



01/06/2025

Anno Accademico 2024/2025

# BOSTARTER

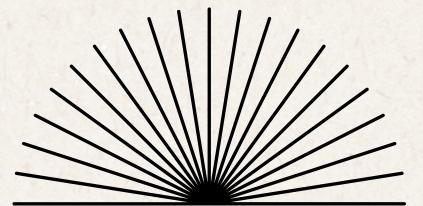
**IMPLEMENTAZIONE DI UN WIS CON STACK AMP**

**PROGETTO:**

Progetto del corso di Basi di Dati

**AUTORI:**

Francesco Maria Fuligni  
Roberto Zanolli



# Indice

03	<b>Specifiche</b>
06	<b>Progettazione della base di dati</b>
08	<b>Implementazione della base di dati</b>
13	<b>Implementazione del WIS</b>
19	<b>DEMO</b>

- 01** Ogni utente di BOSTARTER è identificato da un'email univoca, un nickname e una password. A questi campi si aggiungono nome, cognome, anno e luogo di nascita. Gli utenti possono anche inserire le proprie skill di curriculum, indicate come coppie <competenza, livello>. La lista delle competenze è comune a tutti e definita dagli amministratori.
- 02** La piattaforma prevede utenti “standard”, “creatori” e “amministratori”. Gli amministratori possiedono un codice di sicurezza e il privilegio di gestire la lista di competenze. Gli utenti creatori possiedono, oltre ai campi base, un contatore del numero di progetti (**ridondanza concettuale**) e un punteggio di affidabilità.
- 03** Le competenze sono un insieme di stringhe condiviso, gestito dagli amministratori. Ogni utente dichiara quali competenze possiede e a quale livello (0-5). Questo sistema è utile – nei progetti software – a verificare l'idoneità di un utente a presentare una candidatura per un profilo.

# Specifiche

Requisiti

**04** Solo i creatori possono inserire un progetto, che possiede nome univoco, descrizione, data di inserimento, foto, budget da raggiungere, data limite e stato (aperto/chiuso). Il progetto resta aperto finché i finanziamenti non superano il budget o finché non si supera la data limite; in entrambi i casi diventa automaticamente chiuso e non accetta più finanziamenti.

**05** I progetti hardware includono, oltre ai campi comuni, una lista di componenti con nome univoco, descrizione, prezzo e quantità. I progetti software elencano i profili richiesti, ciascuno caratterizzato da skill <competenza, livello>.

**06** Ogni progetto offre una serie di reward (codice univoco, breve descrizione, foto). Un utente può finanziare più volte lo stesso progetto, fornendo importo e data. Una volta che la somma dei finanziamenti supera il budget o si raggiunge la data limite, lo stato passa a “chiuso” e non sono più ammessi nuovi finanziamenti. Ad ogni finanziamento è associata una reward.

## Specifiche

Requisiti

**07** Gli utenti possono commentare un progetto; un commento è composto da un id univoco, una data e un testo. Il creatore del progetto può rispondere a ogni commento sul proprio progetto, inserendo al massimo una risposta per commento.

**08** Un utente può candidarsi per uno o più profili di un progetto software solo se possiede per ogni skill richiesta un livello pari o superiore a quello indicato. Le candidature valide vengono presentate al creatore, che può accettare o rifiutare ciascun candidato.

