

Quiz Web

| | | |
|-------|---|----|
| 1 | Introduzione..... | 3 |
| 1.1 | Informazioni sul progetto..... | 3 |
| 1.2 | Scopo | 3 |
| 2 | Analisi | 4 |
| 2.1 | Analisi del dominio | 4 |
| 2.2 | Analisi e specifica dei requisiti | 4 |
| 2.2.1 | Spiegazione elementi tabella dei requisiti:..... | 8 |
| 2.3 | Use case | 9 |
| 2.4 | Pianificazione | 10 |
| 2.5 | Analisi dei mezzi..... | 11 |
| 2.5.1 | Software | 11 |
| 2.5.2 | Hardware..... | 11 |
| 3 | Progettazione | 12 |
| 3.1 | Design dell'architettura del sistema | 12 |
| 3.1.1 | Struttura generale del sistema..... | 12 |
| 3.1.2 | Moduli/componenti applicativi..... | 12 |
| 3.1.3 | Flusso di dati principale | 13 |
| 3.1.4 | Site map | 13 |
| 3.2 | Design dei dati e database..... | 14 |
| 3.2.1 | Schema logico | 14 |
| 3.3 | Design delle interfacce | 14 |
| 3.4 | Design procedurale | 22 |
| 3.4.1 | Routing principale | 22 |
| 3.4.2 | Routing admin | 23 |
| 3.4.3 | Controllo accessi..... | 23 |
| 3.4.4 | Logica di valutazione quiz..... | 23 |
| 4 | Implementazione | 24 |
| 4.1 | Struttura progetto | 24 |
| 4.2 | Implementazione del Database..... | 25 |
| 4.3 | Implementazione del pannello di amministrazione | 26 |
| 4.3.1 | Implementazione grafica..... | 27 |
| 4.4 | Implementazione dei quiz | 29 |
| 4.5 | Sistema di Login..... | 30 |
| 4.6 | Helper personalizzati..... | 33 |
| 5 | Test..... | 35 |
| 5.1 | Protocollo di test..... | 35 |
| 5.2 | Risultati test..... | 41 |
| 5.3 | Mancanze/limitazioni conosciute..... | 41 |
| 6 | Consuntivo..... | 41 |
| 7 | Conclusioni | 43 |
| 7.1 | Sviluppi futuri..... | 43 |
| 7.2 | Considerazioni personali..... | 43 |
| 8 | Glossario | 44 |
| 9 | Bibliografia | 45 |
| 9.1 | Sitografia | 45 |
| 10 | Indice delle immagini | 45 |

1 Introduzione

1.1 Informazioni sul progetto

- Allievo Ryan Pinana
- Docente: Ingrid Paola Anna Cereda
- SAMT Progetti
- 12/09/25 – 19/12/25.

1.2 Scopo

Lo scopo del progetto è riuscire a creare un sito web su cui poter giocare a dei quiz le cui domande sono salvate su un DB esterno, proprio a questo proposito un ulteriore scopo è imparare ad usare il framework Node.js insieme ad Express e MongoDB per poter riuscire a far comunicare un sito web ad un DB.

2 Analisi

2.1 Analisi del dominio

Al momento esistono già molti siti web che ospitano cataloghi di quiz con diversi argomenti, ad esempio sporcle.com è uno dei più famosi siti per trovare quiz incentrati su vari argomenti della geografia, il mio sito vuol offrire una versione più semplificata e semplice di sporcle puntando ad offrire un catalogo di quiz che potrebbero risultare interessanti a tutti i tipi di utenti, dagli appassionati di geografia ai fanatici degli anime, dai bambini agli adulti.

Per poter far in modo che il sito possa essere utilizzato da tutti ovunque dovrà essere responsive, per questo motivo devo utilizzare oltre HTML anche CSS per dargli un design semplice e responsive per qualsiasi dispositivo, inoltre per il salvataggio delle domande per non salvarle tutte in dei file JS che sarebbe troppo semplice, farò in modo che le domande ed i quiz vengano salvati su un DB, e per fare in modo che interagisca con la pagina e possa mostrare le domande all'utente uso Node.JS con Express e MongoDB che permettono ad una pagina web di interagire con un DB senza usare PHP.

2.2 Analisi e specifica dei requisiti

Il progettista, dopo aver ricevuto il mandato, in collaborazione con il committente redige una lista di requisiti. Durante questi incontri, tramite interviste (da inserire nei diari), il progettista deve cercare di rispondere alle seguenti domande:

- Quali sono i bisogni del committente?
- Quali funzioni deve svolgere il prodotto?
- Come devono essere implementate?
- L'utente, come vorrebbe/dovrebbe interagire con il prodotto?
- Come verrà utilizzato il prodotto?
- Che tipo di interfaccia si immagina?
- Che prestazioni minime deve fornire il prodotto?
- Che grado di sicurezza deve avere il prodotto?
- ...

In base alla lista dei requisiti e all'analisi degli stessi, il progettista redige una *specifiche dei requisiti* in cui elenca e descrive in modo dettagliato quali sono le funzionalità che il prodotto fornirà. La specifica dovrebbe essere abbastanza dettagliata da poter essere utilizzata come base per lo sviluppo, ma non troppo; ad esempio non dovrebbe contenere dettagli di implementazione, o definizioni dettagliate dell'interfaccia grafica a meno che questi non siano considerati cruciali. Non si deve scordare che i requisiti non rappresentano delle attività bensì delle caratteristiche che il prodotto dovrà possedere.

Qua di seguito sono rappresentati i requisiti

| ID: REQ-01 | |
|------------|--|
| Nome | utilizzo DB locale |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Creazione ed utilizzo di un DB locale per salvare i dati |

| ID: REQ-02 | |
|------------------------|---|
| Nome | Login per l'amministratore |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Login unico per l'amministratore e abilita la modifica |
| Sotto requisiti | |
| 001 | Si necessita di creare delle credenziali per l'amministratore |

| ID: REQ-03 | |
|------------------------|-----------------------------------|
| Nome | Creazione delle domande |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Solo da parte dell'amministratore |
| Sotto requisiti | |
| 001 | Req-01 |
| 002 | Req-02 |

| ID: REQ-04 | |
|------------------------|-----------------------------------|
| Nome | Modifica delle domande |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Solo da parte dell'amministratore |
| Sotto requisiti | |
| 001 | Req-01 |
| 002 | Req-02 |

| ID: REQ-05 | |
|-----------------|-----------------------------------|
| Nome | Eliminazione delle domande |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Solo da parte dell'amministratore |

| Sotto requisiti | |
|-----------------|--------|
| 001 | Req-01 |
| 002 | Req-02 |

| ID: REQ-06 | |
|-----------------|-----------------------------------|
| Nome | Creazione dei quiz |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Solo da parte dell'amministratore |
| Sotto requisiti | |
| 001 | Req-01 |
| 002 | Req-02 |
| 003 | Req-03 |

| ID: REQ-07 | |
|-----------------|-----------------------------------|
| Nome | Modifica dei quiz |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Solo da parte dell'amministratore |
| Sotto requisiti | |
| 001 | Req-1 |
| 002 | Req-2 |
| 003 | Req-03 |

| ID: REQ-08 | |
|-----------------|-----------------------------------|
| Nome | Eliminazione dei quiz |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Solo da parte dell'amministratore |
| Sotto requisiti | |
| 001 | Req-01 |

| | |
|-----|--------|
| 002 | Req-02 |
| 003 | Req-03 |

| ID: REQ-09 | |
|-----------------|---|
| Nome | Interfaccia UI che mostri i quiz e le domande |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Amministratore può vedere pulsanti di modifica dei quiz e domande |
| Sotto requisiti | |
| 001 | REQ-01 |
| 002 | REQ-02 |

| ID: REQ-10 | |
|-----------------|--|
| Nome | Caricare le domande durante i quiz |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Le domande vengono caricate dal server |
| Sotto requisiti | |
| 001 | Req-01 |
| 002 | Req-02 |

| ID: REQ-11 | |
|-----------------|--|
| Nome | Barra di navigazione tra gli argomenti |
| Priorità | 1 |
| Versione | 1.0 |
| Note | Nav bar per i vari argomenti |
| Sotto requisiti | |
| 001 | Req-07 funziona |

| ID: REQ-12 | |
|-----------------|---|
| Nome | Interfaccia UI che mostri la lista di quiz giocabili all'utente |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | Quiz già presenti nel DB |

| ID: REQ-13 | |
|-----------------|---------------------------------------|
| Nome | Interfaccia UI del risultato del quiz |
| Priorità | 1 |
| Versione | 1.0 |
| Sotto requisiti | |
| 001 | REQ-12 |

