Alma Mater Studiorum - Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica per il Management

Piattaforma ESQL

DOCUMENTAZIONE SVOLTA DA:
Canghiari Matteo
De Rosa Davide
Ghazanfar Tabish
Nadifi Ossama

Anno Accademico 2023/2024

1 Design Model

1.1 Modello di dominio

1.2 Diagramma dei casi d'uso

METTERE DIAGRAMMA

Registrazione

ID: UC1 - Registrazione

Attori: Utente

Precondizioni:

- Non presente

Main sequence:

1. Utente inserisce e invia i dati
2. Sistema acquisisce e salva i dati
3. Sistema invia esito positivo
4. Utente visualizza esito positivo

Alternative sequence:

1. Utente inserisce e invia i dati non idonei
2. Sistema acquisisce e individua un errore
3. Sistema invia esito negativo
4. Utente visualizza esito negativo

Postcondizioni:

- Sistema salva i dati nella piattaforma

Login

ID: UC2 - Login

Attori: Utente

Precondizioni:

- Utente ha effettuato la registrazione

Main sequence:

1. Utente inserisce e invia i dati
2. Sistema acquisisce e confronta con i dati salvati
3. Sistema nega l'accesso e invia esito positivo
4. Utente visualizza esito positivo

Alternative sequence:

1. Utente inserisce e invia i dati non idonei
2. Sistema acquisisce e individua un errore
3. Sistema invia esito negativo
4. Utente visualizza esito negativo

Postcondizioni:

- Utente autenticato dalla piatttaforma

Logout

ID: UC3 - Logout

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma

Main sequence:

1. Utente effettua la selezione di logout
2. Sistema registra lo stato Utente

Alternative sequence:

1. Non presente

Postcondizioni:

- Utente disconesso dalla piatttaforma

Pagamento

ID: UC4 - Pagamento

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma
- Utente non abbia effettuato alcun pagamento

Main sequence:

1. Utente inserisce e invia i dati
2. Sistema acquisisce i dati
3. Sistema processa pagamento e invia esito positivo
4. Utente visualizza esito positivo

Alternative sequence:

1. Utente inserisce e invia i dati
2. Sistema acquisisce i dati
3. Sistema processa pagamento e invia esito negativo
4. Utente visualizza esito negativo

Postcondizioni:

- Sistema salva lo stato Utente di pagamento

Creazione scenario

ID: UC5 - Creazione scenario

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma

Main sequence:

1. Utente inserisce e invia i dati
2. Sistema acquisisce i dati
3. Sistema controlla i dati e invia esito positivo
4. Utente visualizza esito positivo

Alternative sequence:

1. Utente inserisce e invia i dati non completi
2. Sistema acquisisce i dati
3. Sistema controlla i dati e invia esito negativo
4. Utente visualizza esito negativo

Postcondizioni:

- Sistema salva i dati Scenario

Creazione scelta

ID: UC6 - Creazione scelta

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma
- Utente abbia creato componenti Scenario

Main sequence:

1. Utente inserisce e invia i dati
2. Sistema acquisisce i dati
3. Sistema controlla i dati e invia esito positivo
4. Utente visualizza esito positivo

Alternative sequence:

1. Utente inserisce e invia i dati non completi
2. Sistema acquisisce i dati
3. Sistema controlla i dati e invia esito negativo
4. Utente visualizza esito negativo

Postcondizioni:

- Sistema salva i dati Scelta

Salvataggio storia

ID: UC7 - Salvataggio storia

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma
- Utente abbia creato componenti Scenario
- Utente abbia creato componenti Scelta

Main sequence:

1. Utente richiede salvataggio Storia
2. Sistema acquisisce i dati
3. Sistema controlla i parametri di salvataggio
4. Sistema invia esito positivo
5. Utente visualizza esito positivo

Alternative sequence:

1. Utente richiede salvataggio Storia
2. Sistema acquisisce i dati
3. Sistema controlla i parametri di salvataggio
4. Sistema invia esito negativo
5. Utente visualizza esito negativo

Postcondizioni:

- Sistema salva i dati Storia
- Sistema rende componente Storia giocabile

Gestione storia

ID: UC8 - Gestione storia

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma

Main sequence:

1. Utente richiede rimozione della Storia
2. Sistema riceve richiesta

Alternative sequence:

