



Università
Ca' Foscari
Venezia



RELAZIONE progetto Basi di Dati

GESTIONE ESAMI UNIVERSITARI

Brugnolaro Doriana (876421)
Cantoni Letizia (882713)

SOMMARIO:

SOMMARIO:.....	1
1. Introduzione.....	2
2. Funzionalità Principali.....	3
3. Progettazione Concettuale.....	5
4. Progettazione Logica.....	7
5. Query e Trigger principali.....	10
6. Scelte Progettuali.....	15
7. Descrizione pagine applicazione.....	18
8. Contributo.....	23
9. Risorse.....	24

1. Introduzione

In questo progetto abbiamo sviluppato una web application dedicata alla gestione degli esami universitari.

Come prima cosa abbiamo creato uno schema concettuale che successivamente abbiamo trasformato nello schema logico per la base di dati su cui si basa la nostra applicazione.

In merito al Database, ci siamo appoggiate a PostgreSQL.

PostgreSQL è un database a oggetti in cui l'informazione è rappresentata in forma di oggetti, noto anche con l'abbreviazione "Postgres". Si distingue nel settore delle basi di dati per le sue caratteristiche uniche nel suo genere, che lo pongono per alcuni aspetti all'avanguardia nel settore delle basi di dati.

Per quanto riguarda il back-end della nostra applicazione, abbiamo utilizzato PostgreSQL, Flask e SQLAlchemy per gestire in modo ottimale l'intera parte server della nostra applicazione.

Per quanto riguarda il front-end della nostra applicazione, abbiamo fatto affidamento su HTML, CSS e JavaScript. Abbiamo adottato un design minimale ma accattivante, con l'obiettivo di offrire agli utenti la massima semplicità nell'utilizzo della nostra piattaforma.



In questa sezione forniremo una breve panoramica delle funzionalità generali della nostra applicazione. Successivamente, approfondiremo dettagliatamente ciascuna di esse nel capitolo 2.

Gli utenti della nostra applicazione, si suddividono principalmente in due categorie: i professori e gli studenti.

I professori avranno la possibilità di eseguire diverse azioni, tutte finalizzate alla gestione degli esami universitari, ad esempio potranno creare degli esami con le prove che li compongono, registrare le valutazioni delle prove svolte dagli studenti e altro ancora.

Gli studenti avranno accesso alla gestione della loro carriera universitaria, comprese funzioni quali l'iscrizione agli esami, la visualizzazione e l'accettazione delle valutazioni relative a tali esami, e altro ancora.

2. Funzionalità Principali

Di seguito verranno descritte le funzionalità principali che abbiamo implementato nella nostra applicazione.

- **Registrazione e login dell'utente:**

- Un utente non registrato non può accedere all'applicazione, deve quindi creare un nuovo account registrandosi tramite apposito form.
- Una volta creato l'account in cui l'utente specificherà il proprio ruolo (professore o studente) avrà accesso alla propria area riservata che sarà diversa in base al ruolo inserito in fase di registrazione e avrà diverse funzionalità.

- **Creazione di Esami e prove:**

- I docenti possono creare nuovi esami.
- Un esame per esistere dev'essere composto da almeno una prova, che sarà creata subito dopo l'esame.

- **Configurazione delle Prove:**

- I docenti possono configurare le prove, definendo i criteri per la valutazione finale.
- Ogni prova può avere una diversa tipologia (scritto, orale, progetto...) e un diverso tipo di valutazione peso del voto, bonus al voto).

- **Assegnazione Data:**

- Il professore assegna a tutte le prove che ha creato la data che preferisce in cui ogni prova venga sostenuta (ad eccezione di weekend o date già trascorse).

- **Visualizzazione e modifica:**

- Il professore può visualizzare tutte le prove da lui e dagli altri professori create, con i loro dettagli e la data a loro assegnata, potrà inoltre apportare modifiche alle prove da lui create in caso di errore o altro.

- **Prove e Superamento:**

- Ogni prova ha una data di superamento e una data di scadenza (la data di scadenza è calcolata in base alla data di superamento).
- Se una prova viene superata in un appello successivo, invalida la prova precedente.

- **Iscrizione Prove:**

- Ogni studente può iscriversi alle prove create dai professori in uno degli appelli disponibili.

- **Visualizzazione e cancellazione:**

- Lo studente può visualizzare tutte le prove a cui si è iscritto e può decidere di cancellare la sua iscrizione fino a 3 giorni prima.

- **Registrazione dei Voti parziali:**

- Quando è passata la data in cui una prova è stata sostenuta, il professore può inserire le valutazioni di tutti gli studenti iscritti.

- **Registrazione dei Voti totali:**

- Quando tutte le prove sono superate e ancora valide, è possibile registrare il voto dell'esame.

- **Visualizzazione dello Stato:**

- Il sistema fornisce la possibilità di visualizzare lo stato degli studenti, incluse le prove valide superate e lo storico delle prove sostenute.