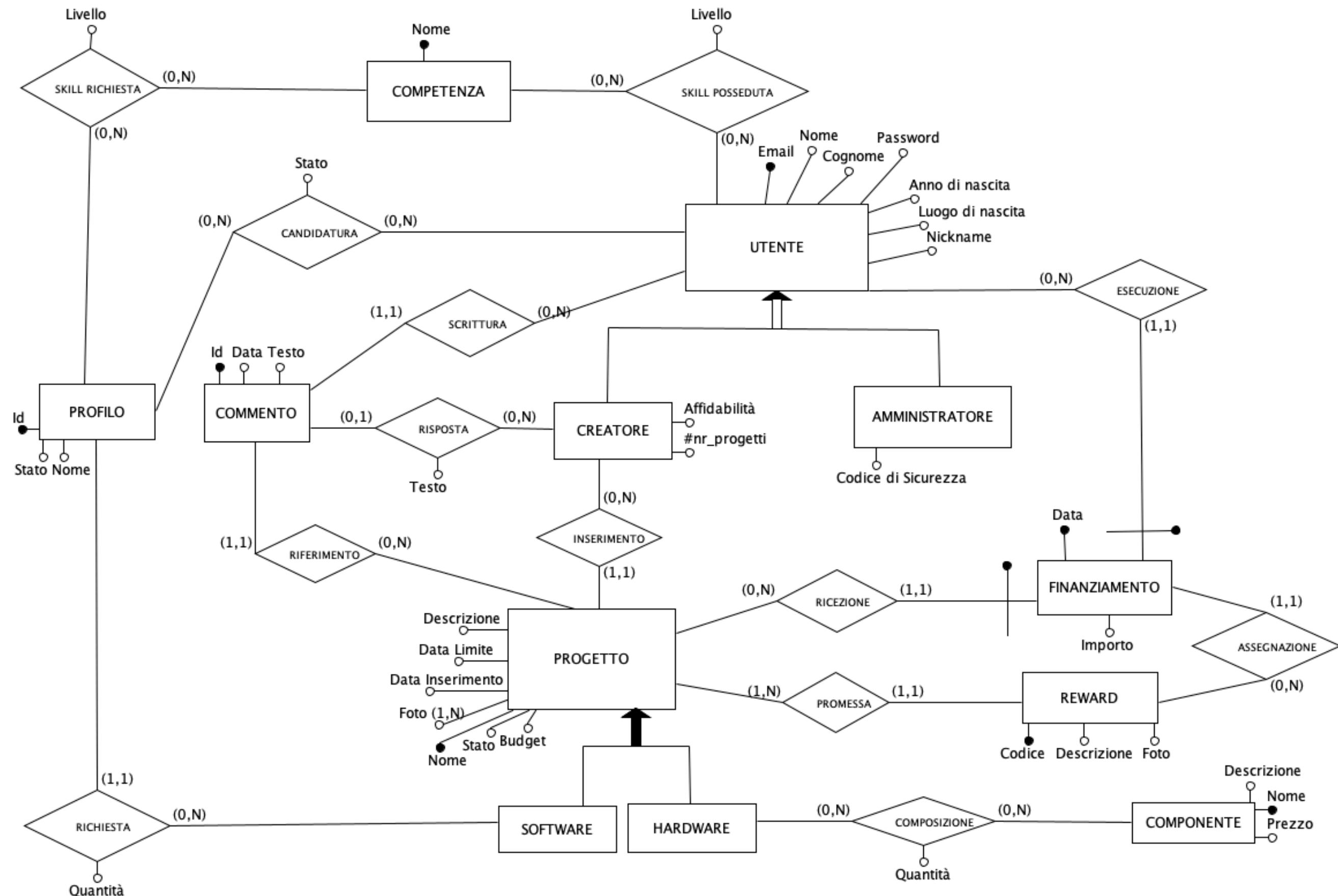
**09** Per tracciare tutte le operazioni di inserimento nella base di dati viene utilizzato un log in una collezione MongoDB. Ogni evento di inserimento è registrato come messaggio di testo dal logger.