1. Non presente

Postcondizioni:

- Sistema elimina componente Storia

Gestione scenario

ID: UC9 - Gestione scenario

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma

Main sequence:

1. Utente richiede rimozione dello Scenario
2. Sistema riceve richiesta

Alternative sequence:

1. Utente richiede modifica testuale dello Scenario
2. Sistema riceve richiesta
3. Utente inserisci i dati sostitutivi
4. Sistema acquisisce modifiche

Postcondizioni:

- Sistema elimina componente Scenario
- Sistema modifica Scenario

Gestione scelta

ID: UC10 - Gestione scelta

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma

Main sequence:

1. Utente richiede rimozione della Scelta
2. Sistema riceve richiesta

Alternative sequence:

1. Utente richiede modifica testuale della Scelta
2. Sistema riceve richiesta
3. Utente inserisci i dati sostitutivi
4. Sistema acquisisce modifiche

Postcondizioni:

- Sistema elimina componente Scelta
- Sistema modifica Scelta

Gestione storia

ID: UC11 - Gestione storia

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma
- Utente abbia effettuato un pagamento
- Storia deve essere giocabile

Main sequence:

1. Utente seleziona una Storia giocabile
2. Sistema mostra Scenario composto dalle sue Scelte
3. Utente indica Scelta
4. Sistema acquisisce la Scelta
5. Sistema mostra Scenario successivo

Alternative sequence:

1. Utente seleziona una Storia giocabile
2. Sistema mostra Scenario composto dalle sue Scelte
3. Utente indica Scelta
4. Sistema acquisisce la Scelta
5. Sistema mostra Scenario finale

Postcondizioni:

- Sistema salva la Partita

Gestione partita

ID: UC12 - Gestione partita

Attori: Utente

Precondizioni:

- Utente autenticato dalla piattaforma
- Utente abbia effettuato un pagamento
- Storia deve essere giocabile

Main sequence:

1. Utente richiede rimozione Partita inerente alla Storia
2. Sistema riceve richiesta

Alternative sequence:

1. Utente seleziona una Partita da riprendere
2. Sistema mostra Scenario composto dalle sue Scelte
3. Utente indica Scelta
4. Sistema acquisisce la Scelta
5. Sistema mostra Scenario Successivo

Postcondizioni:

- Sistema rimuove la Partita
- Utente riprende la Partita

Creazione oggetto

- ID: UC13 - Creazione oggetto
- Attori: Utente
- Precondizioni:
  - Utente autenticato dalla piattaforma
- Main sequence:
  1. Utente inserisce e invia i dati
  2. Sistema acquisisce i dati
  3. Sistema controlla i dati e invia esito positivo
  4. Utente visualizza esito positivo
- Alternative sequence:
  1. Utente inserisce e invia i dati non completi
  2. Sistema acquisisce i dati
  3. Sistema controlla i dati e invia esito negativo
  4. Utente visualizza esito negativo
- Postcondizioni:
  - Sistema salva i dati dell'Oggetto

Gestione oggetto

- ID: UC14 - Gestione oggetto
- Attori: Utente
- Precondizioni:
  - Utente autenticato dalla piattaforma
- Main sequence:
  1. Utente richiede rimozione dell'Oggetto

2. Sistema riceve richiesta

- Alternative sequence:

1. Utente richiede modifica testuale dell'Oggetto
2. Sistema riceve richiesta
3. Utente inserisce modifiche
4. Sistema acquisisce modifiche

- Postcondizioni:

    - Sistema rimuove componente Oggetto
    - Sistema modifica componente Oggetto

1.3 Burndown chart

2 Manuale dell'utente

Home

Sezione a cui si accede avviando l'applicativo. All'interno di questa è possibile ottenere una breve descrizione del progetto, visualizzare le specifiche e la repository GitHub, ed infine accedere alla relazione. Dalla navbar si può accedere alle schermate di *Accedi* e *Registrati*. **METTERE FOTO**

Registrazione

Sezione che permette di registrarsi alla piattaforma. Offre la possibilità di accedere alla schermata di *Accedi* sia tramite la schermata principale che tramite navbar. **METTERE FOTO**

Accedi

Sezione che permette di autenticarsi alla piattaforma. Offre la possibilità di accedere alla schermata di *Registrazione* sia tramite la schermata principale che tramite navbar. **METTERE FOTO**

Homepage storJ

Sezione a cui si accede in seguito all'autenticazione. Permette di visualizzare una breve descrizione delle azioni eseguibili. Attraverso la navbar si potranno raggiungere le schermate relative a *Paga*, *Crea*, ed infine effettuare il *Logout*. **METTERE FOTO** Nel caso in cui pagamento sia già stato effettuato, la voce *Paga* sarà sostituita da *Gioca*.