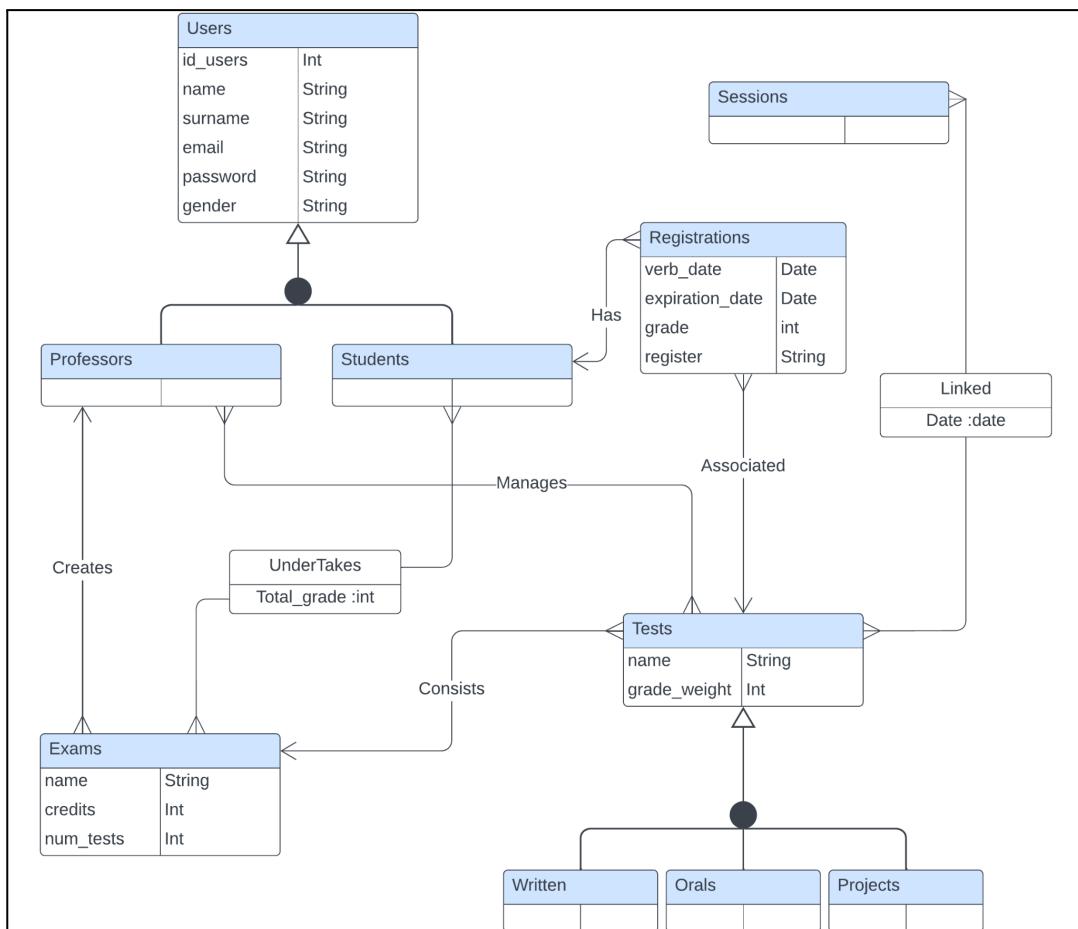
- **Pagina dedicata gestione errori:**

- Al fine di migliorare la trasparenza e la gestione degli errori nel nostro progetto, abbiamo deciso di implementare una pagina dedicata. Questa pagina sarà specificamente progettata per affrontare eventuali errori che potrebbero verificarsi durante l'esecuzione del sistema. È importante sottolineare che la creazione di quest'ultima non riflette una mancanza di considerazione da parte nostra, ma piuttosto un'attenzione riguardo all'utente che andrà ad utilizzare la nostra applicazione perché trovandosi in questa pagina avrà la possibilità di svolgere comunque varie azioni come tornare indietro, eseguire il logout, andare alla homepage, oppure andare nel suo profilo.

3. Progettazione Concettuale

Per convenzione abbiamo creato i nostri schemi (e l'applicazione) in lingua inglese.

Di seguito mostriamo lo schema che abbiamo creato nella **Progettazione Concettuale**:



Lo schema concettuale è composto da un insieme di entità e associazioni.

Di seguito una breve spiegazione delle nostre collezioni.

CLASSI

- **Users**: Per ogni utente, verranno salvati un insieme di dati essenziali, tra cui un codice identificativo, il nominativo completo, l'indirizzo email, la password di accesso, il genere e il ruolo.
L'identificativo univoco sarà utilizzato come chiave primaria per distinguere in modo univoco ciascun utente nel sistema.
L'indirizzo email e la password saranno le credenziali utilizzate durante la procedura di accesso all'applicazione, garantendo l'autenticazione sicura dell'utente.
Infine, il ruolo svolge un ruolo importante nell' identificare se un utente è uno studente o un docente.
- **Exams**: Per ogni esame verrà salvato: il nome dell'esame, il numero dei crediti ad esso assegnati, il numero dei test di cui esso è composto e il voto totale. Ogni esame sarà

identificato da un codice univoco. Il voto totale rappresenterà la media pesata dei voti ottenuti nei singoli test.

- **Tests:** Di ogni test verrà salvato: il nome del test, il suo tipo (scritto-orale-progetto), il bonus e il peso che quel voto avrà sull'esame finale.
- **Registrations:** Di ogni test verrà salvato: il nome del test, la sua obbligatorietà, il suo tipo (scritto-orale-progetto), il bonus e il peso che quel voto avrà sull'esame finale.
- **Sessions:** Di ogni test verrà salvato: il nome del test, la sua obbligatorietà, il suo tipo (scritto-orale-progetto), il bonus e il peso che quel voto avrà sull'esame finale.