# Specifiche

Requisiti

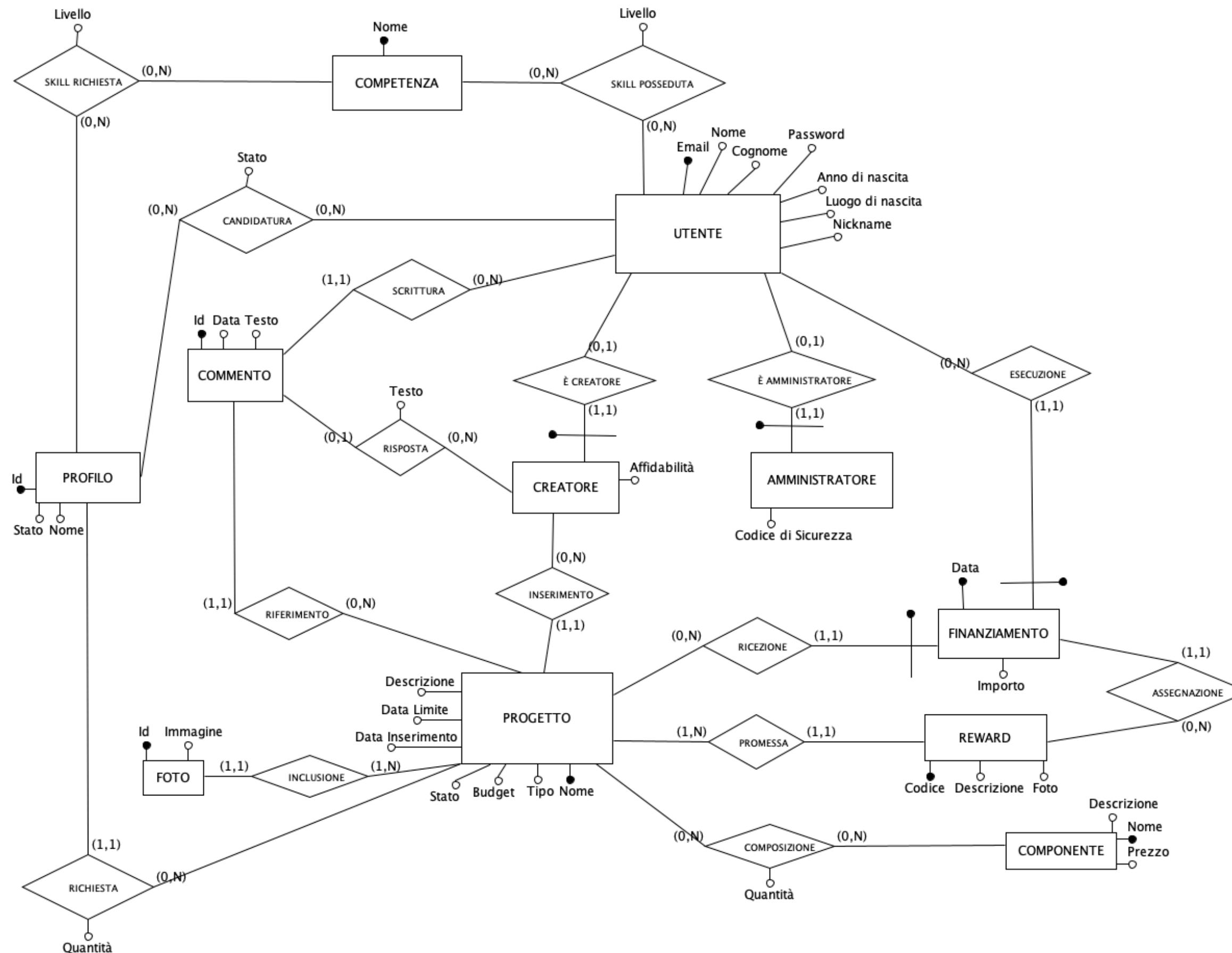
# Progettazione della base di dati

Diagramma  
ER



# Progettazione della base di dati

Diagramma ER  
ristrutturato



# Implementazione della base di dati

Esempio di DDL

```
CREATE TABLE IF NOT EXISTS PROGETTO (
    nome VARCHAR(32) PRIMARY KEY,
    descrizione VARCHAR(255) NOT NULL,
    budget DECIMAL(16,2) NOT NULL,
    data_inserimento DATE NOT NULL,
    data_limite DATE NOT NULL,
    stato ENUM ('APERTO', 'CHIUSO') DEFAULT 'APERTO',
    tipo ENUM ('SOFTWARE', 'HARDWARE') NOT NULL,
    email_utente_creatore VARCHAR(32) NOT NULL REFERENCES UTENTE_CREATORE(email_utente),
    somma_raccolta DECIMAL(16,2) DEFAULT 0
) ENGINE = 'InnoDB';
```