2.2.1 Spiegazione elementi tabella dei requisiti:

ID: identificativo univoco del requisito

Nome: breve descrizione del requisito

Priorità: indica l'importanza di un requisito nell'insieme del progetto, definita assieme al committente. Ad esempio, poter disporre di report con colonne di colori diversi ha priorità minore rispetto al fatto di avere un database con gli elementi al suo interno. Solitamente si definiscono al massimo di 2-3 livelli di priorità.

Versione: indica la versione del requisito. Ogni modifica del requisito avrà una versione aggiornata.

Sulla documentazione apparirà solamente l'ultima versione, mentre le vecchie dovranno essere inserite nei diari.

Note: eventuali osservazioni importanti o riferimenti ad altri requisiti.

Sotto requisiti: elementi che compongono il requisito.

2.3 Use case

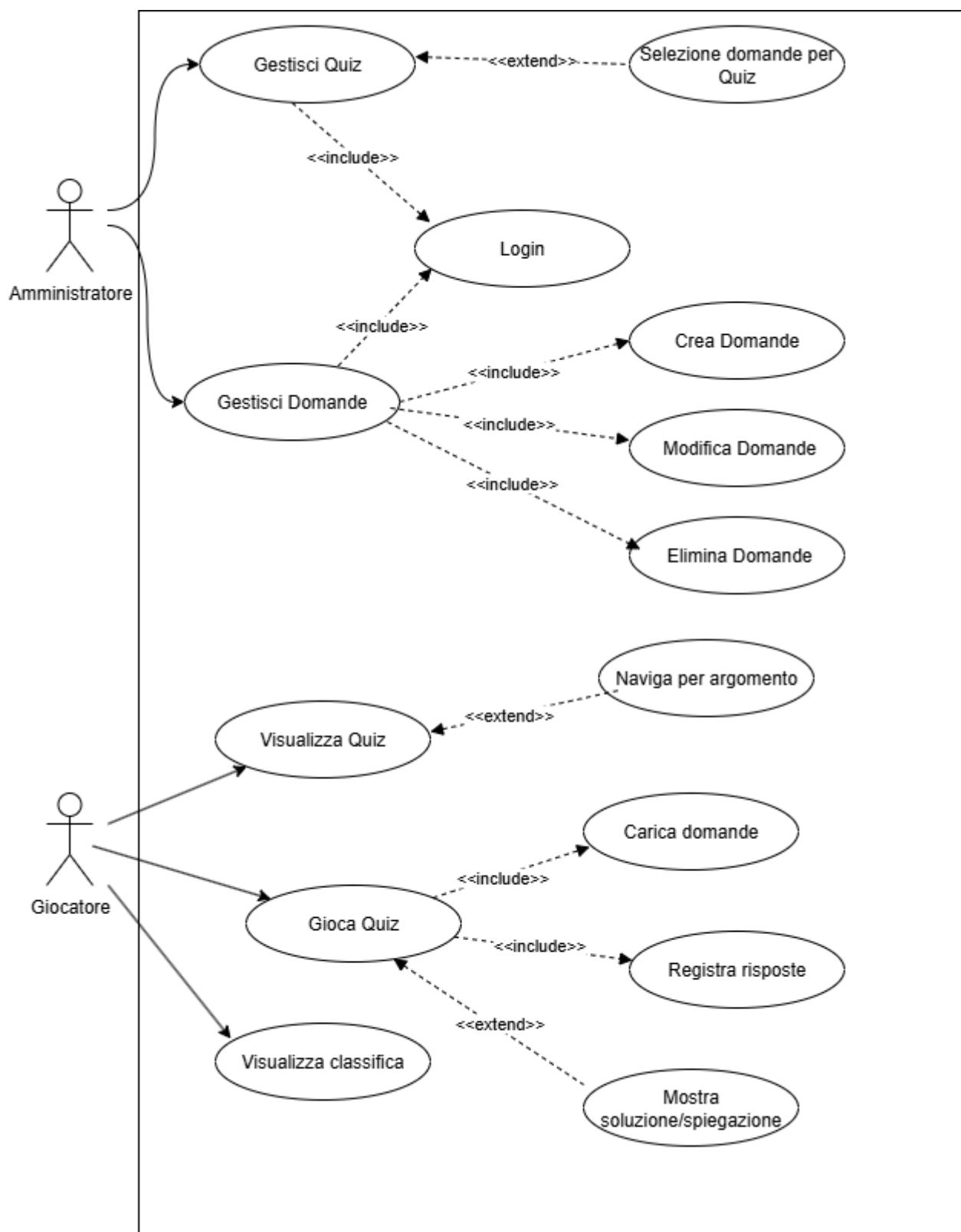


Figura 1 - Use Case

2.4 Pianificazione

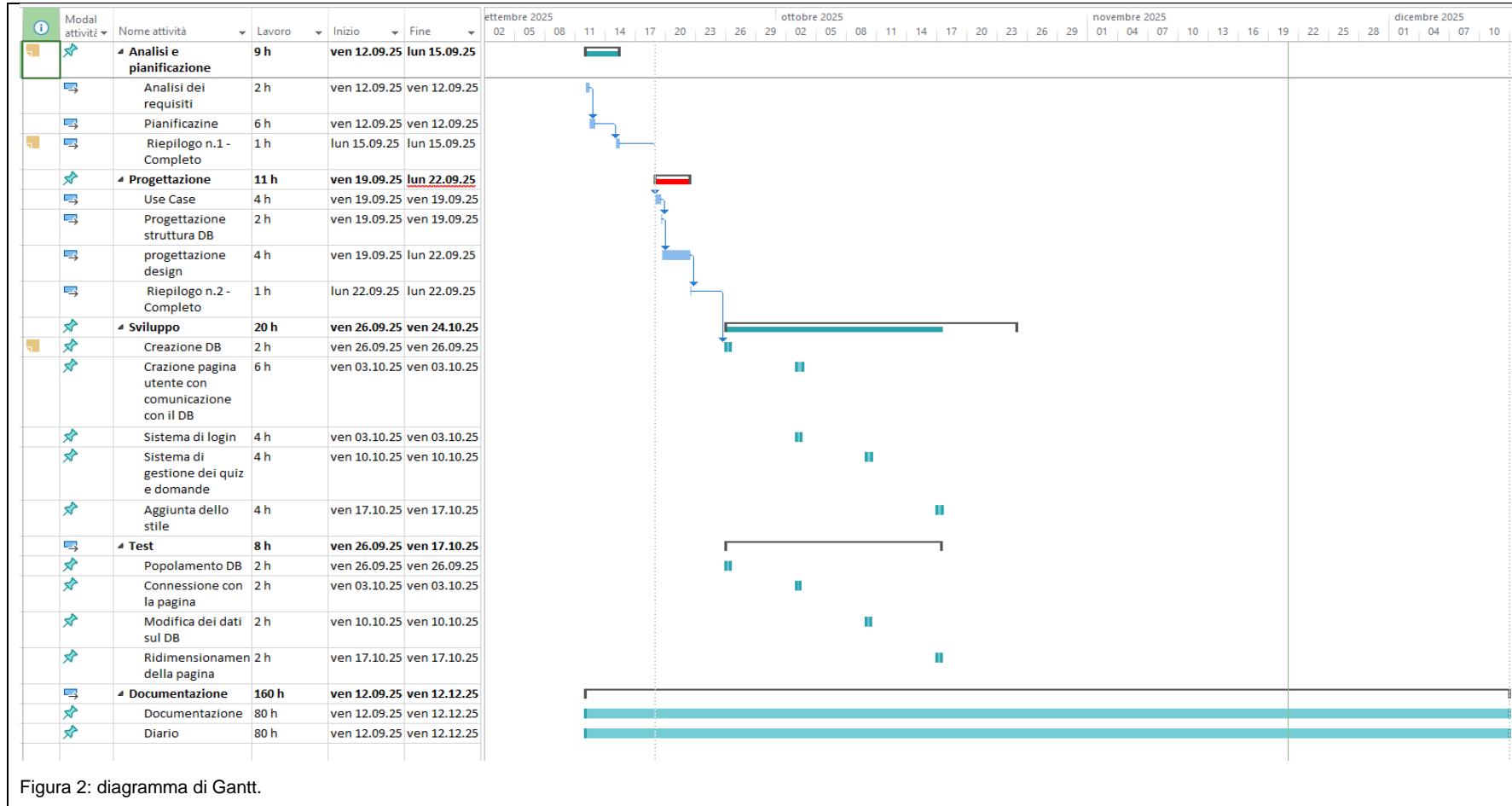


Figura 2: diagramma di Gantt.

2.5 Analisi dei mezzi

2.5.1 Software

Per realizzare questo progetto sono stati utilizzati diversi strumenti, ognuno ha avuto un ruolo specifico per la creazione dell'applicazione web, qui di seguito sono elencati e descritti i vari software utilizzati:

- **Visual Studio Code 1.106**
 - Uno dei migliori editor di testo per file di codice, qui utilizzato per scrivere JavaScript, HTML, CSS e anche usato per fare ordine nella struttura del progetto.
- **MongoDB Compass**
 - Software utilizzato per visualizzare il DB con i suddetti documenti al suo interno, qui usato per vedere il risultato delle azioni dell'amministratore.
- **Node.js v22.20.0**
 - Runtime di JavaScript utilizzato per poter far eseguire codice di esso a lato server (Backend) con package manager **npm**.
- **HTML5 / CSS3**
 - Linguaggi di markup utilizzati per la costruzione dell'interfaccia grafica ed il layout delle pagine.
- **Express.js 4.19.2**
 - Framework backend usato per gestire il routing, middleware e la struttura MVC dell'applicazione.
- **Express-cookie-session**
 - Utilizzato per gestire le sessioni e l'autenticazione dell'amministratore.
- **Hbs (Handlebars)**
 - Motore di templating per la creazione di pagine HTML dinamiche che cominciano con Node.js, include anche helper personalizzati per far funzionare il progetto (eq, gte, includes).
- **Mongoose 8.18.2**
 - ODM per far interagire Node.js con MongoDB tramite schemi e modelli.
- **MongoDB**
 - DB NoSQL utilizzato per salvare: informazioni dell'utente amministratore, quiz (con associate le domande), domande.
- **Git + GitHub**
 - Utilizzati per il versioning del progetto e salvataggio del progetto in esterno con condivisione con la professoressa.

2.5.2 Hardware

2.5.2.1 Piattaforma di esecuzione