SOTTOCLASSI

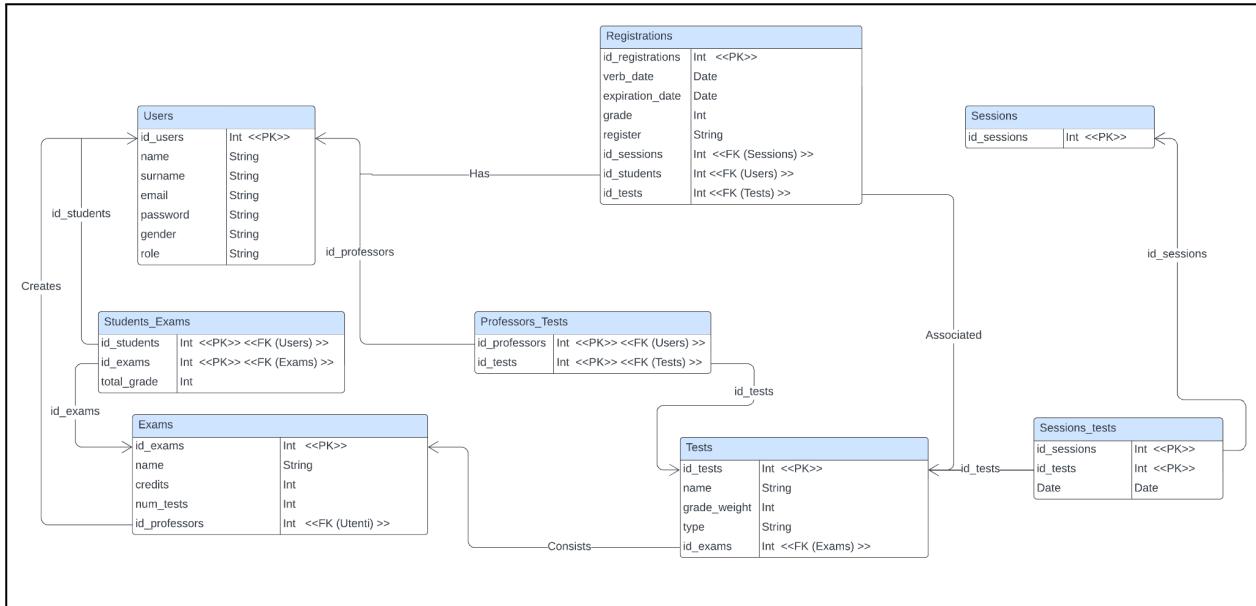
- **Professors** e **Students** sono sottoclassi di **Users**.
- **Written Orals** e **Projects** sono sottoclassi di **Tests**.

ASSOCIAZIONI

Le Associazioni sono: **Creates**,**Consists**,**Manages**,**Has**,**Associated**,**Linked**.

- Professors → **Creates** → Exams (**1:N**)
Un professore crea uno o più esami.
Un esame è creato da un professore.
- Exams → **Consists** → Tests (**1:N**)
Un esame consiste di una o più prove.
Una prova compone un esame.
- Professors → **Manages** → Tests (**N:N**)
Un professore gestisce uno o più esami.
Un esame è gestito da uno o più professori.
- Students → **Has** → Registrations (**1:N**)
Uno studente ha una o più registrazioni.
Una registrazione è di uno studente.
- Students → **UnderTakes** → Exams(**N:N**)
Uno studente svolge uno o più esami.
Un esame è svolto da uno o più studenti.
- Registrations → **Associated** → Tests (**N:1**)
Una prova è associata ad una o più registrazioni.
Una registrazione è associata ad una prova.
- Tests → **Linked** → Sessions (**N:N**)
Una prova è associata(linked) ad una o più sessioni.
Una sessione è associata ad una o più prove.

4. Progettazione Logica



Il nostro modello Concettuale è stato tradotto in Quello Logico utilizzando le regole e i metodi visti durante il corso, di seguito un approfondimento di alcuni punti degni di attenzione.

RAPPRESENTAZIONE GERARCHIE TRA CLASSI

Abbiamo utilizzato il modo della **relazione unica** per rappresentare le sottoclassi: **Professors - Students - Written - Orals e Projects**.

TRASFORMAZIONE DELLE ASSOCIAZIONI N:N

Abbiamo trasformato poi le relazioni **Manages**, **UnderTakes** e **Linked** arrivando ad avere le classi: **Sessions_tests**, **Students_Exams** e **Professors_tests**.

PRIMARY KEYS (PK) e FOREIGN KEYS (FK)

- **Users**: id_users (PK)
- **Exams**: id_exams (PK), id_professors (FK Users)
- **Tests**: id_tests (PK), id_exams (FK Exams)
- **Registrations**: id_registrations (PK), id_sessions (FK Sessions), id_students (FK Users), id_tests (FK Tests)
- **Sessions**: id_sessions (PK)
- **Sessions_tests**: id_sessions (PK) (FK Sessions), id_tests (PK) (FK Tests)

- **Students_exams:** id_students (PK) ([FK Users](#)), id_exams (PK) ([FK Exams](#))
- **Professors_tests:** id_professors (PK) ([FK Users](#)), id_tests (PK) ([FK Tests](#))

