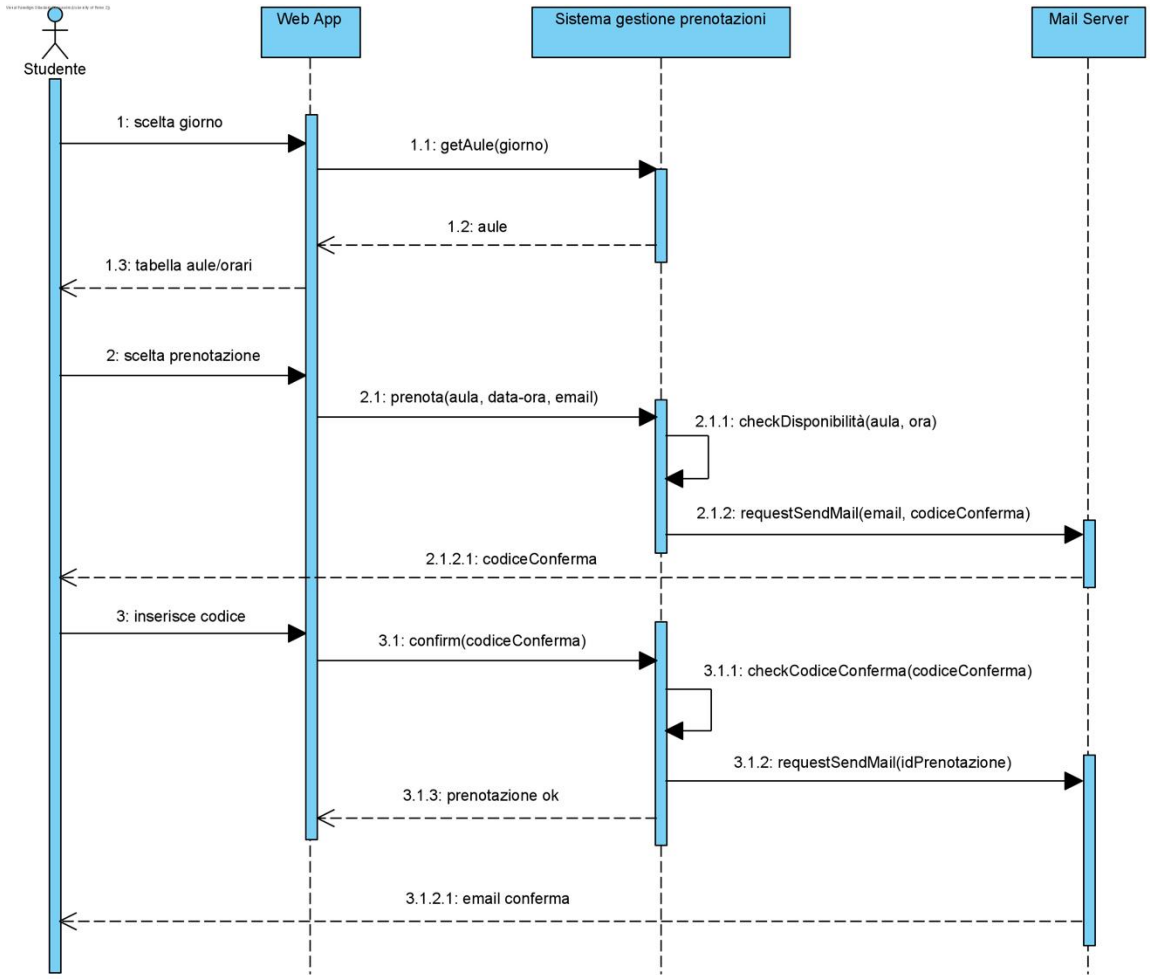
Il **sequence diagram** è una rappresentazione formale che mostra uno scenario di interazione tra un attore (nel nostro caso lo studente) ed una serie di oggetti del sistema.
Tale diagramma è caratterizzato da due assi, di cui quello orizzontale mostra gli oggetti che partecipano all'interazione, mentre quello verticale mostra l'evoluzione dell'oggetto nel tempo.
È quindi descritta una sequenza deterministica di interazioni che si susseguono tra oggetti e attori nel corso del tempo. Queste interazioni non sono altro che scambi di messaggi.

Lo scambio di un messaggio è rappresentato da una freccia il cui verso indica la direzione della comunicazione. Si distinguono più tipi fondamentali di scambi di messaggi.
In particolare notiamo nel nostro sequence diagram uno scambio di messaggi di tipo *sincrono* e *di risposta*.

Uno scambio di messaggi è sincrono, raffigurato con la freccia "completa", se il mittente si blocca in attesa di una risposta. Invece uno scambio di messaggio è "di risposta" ed è raffigurato con la freccia

“tratteggiata”, se il **mittente** lo effettua come risposta ad uno scambio di messaggio di cui era destinatario.

3.2.1 Procedure di Scelta e Prenotazione

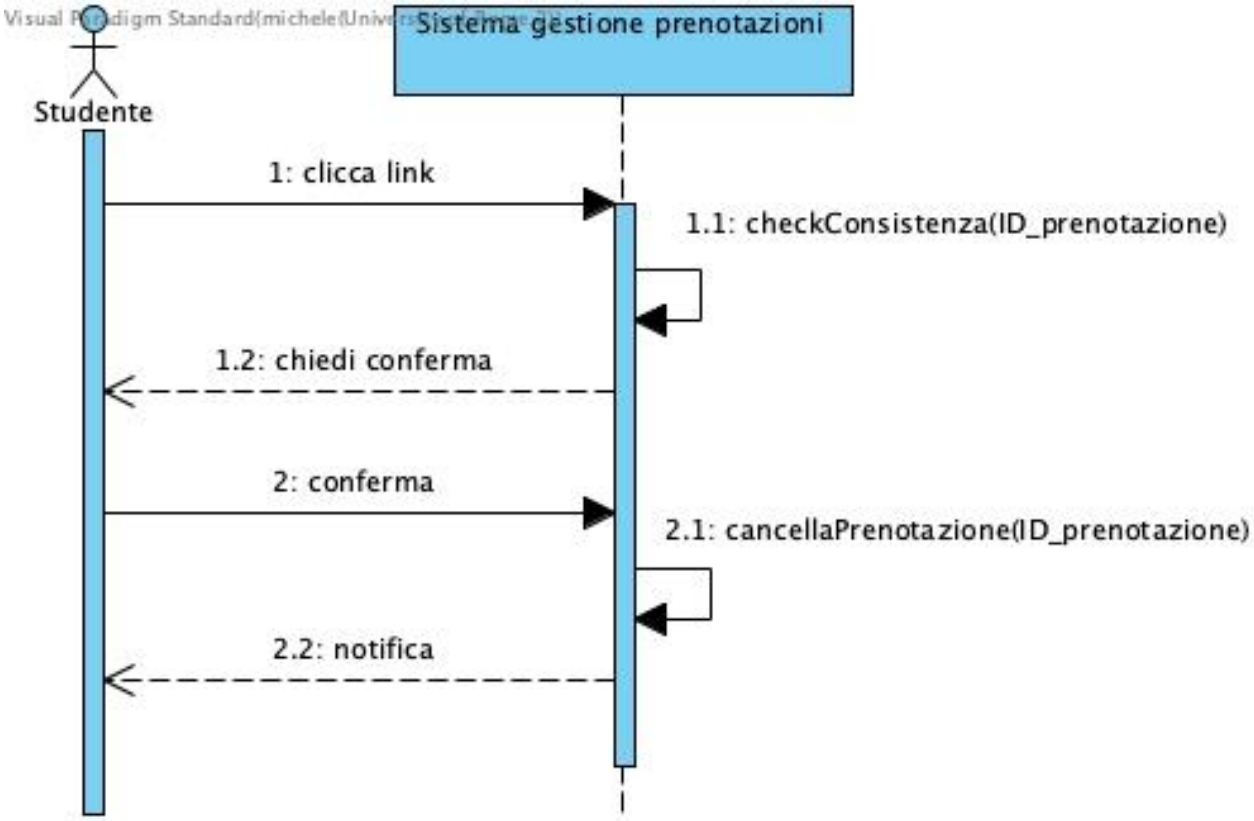


In questo primo Sequence Diagram vengono mostrate le procedure di **scelta** e di **prenotazione** di un'aula. Abbiamo scelto di rappresentare entrambe le procedure in un unico diagramma per via della loro natura strettamente correlata. Inoltre specifichiamo che questo diagramma descrive il comportamento del caso in cui non si presentino eccezioni.
Descriviamo in ordine le seguenti fasi:

