Gestione Progetti: Un'Applicazione Robusta e Testata

Elena Zerbin

17 luglio 2024

1. Introduzione

Benvenuti alla presentazione del nostro sistema di Gestione Progetti.

2. Architettura del Database

Diagramma ER

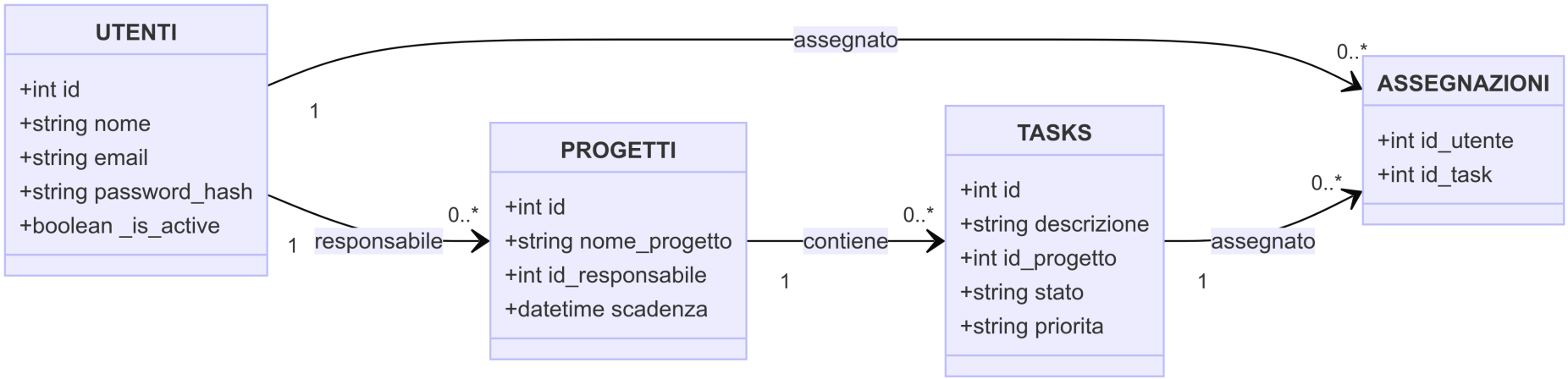


Figura: Diagramma ER del nostro database

3. Funzionalità Principali

1. Registrazione e autenticazione utenti
2. Creazione e gestione di progetti
3. Aggiunta e gestione di task
4. Assegnazione di task agli utenti
5. Monitoraggio dello stato e della priorità
6. Gestione delle scadenze

4. Tecnologie Utilizzate

- Backend: Python con Flask
- Database: MySQL
- Testing: pytest, Hypothesis

Utilizzo Hypothesis per il property-based testing, che ci permette di generare automaticamente una vasta gamma di input di test.

```python
# Strategies for generating test data
name_strategy = st.text(min_size=1, max_size=50).filter(lambda x: x.strip() != "")
email_strategy = st.emails()
password_strategy = st.text(min_size=8, max_size=20)
project_name_strategy = st.text(min_size=1, max_size=50).filter(lambda x: x.strip() != "")
project_description_strategy = st.text(min_size=1, max_size=200)
date_strategy = st.dates(min_value=datetime.now().date() + timedelta(days=1), max_value=datetime.now().date() + timedelta(days=365))
task_description_strategy = st.text(min_size=1, max_size=100).filter(lambda x: x.strip() != "")
task_status_strategy = st.sampled_from(['Todo', 'In Progress', 'Done'])
task_priority_strategy = st.sampled_from(['Low', 'Medium', 'High'])
```

Figura: Strategie per la generazione di dati di prova

Questi test ci permettono di:

- Verificare la robustezza delle nostre funzioni con input diversificati
- Scoprire edge case che potrebbero sfuggire ai test tradizionali
- Assicurare che le nostre funzioni rispettino determinate proprietà invarianti

Questi test coprono l'intero flusso dell'applicazione, dalla registrazione dell'utente alla creazione e gestione di progetti e task.