CLASSI NELLA NOSTRA APPLICAZIONE

```

class Users(Base):
    __tablename__='users'
    id=Column(Integer, primary_key=True, autoincrement=True)
    name=Column(String, nullable=False)
    surname=Column(String, nullable=False)
    email=Column(String, nullable=False)
    password=Column(String, nullable=False)
    gender=Column(String, nullable=False)
    role=Column(String, nullable=False)

class Exams(Base):
    __tablename__='exams'
    id_exam=Column(Integer, primary_key=True, autoincrement=True)
    name=Column(String, nullable=False)
    credits=Column(Integer, nullable=False)
    num_tests=Column(Integer, nullable=False)
    id_professors= Column(Integer, ForeignKey(Users.id), nullable=False)

class Tests(Base):
    __tablename__='tests'
    id_test=Column(Integer, primary_key=True,autoincrement=True)
    name=Column(String, nullable=False)
    type=Column(String, nullable=False)
    bonus=Column(Integer, nullable=False)
    grade_weight=Column(Integer, nullable=False)
    id_exams=Column(Integer, ForeignKey(Exams.id_exam), nullable=False)

class Sessions(Base):
    __tablename__='sessions'
    id_session=Column(Integer, primary_key=True, autoincrement=True)

class Sessions_tests(Base):
    __tablename__='sessions_tests'
    id_sessions=Column(Integer, ForeignKey('sessions.id_session'), primary_key=True)
    id_tests= Column(Integer, ForeignKey('tests.id_test'), primary_key=True)
    date=Column(DATE)
    __table_args__ = (
        PrimaryKeyConstraint('id_sessions', 'id_tests'),
    )

class Registrations(Base):
    __tablename__='registrations'
    id_registrations=Column(Integer, primary_key=True, autoincrement=True)
    verb_date=Column(DATE)
    expiration_date=Column(DATE)
    grade=Column(Integer)
    register=Column(String)
    id_students=Column(Integer, ForeignKey(Users.id))
    id_tests=Column(Integer, ForeignKey(Tests.id_test))

```

```
    id_sessions=Column(Integer, ForeignKey(Sessions.id_session))

class Professors_tests(Base):
    __tablename__='professors_tests'
    id_professors = Column(Integer, ForeignKey('users.id'), primary_key=True)
    id_tests = Column(Integer, ForeignKey('tests.id_test'), primary_key=True)
    __table_args__ = (
        PrimaryKeyConstraint('id_professors', 'id_tests'),
    )
class Students_exams(Base):
    __tablename__='students_exams'
    id_students = Column(Integer, ForeignKey('users.id'), primary_key=True)
    id_exams = Column(Integer, ForeignKey('exams.id_exam'), primary_key=True)
    total_grade= Column(Integer , nullable=false)
    __table_args__ = (
        PrimaryKeyConstraint('id_students', 'id_exams'),
    )
```

5. Query e Trigger principali

In questa sezione andiamo a mostrare alcune delle query e dei trigger principali che abbiamo implementato nella nostra applicazione, per sintesi non andremo ad inserirli tutti.

QUERY PRINCIPALI

Questa query ci va a restituire tutti gli esami, con i loro dettagli, che un professore (colui che sta utilizzando l'applicazione) ha creato.

```
lists = ( session.query(Exams).filter(Exams.id_professors == current_user.id).all() )
```

Questa query ci va a restituire tutti i dettagli delle prove e dei professori che hanno creato quei test.

```
t= aliased(Tests)
pt=aliased(Professors_tests)
st=aliased(Sessions_tests)
tests=(session.query(t.id_test, t.name, t.type, st.date, pt.id_professors)
       .join(pt, t.id_test== pt.id_tests)
       .join(st,st.id_tests==t.id_test ).all())
```

Questa query ci va a restituire tutti i test che un professore (il professore che sta utilizzando l'applicazione) ha creato e che gli studenti hanno svolto a cui non è ancora stato assegnato un voto.

```
tests = (
    session.query(
        Tests.name.label('Test_Name'),
        Users.name.label('User_Name'),
        Tests.id_test,
        Tests.type,
        Tests.bonus,
        Registrations.verb_date,
        Registrations.id_registrations
    )
    .join(Registrations, Registrations.id_tests == Tests.id_test)
    .join(Users, Registrations.id_students == Users.id)
    .join(Professors_tests, Tests.id_test == Professors_tests.id_tests)
    .filter(Professors_tests.id_professors == current_user.id)
    .filter(Registrations.grade==None ).all()
```

Questa query ci va a restituire tutti gli esami che lo studente che sta usando l'applicazione ha svolto.

```
lists = (
    session.query(
        Exams.id_exam,
```

```

        Exams.name.label('exam_name'),
        Exams.credits.label('exam_credits'),
        Users.id.label('professor_id'),
        Users.name.label('professor_name'),
        Students_exams.total_grade
    )
    .join(Tests, Tests.id_exams == Exams.id_exam)
    .join(Users, Exams.id_professors == Users.id)
    .join(Students_exams, Exams.id_exam == Students_exams.id_exams)
    .filter(Students_exams.total_grade > 17, Students_exams.id_students ==
current_user.id)
    .distinct(Exams.id_exam, Exams.name)
).all()

```

Questa query ci va a restituire la lista delle iscrizioni che uno studente (colui che sta utilizzando l'applicazione) ha effettuato, tutte le informazioni riguardanti i test a cui si è iscritto e la data.

```

subs = (
    session.query(
        Tests.name.label('test_name'),
        Users.name.label('student_name'),
        Tests.id_test,
        Tests.type,
        Registrations.verb_date,
        Registrations.id_registrations
    )
    .join(Registrations, Registrations.id_tests == Tests.id_test)
    .join(Users, Registrations.id_students == Users.id)
    .filter(Users.id == current_user.id).distinct().all()
)