# Implementazione della base di dati

## Esempi di Stored Procedure

```
-- Procedura per verificare se un utente è creatore
DROP PROCEDURE IF EXISTS verifica_creatore;

DELIMITER //
CREATE PROCEDURE verifica_creatore(
IN in_email VARCHAR(32),
OUT esito BOOLEAN
)
BEGIN
SELECT EXISTS(SELECT 1 FROM UTENTE_CREATORE WHERE email_utente = in_email) INTO esito;
END //
DELIMITER ;

-- Procedura per l'inserimento di un nuovo progetto (solo creatori)
DROP PROCEDURE IF EXISTS crea_progetto;

DELIMITER //
CREATE PROCEDURE crea_progetto(
IN in_nome VARCHAR(32),
IN in_descrizione VARCHAR(255),
IN in_budget DECIMAL(16,2),
IN in_data_limite DATE,
IN in_tipo ENUM ('SOFTWARE', 'HARDWARE'),
IN in_email_creatore VARCHAR(32),
OUT esito BOOLEAN
)
BEGIN
CALL verifica_creatore(in_email_creatore, esito);
IF esito THEN
IF CURDATE() > in_data_limite THEN
INSERT INTO PROGETTO (nome, descrizione, budget, data_inserimento, data_limite, stato, tipo, email_utente_creatore)
VALUES (in_nome, in_descrizione, in_budget, CURDATE(), in_data_limite, 'CHIUSO', in_tipo, in_email_creatore);
ELSE
INSERT INTO PROGETTO (nome, descrizione, budget, data_inserimento, data_limite, stato, tipo, email_utente_creatore)
VALUES (in_nome, in_descrizione, in_budget, CURDATE(), in_data_limite, 'APERTO', in_tipo, in_email_creatore);
END IF;
END IF;
END //
DELIMITER ;
```

# Implementazione della base di dati

## Esempio di Trigger

```
-- Trigger per cambiare lo stato di un progetto al raggiungimento del budget
DROP TRIGGER IF EXISTS aggiorna_stato_progetto;

DELIMITER //
CREATE TRIGGER aggiorna_stato_progetto
AFTER INSERT ON FINANZIAMENTO
FOR EACH ROW
BEGIN
DECLARE budget_progetto DECIMAL(16,2);
DECLARE totale_finanziamenti DECIMAL(16,2);
SELECT budget INTO budget_progetto
FROM PROGETTO
WHERE nome = NEW.nome_progetto;
SELECT SUM(importo) INTO totale_finanziamenti
FROM FINANZIAMENTO
WHERE nome_progetto = NEW.nome_progetto;
IF totale_finanziamenti >= budget_progetto THEN
UPDATE PROGETTO
SET stato = 'CHIUSO'
WHERE nome = NEW.nome_progetto;
END IF;
END //
DELIMITER ;
```

# Implementazione della base di dati

Esempio di Evento

```
-- Evento per chiudere i progetti scaduti (eseguito ogni giorno)
DROP EVENT IF EXISTS chiudi_progetti_scaduti;

DELIMITER //
CREATE EVENT chiudi_progetti_scaduti
ON SCHEDULE EVERY 1 DAY
STARTS CURRENT_TIMESTAMP
DO
BEGIN
UPDATE PROGETTO
SET stato = 'CHIUSO'
WHERE stato = 'APERTO'
AND data_limite < CURDATE();
END //
DELIMITER ;
```

