Raccolta dei testi dei progetti per l'esame di Applicazioni dinamiche per il Web per AA 2021/2022

Aggiornato al 13 luglio 2022

1 Progetto per gli appelli di giugno e luglio 2022

Considerate la base di dati all'indirizzo `https://www.w3resource.com/sql/sql-table.php` che rappresenta gli ordini fatti dai clienti tramite degli agenti.

Si realizzi un'applicazione web dinamica in cui:

1. È possibile gestire gli ordini da parte di più *attori*.
2. Se l'attore è un *cliente*, può vedere solo i propri ordini fatti.
3. Se l'attore è un *agente*, può vedere, modificare e cancellare solo gli ordini da lui gestiti. Può, inoltre, aggiungere un ordine.
4. Se l'attore è un *dirigente*, può vedere e modificare tutte le 3 entità della base di dati.

L'applicazione quindi deve, in una base di dati separata, gestire gli account dei possibili utenti che accedono alla applicazione mediante autenticazione per garantire il principio che la base di dati di dominio deve essere separata dalla base di dati di funzionamento dell'applicazione.

Nel caso 2., il cliente deve poter vedere la lista dei suoi ordini ordinabile al volo secondo le diverse colonne (ogni ordine deve riportare anche l'agente cliccabile. Cliccando sul nome dell'agente si deve aprire una div che dà informazioni di contatto dell'agente)

Analogamente per il caso 3, dove gli ordini riportano il nome del cliente invece dell'agente. Il nome del cliente deve essere cliccabile nella stessa maniera in cui lo è l'agente nel case 2. Per limitare il progetto, **non** si richiede che un agente possa aggiungere o modificare la tabella clienti.

Nel caso 4, un dirigente dovrebbe poter gestire gli ordini, gli agenti e i clienti. Per limitare il progetto, si richiede **solo** che la visualizzazione degli ordini sia completa (con cliente e agente) con solo la possibilità di modificare gli ordini presenti.

Ulteriori richieste di progetto:

1. Il gruppo di sviluppo **deve** essere di 2-3 studenti in cui le responsabilità di progetto devono essere ben definite.
2. Il framework da usare è Spring Boot+GraphQL per realizzare i microservizi.
3. Il DBMS deve essere PostregSQL ≥ 12.
4. Il lato client deve essere Vue o React (String Boot è facilmente integrabile con Vue).
5. L'applicazione deve essere accessibile anche via tablet o smartphone (usare CSS per rendere responsive).
6. Garantire il livello AA WCAG 2.1 solo per le pagine accessibili dai clienti.

Modalità di presentazione all'esame

- Preparare alcune slide in cui si spiega come funziona Spring Boot (questo serve per valutare quanto uno studente ha capito del framework).
- Preparare alcune slide in cui si mostra i servizi resi disponibili secondo il modello GraphQL.
- Preparare alcune slide in cui si mostra il processo di validazione AA WCAG 2.1 (che strumenti si sono scelti, che prove si sono fatte). Queste prove si devono poi mostrare dal vivo.
- È necessario saper dimostrare di essere in grado di presentare/modificare qualsiasi riga del codice sorgente a richiesta del docente.
- È possibile che venga richiesto di aggiungere una funzionalità al volo (di piccola entità).

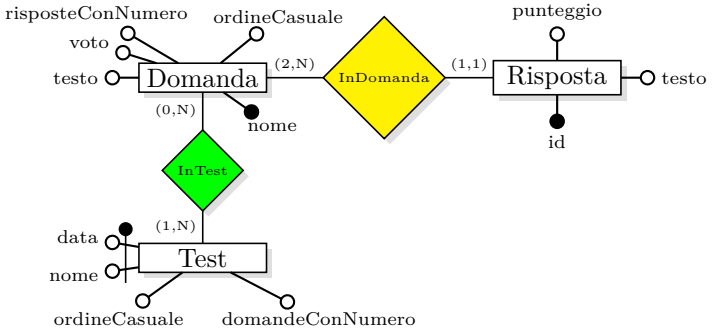
Ricordo, infine, le raccomandazione fatte alla presentazione del corso:

- Il progetto deve funzionare esattamente come da specifiche
- Il progetto deve essere sviluppato su un proprio PC (per chi non ha PC, ci sono i PC dei laboratori)
- Il progetto deve venire presentato da tutto il gruppo insieme.
- Non sono ammesse tecnologie che richiedano compilazioni o procedure complesse di aggiornamento del codice eseguito. Deve essere possibile aprire un file sul server e modificarlo al volo durante la prova. Web packer, minimizzatori ed altre scorciatoie non sono ammesse.
- Si devono evitare i classici anti-pattern (`https://en.wikipedia.org/wiki/Anti-pattern`) in fase di analisi e sviluppo! Quelli non tollerati:
  - Cargo Cult programming
  - Coding by exception
  - Copy and Paste programming
  - Hard Code
  - Magic Numbers and Strings - Reinventing the wheel

2 Progetto per gli appelli di settembre 2022 e febbraio 2023

Si vuole costruire un prototipo di applicazione web per la creazione e somministrazione di test basati su domande a risposte multiple.

Lo schema entità-relazione è il seguente:



Una possibile schema fisico (senza indici e controlli di numerosità) è il seguente:

```sql
CREATE TABLE Test (
   data TIMESTAMP NOT NULL,
   nome VARCHAR NOT NULL,
   ordineCasuale BOOLEAN DEFAULT FALSE, --le domande devono essere presentate in ordine casuale
   domandeConNumero BOOLEAN DEFAULT FALSE, --le domande devono essere numerate
   PRIMARY KEY(data,nome)
);
CREATE TABLE Domanda (
   nome VARCHAR PRIMARY KEY,
   testo VARCHAR NOT NULL,
   punti DECIMAL(5,2), -- quanti punti vale la domanda. Esempio: 2.0
   ordineCasuale BOOLEAN DEFAULT FALSE, -- le risposte devono essere presentate in ordine casuale
   risposteConNumero BOOLEAN DEFAULT FALSE --le risposte devono essere numerate
);
CREATE TABLE Risposta (
    id serial PRIMARY KEY,
    testo VARCHAR NOT NULL,
    punteggio DECIMAL(5,4) CHECK (punteggio <= 1.0),  -- percentuale dei punti della domanda
    domanda VARCHAR REFERENCES Domanda
);
CREATE TABLE InTest (
    domanda VARCHAR REFERENCES Domanda,
    dataTest TIMESTAMP NOT NULL,
    nomeTest VARCHAR NOT NULL,
    FOREIGN KEY (dataTest, nomeTest) REFERENCES Test
);
```

Si chiede di mantenere i nomi delle tabelle e degli attributi.

Alcuni vincoli che si devono rispettare e che sono parzialmente espressi nello schema fisico:

- Un test è definito da almeno una domanda (c'è).
- Una domanda deve avere almeno due risposte (c'è).
- Almeno una risposta di una domanda deve avere come punteggio 1.0 (che equivale al 100%) che rappresenta la risposta esatta.
- Una domanda può contenere anche più di una risposta con punteggio 1.0 (più risposte esatte).
- Un test deve presentare le domande in ordine sparso se `ordineCasuale` è vero.
- Un test deve numerare le domande se `domandeConNumero` è vero (il numero è l'ordine di presentazione).
- Una domanda deve presentare le risposte in ordine sparso se `ordineCasuale` è vero.
- Una domanda deve numerare le risposte se `risposteConNumero` è vero (il numero è l'ordine di presentazione).
- Il punteggio totale di un test è dato dalla somma dei punti delle risposte.
  Esempio: Si supponga che ci sia un test che ha 10 domande, dove 5 valgono 2 punti mentre le altre 5 valgono 1 punto. Un utente risponde a tutte le domande da 2 punti con una risposta parzialmente esatta (punteggio 0.5 o 50%) e a tutte le domande da un 1 punto in modo esatto. Il punteggio ottenuto è 10/15.