```

Questa query ci va a restituire la lista delle valutazioni positive delle **prove** effettuate dallo studente (che sta utilizzando l'applicazione) con tutti i dettagli inerenti ad esse.

```

lists = (
    session.query(
        Registrations.id_tests,
        Tests.name.label('test_name'),
        Tests.type,
        Tests.bonus,
        Registrations.verb_date,
        Registrations.grade,
        Users.id.label('professor_id'),
        Users.name.label('professor_name'),
    )
    .join(Tests, Registrations.id_tests == Tests.id_test)
    .join(Exams, Tests.id_exams == Exams.id_exam)
    .join(Users, Exams.id_professors == Users.id)
    .filter(Registrations.grade > 17, Registrations.id_students == current_user.id)
)

```

```
.all()  
)
```

Questa query ci va a restituire la lista delle valutazioni positive degli **esami** effettuati dallo studente (che sta utilizzando l'applicazione) con tutti i dettagli inerenti ad essi.

```
lists = (  
    session.query(  
        Exams.id_exam,  
        Exams.name.label('exam_name'),  
        Exams.credits.label('exam_credits'),  
        Users.id.label('professor_id'),  
        Users.name.label('professor_name'),  
        Students_exams.total_grade  
    )  
    .join(Tests, Tests.id_exams == Exams.id_exam)  
    .join(Users, Exams.id_professors == Users.id)  
    .join(Students_exams, Exams.id_exam == Students_exams.id_exams)  
    .filter(Students_exams.total_grade > 17, Students_exams.id_students ==  
current_user.id)  
    .distinct(Exams.id_exam, Exams.name)  
).all()
```

TRIGGER PRINCIPALI

Questo trigger controlla che tutti i voti dei test che compongono un esame siano sufficienti prima di registrare il voto finale nella tabella esami

```
CREATE OR REPLACE FUNCTION check_exam_passed() RETURNS TRIGGER AS $$  
BEGIN  
    IF EXISTS (  
        SELECT registrations.grade  
        FROM registrations  
        JOIN tests ON registrations.id_tests = tests.id_test  
        JOIN exams ON exams.id_exam = tests.id_exams  
        WHERE registrations.grade < 18  
    ) THEN  
        RAISE EXCEPTION 'Impossibile inserire o aggiornare l''esame';  
    END IF;  
    RETURN NEW;  
END;  
$$ LANGUAGE plpgsql;  
  
DROP TRIGGER IF EXISTS final_grade ON students_exams;  
CREATE TRIGGER final_grade  
BEFORE INSERT OR UPDATE ON exams
```

```

FOR EACH ROW
EXECUTE FUNCTION check_exam_passed();

```

Questo trigger controlla che i test che compongono un esame non siano scaduti quando andiamo ad inserire il voto totale nella tabella esami.

```

CREATE OR REPLACE FUNCTION check_date_expiration() RETURNS TRIGGER AS $$

BEGIN
    IF EXISTS(
        SELECT registrations.expiration_date
        FROM registrations
        JOIN tests ON registrations.id_tests = tests.id_test
        JOIN exams ON exams.id_exam = tests.id_exams
        WHERE registrations.expiration_date < CURRENT_DATE
    )THEN
        RAISE NOTICE 'La prova è scaduta, non può quindi essere utilizzata per il voto totale';
    END IF;
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS expiration_date_trigger ON students_exams;
CREATE TRIGGER expiration_date_trigger
BEFORE INSERT OR UPDATE
ON exams
FOR EACH ROW
EXECUTE FUNCTION check_date_expiration();

```

Questo trigger controlla che la somma dei pesi (grade weight) che ciascun test possiede sulla media del voto finale sia uguale a 100.

```

CREATE OR REPLACE FUNCTION grade_weight() RETURNS TRIGGER AS $$

DECLARE
numtestrequired INTEGER;
total INTEGER;
numtest INTEGER;
BEGIN
    SELECT exams.num_tests
    INTO numtestrequired
    FROM exams
    WHERE exams.id_exam= NEW.id_exam;

    SELECT COUNT (*)
    INTO numtest

```

```

FROM tests
WHERE id_exams=NEW.id_exams;

IF numtestrequired=numtest THEN
  SELECT SUM(grade_weight)
  INTO total
  FROM tests
  WHERE id_exams=NEW.id_exams;
  IF total != 100
    THEN RAISE EXCEPTION 'Il totale del peso dei voti non corrisponde a 100';
    END IF;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS control_grade_weight ON tests;
CREATE TRIGGER control_grade_weight
AFTER INSERT OR UPDATE ON tests
FOR EACH ROW
EXECUTE FUNCTION grade_weight();

```

Questo trigger controlla che non vengano creati due esami con lo stesso nome

```

CREATE OR REPLACE FUNCTION double_exam() RETURNS TRIGGER AS $$%
BEGIN
  IF EXISTS (SELECT * FROM exams WHERE name=NEW.name)
    THEN RAISE NOTICE 'impossibile creare un altro esame con lo stesso nome';
    END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER no_double_exam
BEFORE INSERT OR UPDATE ON exams
FOR EACH ROW
EXECUTE FUNCTION double_exam();

```

Questo trigger serve per l'invalidazione di una possibile precedente iscrizione ad un test, quando un utente si iscrive nuovamente.

```

CREATE FUNCTION invalidation_exam() RETURNS TRIGGER AS $$%
BEGIN
  IF EXISTS (

```

```

SELECT r.id_registrations
  FROM registrations r J
 WHERE r.id_students=NEW.id_students AND r.id_tests=NEW.id_tests AND r.verb_date<
NEW.verb_date)
 THEN
  DELETE FROM registrations
 WHERE id_students=NEW.id_students AND id_tests=NEW.id_tests AND verb_date<
NEW.verb_date;
END IF;
RETURN NEW;
END;
$$LANGUAGE plpgsql;

CREATE TRIGGER invalidation
AFTER INSERT OR UPDATE ON registrations
FOR EACH ROW
EXECUTE FUNCTION invalidation_exam();

```

6. Scelte Progettuali

SQLAlchemy

Per creare le nostre query abbiamo utilizzato SQLAlchemy, una potente libreria ORM (Object-Relational Mapping).