1. L’utente chiede all’applicazione di mostrare le aule in un giorno a scelta.
    1.1. L’applicativo lato client, con il quale l’utente si è interfacciato, fa una richiesta al server di gestione delle prenotazioni e chiede le aule disponibili nel giorno indicato dall’utente.
    1.2. Il server di gestione delle prenotazioni restituisce le disponibilità del giorno richiesto.
    1.3. Il client mostra all’utente una tabella con tutte le disponibilità del giorno richiesto, diviso in fasce orarie.

2. L’utente clicca nella casella che specifica aula e ora della prenotazione desiderata e immette la sua mail universitaria.
    2.1. Il client fa una richiesta di prenotazione aula al server, specificando aula, data/ora e mail.
        2.1.1. Il server effettua un controllo per verificare l’effettiva disponibilità dell’aula.
        2.1.2. Se il controllo va a buon termine, il server genera un **codice di conferma**, e invia una richiesta a un server mail universitario, per avviare la procedura di conferma della prenotazione. In questa richiesta sono specificati email dello studente che prenota e il codice di conferma generato.
            2.1.2.1. Se l’email universitaria corrisponde ad una credenziale valida, il server mail invia una mail all’utente con un codice di conferma.
3. L’utente inserisce il codice di conferma richiesto dall’applicazione. Questo dovrà essere inserito entro un tempo prestabilito, altrimenti la procedura di prenotazione non andrà a buon fine.
    3.1. L’applicazione invia al server il codice di conferma inserito dall’utente.
        3.1.1. Se il codice è corretto, la procedura è considerata terminata con esito positivo. Il server aggiorna quindi il database delle prenotazioni e genera un **idPrenotazione**.
        3.1.2. Il server di gestione delle prenotazioni invia al server mail una richiesta di notifica allo studente che ha prenotato. In questa richiesta sono ovviamente annotati i dati della prenotazione.
            3.1.2.1. Il server mail invia allo studente una mail di notifica in cui sono presenti la conferma dell’avvenuta prenotazione, un riepilogo di giorno, ora e aula prenotata e in fine un link necessario per l’annullamento della prenotazione (vedi dopo).
        3.1.3. Il server notifica l’applicazione client dell’avvenuta prenotazione.

3.2.2 Procedure di Annullamento Prenotazione

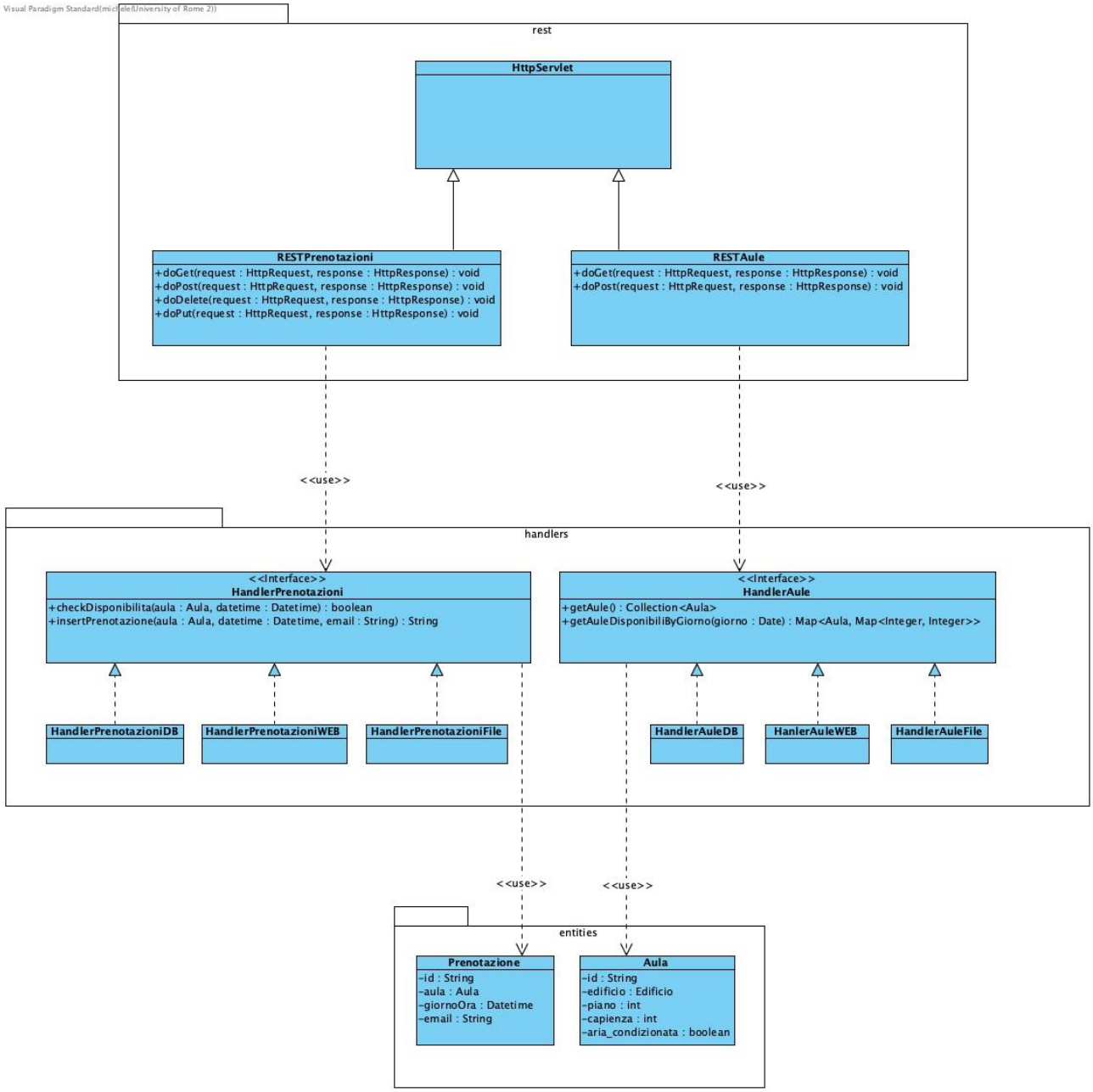


Di seguito verrà descritta in linguaggio naturale la procedura di **annullamento prenotazione** mostrata nel diagramma precedente.

1. Lo studente segue il link di annullamento ricevuto nell'email di notifica al termine della fase di prenotazione. Cliccando il link, sarà inviata una richiesta di annullamento al server di gestione delle prenotazioni.
    1.1. Nella richiesta è presente l'idPrenotazione, del quale il server verificherà la validità.
    1.2. Se l'id è valido, il sistema risponde all'utente mandando una pagina web in cui richiede una conferma (per sicurezza).
2. Se lo studente è veramente interessato ad annullare, lo conferma al server.
    2.1. Il server cancella la prenotazione.
    2.2. Il server notifica l'avvenuta rimozione della prenotazione.