Si realizzi un'applicazione web dinamica in cui:

1. Ci sono due tipi di utenti: docenti e studenti.
2. I docenti possono creare test (quindi creare domande e risposte) e possono fare i test.
3. Gli studenti possono solo fare i test.
4. Un docente quando si collega deve poter vedere i test presenti, decidere se fare un test o se crearne uno nuovo.
5. La creazione di un test deve garantire che vengano rispettati tutti i vincoli di cui sopra.
6. Uno studente quando si collega deve poter vedere i test presenti e poter eseguire un test.
7. Quando uno studente esegue un test, ogni domanda viene presentata in una pagina separata e ogni volta che si passa alla domanda successiva o precedente, lo stato del test (risposte date) deve essere salvato. Questo per garantire che se per qualche motivo il collegamento si interrompe, lo studente ricollegandosi può continuare con il test senza perdere nulla. (Questa richiesta richiede di salvare lo stato in modo opportuno, si chiede di usare la base di dati contenente i dati dell'applicazione per salvare lo stato, no la base di dati contenente le domande).
8. Alla fine del test, si deve presentare una pagina riassuntiva con il punteggio ottenuto e la lista delle domande specificando per ciascuna la risposta esatta e la/le risposta data. In questo modo lo studente capisce il punteggio. Non si richiede di salvare questo dato.
9. Le operazioni di interrogazione/modifica base di dati delle domande devono essere accessibili anche via GraphQL come microservizi solo da utenti autenticati. Se è un professore, può accedere a tutte le informazioni, se è uno studente, può accedere a tutto tranne i punteggi delle risposte (altrimenti capirebbe qual è/sono la/le risposte esatte).

L'applicazione quindi deve, in una base di dati separata, gestire gli account dei possibili utenti che accedono alla applicazione mediante autenticazione. Questo per garantire il principio che la base di dati di dominio deve essere separata dalla base di dati di funzionamento dell'applicazione.

Ulteriori richieste di progetto:

1. Il gruppo di sviluppo **deve** essere di 2-3 studenti in cui le responsabilità di progetto devono essere ben definite.
2. Il framework da usare è Spring Boot+GraphQL per realizzare i microservizi.
3. Il DBMS deve essere PostregSQL ≥ 12.
4. Il lato client deve essere React.
5. L'applicazione deve essere accessibile anche via tablet o smartphone (usare CSS per rendere responsive).
6. Garantire il livello AA WCAG 2.1 solo per le pagine di esecuzione dei test.

Modalità di presentazione all'esame

- Preparare alcune slide in cui si spiega come si è organizzata l'applicazione rispetto al framework Spring e React (quali moduli si sono usati, come si sono configurati i framework, etc.)
- Preparare alcune slide in cui si mostra i micro-servizi resi disponibili secondo il modello GraphQL.
- Preparare alcune slide in cui si mostra il processo di validazione AA WCAG 2.1 (che strumenti si sono scelti, che prove si sono fatte). Queste prove si devono poi mostrare dal vivo. Attenzione: esistono delle check-list previste dal WCAG 2.1, si devono usare!
- È necessario saper dimostrare di essere in grado di presentare/modificare qualsiasi riga del codice sorgente a richiesta del docente.

Ricordo, infine, le raccomandazione fatte alla presentazione del corso:

- Il progetto deve funzionare esattamente come da specifiche
- Il progetto deve essere sviluppato su un proprio PC (per chi non ha PC, ci sono i PC dei laboratori)

- Il progetto deve venire presentato da tutto il gruppo insieme.
- Non sono ammesse tecnologie che richiedano compilazioni o procedure complesse di aggiornamento del codice eseguito. Deve essere possibile aprire un file sul server e modificarlo al volo durante la prova. Web packer, minimizzatori ed altre scorciatoie non sono ammesse.
- Si devono evitare i classici anti-pattern (`https://en.wikipedia.org/wiki/Anti-pattern`) in fase di analisi e sviluppo! Quelli non tollerati:
  - Cargo Cult programming
  - Coding by exception
  - Copy and Paste programming
  - Hard Code
  - Magic Numbers and Strings - Reinventing the wheel