I Vantaggi delle query SQLAlchemy:

- Astrazione**

SQLAlchemy offre un'interfaccia Python di alto livello per interagire con i database, astraendo la complessità SQL sottostante. Questa astrazione semplifica l'interazione del database, riducendo la necessità di codice specifico del database.

- Leggibilità e manutenibilità**

Il codice SQLAlchemy tende ad essere più leggibile e più facile da mantenere rispetto alle query SQL grezze. Sfrutta classi e oggetti Python per rappresentare tabelle e righe di database, rendendo il codice intuitivo e autoesplicativo.

- Portabilità**

Con SQLAlchemy, puoi passare senza problemi tra vari motori di database (ad esempio, SQLite, PostgreSQL, MySQL) semplicemente modificando un'impostazione di configurazione. Questa portabilità può far risparmiare notevoli sforzi di sviluppo.

- Sicurezza (Protezione dalle SQL-injection)**

SQLAlchemy fornisce una difesa integrata contro gli attacchi di SQL injection. L'input dell'utente viene automaticamente sanificato proteggendo il sistema.

WTForms

Nella realizzazione del nostro progetto, per quanto riguarda la gestione dei form abbiamo utilizzato i **WTForms**;

WTForms è una libreria flessibile di convalida e rendering di moduli per Python sviluppo web. Uno dei principali vantaggi dell'utilizzo di WTForms è la:

- **Sicurezza (Protezione da Cross-Site Request Forgery (CSRF))**

Può funzionare con qualsiasi framework e modello web motore che scegli. Supporta la convalida dei dati, la protezione CSRF, internazionalizzazione (I18N) e altro ancora.

Di seguito mostriamo un esempio dell'utilizzo dei WTForms nella nostra applicazione.

Esempio:

```
class RegistrationForm(Form):
    firstName = StringField('First Name', [validators.Length(min=1, max=50)])
    lastName = StringField('Last Name', [validators.Length(min=1, max=50)])
    email = StringField('Email Address', [validators.Email(message="This field is required")])
    password = PasswordField('Password', [validators.Length(min=6, max=50)])
    role = SelectField('Role', choices=[('professor', 'Professor'), ('student', 'Student')])
    gender = RadioField('Gender', choices=[('female', 'Female'), ('male', 'Male'), ('other', 'Other')])
    submit = SubmitField('Create Account')
```

Questo esempio mostra il form utilizzato per la registrazione (il "signup")(file **forms.py**).

```
<form class="mainForm" action="/signup" method="POST">
    {{ form.csrf_token }}
    {{ form.firstName.label }} {{ form.firstName }}
    {{ form.lastName.label }} {{ form.lastName }}
    {{ form.email.label }} {{ form.email }}
    {{ form.password.label }} {{ form.password }}
    {{ form.role.label }} {{ form.role }}
    {{ form.gender.label }} {{ form.gender }}
    {% for field, errors in form.errors.items() %}
        {% for error in errors %}
            <div class="alert alert-danger">
                {{ error }}
            </div>
        {% endfor %}
    {% endfor %}
    {{ form.submit (class="btn btn-outline-primary")}}
</form>
```

Questo sarà il nostro codice dentro il file **signup.html**, {{ form.csrf_token }} questo include un CSRF token per proteggerci dai Cross-Site Request Forgery (CSRF)(file **signup.html**).

RUOLI

Nel nostro progetto abbiamo poi deciso di implementare i ruoli in PostgreSQL, quelli che abbiamo implementato sono i seguenti: admin, professor, student.

PostgreSQL gestisce i permessi di accesso al database utilizzando il concetto di ruoli .

Un ruolo può essere considerato come un utente del database o un gruppo di utenti del database, a seconda di come è impostato il ruolo. I ruoli possono possedere oggetti di database (ad esempio tabelle e funzioni) e possono assegnare privilegi su tali oggetti ad altri ruoli per controllare chi ha accesso a quali oggetti. Inoltre, è possibile concedere l'appartenenza ad un ruolo ad un altro ruolo, consentendo così al ruolo membro di utilizzare i privilegi assegnati ad un altro ruolo.

Esempio:

```
DROP USER IF EXISTS admin;
DROP USER IF EXISTS professor;
DROP USER IF EXISTS student;

CREATE USER admin WITH SUPERUSER CREATE ROLE PASSWORD 'admin';
CREATE USER professor WITH PASSWORD 'professor';
CREATE USER student WITH PASSWORD 'student';

GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE users, exams, tests, registrations,
sessions_tests, sessions, professors_test TO admin;
GRANT SELECT, INSERT, UPDATE, DELETE ON TABLE exams, tests, registrations,
sessions_tests, sessions, professors_test TO professor;
GRANT SELECT ON TABLE exams, tests, registrations, sessions_tests, sessions,
professors_test TO student;
```

(file **roles.sql**)

FLASK LOGIN

Nel nostro progetto abbiamo utilizzato Flask-Login. Esso fornisce la gestione delle sessioni utente per Flask. Si occupa delle attività di accesso, disconnessione e memorizzazione delle sessioni degli utenti per periodi di tempo prolungati.