Il progetto è pensato per essere eseguito su qualsiasi piattaforma che supporti Node.js e dei browser moderni.

Quindi compatibile con:

- Windows 10/11
- macOS
- Linux (Ubuntu/Arch/Mint)

La parte server che fa da host per l'applicazione richiede esclusivamente:

- Node.js installato
- Accesso ad un database MongoDB locale o remoto

La parte client richiede:

- Un browser moderno (Chrome, Firefox, Opera, Brave)

2.5.2.2 Hardware utilizzato per lo sviluppo

- **PC scolastico con le seguenti specifiche**
 - CPU: 13th Gen Intel(R) Core(TM) i7-13700 2.10 GHz
 - RAM: 32.0 GB
 - Spazio d'archiviazione: 512 GB SSD
 - GPU: Nvidia T400 4GB
 - OS: Windows 11 Education 23H2
- **Limitazioni hardware**
 - Nessun hardware dedicato richiesto
 - GPU non richieste ne risorse particolari
 - Il DB è ospitato localmente: macchine con poca memoria (meno di 4GB) potrebbero avere performance ridotte

3 Progettazione

3.1 Design dell'architettura del sistema

3.1.1 Struttura generale del sistema

Il progetto è una web app sviluppata con architettura **client/server**, composta da:

- **Client (front-end)**
 - Browser web
 - Pagine HTML generate tramite Handlebars (HBS)
 - CSS personalizzato
 - Moduli per interazione con quiz, domande e login admin
- **Server (back-end)**
 - Applicazione Node.js con framework Express
 - Routing server-side
 - Middleware per sessioni e autenticazione
 - Gestione CRUD (Create, Read, Update, Delete) di quiz e domande
 - Comunicazione con il database
- **Database**
 - MongoDB (NoSQL)
 - Raccolte principali:
 - Users
 - Questions
 - Quizzes

3.1.2 Moduli/componenti applicativi

| Componente | Funzione |
|---------------------|--|
| app.js | Server principale, setup middleware, routing, avvio |
| routes/ | Definizione delle rotte per pagine pubbliche, admin e autenticazione |
| models/ | Schema Mongoose per le raccolte MongoDB |
| views/ | Template Handlebars per l'interfaccia |
| public/ | File statici (in questo caso solo il file CSS) |
| middleware/admin.js | Controllo accesso area admin |

3.1.3 Flusso di dati principale

Utente non autenticato

1. Visita il sito
2. Visualizza la lista dei quiz
3. Sceglie un quiz
4. Risponde alle domande
5. Il server calcola:
 - Risposte corrette
 - Punteggio
 - Valutazione
6. Viene mostrata la schermata finale dei risultati

Amministratore

1. Effettua login
2. Accede alla dashboard
3. Può eseguire CRUD su:
 - Domande
 - Quiz

3.1.4 Site map

Public:

- /
- /quiz
- /quiz/:id
- /quiz/:id/result

Admin

- /login
- /admin
- /admin/questions/new
- /admin/questions/:id/edit
- /admin/quizzes/new
- /admin/quizzes/:id/edit

3.2 Design dei dati e database

3.2.1 Schema logico

Qui di seguito è mostrato lo schema logico del mio DB mongoDB che ho utilizzato per il progetto.