3.3 Diagramma delle classi

I Diagrammi delle classi descrivono il tipo degli oggetti che compongono il sistema e le relazioni statiche esistenti tra loro. Sono adatti per la rappresentazione di architetture software dettagliate.



La struttura del nostro diagramma è composta da tre package: *Rest*, *Handlers* ed *Entities*. Che a loro volta interagiscono fra di loro e contengono diverse classi che descriveremo di seguito.

3.3.1 Uso dei principi REST

Nel nostro sistema facciamo riferimento al **REST** (Representational State Transfer), uno stile di progettazione utilizzato a partire dagli anni 2000 (anno in cui fu introdotto), basato su HTTP.
I principi REST utili al nostro sistema sono:

- La netta distinzione logica e progettuale tra Client e Server (come si può notare dai Sequence Diagram).
- L'identificazione univoca delle risorse tramite URI.
- Uniformità d'interfacciamento alle risorse tramite i metodi HTTP (GET, POST, DELETE, PUT). Ad esempio, nella classe *RESTAule,* il metodo doGet intercetta una richiesta HTTP del tipo "GET /safe-study-service/aule/2021-06-22" e restituisce una collezione delle disponibilità delle aule riferite alla data 2021-06-22.

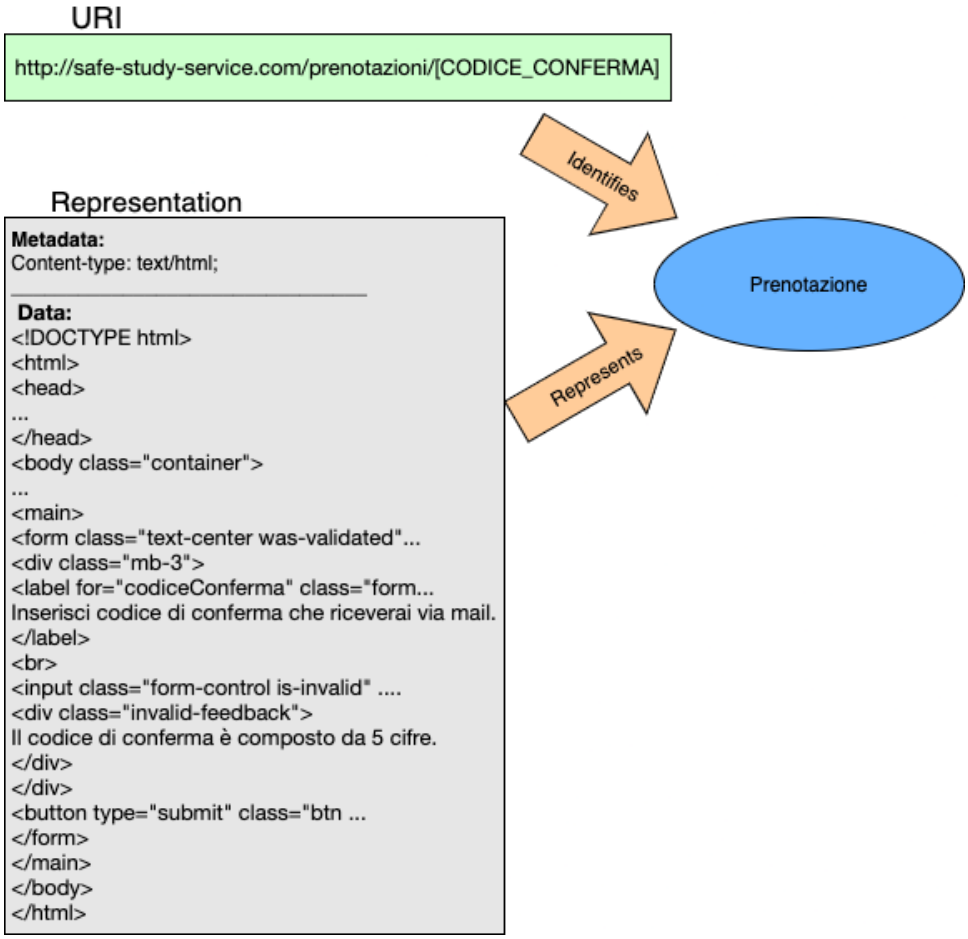
3.3.2 Uso di Java Servlet

Per l'implementazione del lato server abbiamo pensato che fosse una buona idea fare uso delle Servlet Java.
Una Servlet è un programma in esecuzione scritto in Java, localizzato sul server, che permette di gestire le richieste dei client. Un utente interessato a conoscere quali siano gli orari di disponibilità delle aule, può prenotarle ed eventualmente annullarne la prenotazione interfacciandosi con le Servlet adibite ai rispettivi compiti.

La Servlet utili per la realizzazione del server di prenotazione sono:

1. *RestPrenotazioni*
2. *RestAule*

3.3.2.1 RestPrenotazione

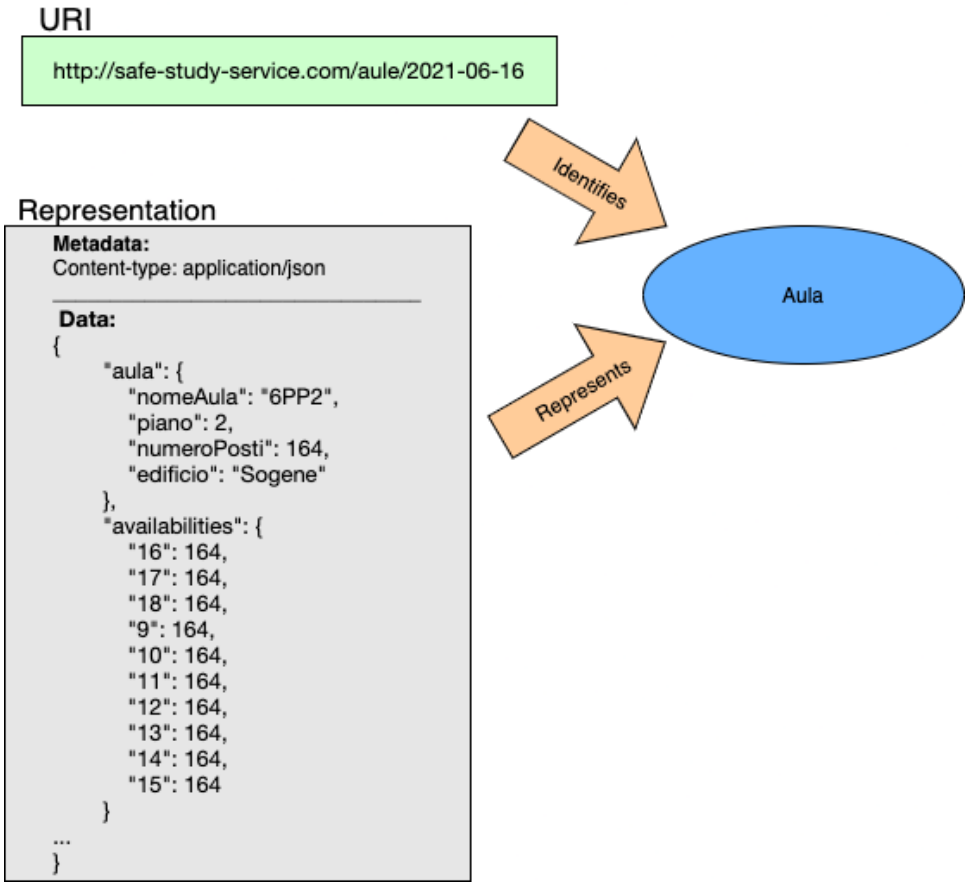


RestPrenotazione è l'insieme di operazioni che riguardano la prenotazione dell'aulai (ncluso il suo annullamento).
I metodi offerti dalla servlet in questione sono

Metodo	Descrizione

doGet	Avviare la fase di prenotazione, generando in caso di disponibilità un codice di conferma.
doPost	Conferma la prenotazione tramite codice di conferma inserito dall'utente, e registra la prenotazione.
doDelete	Annulla una prenotazione.