# Implementazione della base di dati

Esempio di Vista

```
-- Vista per tutti i progetti con la prima foto associata
DROP VIEW IF EXISTS progetti_con_foto;

CREATE VIEW progetti_con_foto AS
WITH prima_foto AS (
    SELECT f.immagine, f.nome_progetto
    FROM FOTO f
    WHERE f.id = (
        SELECT MIN(f2.id)
        FROM FOTO f2
        WHERE f2.nome_progetto = f.nome_progetto
    )
)
SELECT
    p.*,
    pf.immagine
    FROM PROGETTO p
    LEFT JOIN prima_foto pf
    ON p.nome = pf.nome_progetto;
```

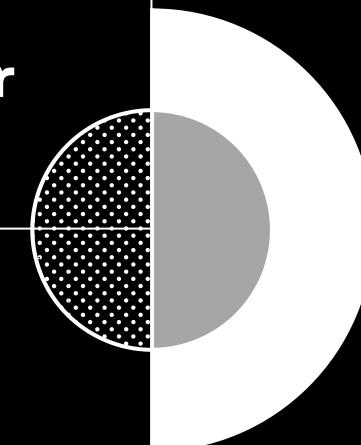
src
config
Authentication.php
Database.php
MongoLogger.php
Router.php
controllers
AdminLoginController.php
CreateProjectController.php
DashboardController.php
HomeController.php
LoginController.php
LogoutController.php
ProjectDetailController.php
RegisterController.php
js
models
Competence.php
Component.php
Profile.php
Project.php
Statistic.php
User.php
public
style
views
components
admin-login.php
create-project.php
dashboard.php
home.php
login.php
project-detail.php
register.php
.htaccess
index.php
routes.php
vendor
.env

# Implementazione del WIS

13

MVC

Fornisce i metodi per le operazioni sul database Gestisce i log delle operazioni	Model
Definisce il layout della pagina e la presentazione dei dati	View
Recupera i dati per popolare la view Richiama i metodi del model, gestendo gli eventi	Controller



# Implementazione del WIS

View

```
1 <form action="/login" method="post">
2     <div class="form-group">
3         <label for="email">Email</label>
4         <input type="email" class="form-control" id="email" name="email" required>
5     </div>
6     <div class="form-group">
7         <label for="password">Password</label>
8         <input type="password" class="form-control" id="password" name="password" required>
9     </div>
10    <button type="submit" class="btn btn-primary btn-block">Accedi</button>
11</form>
```

# Implementazione del WIS

Controller

```

1 function handleLogin() {
2     $db = new Database();
3     $conn = $db->getConnection();
4     $userModel = new User($conn);
5
6     $email = isset($_POST['email']) ? trim($_POST['email']) : '';
7     $password = isset($_POST['password']) ? trim($_POST['password']) : '';
8
9     if (empty($email) || empty($password)) {
10         $_SESSION['error'] = "Inserisci sia email che password.";
11         header('Location: /login');
12         exit;
13     }
14
15     if ($userModel->isAdmin($email)) {
16         $_SESSION['info'] = "Gli amministratori devono usare il login amministratore.";
17         header('Location: /admin-login');
18         exit;
19     }
20
21     $hashedPassword = hash('sha256', $password);
22     $loginResult = $userModel->login($email, $hashedPassword);
23
24     if ($loginResult['success']) {
25         $userData = $loginResult['data'];
26         $token = bin2hex(random_bytes(32));
27
28         $_SESSION['user_id'] = $userData['email'];
29         $_SESSION['user_name'] = $userData['nome'] . ' ' . $userData['cognome'];
30         $_SESSION['user_nickname'] = $userData['nickname'];
31         $_SESSION['user_type'] = $userModel->isCreator($email) ? 'creator' : 'user';
32         $_SESSION['auth_token'] = $token;
33         $_SESSION['token_expiration'] = time() + (60 * 60);
34
35         header('Location: /dashboard');
36         exit;
37     } else {
38         $_SESSION['error'] = "Email o password non validi.";
39         header('Location: /login');
40         exit;
41     }
42 }
```