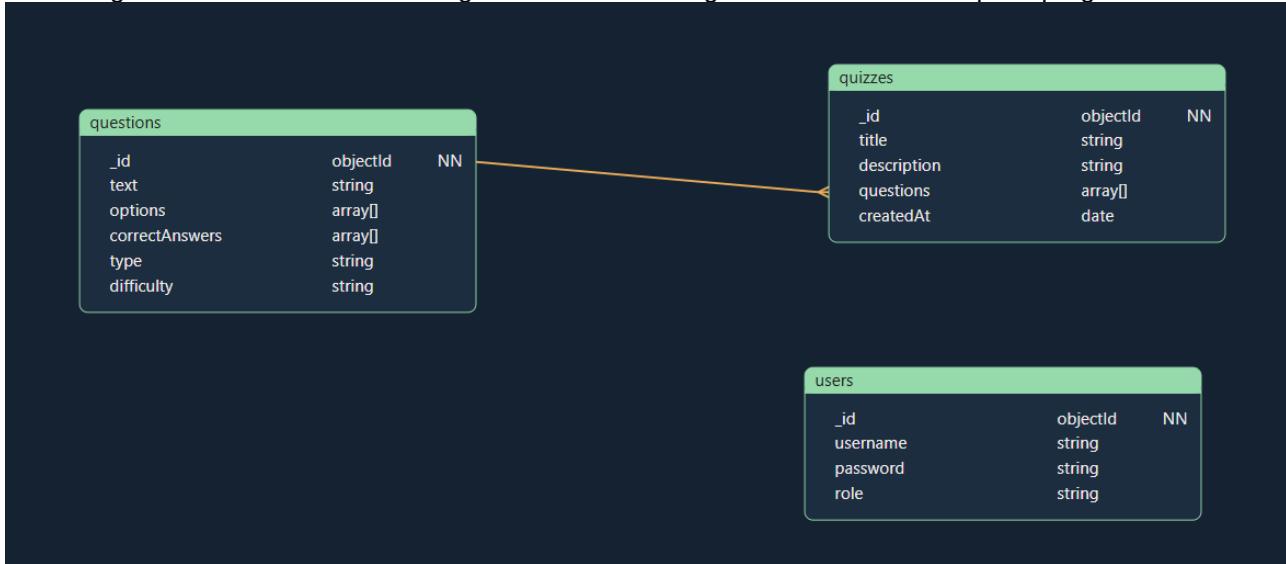


Figura 3 - Diagramma E-R del DB

La collezione **questions** conterrà un ID identificativo, il testo della domanda, un array di opzioni tra cui si potrà scegliere, un **array** che conterrà gli indici delle risposte corrette, ho messo array perché in caso la domanda fosse risposta multipla può così contenere più indici, il tipo della domanda che potrà essere a risposta **sigola o multipla** e la difficoltà della domanda così da poter dire all'utente che sta giocando se la domanda è facile, media o difficile.

Poi ho la collezione quizzes che conterrà un ID identificativo, il titolo e la descrizione del quiz, un array degli ID delle domande che sono contenute nel quiz così che nel codice posso andare a prendere tutte le informazioni della domanda partendo dall'indice e la data di creazione del quiz.

3.3 Design delle interfacce

Qui di seguito sono elencati e descritti i mockup di tutte le pagine del progetto.

Home:

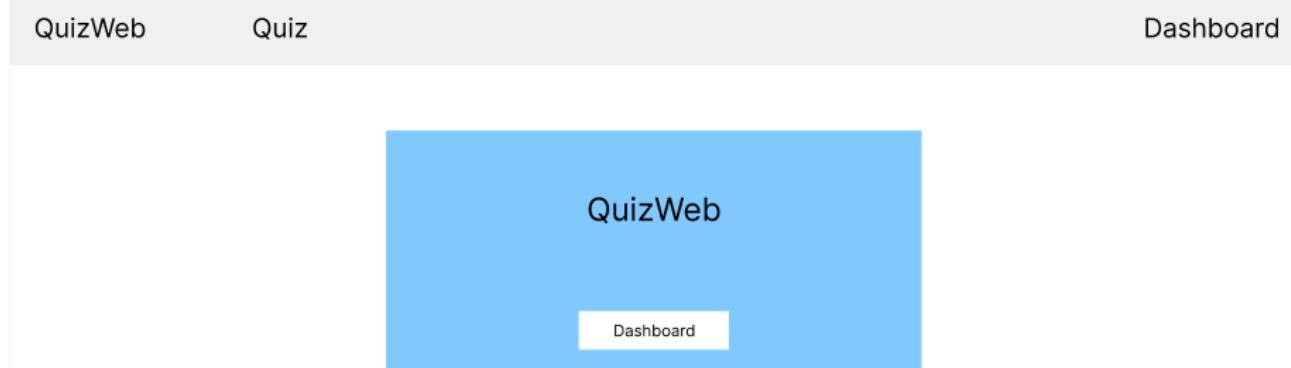


Figura 4 - Mockup Home

Una pagina semplice che dà il benvenuto all'utente nel sito, da qui l'utente potrà andare alla lista dei quiz o se è loggato come amministratore potrà andare alla dashboard.

Lista dei Quiz:

The mockup shows a header bar with three tabs: "QuizWeb", "Quiz", and "Dashboard". Below this, there are three separate card-like boxes, each containing "Titolo" and "Descrizione" fields and a "Gioca" button at the bottom.

| Titolo | Descrizione | Gioca |
|--------|-------------|-------|
| Titolo | Descrizione | Gioca |
| Titolo | Descrizione | Gioca |

Figura 5 - Mockup lista dei quiz

Qui verrà mostrata una lista di tutti i quiz disponibili nel DB e l'utente potrà scegliere quale giocare.

Pagina di gioco:

QuizWeb

Quiz

Dashboard

Titolo

Descrizione

Domanda

Opzione

Opzione

Opzione

Opzione

Invia risposte

Figura 6 - Mockup pagina di gioco

In questa pagina verranno caricate le domande del quiz che ha scelto l'utente, ogni domanda avrà delle opzioni tra cui scegliere e alla fine del quiz l'utente invierà le risposte al sistema che le verificherà.

Pagina dei risultati:

QuizWeb

Quiz

Dashboard

Risultati

Percentuale

Giuste

Parziali

Sbagliate

[Torna alla lista](#)

Figura 7 - Mockup pagina risultati

In questa pagina verrà mostrato il risultato del quiz dell'utente mostrando quante risposte sono giuste, quante parzialmente giuste e quante sbagliate e mostrerà anche la percentuale di correttezza che ha ottenuto l'utente.

Dashboard:

The dashboard interface is divided into two main sections: 'Quiz' and 'Domande'.
Quiz Section: Contains three items, each represented by a grey card with a white border:

- Titolo**: Text input field.
- Descrizione**: Text input field.
- Domande**: Text input field.

Below each card are two buttons: 'Modifica' (white background) and 'Elimina' (red background).
Domande Section: Contains three items, each represented by a grey card with a white border:

- ID Domande**: Text input field.
- Titolo**: Text input field.
- Descrizione**: Text input field.

Below each card are two buttons: 'Modifica' (white background) and 'Elimina' (red background).
At the top center of the dashboard is a button labeled 'Crea Quiz'.

Figura 8 - Mockup Dashboard

La dashboard fungerà da centro di controllo delle domande e dei quiz per l'amministratore che potrà creare nuovi quiz o domande e modificare o eliminare quelle già esistenti, in questa pagina ci potranno entrare solo gli amministratori tramite login.

Form domande:

QuizWeb

Quiz

Dashboard

Crea nuova domanda

Testo

Opzioni

Risposte corrette

Tipo

[Crea domanda](#) [Dashboard](#)

Figura 9 - Mockup form domande

In questa pagina l'amministratore potrà creare o modificare una domanda, se ne crea una nuova dovrà riempire i campi del testo, delle opzioni, delle risposte corrette e del tipo di domanda, se invece ne modifica una i campi si riempiranno automaticamente con quelli della domanda già esistente e l'amministratore dovrà solo modificare i campi che vuole.

