Università degli Studi di Salerno
Corso di Ingegneria del Software

EduCat
Object Design Document
Versione 1.0

Data: 21/12/2025

Coordinatore del progetto:

Nome	Matricola
Isabella Dima	0512119719

Partecipanti:

Nome	Matricola
Isabella Dima	0512119719
Anna Chiara Esposito	0512119143

Scritto da:	Anna Chiara Esposito

Revision History

Data	Versione	Descrizione	Autore
21/12/2025	1.0	Object Design Document: sottosistema	Tutto il team

Indice

1. Introduzione

1.1 Object Design trade-off

Trade-off	Descrizione
Utilizzo di memoria vs Tempo di risposta	Il sistema darà priorità a tempi di risposta più rapidi rispetto a un utilizzo minimo della memoria. A tal fine saranno utilizzati Beans per trasferire dati tra i layer e le operazioni di ricerca verranno delegate al livello di database, alleggerendo il livello di applicazione.
Performance vs Manutenibilità	Il sistema favorirà la manutenibilità a lungo termine rispetto alla performance. Questo sarà realizzato utilizzando dei DAO e separando la logica di business in classi apposite, migliorando così la modularità e testabilità.

1.2 Interface documentation guidelines

Questa sezione definisce le linee guida di documentazione delle interfacce e convenzioni nel codice che verranno adottate nell'Object Design di EduCat.
Queste convenzioni hanno lo scopo di aiutare gli sviluppatori a progettare le interfacce in maniera consistente e facilitare lo sviluppo definendo linee guida di leggibilità e facilità nella comunicazione.
Le convenzioni non si evolveranno nel corso del progetto.

1.1.1. Convenzioni sui nomi

- Le classi sono chiamate con singoli nomi con convenzione PascalCase (es. Utente, Lezione).
- I metodi sono denominati con sintagmi verbali in camelCase (es. login(), prenotaLezione()) mentre i campi e i parametri con sintagmi nominali in camelCase (es. email).
- Costanti ed enumerazioni sono scritti in maiuscolo (es. STATO)

1.1.2. Design delle interfacce

- Le interfacce pubbliche rendono disponibili solo le operazioni strettamente necessarie all'utilizzatore dell'interfaccia.
- I dettagli implementativi sono nascosti alle classi client.

1.1.3. Eccezioni

- Gli errori sono gestiti tramite eccezioni controllate e non valori di ritorno (es. SlotOccupatoException).
- Le eccezioni contengono messaggi che chiarificano la natura dell'errore.

1.1.4. Condizioni di Boundary

- Input invalidi risulteranno in eccezioni.
- Valori nulli, identificatori invalidi e altre condizioni di boundary sono gestite in maniera esplicita.

1.3 Definizioni, acronimi e abbreviazioni

- **Studenti**: utenti che ricercano lezioni private per migliorare la propria preparazione scolastica o universitaria.
- **Genitori**: nel caso di studenti minorenni, i genitori agiscono come referenti per la prenotazione e il pagamento delle lezioni.
- **Tutor**: figure abilitate a offrire lezioni private. Possono registrarsi, gestire il proprio profilo, definire tariffe e disponibilità e ricevere feedback dagli studenti.
- **Package**: collezioni di classi raggruppate insieme.
- **Design Pattern**: soluzioni tipiche ai problemi più comuni nella progettazione del software. Alcuni esempi sono il Facade Pattern per Pattern Strutturali, Command Pattern per Pattern Comportamentali e Abstract Factory per Pattern Creazionali.
- **MVC**: Model-View-Controller: pattern di architettura software che separa la logica di business, la presentazione e l'accesso ai dati persistenti per migliore manutenibilità e maggiore facilità nel debugging.

1.4 References

- ProblemStatement_EduCat v1.1
- RAD_EduCat v1.1
- SDD_EduCat v1.0
- Bernd Bruegge & Allen Dutoit - Object-Oriented Software Engineering: Using UML, Patterns, and Java

2. Packages

Questa sezione descrive la decomposizione del sistema in pacchetti (packages) Java, seguendo l'architettura MVC (Model-View-Controller) e la suddivisione in livelli logici (Interface, Application, Storage) definita nel System Design.
La struttura dei pacchetti è organizzata per separare chiaramente la logica di presentazione, di business e di persistenza.

2.1 Overview dei Pacchetti

Il sistema utilizza il root package it.unisa.educat.