- Tutte le funzionalità del test parziale
- Test di Scenari di Errore Specifici
- Test di Funzionalità Aggiuntive (Modifica Progetti e Tasks)
- Test per API o Endpoints Aggiuntivi
- Test di Eliminazione di Progetti e Tasks Non Esistenti

Note:

- Copertura più completa delle funzionalità e degli scenari di errore.
- Include test specifici per API e gestione di errori.

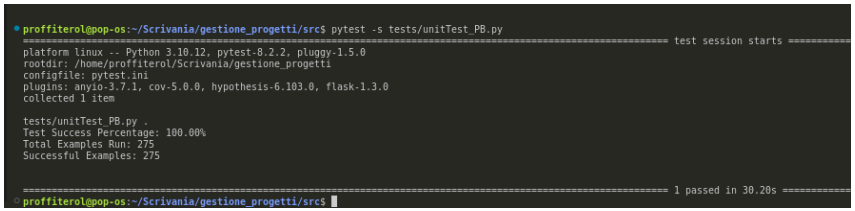
Questi test si concentrano su specifiche funzionalità o scenari, come:

- **Registrazione, Login e Logout:**
  - Registrazione di un nuovo utente.
  - Login con le credenziali dell'utente registrato.
  - Logout dell'utente.
- **CRUD Progetto:**
  - Creazione di un progetto con validazione del nome del progetto.
  - Creazione di un progetto valido.
  - Verifica della creazione del progetto nel database.
  - Eliminazione di un progetto.
  - Verifica dell'eliminazione del progetto nel database.

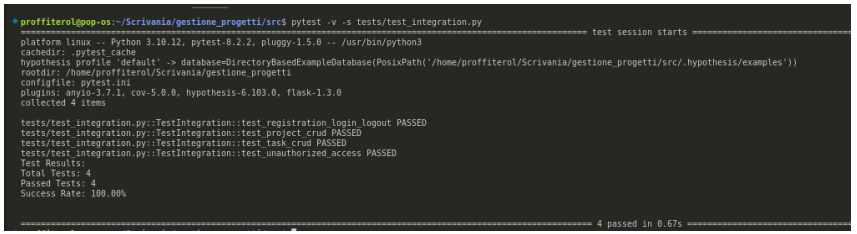
- **CRUD Task:**
  - Creazione di un progetto per associarlo ai task.
  - Creazione di un task valido associato al progetto.
  - Verifica della creazione del task nel database.
  - Eliminazione di un task.
  - Verifica dell'eliminazione del task nel database.
- **Accesso Non Autorizzato:**
  - Tentativo di accesso alla dashboard senza autenticazione.
  - Tentativo di aggiungere un progetto senza autenticazione.
  - Tentativo di aggiungere un task senza autenticazione.

5. Test e Qualità del Codice

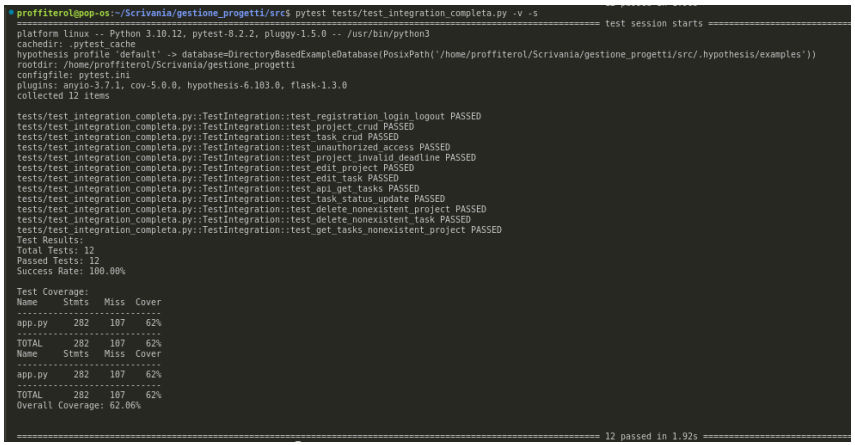
Risultati dei Test



(a) Property based Testing



(b) test integration parziale



(c) test integration completa

Utilizziamo il modulo coverage di Python per misurare la copertura dei nostri test.
Processo:

- Esecuzione dei test con copertura attivata
- Generazione di un report dettagliato
- Analisi delle aree del codice non coperte dai test

```
Test Results:
Total Tests: 12
Passed Tests: 12
Success Rate: 100.00%

Test Coverage:
```

Name	Stmts	Miss	Cover
app	282	107	62%
TOTAL	282	107	62%

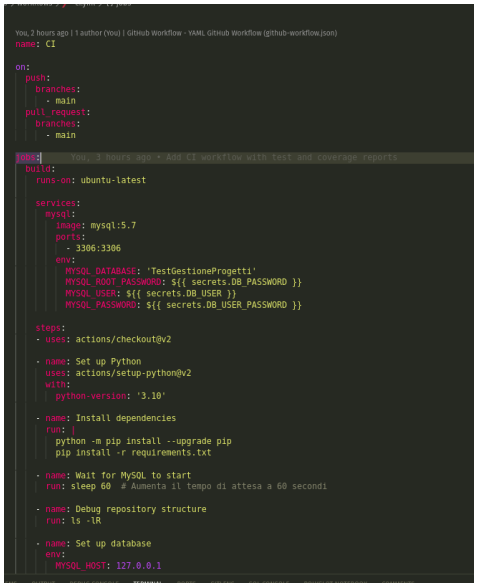
```
Overall Coverage 62.06%
```

6. Continuous Integration e Continuous Delivery

Pipeline CI/CD

- La pipeline CI/CD automatizza il processo di build, test e deploy del codice.
- Utilizziamo GitHub Actions per implementare la pipeline.

Configurazione della Pipeline:



(a) Config. della Pipeline parte 1

(b) Config. della Pipeline parte 2

- La pipeline CI/CD automatizza il processo di build, test e deploy del codice.
- Utilizziamo GitHub Actions per implementare la pipeline e Docker per il deployment.

Configurazione della Pipeline:

```
- name: Log in to Docker Hub
  uses: docker/login-action@v2
  with:
    username: ${{ secrets.DOCKER_USERNAME }}
    password: ${{ secrets.DOCKER_PASSWORD }}

- name: Build and push Docker image
  run: |
    IMAGE_NAME=${{ secrets.DOCKER_USERNAME }}/gestione_progetti
    IMAGE_TAG=${GITHUB_SHA::7}
    docker build -t $IMAGE_NAME:$IMAGE_TAG .
    docker push $IMAGE_NAME:$IMAGE_TAG
    docker tag $IMAGE_NAME:$IMAGE_TAG $IMAGE_NAME:latest
    docker push $IMAGE_NAME:latest
```

7. Deployment in Container

Dockerfile

- Utilizziamo Docker per creare un'immagine containerizzata dell'applicazione.
- Ecco il Dockerfile usato per costruire l'immagine.

Dockerfile:

```
# Usa un'immagine base Python
FROM python:3.10-slim

# Imposta la directory di lavoro nel container
WORKDIR /app

# Copia i file requirements.txt e installa le dipendenze
COPY requirements.txt requirements.txt
RUN pip install --no-cache-dir -r requirements.txt

# Copia il resto del codice dell'applicazione nel container
COPY . .

# Espone la porta su cui gira l'applicazione
EXPOSE 5000

# Comando per eseguire l'applicazione
CMD ["python", "src/app.py"]
```

Grazie!

Grazie per l'attenzione