Form quiz:

QuizWeb Quiz Dashboard

Crea nuovo Quiz

Titolo

Descrizione

Seleziona domande

Domanda

Domanda

Domanda

Crea quiz **Dashboard**

Figura 10 - Mockup form quiz

In questa pagina l'amministratore potrà creare un quiz o modificarne uno, come prima se ne crea uno dovrà mettere il titolo del quiz, la sua descrizione e scegliere le domande da mettere nel quiz, se ne modifica uno i campi verranno riempiti automaticamente e nella selezione dei quiz verranno già selezionati i quiz già associati.

Login:

QuizWeb Quiz Dashboard

Login

The mockup shows a light gray rectangular form. At the top left is the word "Username" above a white input field. Below it is the word "Password" above another white input field. At the bottom center is a dark gray button labeled "Entra". At the bottom left of the form is a link "Torna alla home".

Figura 11 - Mockup login

In questa pagina l'amministratore potrà autenticarsi per poter accedere alla Dashboard tramite username e password, se sono corretti potrà entrare nella dashboard, se no uscirà un messaggio di errore che dice username o password non corretti.

3.4 Design procedurale

3.4.1 Routing principale

| Metodo | Percorso | Descrizione |
|--------|------------------|---------------------|
| GET | / | Home page |
| GET | /quiz | Lista quiz |
| GET | /quiz/:id | Esecuzione quiz |
| POST | /quiz/:id/submit | Valutazione risorse |

3.4.2 Routing admin

| Metodo | Percorso | Funzione |
|--------|-----------------------------|-----------------------|
| GET | /admin | Dashboard |
| GET | /admin/questions/new | Form nuova domanda |
| POST | /admin/questions/new | Crea domanda |
| GET | /admin/questions/:id/edit | Form modifica domanda |
| POST | /admin/questions/:id/delete | Elimina domanda |
| GET | /admin/quizzes/new | Form nuovo quiz |
| POST | /admin/quizzes/new | Salva quiz |
| POST | /admin/quizzes/:id/delete | Elimina quiz |

3.4.3 Controllo accessi

- Nessun login richiesto per giocare
- Login **obbligatorio solo per l'amministratore**
- Viene usato:
 - Session cookie-based
 - Middleware requireAuthAdmin che verifica user.role === “Admin”
- Se non valido:
 - Redirect automatico a /login

3.4.4 Logica di valutazione quiz

Per ogni domanda:

- Confronta le risposte dell’utente con la lista delle risposte corrette
- Se tutte corrette → +1 corrette
- Se parzialmente corrette → +1 parziali
- Altrimenti → +1 sbagliate

Punteggio finale = (risposte corrette / totale) * 100

Describe i concetti dettagliati dell’architettura/sviluppo utilizzando ad esempio:

- Diagrammi di flusso e Nassi.
- Tabelle.
- Classi e metodi.
- Tabelle di routing
- Diritti di accesso a condivisioni ...

Questi documenti permetteranno di rappresentare i dettagli procedurali per la realizzazione del prodotto.

4 Implementazione

4.1 Struttura progetto

Il progetto è stato sviluppato utilizzando una tipica struttura MVC (Model – View - Controller), tipica delle applicazioni Node.js con Express:

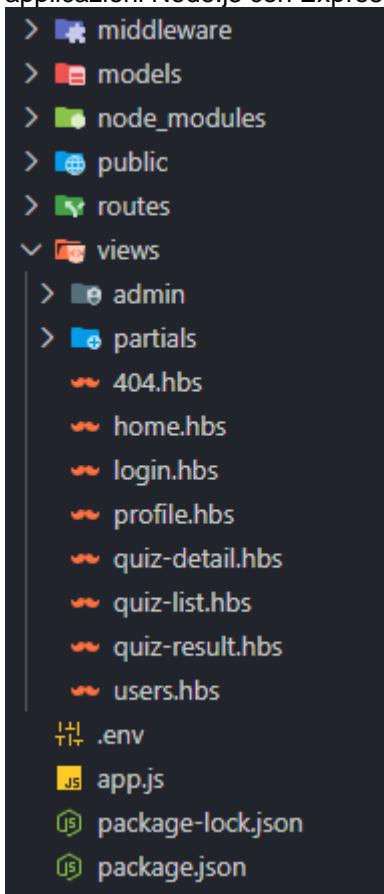


Figura 12 - Struttura progetto

app.js

È il file principale dell'applicazione.

Qui vengono:

- Inizializzati Express e i middleware,
- Configurato Handlebars come engine delle viste
- Importa le route,
- Avvia il server su una porta specifica (3000)
- Registra helper personalizzati

.env

File utilizzato per salvare variabili d'ambiente, ad esempio:

```
! .env
1 PORT=3000
2 SESSION_KEYS=secret1,secret2
3 MONGO_URI=mongodb://127.0.0.1:27017/quizdb
```

Figura 13 - File .env

/models

Contiene **Schemi mongoose** utilizzati per rappresentare i dati nel database MongoDB:

- User.js → schema utenti
- Quiz.js → schema quiz
- Question.js → schema domande

Sono componenti che definiscono la struttura dei dati in modo centralizzato

/middleware

Contiene il file **auth.js** con funzioni che vengono eseguite tra la richiesta e la risposta:

- Controllo login,
- Controllo ruolo admin,
- Salvataggio sessione

/public

Contiene i file statici (file che rimango così), nel mio caso styles.css per lo stile delle pagine del sito

/views

Questa cartella contiene tutte le pagini visualizzabili dal browser, realizzate in Handlebars (.hbs)

/views/partials

Contiene componenti grafici riutilizzabili in più pagine, come **_nav.hbs** per la barra di navigazione e **_head.hbs** per la sezione <head> delle pagine.

/views/admin

Contiene pagine riservate alla gestione dei contenuti da parte dell'amministratore, nel mio caso la gestione di quiz e domande.

4.2 Implementazione del Database

Il database scelto è MongoDB, che è un database NoSQL, ciò vuol dire che non utilizza la classica struttura di tabelle solide e le relazioni. MongoDB essendo un DB NoSQL ha delle collezioni, che sarebbero le tabelle e i documenti che sarebbero i dati, noi possiamo dare una struttura alla collezione ma i documenti non sono vincolati a rispettarla, ad esempio potrei avere un documento con nome, cognome, eta e diploma ed un altro con solo i primi tre e non crea problemi. Ho scelto questo tipo di DB proprio per la struttura dei dati flessibile e facilmente estendibile.

Per far lavorare MongoDB e Node.js si usa il package Mongoose che ci permette di collegarci a MongoDB, creare schemi e manipolare i documenti all'interno del DB.

Di seguito c'è un esempio di uno schema MongoDB del progetto.

```
models > question.js > questionSchema > difficulty > enum
 1 const mongoose = require("mongoose")
 2
 3 const questionSchema = new mongoose.Schema({
 4   text: {type: String, required: true},
 5   options: [{type: String, required: true}],
 6   correctAnswers: [{type: Number, required: true}],
 7   type: {type: String, enum: ["single", "multiple"], default: "single"},
 8   difficulty: {type: String, enum: ["facile", "media", "difficile"], default: "media"}
 9 }
10
11 module.exports = mongoose.model("Questions", questionSchema)
```

Figura 14 - Schema domande

Come vedete io dico che lo schema delle domande ha bisogno obbligatoriamente del testo, delle opzioni e della/delle risposte corrette, mentre se non si mettono gli altri due campi, riceveranno un valore di default.