Flask-Login:

- Memorizza l'ID dell'utente attivo nella sessione di Flask e consente di effettuare facilmente il log-in e il log-out.
- Consente di limitare le visualizzazioni agli utenti connessi (o disconnessi). (`login_required`)
- Aiuta a proteggere le sessioni degli utenti dal furto dei cookie.

Sempre per quanto riguarda l'autenticazione, per conservare la password nel database in modo sicuro e non in chiaro abbiamo utilizzato la libreria **flask_bcrypt** che fornisce utilità di hashing bcrypt per l'applicazione.

Abbiamo utilizzato **Flask Blueprints** per poter suddividere il progetto in pagine diverse in modo da facilitarne la lettura.

7. Descrizione pagine applicazione

In questa sezione andremo a dare una spiegazione più dettagliata delle pagine e del funzionamento della nostra applicazione.

Aprendo la nostra applicazione si viene reindirizzati alla **homepage** dove viene descritta la nostra università e dove si possono trovare il login e il signup.

Schiacciando su **Signup** ci si può registrare e successivamente essere indirizzati al login.

Con il **Login** si può accedere al menù principale dei professori, se chi ha effettuato l'accesso è un professore, altrimenti al menù degli studenti se chi ha effettuato l'accesso è uno studente.

→ **Menù Professori:** In altro troveremo un menù con "Home","Profile","Logout"

Scrollando in basso troveremo le seguenti Pagine:

- **Exams Creation:** in questa pagina il professore potrà creare gli esami che vuole e le prove di cui sono composti.
- **Tests dates assignment:** in questa pagina il professore può decidere in quali date si terranno le prove create da lui in precedenza.
- **Tests Created:** in questa pagina il professore può visualizzare e anche modificare la lista delle prove che ha creato e la data che gli ha assegnato.
- **Tests grade registration:** in questa pagina il professore avrà la lista di studenti che hanno partecipato ad una sua prova e sarà in grado di inserire la valutazione che gli ha assegnato.
- **Students status records:** in questa pagina il professore è in grado di visualizzare l'elenco degli studenti con le prove da loro effettuate, anche se il voto è insufficiente,basta che la loro validità sia ancora attiva.
- **Tests grade registration:** in questa pagina il professore è in grado di visualizzare l'elenco degli studenti che sono in grado di avere il voto dell'esame registrato e registrarlo.
- **Account:** in questa pagina il professore può accedere al suo profilo.
- **Help:** in questa pagina vengono spiegate tutte le pagine del menù e determinate regole a cui deve attenersi il professore.

→ **Menù Studenti:** In altro troveremo un menù con "Home","Profile","Logout"

Scrollando in basso troveremo le seguenti Pagine:

- **Registration for Tests:** in questa pagina uno studente può iscriversi ad una o più prove.

- **Reservations made:** in questa pagina uno studente può visualizzare l'elenco delle prove a cui si è iscritto e può decidere se annullare la sua registrazione se la data della prova è almeno a tre giorni di distanza.
- **Results board:** in questa pagina uno studente può visualizzare le prove che ha svolto con esito positivo.
- **Career:** in questa pagina uno studente può visualizzare tutti gli esami che ha passato nella sua carriera ed i loro dettagli.

A titolo esemplificativo di seguito mostriamo solo qualche schermata del nostro progetto.
La nostra applicazione verrà mostrata nella sua interezza nel video dimostrativo consegnato.

Homepage, la schermata iniziale

Università Astra

Formiamo le menti di domani

Riguardo la nostra università:

L'Università Astra è un'istituzione d'avanguardia che ridefinisce l'educazione superiore nel XXI secolo. Situata in un campus moderno e accogliente, questa università si distingue per l'approccio all'avanguardia alla formazione e per l'attenzione costante all'evoluzione delle esigenze globali e alle sfide emergenti.

Missioni e Valori: La nostra missione è quella di fornire un'educazione di alta qualità che prepari gli studenti a eccellere in un mondo in rapida trasformazione. I nostri valori fondamentali includono l'innovazione, l'accessibilità, l'integrità e la collaborazione.

Programmi Accademici Avanzati: Offriamo una vasta gamma di programmi accademici che spaziano dalle



Programmi Accademici Avanzati: Offriamo una vasta gamma di programmi accademici che spaziano dalle scienze umane alle tecnologie emergenti, garantendo che gli studenti abbiano accesso a un'istruzione di livello mondiale in settori chiave. I nostri docenti sono esperti nel loro campo e sono impegnati a guidare gli studenti verso l'eccellenza.

Tecnologie all'Avanguardia: L'Università Innovativa del Futuro integra tecnologie all'avanguardia in ogni aspetto dell'esperienza educativa. Dalle aule intelligenti ai laboratori virtuali, forniamo agli studenti gli strumenti necessari per affrontare le sfide del futuro.