Paga

Sezione che permette di effettuare il pagamento, in modo da poter sbloccare tutte le funzionalità della piattaforma. Sarà necessario inserire l'ammontare del pagamento, titolare e numero della carta, ed infine il relativo CVV. Potrebbe essere necessario effettuare più tentativi per far sì che vada a buon fine. **METTERE FOTO**

Storie

Sezione che consente la gestione delle storie. Permette di visualizzare le storie create in precedenza. Ad ognuna di esse sono assegnati due bottoni, necessari per la *modifica* e l'*eliminazione*. È presente una navbar contenente il collegamento alla schermata dedicata alla *creazione* delle storie. Nel caso la storia sia stata salvata, la *modifica* permetterà solo di cambiare i testi all'interno di questa. **METTERE FOTO**

Form creazione storia

Sezione che permette la *creazione* di una storia. È necessario inserire il titolo e la categoria. **METTERE FOTO**

Scenari

Sezione che consente la *gestione* di una storia specifica. Permette di visualizzare gli scenari appartenenti alla storia. Ad ognuno di essi sono associate le relative informazioni e due bottoni, utilizzati per l'accesso alla *gestione* delle scelte e per la *cancellazione* dello scenario. È presente una navbar che permette di salvare la storia e accedere alle sezioni relative alla *creazione* di uno scenario, e infine alla *gestione* degli oggetti. **METTERE FOTO** Una volta effettuato il salvataggio della storia, sarà possibile solo modificare i testi relativi ad essa. **METTERE FOTO**

Form creazione scenario

Sezione che permette la *creazione* di uno scenario. È necessario inserire i dati relativi ad esso, quindi il testo, la tipologia di scenario, la tipologia di risposta ed infine indicare la presenza o meno del drop di un oggetto.

Nota bene: è necessario aver già creato l'oggetto che si vuole far rilasciare dallo scenario. **METTERE FOTO**

Indovinello

Sezione che permette la *gestione* della risposta ad un indovinello. È possibile visualizzare i dati relativi a quest'ultimo, quindi il testo, la risposta associata ed infine gli scenari destinazione in seguito alla risposta corretta piuttosto che alla risposta errata. **METTERE FOTO** È possibile cancellare l'indovinello, se presente, e attraverso navbar ci sarà la possibilità di crearne uno nuovo. **METTERE FOTO**

Form creazione indovinello

Sezione che permette la *creazione* della risposta ad un indovinello. È necessario inserire il testo del quesito, la risposta attesa ed infine gli scenari di destinazione in seguito alla risposta corretta piuttosto che alla risposta errata. **METTERE FOTO**

Scelta multipla

Sezione che permette la *gestione* delle risposte ad una domanda a scelta multipla. È possibile visualizzare i dati relativi alle opzioni già presenti, quindi il testo, lo scenario successivo alla risposta, e la presenza o meno di un oggetto richiesto.

Nota bene: è necessario aver già creato l'oggetto richiesto.

È possibile inoltre cancellare le opzioni, se presenti, e attraverso navbar ci sarà la possibilità di crearne nuove. **METTERE FOTO**

Form creazione multipla

Sezione che permette la *creazione* della risposta ad una domanda a scelta multipla. È necessario inserire il testo del quesito, lo scenario successivo alla risposta, e la presenza o meno di un oggetto richiesto. **METTERE FOTO**

Oggetti

Sezione che permette la *gestione* degli oggetti di una storia. È possibile visualizzare i dati relativi ad ognuno, quindi il titolo e la descrizione, ed un bottone che permette la cancellazione. Attraverso navbar ci sarà la possibilità di creare nuovi oggetti. **METTERE FOTO**

Form creazione oggetti

Sezione che permette la *creazione* di un oggetto. È necessario inserire i dati relativi ad esso, quindi il titolo e la descrizione. **METTERE FOTO**

Storie giocabili

Sezione che permette la *selezione* della storia da giocare. È possibile visualizzare tutte le storie giocabili con le varie informazioni, permettendo la ricerca attraverso dei filtri. **METTERE FOTO**

Per ogni storia presente ci sarà la possibilità di *avviare* la partita, o in alternativa di *visualizzare* l'*anteprima* di essa.