4.3 Implementazione del pannello di amministrazione

Il pannello admin, che solo lui può usare, consente:

- Gestione dei quiz (creazione, modifica ed eliminazione)
- Gestione delle domande (creazione, modifica ed eliminazione)

Tutte queste operazioni sono contenute nel file route admin.js dove per ogni richiesta viene chiamato il metodo adatto per essa, di seguito vediamo come funziona la creazione delle domande.

```
30 //Mostra il form di creazione della domanda
31 router.get("/questions/new", requireAuthAdmin, async(req, res) => {
32   res.render("admin/question-form", {action: "create"})
33 })
34
35 //Crea nuova domanda
36 router.post("/questions/new", requireAuthAdmin, async(req, res) => {
37   try{
38     const {text, option, correctAnswers, type, difficulty} = req.body
39
40     const question = new Question({
41       text,
42       options: option.split(",").map(o => o.trim()),
43       correctAnswers: correctAnswers
44         .split(",")
45         .map(n => parseInt(n.trim()))
46         .filter(n => !isNaN(n)),
47       type,
48       difficulty
49     })
50
51     await question.save()
52     res.redirect("/admin")
53   }catch(err){
54     console.error("Errore creazione domanda:", err)
55     res.status(500).send("Errore nel salvataggio della domanda")
56   }
57 })
```

Figura 15 - Funzione creazione domanda

I file routes contengono tutti i metodi GET e POST che possiamo modificare, il metodo GET qua è usato chiamato se l'admin va alla pagina sottopagina **/admin/questions /new** caricherà le pagine del form per la creazione delle domande. Mentre se da quella pagina riceve qualcosa fale seguenti operazioni:

Prima di tutto prova a prendere da **req.body** i dat che arrivano dal form tramite il metodo POST, poi crea una nuova domanda riempiendo i campi necessari, il campo opzioni dato che viene passato come una stringa con delle virgolette vengono separate e trasformate in un array di stringhe che contengono le possibili risposte, stessa cosa per l'indice della risposta corretta, in caso di virgola vengono separati in un array, controlla che sia un numero e che non sia vuoto. Poi aspetta che salva la domanda nel DB e torna nella dashboard, se succede qualche errore segnala l'errore e non salva la domanda.

4.3.1 Implementazione grafica

Per l'implementazione grafica dell'interfaccia ho usato delle Handlebars (.hbs), che permettono la comunicazione tra il server che usa Node.js e il front-end che usa HTML.

Di seguito c'è una parte del file dashboard.hbs che mostra come vengono visualizzate le varie domande nella dashboard.

```
49 <section class="admin-section">
50   <div class="admin-section-header">
51     <h2>Domande</h2>
52     <a class="button primary" href="/admin/questions/new">Crea nuova domanda</a>
53   </div>
54
55   {{#if questions.length}}
56     <div class="card-grid">
57       {{#each questions}}
58         <div class="card">
59           <h3>{{this.text}}</h3>
60           <p class="meta"><strong>Tipo:</strong> {{this.type}}</p>
61           <p class="meta"><strong>Opzioni:</strong> {{this.options}}</p>
62           <p class="meta"><strong>Corrette:</strong> {{this.correctAnswers}}</p>
63           <p class="meta"><strong>Difficoltà:</strong> {{this.difficulty}}</p>
64
65           <div class="card-actions">
66             <a class="button small" href="/admin/questions/{{this._id}}/edit">Modifica</a>
67             <form action="/admin/questions/{{this._id}}/delete" method="POST" class="inline-form">
68               <button type="submit" class="danger small" onclick="return confirm('Sei sicuro di voler eliminare questa domanda?')">Elimina</button>
69             </form>
70           </div>
71         </div>
72       {{/each}}
73     </div>
74   {{else}}
75     <p class="empty">Nessuna domanda presente. <a class="button" href="/admin/questions/new">Crea una</a>.</p>
76   {{/if}}
77 </section>
78 </main>
79 </body>
80 </html>
```

Figura 16 - .hbs della dashboard, parte domande

Come vediamo è simile a PHP solo con delle sintassi leggermente diverse, all'inizio della sezione non faccio nulla di speciale solo do un titolo alla sezione con un pulsante per creare una domanda, poi controllo se ci sono già delle domande dalla variabile `questions` che viene passata dal file `admin.js` grazie al metodo seguente.

```
//DASHBOARD ADMIN
router.get("/", requireAuthAdmin, async (req, res) => {
  try {
    // Recupera tutti i quiz e domande senza usare populate
    const quizzes = await Quiz.find().lean()
    const questions = await Question.find().lean()

    //Formatta la data in dd/mm/yyyy
    quizzes.forEach(q => {
      const d = new Date(q.createdAt)
      //Crea nuovo campo nel quiz con la data formattata
      q.formattedDate = d.toLocaleDateString("it-IT", {
        day: "2-digit",
        month: "2-digit",
        year: "numeric"
      })
    })

    // Renderizza la dashboard
    res.render("admin/dashboard", {
      title: "Dashboard Admin",
      quizzes,
      questions,
      user: { name: "Administrator", role: "Admin" }
    })
  } catch (err) {
    console.error("Errore nel caricamento dashboard admin:", err)
    res.status(500).send("Errore nel caricamento della dashboard admin.")
  }
})
```

Figura 17 - Funzione di caricamento della dashboard

Come vediamo il metodo prova a prendere tutte le domande ed i quiz che ci sono nel DB e poi renderizza la pagina della dashboard con in più le variabili quizzes e questions da poter usare.

Tornando a dashboard.hbs poi se ci sono domande creo un ciclo per far sì che ogni domanda abbia la sua "carta" con dentro le sue informazioni andando a leggere gli attributi di questions.

```
{#each questions}
<div class="card">
  <h3>{{this.text}}</h3>
  <p class="meta"><strong>Tipo:</strong> {{this.type}}</p>
  <p class="meta"><strong>Opzioni:</strong> {{this.options}}</p>
  <p class="meta"><strong>Corrette:</strong> {{this.correctAnswers}}</p>
  <p class="meta"><strong>Difficoltà:</strong> {{this.difficulty}}</p>

  <div class="card-actions">
    <a class="button small" href="/admin/questions/{{this._id}}/edit">Modifica</a>
    <form action="/admin/questions/{{this._id}}/delete" method="POST" class="inline-form">
      <button type="submit" class="danger small" onclick="return confirm('Sei sicuro di voler eliminare questa domanda?')">Elimina</button>
    </form>
  </div>
</div>
{{/each}}
```

Figura 18 - Focus file .hbs parte domande

4.4 Implementazione dei quiz

Per poter giocare ai quiz ho creato una pagina che mostra una lista di tutti i quiz disponibili nel DB, di seguito è mostrato il codice.

```

2   <!DOCTYPE html>
3   <head>
4       {{>} _head title="Lista Quiz"}
5   </head>
6   <body>
7       {{>} _nav}
8
9       <main class="container">
10      <h1>Quiz disponibili</h1>
11
12      {{#if quizzes}}
13          <div class="quiz-grid">
14              {{#each quizzes}}
15                  <div class="quiz-card">
16                      <h3>{{this.title}}</h3>
17                      <p>{{this.description}}</p>
18                      <form action="/quiz/{{this._id}}" method="get">
19                          <button type="submit">Vai al quiz</button>
20                      </form>
21                  </div>
22              {{/each}}
23          </div>
24      {{else}}
25          <p>Non ci sono quiz disponibili</p>
26      {{/if}}
27  </main>
28 </body>
```

Figura 19 - .hbs lista dei quiz

Potete vedere che è molto semplice, quando entro nella pagina il sistema controlla se ci sono dei quiz, se ci sono li carica mostrando il titolo, la descrizione ed il bottone per giocare al quiz. Quando si entra nella pagina viene chiamata la seguente funzione per permettere di listare i quiz.

```

6 //Lista tutti i quiz con le domande
7 router.get("/", async (req, res) => {
8     try{
9         const quizzes = await Quiz.find()
10        console.log(quizzes)
11        res.render("quiz-list", {quizzes})
12    }catch(err){
13        res.status(500).json({error: err.message})
14    }
15})
```

Figura 20 - Funzione per caricare i quiz

Mette nella variabile **quizzes** tutti i quiz presenti e poi carica la pagina **quiz-list** dandogli la variabile quizzes da utilizzare.