# Implementazione del WIS

Model

```
1 class User {
2     private $conn;
3     private $logger;
4
5     public function __construct($db) {
6         $this->conn = $db;
7         $this->logger = new \MongoLogger();
8     }
9
10    /**
11     * Esegue il login di un utente.
12     *
13     * @param string $email Email dell'utente.
14     * @param string $hashedPassword Password hashata.
15     * @return array ['success' => bool, 'data' => array|null]
16     */
17    public function login($email, $hashedPassword) {
18        $email = strtolower($email);
19        try {
20            $stmt = $this->conn->prepare("CALL autenticazione_utente(:email, :password, @autenticato)");
21            $stmt->bindParam(':email', $email);
22            $stmt->bindParam(':password', $hashedPassword);
23            $stmt->execute();
24
25            $result = $this->conn->query("SELECT @autenticato as autenticato")->fetch(PDO::FETCH_ASSOC);
26
27            if ($result['autenticato']) {
28                $userData = $this->getData($email);
29                return ['success' => true, 'data' => $userData['data']];
30            }
31
32            return ['success' => false, 'data' => null];
33        } catch (PDOException $e) {
34            error_log($e->getMessage());
35            return ['success' => false, 'data' => null];
36        }
37    }
38}
```

# Implementazione del WIS

PDO

```
1 class Database {
2     private $host;
3     private $db_name;
4     private $username;
5     private $password;
6     private $conn = null;
7
8     public function __construct() {
9         $this->host = $_ENV['DB_HOST'];
10    $this->db_name = $_ENV['DB_NAME'];
11    $this->username = $_ENV['DB_USERNAME'];
12    $this->password = $_ENV['DB_PASSWORD'];
13 }
14
15 /**
16 * Stabilisce e restituisce una connessione PDO al database.
17 *
18 * @return PDO|null Oggetto PDO se la connessione ha successo, null altrimenti.
19 */
20 public function getConnection() {
21     try {
22         $this->conn = new PDO("mysql:host=" . $this->host . ";dbname=" . $this->db_name, $this->username, $this->password);
23         $this->conn->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
24     } catch(PDOException $e) {
25         echo " Errore durante la connessione: " . $e->getMessage();
26     }
27     return $this->conn;
28 }
29 }
```

```
1 class User {
2     private $conn;
3     private $logger;
4
5     public function __construct($db) {
6         $this->conn = $db;
7         $this->logger = new \MongoLogger();
8     }
9
10    /**
11     * Esegue il login di un utente.
12     *
13     * @param string $email Email dell'utente.
14     * @param string $hashedPassword Password hashata.
15     * @return array ['success' => bool, 'data' => array|null]
16     */
17    public function login($email, $hashedPassword) {
18        $email = strtolower($email);
19        try {
20            $stmt = $this->conn->prepare("CALL autenticazione_utente(:email, :password, @autenticato)");
21            $stmt->bindParam(':email', $email);
22            $stmt->bindParam(':password', $hashedPassword);
23            $stmt->execute();
24
25            $result = $this->conn->query("SELECT @autenticato as autenticato")->fetch(PDO::FETCH_ASSOC);
26
27            if ($result['autenticato']) {
28                $userData = $this->getData($email);
29                return ['success' => true, 'data' => $userData['data']];
30            }
31
32            return ['success' => false, 'data' => null];
33        } catch (PDOException $e) {
34            error_log($e->getMessage());
35            return ['success' => false, 'data' => null];
36        }
37    }
38}
```

# Implementazione del WIS

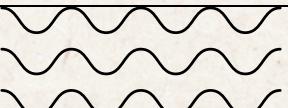
MongoLogger

**Clicca *qui* per provare BOSTARTER!**

**AUTORI**

Francesco Maria Fuligni

Roberto Zanolli



*grazie*

# Resource Page

USE THESE DESIGN RESOURCES IN YOUR  
CANVA PRESENTATION. HAPPY DESIGNING!

DELETE OR HIDE THIS PAGE BEFORE  
PRESENTING.