Gioca

Sezione in cui avviene lo *svolgimento* della partita. All'interno della schermata sarà presente il testo dello scenario ed in base alla tipologia della domanda verranno visualizzati i bottoni contenenti le opzioni di risposta piuttosto che l'area testuale per inserire la risposta all'indovinello. **METTERE FOTO**

METTERE FOTO

Attraverso la barra di navigazione sarà sempre possibile consultare l'*inventario* della partita. **METTERE FOTO**

Partite giocate

Sezione in cui sono presenti le partite avviate dall'utente. Per ogni partita sarà possibile riprendere la storia dallo scenario in cui era stata interrotta oppure rimuovere i dati di gioco relativi ad essa. **METTERE FOTO**

2.1 Setup e deploy

Unico requisito per il deploy dell'applicativo è **Docker**. Una volta scaricata la *repository* da *GitHub*, sarà sufficiente eseguire all'interno della cartella il comando **docker-compose up –build**; quest ultimo permetterà a *Docker* di eseguire la fase di *build* e successivamente di *avviare* i *container* relativi ai diversi servizi che compongono l'applicativo.

Per terminare l'esecuzione, eseguire all'interno della cartella precedente il comando **docker-compose down**, il quale terminerà ed eliminerà i *container* dell'applicazione.

È stato scelto di utilizzare *Docker* e *Docker-Compose* per il deploy dell'applicativo per avere una maggiore facilità di *deploy* e per permettere l'utilizzo del servizio senza dover installare in locale alcuna dipendenza software, come *Maven*, *NodeJS* o *PostgreSQL*.

È possibile osservare all'interno della repository i *Dockerfile* inerenti ai vari servizi, i quali permettono a *Docker* di avviare i *container*. Per ottenere un avvio simultaneo di tutti i servizi viene utilizzato un file *docker-compose.yml*, il quale permette l'avvio dell'applicativo con un singolo comando.

Un esempio di *Dockerfile* appartiene allo *Spring Boot* del backend, il quale si suddivide in due fasi:

```
FROM maven:3.8.3-openjdk-17 AS spring-builder
WORKDIR /usr/src/app
COPY . /usr/src/app
RUN mvn clean package

FROM eclipse-temurin:17
WORKDIR /opt/app
COPY --from=spring-builder /usr/src/app/target/storj-1.0.0.jar /opt/app/storj.jar
EXPOSE 8080
CMD ["java", "-jar", "/opt/app/storj.jar"]
```

La prima fase esegue il comando *Maven* per ottenere il file *.jar* eseguibile. La seconda avvia un *container* eseguendo l'applicativo di backend.

Passando invece al *docker-compose.yml*, la suddivisione è netta tra i servizi:

```
version: '3'

services:
  db:
    image: 'postgres:16'
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD=root
      - POSTGRES_DB=storj
    container_name: db
    networks:
      - storjnetwork
    volumes:
      - ./db/volumes/data:/var/lib/postgresql/data
      - ./db/sql/:/docker-entrypoint-initdb.d/
```

```
aymentservice:
    build: payment
    container_name: paymentservice
    depends_on:
    - app
    networks:
    - storjnetwork
    ports:
    - "6789:6789"

app:
    build: backend
    container_name: storj
    depends_on:
    - db
    networks:
    - storjnetwork
    ports:
    - "8080:8080"

angular:
    build: frontend
    container_name: angular
    depends_on:
    - app
    networks:
    - storjnetwork
    ports:
    - "4201:4200"

networks:
    storjnetwork:
      driver: bridge
```

Come si può notare, vengono avviati quattro servizi, distinti in:

- **Database**, servizio inerente al *DBMS* scelto per la persistenza dei dati; in questa casistica ricade in *PostgreSQL*. Vengono quindi configurati alcuni valori di *environment* per permettere la creazione del database all'interno del *container*. Vengono inoltre utilizzati i *volumi* per garantire la persistenza dei dati e per creare lo schema del database al primo avvio del *container*
- **Paymentservice**, servizio di pagamento esterno richiesto dalla traccia. Viene avviato un *container Docker* che avvia l'eseguibile fornito nelle specifiche, esponendo la chiamata *API* per il pagamento
- **App**, backend dell'applicativo, realizzato con *Spring Boot*. L'*image* relativa al suo *container* viene realizzata attraverso il *Dockerfile* osservato in precedenza
- **Angular**, frontend dell'applicativo, realizzato con *Angular*. Come avviene per il backend, l'*image* relativa al suo *container* viene realizzata attraverso un *Dockerfile*, presente nella cartella del frontend

Si può notare infine l'utilizzo dei networks di *Docker*, i quali permettono ai diversi container di comunicare tra loro. Vengono inoltre esposte diverse porte per permettere l'accesso ai servizi offerti dai *container* sulla *macchina host*.

2.2 Frontend

Per il frontend è stato utilizzato il framework **Angular**, in grado di garantire un approccio orientato a facilitare lo sviluppo delle interfacce grafiche. *Angular* fornisce un elevato livello di *reusability* del codice, grazie alla suddivisione dell'architettura che lo contraddistingue in differenti *componenti*; inoltre il framework è caratterizzato da un'ulteriore capacità, la quale consiste in azioni automatiche di *sincronizzazione* tra la *User Interface* e il modello dati. Infine, uno dei fattori principali è dettato dalla *Dependency Injection*, poiché facilita la manipolazione dei legami posti tra *componenti* e *servizi* che compongono l'applicazione.

Come è stato già citato prima, il progetto è stato concepito seguendo un'architettura modulare, che divide l'applicazione in *componenti* e *servizi*. Questo approccio favorisce la separazione delle responsabilità, migliorando la manutenibilità e la scalabilità del codice. Tuttavia, è bene individuare quali siano i punti salienti su cui occorre soffermarsi, descritti come segue:

- **Componenti**, rappresentano le parti visive dell'applicazione, ognuna incaricata di gestire una specifica funzionalità o area dell'interfaccia utente. Nell'applicazione sono presenti tre elementi fondamentali, suddivisi in:
  - **Form**, attuati per consentire all'utente di creare oppure modificare i vari componenti delle proprie storie
  - **Navbar**, elemento imprescindibile per la navigazione tra le pagine che compongono il sito; dinamica rispetto all'interazioni precedenti effettuate dall'utente
  - **Pop-up**, entità grafica visualizzata esclusivamente all'interno della fase di gioco, utilizzata per mostrare informazioni inerenti alla storia selezionata

  Oltre alle tre citate, sono molteplici le *pagine* presenti, le quali non sono contraddistinte da mansioni specifiche ma possiedono un ruolo fondamentale, poiché illustrano tutti i *componenti* definiti dai vari utenti
- **Servizi**, tramite per acquisire i dati dal backend, posti all'interno del database e comunicati dalle *API* (per ulteriori approfondimenti consultare sezione backend). Fungono da intermediari, sovrapposti tra i componenti e le *API*, affinché i dati possano essere forniti e successivamente visualizzati a schermo
- **LocalStorage**, strumento versatile, inerente non solo a fasi di sviluppo, principalmente attuato per azioni di *debugging*, ma anche a funzionalità mirate alla persistenza dei dati, garantendo un'esperienza all'utente fluida e consistente. Uno degli utilizzi principi riguarda il controllo dell'autenticazione dell'utente, necessario per un corretto funzionamento delle guardie
- **Guardie**, funzionalità built-in del framework, stabilite per circoscrivere il reindirizzamento tra le pagine. Nell'applicazione sono presenti due tipologie di *guardie*, in cui la prima, denominata **AccessGuard**, definisce se l'utente sia autenticato o meno; mentre la seconda, chiamata **PaymentAccessGuard**, impedisce all'utente di accedere al form *payment-page*, qualora abbia già effettuato un pagamento andato a buon fine
- **Classi**, utilizzate per definire l'architettura interna dei vari oggetti manipolati dall'app, inerenti alla struttura del database di riferimento (ad esempio scenari, storie oppure partite)

Backend

Per il backend è stato utilizzato il framework **Spring**, principalmente attuato per velocizzare le fasi di sviluppo. *Spring* inoltre fornisce funzionalità molto comode per lo sviluppo di backend, come la facilità di sviluppo di **REST API** ed il collegamento ad un **database**. Inoltre, garantisce l'approccio relativo alla **Dependency Injection**, attraverso il quale il contenitore *Spring* "inietta" oggetti in altre "dipendenze". In poche parole, ciò consente un accoppiamento libero dei componenti e sposta la responsabilità della gestione dei componenti al framework.