3.3.2.2 RestAule



RESTAule fornisce l'insieme di operazioni che riguardano le informazioni sulle aule (disponibilità, orari, capienza, …).

Metodo	Descrizione
doGet	Restituisce le informazioni sulle aule con relative disponibilità, in base al giorno specificato.
doPost	Inserisce nel sistema una nuova aula.

3.3.3 Handler

Le informazioni sulle aule possono essere contenute in un DB, su file o su un altro server. Le modalità di accesso ai dati sono però **invisibili** alla servlet, che si avvale di una interfaccia detta **Handler**

(*HandlerAule* o *HandlerPrenotazioni*) il cui compito è quello di accedere ai dati, nascondendo il dettaglio implementativo (*Information Hiding*).

Le suddette interfacce sono:

- *HandlerPrenotazioni*
- *HandlerAule*

3.3.3.1 HandlerPrenotazioni

Dipendentemente dal tipo di risorsa contenente le informazioni, l'interfaccia è implementata dalla classi *HandlerPrenotazioniDB*, *HandlerPrenotazioniWeb* o *HandlerPrenotazioniFile*.

Con questa interfaccia, che fa uso della classe *Prenotazione*, si possono eseguire i seguenti metodi:

Nome metodo	Descrizione
checkDisponibilità	Prende in input aula, data e ora. Verifica se l'aula è disponibile ritornando TRUE o FALSE.
insertPrenotazione	Prende in input aula, data, ora e email. Inserisce la prenotazione restituendo un codice di conferma.
deletePrenotazione	Rimuove una prenotazione che deve essere annullata.

La classe Prenotazione contiene i seguenti campi:

Nome	Descrizione
id	Una stringa che identifica univocamente la prenotazione.
aula	Nome dell'aula prenotata.
giornoOra	Giorno e ora di prenotazione.
email	Una stringa che contiene l'e-mail del prenotante.

3.3.3.2 HandlerAule

A seconda del tipo di risorsa contenente le informazioni, l'interfaccia viene implementata dalla classi *HandlerAuleDB*, *HandlerAuleWeb* o *HandlerAuleFile*.

Tramite questa interfaccia, che fa uso della classe *Aula*, si possono eseguire i metodi:

Nome metodo	Descrizione
getAule	Ritorna l'insieme delle aule con relative informazioni.
getAuleDisponibiliByGiorno	Prende in input un giorno e ritorna una mappa delle aule disponibili in tale giorno con relativa disponibilità di orari e posti.

La classe Aula contiene i seguenti campi:

Nome	Descrizione
id	Stringa che identifica univocamente l'aula.
edificio	Edificio dell'università in cui si trova l'aula
piano	Piano in cui si trova aula.
capienza	Numero massimo di persone che possono trovarsi contemporaneamente nell'aula.
aria_condizionata	Presenza o meno di aria condizionata nell'aula.

4 Sviluppo

Nello sviluppo di questo sistema software ci siamo avvalsi di diverse tecnologie.

Innanzitutto, come visto in precedenza, il protocollo usato per la comunicazione tra applicazione e server di gestione è basato su REST.

Il lato client è stato sviluppato con le classiche tecnologie HTML5/CSS/JavaScript in versione Vanilla; non è stato necessario l'ausilio di particolari frameworks. Per quanto riguarda la fase di "scelta aule" abbiamo implementato una pagina *dinamica*. Dopo aver specificato il giorno desiderato, vengono generate le tabelle delle disponibilità "on-the-fly", senza alcun refresh della pagina web.
In questo modo vengono soddisfatti i requisiti di **Responsiveness** e **Usability**. Durante la fase di scelta si potrebbero voler scegliere altri giorni ma, se ogni volta che viene cambiato giorno avviene un tedioso refresh, l'utente potrebbe essere disincentivato ad usare l'app.

Per quanto riguarda il lato server, abbiamo fatto uso del Server Web Apache Tomcat, unito alle Java Servlet, in quanto offrono uno stile architetturale che più rispecchia il class diagram descritto.
La scelta di Java come linguaggio di sviluppo dell'applicazione è dovuta a diversi fattori. Innanzitutto il paradigma OO ci ha permesso di implementare in modo "diretto" gli schemi costruiti durante la fase di analisi e progettazione. Inoltre, non essendo inizialmente sicuri su quale macchina avrebbe eseguito l'applicazione, la portabilità di Java ci ha permesso di risparmiarci qualche "mal di testa". Qualunque fosse stata la macchina, il server Tomcat avrebbe fatto girare l'applicazione senza problemi.

Tutti i dati utili al sistema sono stati salvati in un DataBase, perciò le interfacce **HandlerAule** e **HandlerPrenotazioni** sono state implementate dalle specializzazioni DB.

Nello specifico, il DBMS usato è MariaDB, in quanto il server gira su una macchina con kernel Linux (non essendo MySQL disponibile per il sistema). Nella seguente figura è rappresentato il diagramma E/R del database, semplice e minimale. Non è presente l'entità EDIFICIO, avendo progettato il sistema **solo** per la macroarea di Scienze. Si potrebbe facilmente ampliare il database con le aule presenti in altri edifici con una nuova entità.

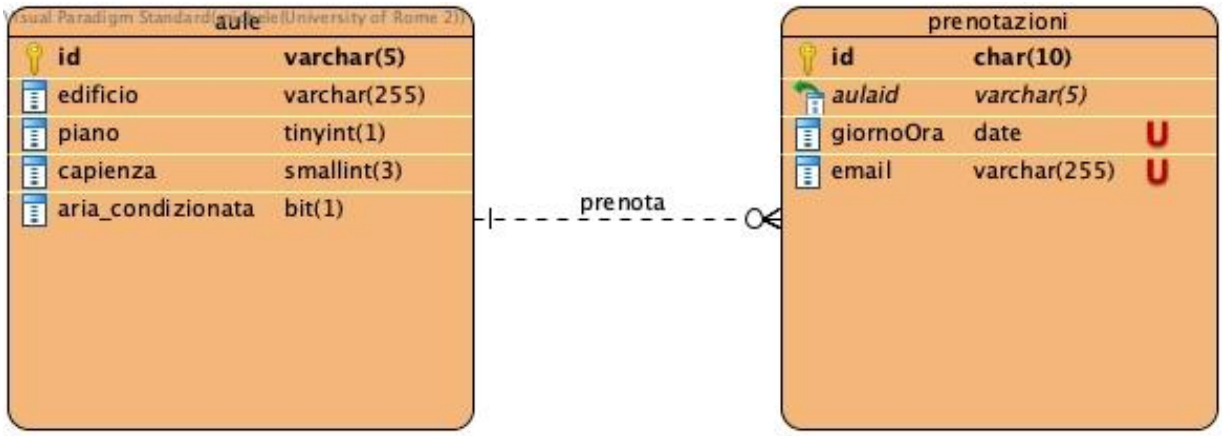


Figura 4.1 Diagramma E/R

L'invio dei messaggi di prenotazione e conferma è gestito da un server Mail che, essendo costituito da varie componenti, viene implementato con diversi programmi:

- **Postfix** è il Mail Transfer Agent (MTA) che si occupa di inviare le e-mail utilizzando SMTP. Costruito il messaggio e determinato il destinatario, il tutto viene passato a Postfix che si occupa del resto.
- Inizialmente era stato configurato anche **Dovecot**, un Mail Delivery Agent (MDA) che si occupa di trasferire le mail nelle mailbox dei destinatari. Una successiva revisione dei requisiti e degli obiettivi del sistema, ne ha portato alla disattivazione, non essendoci il bisogno di ricevere mail da parte degli studenti.
- **OpenDKIM** (DomainKeys Identified Mail) è un sistema che fa in modo che le mail inviate dal sistema siano autenticate. In questo modo viene garantito il requisito della **security**. I messaggi vengono invece firmati da una chiave privata dal mittente. Tale chiave viene controllata dal ricevente consultando i record TXT del server DNS di dominio del mittente, confermandone la provenienza. Un esempio di header mail inviato con questo sistema è mostrato di seguito.

```
Received: by mail.ablant.xyz (Postfix, from userid 1004)
    id 894DD102A1; Fri, 11 Jun 2021 12:06:54 +0000 (UTC)
DKIM-Signature: v=1; a=rsa-sha256; c=relaxed/simple; d=ablant.xyz; s=mail;
    t=1623413214; bh=DJtYjzyGxNwJ0zOWSbCXAIboWwW60UKuX2NtMCogarY=;
    h=Subject:To:Date:From:From;
    b=i4UBmNYYzF7mv5lM32bERgwd8NGvo590hqcnlUohox9eUnXiGPQjFrxUUJjuxOeVF
     ZO5qiU9XiUaWG5fI9V+N9I6+NCug4dgWjLteNg0gVAZqE6S8YAxrEtlaKQoo8RP2uB
     jV9ksa0CCyBO3sv1W68kM819yvsgo4GsGnq1HmnREfa9HmVcroRMSSZ6Z/R+FKUMSV
     nddxW/lcaDE/hgKcNtfrRXHtCnOA+SqAOd6K7Oz1O9rstQN8//tbwSU/69GRjP/Spm
     0eqZj+AoqDhYeAY4WfF+4vk+Er8ntBFTQmywgcliC1Wbks+TVRwIVBOcxlJksZOsDV
     c7Lna0nGGA2UA==
```

Nonostante l'utilizzo di OpenDKIM, che previene l'utilizzo del nostro dominio da parte degli spammer, le nostre email vengono comunque flaggate come spam nella maggior parte dei casi.

Per concludere, per la gestione del progetto, abbiamo fatto uso di **Git**, il famoso strumento per il controllo di versione distribuito.

5 Modello di ciclo di vita

5.1 Il modello scelto

Il modello di ciclo di vita adoperato nello sviluppo del nostro software potremmo di descriverlo come un ibrido tra il modello a **prototipo rapido** e il **modello incrementale senza overall architecture**.

Inizialmente, non avendo la certezza che il nostro progetto fosse accettato come idoneo all'esame, la fase di analisi dei requisiti e definizione degli obiettivi è stata fatta in modo sommario e a bassa risoluzione, tramite un semplice documento di **outline description** (descrizione generale).
L'architettura generale dell'applicazione non era stata definita ma, nonostante questo, è stata sviluppata una prima bozza dell'interfaccia grafica e del sistema di scelta delle aule che simulasse il comportamento dell'applicazione negli *use cases* reali.
Grazie a questo **prototipo rapido**, abbiamo fatto molta chiarezza su **cosa** volessimo da un'applicazione del genere. Infatti, abbiamo visto questo prototipo dal punto di vista di uno studente (il cliente), e non dal punto di vista dello sviluppatore.

Una volta validato il prototipo, è iniziata la definizione dei requisiti a una grana più fine e l'espansione del prototipo iniziale con le funzionalità che man mano si presentavano come necessarie.

5.2 Le build del progetto

Una volta ottenuto un prototipo rapido che ha chiarito le idee sui requisiti che doveva avere il nostro sistema, abbiamo cominciato a sviluppare seguendo un **modello incrementale senza overall architecture**.

Abbiamo sviluppato, validato e integrato i seguenti build:

1. User Interface che consentiva la scelta del giorno da parte dello studente, e la relativa visualizzazione dei dati.
2. Un primo servizio lato Server che consentiva l'estrazione di informazioni dal DB, utili al client per costruire la tabella delle disponibilità.
3. Implementazione lato Client della procedura di prenotazione (inserimento mail, inserimento codice conferma, notifiche di avvenuta conferma, notifiche errori, ecc...).
4. Un ulteriore servizio lato Server che consentisse di gestire l'intera fase di prenotazione, dalla richiesta alla convalida.
5. Un server mail utile a notificare agli studenti la mail con il codice di conferma da inserire in fase di prenotazione, e l'eventuale mail di conferma.

Per via della semplicità strutturale dell'interfaccia utente, c'è stato un largo riuso del codice usato per implementare il prototipo rapido. A livello logico, il prototipo rapido rispettava già tutte le caratteristiche del primo build, perciò abbiamo solamente dovuto fare dei ritocchi dal punto di vista stilistico.
Purtroppo invece per le altre componenti, non è stato possibile riciclare troppo codice ☹.

5.3 Suddivisione del lavoro

Essendo il nostro team composto da 5 persone, è stato possibile dividerci in sotto-squadre, da una o due persone, ognuna delle quali si sarebbe occupata di una particolare fase dello sviluppo dei build. Chiaramente, non essendo i ruoli rigidamente definiti, è stato presente un certo movimento

trasversale dei ruoli nelle varie fasi. Nonostante ciò, il lavoro di specifica, sviluppo e rilascio dei build è stato abbastanza coordinato.

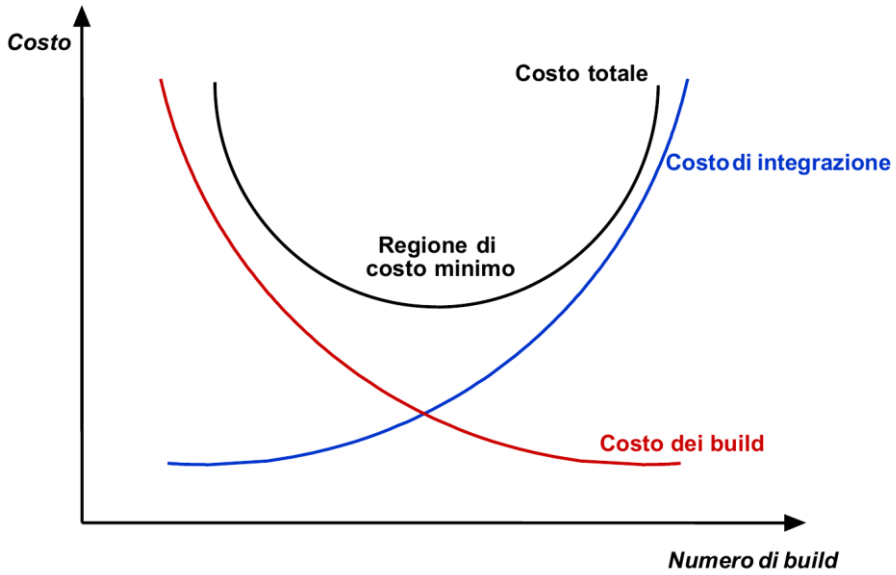
Mentre una sotto-squadra si occupava di implementare una particolare feature, un altro team era occupato nella definizione di nuove specifiche, mentre un terzo team dell'integrazione con le precedenti versioni.

I moduli implementati ad ogni build sono stati di dimensione relativamente ridotta e il loro singolo sviluppo non ha impiegato molto tempo. Il vero problema, in termini di effort, è stata l'integrazione e le varie 'incongruenze' che si presentavano, soprattutto dovute ai diversi ambienti di sviluppo utilizzati dai membri del team. Per esempio, un problema riscontrato è stato che su Windows i nomi delle tabelle in MySQL non sono case-sensitive, invece su Linux lo sono, perciò più di una volta sono avvenute incomprensioni tra chi si occupava di scrivere le query e chi di integrarle e testarle nel sistema.