Ora ho fatto una cosa simile per quando giochiamo al quiz e bisogna mostrare le varie domande, ho fatto un semplice controllo per dire se ci sono domande di caricarle mostrando testo, difficoltà e opzioni, ma c'è anche il seguente controllo.

```
 {{#if (eq this.type "single")}}
  {{#each this.options}}
    <label>
      <input type="radio" name="q{{@../index}}" value="{{@index}}">
      {{this}}
    </label><br>
  {{/each}}
{{else}}
  {{#each this.options}}
    <label>
      <input type="checkbox" name="q{{@../index}}" value="{{@index}}">
      {{this}}
    </label><br>
  {{/each}}
{{/if}}
```

Figura 21 - Funzione per caricare le domande

Questo controllo è per vedere se la domanda è una domanda con risposte multiple o no, se è multipla carica gli input per scegliere le opzioni come **checkbox** ese no come **radio button**, per caricare tutte le domande uso un ciclo foreach come ho fatto per le domande ed i quiz.

Poi prendono come nome e valore prendono l'indice dell'opzione nella domanda così poi quando si inviano le risposte si può confrontare per ogni opzione l'indice della risposta dell'utente e l'indice corretto.

4.5 Sistema di Login

Essendo che io uso Node.js, il sistema di login utilizza un middleware per garantire che soltanto l'amministratore che è loggato possa andare nella dashboard.

Qui di seguito mostro il frontend della pagina di login.

```
8   <main class="login-container">
9     <div class="login-card">
10    {{#if msg}}<p class="notice">{{msg}}</p>{{/if}}
11    <h1>Login</h1>
12    <p class="login-subtitle">Inserisci le tue credenziali per accedere alla Dashboard</p>
13    {{#if error}}<p class="error">{{error}}</p>{{/if}}
14
15    <form action="/login" method="post" class="login-form">
16      <label for="username">Username</label>
17      <input id="username" type="text" name="username" required>
18      <label for="password">Password</label>
19      <input id="password" type="password" name="password" required>
20      <button type="submit" class="button primary full-width">Entra</button>
21    </form>
22
23    <p class="login-footer"><a href="/">Torna alla home</a></p>
24  </div>
25
26  </main>
27 </body>
28 </html>
```

Figura 22 - .hbs del Login

Come vediamo è un **form** molto semplice che invierà in post a **/login** lo username e la password. Il file auth.js prenderà il post da **/login** come vediamo di seguito.

```
1  const express = require('express')
2  const { loginUser, logoutUser } = require('../middleware/auth')
3
4  const router = express.Router();
5
6  router.get("/login", async (req, res) => {
7    res.render("login", {title: "Login Amministratore"})
8  })
9
10 // Logout
11 router.get("/logout", (req, res) => {
12   req.session = null
13
14   res.redirect("/")
15 });
16
17 router.post("/login", loginUser)
18 router.post("/logout", logoutUser)
19
20 module.exports = router
```

Figura 23 - funzioni per caricare il login

Vediamo che il metodo post del login usa il metodo **loginUser** che arriva dal file middleware **auth.js** in cui ho deciso di mettere i due metodi di login e logout.
Qui di seguito è mostrato il metodo del login.

```
41 // Funzione per login
42 async function loginUser(req, res) {
43   const { username, password } = req.body;
44
45   try {
46     const user = await User.findOne({ username, password })
47
48     if (!user) {
49       return res.render("login", {
50         title: "Login Amministratore",
51         error: "Credenziali non valide."
52       });
53     }
54
55     console.log(user._id)
56     req.session.userId = user._id
57     console.log(req.session.userId)
58     res.redirect("/admin")
59   } catch (err) {
60     console.error("Errore login:", err)
61     res.status(500).send("Errore interno del server")
62   }
63 }
```

Figura 24 - Funzione middleware del Login

Vediamo che il metodo prende lo username e la password e poi prova a trovare l'utente all'interno del DB, se non esiste ritorna un errore se no salva l'utente in sessione, così che anche se cambia pagina il server sa che è sempre lo stesso utente e poi lo reindirizza alla dashboard. Se ci sono problemi ritorna un errore generale.

E poi per fare in modo che solo l'amministratore possa entrare nella dashboard ho creato un altro metodo apposta all'interno del middleware.

```
19 // Middleware che permette l'accesso solo all'amministratore
20 async function requireAuthAdmin(req, res, next) {
21   console.log(req.session.userId);
22   if (!req.session || !req.session.userId) {
23     return res.redirect("/login");
24   }
25
26   try {
27     const user = await User.findById(req.session.userId);
28     if (!user || user.role !== "Admin") {
29       return res.status(403).render("403", {
30         title: "Accesso negato",
31         msg: "Solo l'amministratore può accedere a questa sezione."
32       });
33     }
34     next();
35   } catch (err) {
36     console.error("Errore autenticazione admin:", err);
37     res.status(500).send("Errore interno del server");
38   }
39 }
```

Figura 25 - Funzione per richiedere l'admin autenticato

Vediamo che il metodo controlla che ci sia una sessione in corso, se no reindirizza alla pagina di login, poi prova a trovare l'utente e se non è admin nega l'accesso alla pagina, questa parte lo pensata se in futuro si dà la possibilità di avere un account anche per altre persone e non solo admin di controllare se è un admin quello che vuole entrare nella dashboard.

Poi questo metodo viene chiamato dai metodi delle routes di **admin.js** per controllare che chi sta provando a modificare qualcosa sia un admin.

```
10 //DASHBOARD ADMIN
11 router.get("/", requireAuthAdmin, async (req, res) => {
```

Figura 26 - Esempio di funzione che richiede admin

Come vediamo il metodo per caricare la dashboard richiama il metodo che abbiamo visto prima per vedere se è un admin quello che vuole entrare, per importare il metodo basta usare il metodo **require** di JavaScript.

```
4 const { requireAuthAdmin } = require("../middleware/auth.js")
```

Figura 27 - Come importare la funzione per richiedere l'admin

4.6 Helper personalizzati

Nel progetto ho dovuto utilizzare dei helper personalizzati per riuscire a fare dei controlli nei file .hbs per fare in modo che quello che mostri sia corretto, per registrare un nuovo helper bisogna andare in **app.js** e usare la variabile **hbs** istanziata all'inizio ed usare il metodo **registerHelper** per avere un helper da utilizzare in tutti i file hbs.