Pacchetto	Descrizione	Dipendenze
it.unisa.educat.bean	Contiene le classi entità (Java Beans) che rappresentano i dati del dominio (es. Utente, Lezione). Corrisponde ai dati	Nessuna (Data Transfer Objects).

	definiti nel DataBase.	
it.unisa.educat.storage	Contiene le interfacce e le classi DAO (Data Access Object) per l'interazione con il database MySQL tramite JDBC.	it.unisa.educat.bean
it.unisa.educat.business	Contiene le classi Manager che implementano la logica di business e i servizi dei sottosistemi.	it.unisa.educat.storage, it.unisa.educat.bean
it.unisa.educat.control	Contiene le Servlet che gestiscono le richieste HTTP, coordinano i Manager e selezionano la vista di risposta.	it.unisa.educat.business, it.unisa.educat.bean
it.unisa.educat.view	Contiene le pagine JSP (JavaServer Pages) e risorse statiche (HTML/CSS) per l'interfaccia utente.	Utilizza i bean forniti dal control.

2.2 Organizzazione dei File

La struttura delle directory del codice sorgente riflette la gerarchia dei pacchetti:

- src/main/java/it/unisa/educat/bean/
- src/main/java/it/unisa/educat/storage/ (Include classi come GestioneUtenzaDAO, GestioneLezioneDAO)
- src/main/java/it/unisa/educat/business/ (Include classi come GestioneUtenzaManager, RecensioneManager)
- src/main/java/it/unisa/educat/control/ (Include classi come LoginServlet, PrenotazioneServlet)
- src/main/webapp/ (Contiene le JSP e risorse web)

3. Class Interfaces

Questa sezione dettaglia le principali classi e interfacce pubbliche, mappando le entità definite nel RAD e i componenti architetturali del SDD.

3.1 Package it.unisa.educat.bean (Entità)

Queste classi rappresentano il modello dei dati e sono prive di logica di business complessa (POJO).

- Class Utente (Abstract)
    - Rappresenta un utente generico della piattaforma
    - Attributi Pubblici:
        - nome: String

        - cognome: String
        - email: String
        - password: String (Hashed)
        - dataNascita: Date
        - indirizzo: String (Composto da Via, Civico, Città, CAP)
    - Sottoclassi:
        - Studente
        - Genitore
        - Tutor
        - Amministratore Utenti
- Class Lezione
    - Rappresenta una lezione offerta o prenotata
    - Attributi Pubblici:
        - idLezione: int
        - materia: String
        - data: Date
        - durata: float
        - prezzo: float
        - modalitaLezione: String (Enum: ONLINE/PRESENZA)
        - tutor: Tutor
        - città: String
- Class Prenotazione
    - Associa uno studente a una lezione
    - Attributi Pubblici:
        - idPrenotazione: int
        - dataPrenotazione: Date
        - stato: Enum (ATTIVA, ANNULLATA, CONCLUSA)

3.2 Package it.unisa.educat.business (Managers)

Queste classi implementano i servizi definiti nel SDD e i metodi logici descritti nel RAD.

- *Class GestioneLezioneManager*
    - Gestisce la ricerca, prenotazione e gestione delle lezioni
    - Dipendenze:
        - GestioneLezioneDAO
    - Operazioni:

Firma del metodo	Descrizione
public List<Lezione> ricercaLezioni(CriteriRicerca criteri)	Cerca lezioni per materia, città, prezzo, ecc.
ublic void prenotaLezione(int lezioneId, int studenteId, DatiPagamento pagamento) throws SlotOccupatoException	estisce l'acquisto. Verifica la disponibilità dello slot e processa il pagamento
public void annullaLezione(int lezioneId, int userId)	Annulla una lezione e innesca il rimborso se entro i termini (1 giorno di anticipo)
ublic void aggiungiLezione(Lezione lezione)	Permette al Tutor di inserire una nuova disponibilità

3.3 Package it.unisa.educat.storage

Classi responsabili dell'esecuzione delle query SQL mappate nel SDD

- *Interface GestioneLezioneDAO*
  - Operazioni:

Firma del metodo
Lezione doRetrieveById(int id)
List<Lezione> doRetrieveByCriteria(CriteriRicerca criteri)
void doSavePrenotazione(Prenotazione prenotazione)
void doUpdateStatus(int idLezione, String stato)

3.4 Package it.unisa.educat.control (Controllers/Servlets)

Gestiscono il flusso globale del software

- *Class PrenotazioneServlet*
  - Extends HttpServlet
  - Operations:

Firma del metodo	Descrizione
doPost(HttpServletRequest request, HttpServletResponse response)	Riceve i dati di pagamento e invoca GestioneLezioneManager.prenotaLezione()

4. Glossario