Durante l'intero processo, lo sviluppo è stato eseguito in locale sulle macchine di ciascun membro, quindi un ulteriore porzione di effort è stata spesa per rendere il sistema eseguibile su di un server "finale".

5.4 Impatto dei costi

Prendendo come riferimento il grafico presente nelle slide del professore, stimiamo in maniera intuitiva che il costo totale dello sviluppo si può attestare nella regione di costo minimo. Questo perché riteniamo che il numero di build non sia eccessivo, inoltre la loro dimensione (e di conseguenza il singolo costo di realizzazione) è contenuta.



Data la nostra inesperienza in progetti e organizzazione di gruppo, possiamo ritenerci soddisfatti dal fatto che il costo non sia stato ancora maggiore.

6 L'architettura

L'architettura del sistema *Safe Study Service* è chiaramente un'architettura distribuita di tipo **two-tier client/server**. Inoltre l'approccio usato è di tipo **BCE** (**Boundary-Control-Entity**).

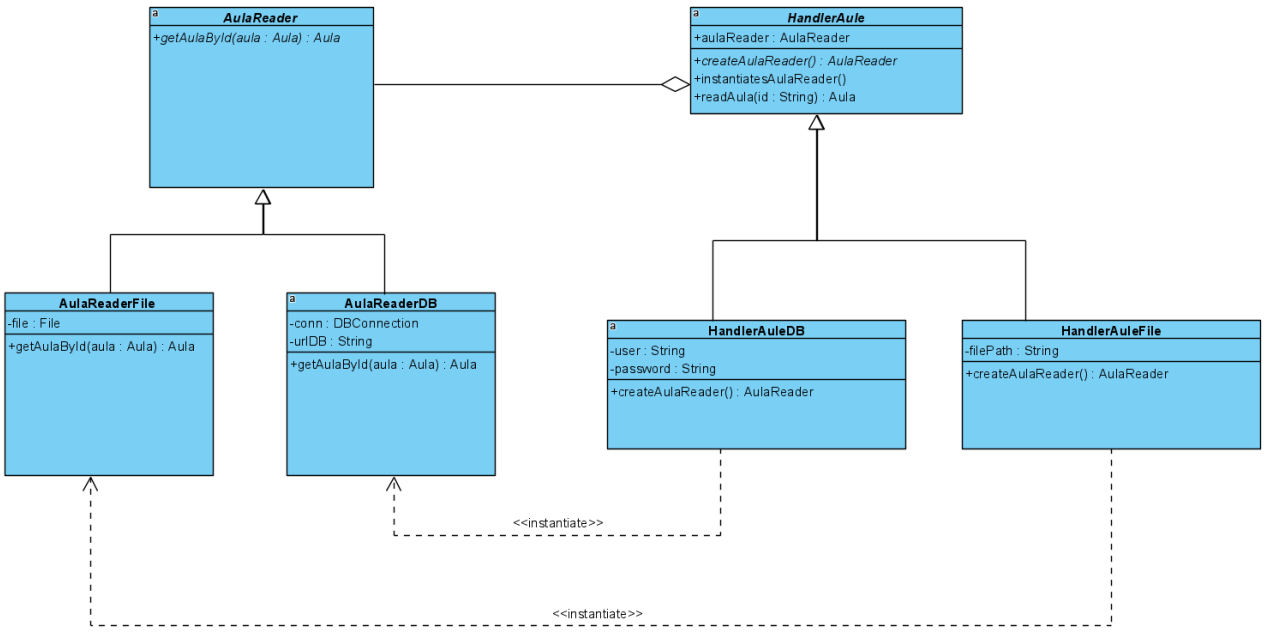
Nello specifico, nel nostro modello il lato client è di tipo **thin-client**, perché esso si occupa solamente del **presentation layer**, ovvero di intercettare le richieste dell'utente, di inoltrarle al server e di fornire una presentazione delle risposte.
Il server invece è appesantito sia dall'**application processing** (ovvero del livello logico che gestisce le funzionalità del servizio) sia del **data management** (ovvero si occupa anche del livello software che ha accesso ai dati).

7 Design Pattern

7.1 Primo Design Pattern

Facendo riferimento al nostro caso specifico, una possibile soluzione progettuale che potrebbe essere applicata per il problema della creazione in un oggetto che acceda in lettura alle entità *Aule* del sistema consiste nell'uso del design pattern **Factory Method**.

Più nello specifico supponiamo di avere una classe *HandlerAule* che ha il compito di restituire le informazioni sulle aule, e che l'accesso alle informazioni sia effettuato da un *AulaReader.*
Il reader delle aule deve poter leggere o da database o da file, e questo compito è assegnato alle classi specifiche *AulaReaderDB* e *AulaReaderFile*.
Dato che *HanderlAule* non sa quale tipo specifico di *AulaReader* usare (perché magari dipende dal sistema in cui gira il SW) delega alle sue sottoclassi specializzate *HanderlAuleDB* e *HanderlAuleFile* il compito di decidere **quale** tipo di *AulaReader* istanziare.



Nel nostro caso i **partecipanti** sono:

- **Creator** -> HandlerAule

- **ConcreteCreator** -> HandlerAuleDB, HandlerAuleFile
- **Product** -> AulaReader
- **ConcreteProduct** -> AulaReaderDB, AulaReaderFile

Mentre il **factoryMethod** è il metodo `createAulaReader`.

Come possiamo osservare *AulaReaderDB* e *AulaReaderFile* sono delle generalizzazioni di *AulaReader*, il quale a sua volta offre una interfaccia che consente di accedere in lettura alle aule registrate nel sistema. Ognuna delle due generalizzazioni implementa un modo diverso di accedere ai dati, relativamente da database o da file.

Di seguito un esempio di codice dei diversi *AulaReader*

```
// Product
interface AulaReader {
    public Aula getAulaById(String id);
}
```

```
// Concrete Product
public class AulaReaderDB implements AulaReader {
    private Connection conn;
    private String urlDB;
    public AulaReaderDB(String user, String password) {
        Class.forName("com.mysql.jdbc.Driver");
        this.urlDB = "jdbc:mysql://localhost/progettoIS?user="+user+"&password="+password
        this.connection = conn = DriverManager.getConnection(this.urlDB);
    }

    public Aula getAulaById(String id){
        Statement stmt = conn.createStatement();
        String query = "SELECT * FROM aule WHERE id="+id;
        ResultSet resultQuery = stmt.executeQuery(query);

        while (resultQuery.next()){
            if(resultQuery.getString("id").equals(id)){
                String edificio = resultQuery.getString("Edificio");
                int numeroPosti = resultQuery.getString("capienza");
                return new Aula(id, edificio, numeroPosti);
            }
        }
        return null;
    }
}
```

```java
public class AulaReaderFile implements AulaReader{
    private File file;

    public AulaReaderFile(String filePath) {
        this.file = new File(filePath);
    }

    public Aula getAulaById(String id){
        FileReader fr = new FileReader(this.file);
        BufferedReader br = new BufferedReader(fr);
        String line;
        while((line = br.readLine())!=null) {
            /* supponiamo che ogni riga rappresenta i dati di un'Aula
             * (id, edificio, capienza) divi da uno spazio
             */
            String[] data = line.split(" ");
            if(data[0].equals(id)){
                String edificio = data[1];
                int numeroPosti = data[2];
                return new Aula(id, edificio, numeroPosti);
            }
        }
        return null;
    }
}
```

Per quanto riguarda il Creator *HandlerAule*, esso **delega** il compito di quale tipo di *AulaRead* istanziare alle sottoclassi che la estendono (*HandlerAuleDB* e *HandlerAuleFile*) definendo solamente la logica di **quando** accedere all'*AulaReader.*