Passando all'implementazione delle *API*, l'approccio utilizzato è *Contract-First* secondo lo standard **OpenAPI 3.0**. Viene realizzata un'interfaccia, la quale permette ad un plugin di *Spring* di generare automaticamente i diversi metodi che verranno sovrascritti per implementare le funzionalità volute. Sono state realizzate le **CRUD** di tutte le strutture collegate al *database*, con alcune chiamate *custom* per ottenere liste personalizzate da visualizzare nel frontend.

Il collegamento con il *database* è stato realizzato attraverso il plugin **Spring Data JPA**, il quale permette in maniera veloce di collegare il backend con la base di dati.

L'architettura delle chiamate API è la seguente:

- Ogni entità della base di dati possiede una corrispettiva classe **Entity**, che rappresenta la riga all'interno del database come un oggetto *Java*. Per comunicare direttamente col *database*, viene creata un'interfaccia **Repository** per ogni entità, dove vengono specificate le diverse Query
- Attraverso il plugin di *OpenAPI* viene generata per ogni entità un **Model**, il quale indica la struttura con la quale frontend e backend comunicano
- Per passare da *Model* ad *Entity*, e viceversa, viene utilizzato il plugin **MapStruct**, che permette di passare da un *Model* ad un **DTO** (**Data Transfer Object**) e da *DTO* ad *Entity*, e viceversa. Separando *Model* ed *Entity* otteniamo un codice più pulito, non utilizzando nella *business logic* del backend alcun *Model*
- I diversi metodi della *Repository* vengono *wrappati* da una classe **Service**, che riprende tutte le possibili chiamate al database per quella entità
- I metodi di ogni *Service* vengono ripresi da una classe di **Business Logic**, dove tutta la logica viene implementata
- In conclusione, i metodi di ogni *Business Logic* vengono richiamati dai **Controller**, ovvero le implementazioni delle interfacce generate dallo standard *OpenAPI*. All'interno dei controller non c'è alcuna logica, delegata alle classi di *Business Logic*

L'architettura delle *API* si basa sul **Single Responsibility Principle**, il quale afferma che ogni elemento di un programma (classe, metodo oppure variabile) deve avere una sola responsabilità, e che tale responsabilità debba essere interamente incapsulata dall'elemento stesso.

Infine, per la gestione degli errori possibili durante le chiamate *API* è stato implementato un *ExceptionHandlingController*, permettendo una gestione comoda e dinamica delle eccezioni.

Si osserva ora la sezione dei **test unitari**. Per quanto riguarda la base di dati, è stato

utilizzato il plugin **H2**, creando un **database in-memory** per eseguire tutti i test. Sono stati realizzati *126 test unitari*, ottenendo un **code coverage** del *97%*. I test sono stati effettuati sulle due parti principali del backend, i **servizi** (che comprendono le classi di *Business Logic* e di *Service*, testano quindi tutta la logica di business e le query) ed i **mapper**. È stato inoltre utilizzato il plugin **Jacoco** per ottenere un semplice report: **METTERE LA FOTO**

Nota bene: non è stato possibile ottenere un *code coverage* maggiore a causa della *randomicità* del pagamento, vincolando i test inerenti a quella parte.

2.3 Database

Per la persistenza dei dati della piattaforma è stato utilizzato un *Database Relazionale*. Nel nostro caso abbiamo scelto **PostgreSQL**. La struttura segue di pari passo le *API* descritte nella parte di backend. Sono presenti le seguenti tabelle:

- **utente**, dati inerenti all'utente, come *username*, *password* e se è stato effettuato o meno il *pagamento*
- **storia**, dati inerenti alla storia, come *creatore*, *titolo*, *categoria*, *numero scenari* e se la storia è stata *salvata* come partita giocabile
- **scenario**, dati inerenti allo scenario, come la *storia* a cui appartiene, *testo*, *tipologia* della *risposta* (*Indovinello* oppure *Multipla*) e *tipologia* dello *scenario* (*Iniziale*, *Normale* oppure *Finale*)
- **oggetto**, dati inerenti ad un oggetto, come la *storia* a cui *appartiene*, *nome* e *descrizione*
- **drop**, struttura di supporto, la quale permette di comprendere l'oggetto rilasciato all'interno di uno scenario. Al suo interno troviamo lo *scenario* e l'*oggetto* collegati
- **multipla**, dati inerenti alla singola scelta appartenente ad un insieme di scelte di uno scenario. Uno scenario può avere scelte multiple soltanto se appartiene alla tipologia *Multipla*. Possono esserci un minimo di due scelte per scenario, senza limiti superiori. Al suo interno si trova lo *scenario* a cui appartiene, *testo* della scelta e lo *scenario* di *destinazione*
- **required**, struttura di supporto, attuata per definire quale sia l'oggetto richiesto per effettuare una specifica scelta multipla. Composto da una *scelta* e dall'*oggetto* a cui è collegata
- **indovinello**, dati inerenti all'indovinello appartenente ad uno scenario, necessariamente di tipologia *Indovinello*. Può esserci soltanto un indovinello per scenario. A sua volta è composto dallo *scenario* a cui appartiene, da un *testo*, da una *risposta corretta*, da uno *scenario corretto*, qualora la risposta data sia conforme con la domanda, e da uno *scenario errato*, qualora l'utente dovesse indicare una risposta sbagliata
- **partita**, dati inerenti alla partita, come la *storia* a cui si riferisce, l'*utente* che la stia svolgendo e lo *scenario corrente*
- **inventario**, struttura di supporto, la quale permette di comprendere la lista di oggetti appartenenti ad una partita. Definito da una *partita* e da un *oggetto*

Il database viene avviato tramite il *docker-compose.yml* utilizzato per avviare tutti i servizi della piattaforma. Vengono inoltre utilizzati i *volumi* per garantire la persistenza dei dati e per creare lo schema del database al primo avvio del *container*. Per quest'ultima, si sfrutta

il *docker-entrypoint-initdb.d*, il quale avvia qualsiasi *script SQL* nella directory selezionata. Questo permette la creazione dinamica del *volume* del database al primo avvio del *container*.

2.4 Payment

Per il servizio di pagamento è stato utilizzato l'applicativo fornito nelle specifiche tecniche, il quale espone una *REST API* che simula il funzionamento di un pagamento. La chiamata *API* è stata *wrappata* all'interno delle chiamate del backend, permettendo una gestione più flessibile delle risposte ricevute. L'eseguibile viene eseguito grazie ad un Dockerfile, il quale permette di effettuare il *build* della sua *Docker image* e di essere eseguito con *docker-compose*.

2.5 DevOps

Per quanto riguarda la parte di **Continuous Integration** (*CI*) e di **Continuous Deployment** (*CD*) sono state automatizzate le *pipeline* attraverso **Jenkins**. L'esecuzione di *Jenkins* avviene all'interno di una *Virtual Machine* (Ubuntu Server).

2.6 Tavola media dei volumi

In questa sezione è specificato il numero stimato di istanze per ogni entità e relazione dello schema. I valori sono necessariamente approssimati, ma oltre tutto indicativi. Si prende come riferimento una realtà universitaria.

Concetto	**Tipo**	**Volume**
Utente	Entità	305
Studente	Entità	300
Docente	Entità	5
Tabella_Esercizio	Entità	50
Attributo	Entità	200
Test	Entità	10
Quesito	Entità	100
Domanda_Chiusa	Entità	50
Opzione_Risposta	Entità	150
Domanda_Codice	Entità	50
Sketch_Codice	Entità	50
Messaggio	Entità	610
Messaggio_Studente	Entità	600
Creazione	Relazione	50
Completamento	Relazione	3000
Invio	Relazione	18000
Ricezione	Relazione	10
Pubblicazione	Relazione	6100
Risposta	Relazione	30000
Composizione	Relazione	1000
Afferenza	Relazione	5000
Combinazione	Relazione	10000
Vincolo Integrità	Relazione	40000
Disposizione	Relazione	7500
Soluzione	Relazione	2500

Table 1: Stima della tavola media dei volumi riferita al progetto svolto.