L'Università Innovativa del Futuro è un faro di eccellenza nell'istruzione superiore, preparando gli studenti a guidare il futuro con intelligenza, creatività e responsabilità. Se sei pronto a fare parte di una comunità che abbraccia l'innovazione e l'evoluzione, l'Università



Signup, la pagina di registrazione

Sign Up

First Name

Last Name

Email Address

Password

Role

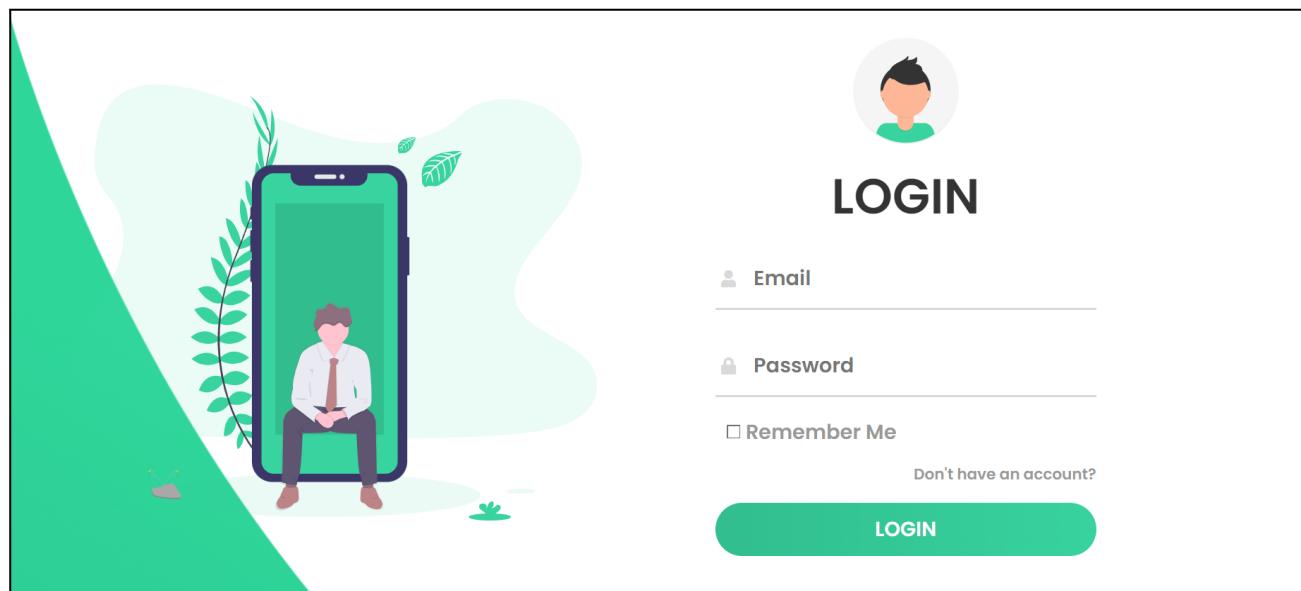
Gender

Female

Male

Other

Login, la pagina di accesso



Menù studenti, la pagina principale per gli studenti

The image shows the main menu page for students. At the top, there is a green header bar with the text "Università Astra" on the left and "Home" "Profile" "Logout" on the right. Below the header, there is a large image of a hand typing on a laptop keyboard. To the left of the keyboard, the text "Welcome bilico" is displayed in a large, bold, black font. The rest of the page is a plain white space.

The image shows a page with four service options for students. Each option is contained within a white box with rounded corners. From left to right: 1. "Registration for Tests": Text says "See the list of the available sessions and Subscribe." 2. "Reservations made": Text says "View the list of reservations made." 3. "Results board": Text says "View all the tests you passed with their details." 4. "Career": Text says "See all the exams you passed in your career with their details." At the bottom right, there is a footer with the text "Università Astra", "A100 Adam Street", "New York, NY 535022", and "Phone: +39 358 461 5399".

Menù professori, la pagina principale per i professori

The screenshot shows the main menu for professors. At the top, there's a green header bar with the text "Università Astra" on the left and "Home Profile Logout" on the right. Below the header is a decorative background image featuring a green plant on the left and a laptop keyboard with a hand on the right. A large, bold "Welcome simone" message is centered at the top of the main content area. The main content is organized into two rows of four boxes each. The first row contains:

- Exams Creation**: Create the exam and the tests it consists of.
- Tests dates assignment**: Assign a session date to the tests created.
- Tests Created**: Here you can see the list of tests you created and their associated dates.
- Tests grades registration**: View the list of tests taken by students and enter ratings.

The second row contains:

- Students status record**: View the student's test and exam record.
- Exams grades registration**: Enter the final grade of the exams taken by the students.
- Account**: Here you can access your profile.
- Help**: Here is an explanation of the pages in this web application.

Error, pagina di gestione errori

The screenshot shows an error page. At the top, there's a dark header bar with the text "Università Astra" on the left and "Home Logout simone" on the right. The main content features a light gray card with a rounded rectangle containing the word "Oops!" in a large, bold, black font. Below "Oops!" is a explanatory text: "If you are seeing this page, it's because you tried to insert some incorrect data, or the page you are trying to access is empty." At the bottom of the card is the text "Please go back and try again." and a blue "Go Back" button. The background of the page has abstract shapes in green, blue, and yellow.

8. Contributo

Per la realizzazione di questo progetto il lavoro è stato suddiviso equamente, principalmente **Brugnolaro Doriana** si è occupata della parte che riguarda lo **studente**, mentre **Cantoni Letizia** di quella che riguarda il **professore**, anche se in entrambi i casi tutte e due abbiamo comunque lavorato su tutto il progetto nella sua interezza.

9. Risorse

Per la creazione del nostro progetto abbiamo utilizzato le lezioni frontali ed i lucidi messi a disposizione dal professore.

Altre risorse che abbiamo utilizzato:

- Real Python
- Flask-SQLAlchemy
- Flask SQLAlchemy (with Examples) - Python Tutorial
- Flask-Login
- BBBootstrap
- Stack Overflow
- Lucid.app

Università Astra