Di seguito una semplice implementazione dei diversi *HandlerAule.* Come esempio, sono stati implementati due metodi `instantiatesAulaReader` e `readAula` che contengono la logica che definisce **quando** un *AulaReader* viene creato, e che mostra come viene delegato alle sottoclassi il compito di **quale** *AulaReader* istanziare.

```java
// Creator
abstract class HandlerAule {
    public AulaReader aulaReader;

    public abstract AulaReader createAulaReader(); // factoryMethod

    public void instantiatesAulaReader() {
        this.aulaReader = this.createAulaReader();
    }

    public Aula readAula(String id) {
        if(this.aulaReader == null) {
            this.instantiatesAulaReader();
        }
        return this.aulaReader.getAulaById(id);
    }
}
```

```java
// Concrete Creator
class HandlerAuleDB extends HandlerAule {
    private String user;
    private String passwd;

    public HandlerAuleDB(String user, String passwd){
        this.user = user;
        this.passwd = passwd;
    }

    public AulaReader createAulaReader() {
        return new AulaReaderDB(user, passwd);
    }
}

class HandlerAuleFile extends HandlerAule {
    private String filePath;

    public HandlerAuleDB(String filePath){
        this.filePath = filePath;
    }

    public AulaReader createAulaReader() {
        return new AulaReaderFile(filePath);
    }
}
```

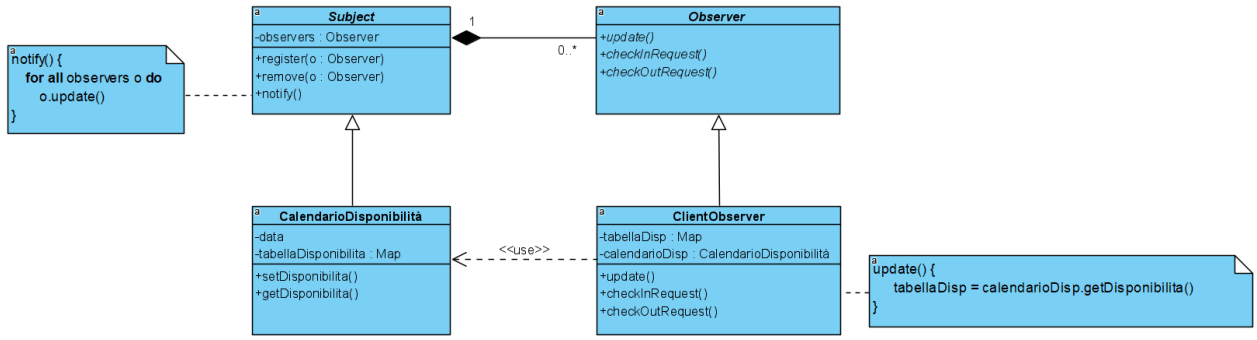
7.2 Secondo Design Pattern

Un secondo problema che può presentarsi è il seguente: supponiamo di voler migliorare il nostro sistema, implementando una interfaccia grafica che cambi in **tempo reale** le disponibilità delle aule nel giorno scelto dallo studente.

Una soluzione preconfezionata per risolvere questo problema consiste nell'uso del design pattern **observer**.

Nel nostro caso, si potrebbe creare come **ConcreteSubject** la classe *CalendarioDisponibilita*, ovvero una classe che conserva tutte le informazioni relative tutte le disponibilità delle aure nelle diverse fasce orarie di uno specifico giorno.

Mentre come **ConcreteObserver** si può implementare la classe **ClientObserver**, ovvero la classe che si occuperà semplicemente di mostrare la tabella con le disponibilità all'utente, e di modificarla qualora ci siano modifiche.



Di seguito una descrizione dei metodi principali

Metodi per **subject**

Nome metodo	Descrizione
register	Registra un observer nella lista degli observer che lo richiedono.
remove	Rimuove un observer di cui non c'è più bisogno.
notify	Notifica in modalità multicast a tutti gli observer registrati.
setDisponibilita	Aggiorno la tabella delle disponibilità.
getDisponibilita	Restituisce una copia della tabella delle disponibilità. Non viene restituita quella originale per evitare modifiche indesiderate.

Metodi per **observer**

Nome metodo	Descrizione
update	Aggiorno lo stato dell'observer a fronte di un cambio di stato del sunject.
checkInRequest	Permette a un'observer di registrarsi presso un subject.
checkOutRequest	Permette a un'observer di disdire la registrazione presso un subject.

Una piccola implementazione esemplificativa del pattern descritto applicato al contesto attuale è la seguente

```
// Subject
public class Subject {
    protected List<Oserver> observers;

    public Subject(){
        // Costruttore di class ...
    }

    public void register(Observer o){
        this.observers.add(o);
    }

    public void remove(Observer o){
        this.observers.remove(o);
    }

    public void notify(){
        for(Observer o : this.observers){
            o.update();
        }
    }
}
```

```
// Concrete Subject
public class CalendarioDisponibilita extends Subject {
    private Date data;
    private Map<Aula, Map<Integer, Integer>> tabellaDisponibilita;

    public CalendarioDisponibilita(){
        super();
        // Costruttore di class ...
    }

    private void setDisponibilita(Aula aula, int ora, int postiDisponibili){
        Map<Integer, Integer> disp = this.tabellaDisponibilita.get(aula);
        if(disp != null)
            disp.put(ora, postiDisponibili);
        notify();
    }

    public Map getDisponibilita(){
        return this.tabellaDisponibilita.clone();
    }
}
```

```java
// Observer
public interface Observer {
    public void update();
    public void checkInRequest();
    public void checkOutRequest();
}

// Concrete Observer
public class ClientObserver implements Observer {
    private Map<Aula, Map<Integer, Integer>> tabellaDisponibilita;
    private CalendarioDisponibilita calendarioDisponibilita;

    public ClientObserver(){
        // costruttore di classe ...
    }

    public void checkInRequest(){
        this.calendarioDisponibilita.register(this);
    }

    public void checkOutRequest(){
        this.calendarioDisponibilita.remove(this);
    }

    public void update(){
        this.tabellaDisponibilita = this.calendarioDisponibilita.getDisponibilita();
    }
}
```

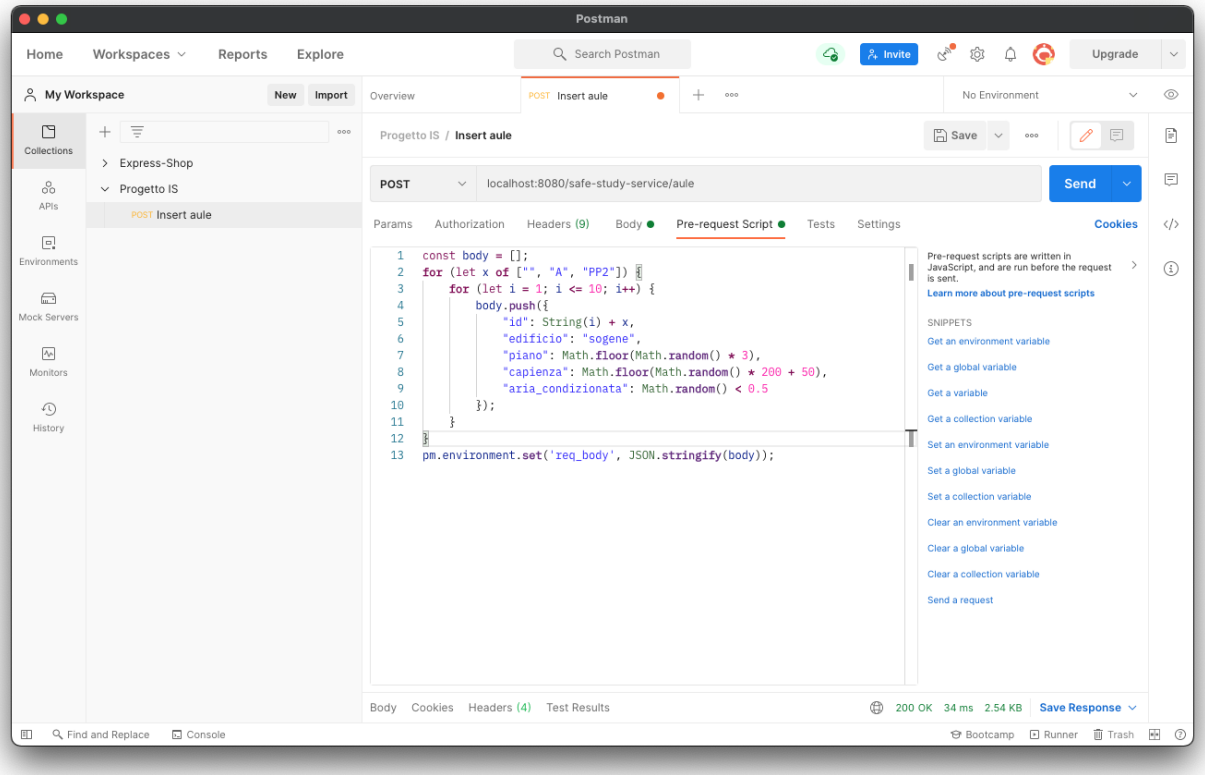
8 Testing

8.1 Dati di prova

Il testing del sistema è stato svolto con un software di nome **Postman**, un comodo strumento che permette di effettuare richieste HTTP.

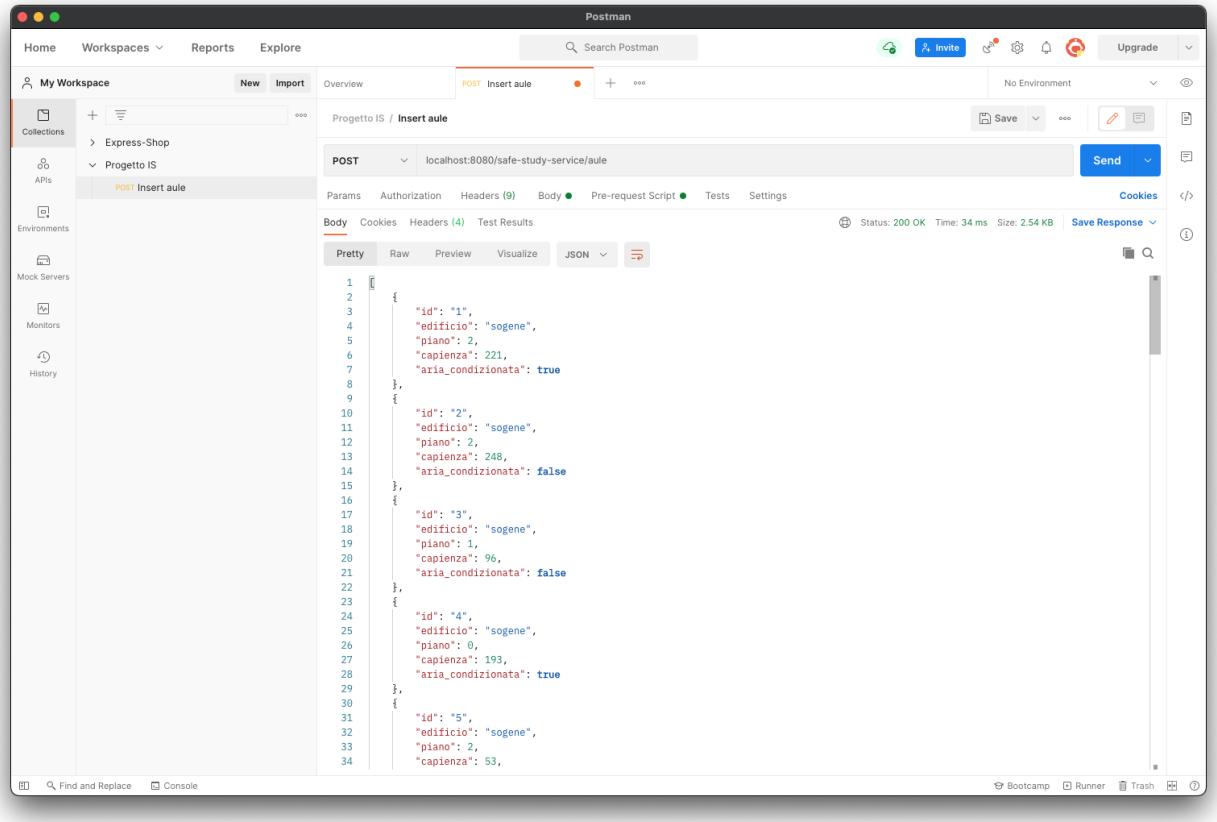
Il suo funzionamento è molto semplice:

1. Si immette nella barra degli indirizzi l'URL di endpoint di una richiesta e si seleziona il metodo HTTP appropriato nell'elenco a discesa.
2. È possibile modificare e/o aggiungere headers della richiesta http.
3. Si aggiungono, opzionalmente, i parametri della richiesta e Postman li concatena nell'URL se la richiesta è una GET, nel body altrimenti.



Postman permette la creazione di script che generano dati da inviare al server. Proprio questo è il metodo che abbiamo utilizzato per popolare il database.

Di seguito una immagine



8.2 Testing della usability

L'usabilità del sistema è stata testata fornendo il prodotto finale a dei colleghi universitari. L'impressione generale è stata che, essendo composto da poche schermate, tutte molto descrittive, il sistema è risultato estremamente comprensibile, ed user-friendly.

8.3 Testing della relaiability

La relaiability è stata testata provando, volontariamente, a fare delle prenotazioni non consistenti, ad esempio, simultaneamente per la stessa aula-giorno-ora.
La consistenza delle prenotazioni è stata garantita da una corretta configurazione del DBMS: come si può vedere dalla Figura [INSERIRE NUMERO DELLA FIGURA DEL DIAGRAMMA E/R], la coppia <email studente, giorno ora prenotazione> è unica, garantendo che un utente non possa prenotare più di un'aula contemporaneamente. La concorrenza è gestita dal sistema transazionale tipico del motore "InnoDB" di MySQL e MariaDB.

9 Rilascio

È possibile trovare una prima implementazione del nostro servizio seguendo il seguente link http://is.ablant.xyz:8080/safe-study-service/

10 Conclusioni

L'esperienza è stata molto significativa poiché trovarsi a lavorare in team per raggiungere un obiettivo comune produce una serie di benefici non solo a livello formativo ma anche personale. Dovendo adeguare il proprio modus operandi a quello degli altri, ognuno ha potuto apprendere nuove prospettive e metodologie di lavoro.
Infine facendo considerazioni sul lavoro svolto, una dinamica sulla quale abbiamo posto particolare attenzione, è stata quella di motivarci a vicenda. Laddove qualcuno di noi in un determinato giorno fosse meno motivato, subito interveniva un altro collega a spronarlo ed incentivarlo sul da farsi. Ed è forse quest'ultimo l'aspetto dal quale abbiamo tratto il più grande vantaggio.

10.1 Sviluppi futuri

- Il sistema è pensato solamente per la Macroarea di Scienze. Si potrebbe facilmente estendere il sistema all'intera università, semplicemente specificando a quale edificio si desidera andare.
- Il sistema consente di prenotare l'aula solamente per un'ora. Sarebbe comodo prenotare per più orari consecutivi in un'unica fase di prenotazione.