Qui di seguito mostro gli helper personalizzati che ho dovuto usare per il progetto.

```
hbs.registerHelper("eq", function (a, b){  
| return a === b  
})
```

Figura 28 - Creazione helper eq

Questo helper funge da controllo se due oggetti sono uguali tra di loro, l'ho dovuto creare perché di base hbs non ha un helper per comparare due oggetti e mi serviva per vedere se le domande erano a risposta singola o multipla così da caricare o checkbox o radio button.

```
{{#if (eq this.type "single")}}
```

Figura 29 - Esempio di uso di eq

Anche impostare subito la difficoltà della domanda esistente se si andava a modificare.

```
<select id="difficulty" name="difficulty">  
| <option value="facile" {{#if (eq question.difficulty "facile")}}selected{{/if}}>Facile</option>  
| <option value="media" {{#if (eq question.difficulty "media")}}selected{{/if}}>Media</option>  
| <option value="difficile" {{#if (eq question.difficulty "difficile")}}selected{{/if}}>Difficile</option>  
</select>
```

Figura 30 - Altro esempio di eq

Un altro helper che ho dovuto creare è l'helper **includes**.

```
hbs.registerHelper("includes", function (array, value) {  
| if(!array){  
| | return false  
| }  
  
| const arr = array.map((v) => String(v))  
| return arr.includes(String(value))  
})
```

Figura 31 - Creazione helper includes

Questo helper controlla se un array contiene un determinato valore, l'ho dovuto usare nel form per la creazione dei quiz perché quando si va a modificare un quiz le domande già presenti devono essere automaticamente selezionate e quindi ho dovuto controllare se quella determinata domanda sia contenuta dentro il quiz, se si la imposto come già selezionata se no non fa niente.

```
{{#if (includes ../quiz.questions this._id)}}checked{{/if}}
```

Figura 32 - Esempio uso includes

Infine l'ultimo helper che ho dovuto creare è stato l'helper gte.

```
hbs.registerHelper("gte", (a, b) => {
  const na = Number(a)
  const nb = Number(b)

  if(!Number.isFinite(na) || !Number.isFinite(nb)){
    return false
  }

  return na >= nb
})
```

Figura 33 - Creazione helper gte

Quest'ultimo helper serve per confrontare se un numero è maggiore o uguale ad un altro, prima controlla se i due numeri siano finiti e poi fa il controllo, l'ho dovuto usare nella pagina del risultato del quiz per fare in modo che se il punteggio fatto sia superiore o uguale ad un determinato punteggio stampa a schermo una determinata frase.

```
<div class="result-message">
  {{#if (gte score 80)}}
    Ottimo lavoro, sei stato bravissimo!!
  {{else if (gte score 60)}}
    Buon Risultato, puoi ancora migliorare!!
  {{else}}
    Questa volta ti è andato male, non arrenderti!!
  {{/if}}
</div>
```

Figura 34 - Esempio uso gte

5 Test

5.1 Protocollo di test

| | | | |
|--------------------------|--|--------------|----------|
| Test Case: | TC-001 | Nome: | Check DB |
| Riferimento: | REQ-01 | | |
| Descrizione: | Verifica che il servizio DB di MongoDB sia attivo | | |
| Prerequisiti: | <ul style="list-style-type: none">Database già creato.Server Node.js attivo. | | |
| Procedura: | <ol style="list-style-type: none">Aprire l'app servizi di windowsCercare il servizio MongoDb Server | | |
| Risultati attesi: | <ul style="list-style-type: none">Il servizio MongoDB Server è attivo | | |

| | | | |
|--------------------------|--|--------------|--|
| Test Case: | TC-002 | Nome: | Login amministratore con credenziali valide. |
| Riferimento: | REQ-02 | | |
| Descrizione: | Verifica che un amministratore registrato possa accedere. | | |
| Prerequisiti: | <ul style="list-style-type: none"> • Utente amministratore creato nel database. • Server Node.js attivo. | | |
| Procedura: | <ol style="list-style-type: none"> 1. Accedere alla pagina di login: /login. 2. Inserire username e password corretti. 3. Premere sul pulsante Entra. | | |
| Risultati attesi: | <ul style="list-style-type: none"> • Il Sistema autentica correttamente l'amministratore. | | |

| | | | |
|--------------------------|--|--------------|---------------------------------|
| Test Case: | TC-003 | Nome: | Visualizzazione della dashboard |
| Riferimento: | REQ-09 | | |
| Descrizione: | Verifica che una volta registrato possa entrare nella dashboard. | | |
| Prerequisiti: | <ul style="list-style-type: none"> • Utente amministratore loggato. • Server Node.js attivo. | | |
| Procedura: | <ol style="list-style-type: none"> 1. Dalla barra di navigazione premere il pulsante Dashboard | | |
| Risultati attesi: | <ul style="list-style-type: none"> • Il Sistema carica correttamente la Dashboard con tutti i quiz e le domande | | |

| | | | |
|--------------------------|--|--------------|--------------------------------------|
| Test Case: | TC-004 | Nome: | Nome amministratore e cambio bottone |
| Riferimento: | REQ-11 | | |
| Descrizione: | Verifica che quando un amministratore fa il login nella barra di navigazione appaiono il pulsante per andare nella dashboard, lo username dell'amministratore e il pulsante di login cambia in logout | | |
| Prerequisiti: | <ul style="list-style-type: none"> • Utente amministratore creato nel database. • Server Node.js attivo. | | |
| Procedura: | <ol style="list-style-type: none"> 1. Ripetere I passaggi del TC-002 | | |
| Risultati attesi: | <ul style="list-style-type: none"> • Il pulsante di login cambia in logout • Lo username dell'amministratore appare nella navbar • Appare il pulsante per andare alla dashboard | | |

| | | | |
|--------------------------|---|--------------|--|
| Test Case: | TC-005 | Nome: | Accesso non autorizzato alle pagine admin. |
| Riferimento: | REQ-02 | | |
| Descrizione: | Verifica che un utente non autenticato non possa accedere direttamente alle pagine protette. | | |
| Prerequisiti: | <ul style="list-style-type: none"> Utente non loggato. | | |
| Procedura: | <ol style="list-style-type: none"> Inserire manualmente nel browser l'URL: localhost:3000/admin. Ripetere la prova con i form di creazione e modifica delle domande e dei quiz. | | |
| Risultati attesi: | <ul style="list-style-type: none"> Redirect immediate al login. Nessun acceso al contenuto riservato. | | |

| | | | |
|--------------------------|---|--------------|---|
| Test Case: | TC-006 | Nome: | Creazione domanda con opzioni multiple. |
| Riferimento: | REQ-03 | | |
| Descrizione: | Verifica che il form di creazione delle domande consenta l'inserimento di domande di tipo diverso. | | |
| Prerequisiti: | <ul style="list-style-type: none"> Utente amministratore loggato. | | |
| Procedura: | <ol style="list-style-type: none"> Accedere al form di creazione della nuova domanda premendo il pulsante Crea nuova domanda. Inserire i campi richiesti e nel tipo scegliere l'opzione multipla Salvare la domanda. | | |
| Risultati attesi: | <ul style="list-style-type: none"> La domanda appare nella lista delle domande esistenti e nel tipo della domanda esce scritto multipla | | |

| | | | |
|--------------------------|---|--------------|--|
| Test Case: | TC-007 | Nome: | Creazione nuovo quiz con domande associate |
| Riferimento: | REQ-06. | | |
| Descrizione: | Test del Sistema di creazione quiz con associazione di più domande | | |
| Prerequisiti: | <ul style="list-style-type: none"> Almeno 3 domande già esistenti. Utente admin loggato. | | |
| Procedura: | <ol style="list-style-type: none"> Accedere al form di creazione del quiz premendo il pulsante Crea nuovo quiz. Inserire titolo e descrizione del quiz. Selezionare alcune domande da associare nel quiz tramite il campo apposite. Salvare. | | |
| Risultati attesi: | <ul style="list-style-type: none"> Il quiz appare nella dashboard mostrando le informazioni su di esso e gli ID delle domande associate. | | |

| | | | | |
|--------------------------|---|---|---|--|
| Test Case: | TC-008 | Nome: | Visualizzazione della lista dei quiz da giocare con data di creazione | |
| Riferimento: | REQ-12. | Descrizione: Verificare la corretta visualizzazione dei quiz nella pagina della lista dei quiz e la data di creazione. | | |
| Prerequisiti: | <ul style="list-style-type: none"> Almeno un quiz già salvato | | | |
| Procedura: | <ol style="list-style-type: none"> Accedere alla pagina della lista dei quiz tramite il pulsante nella navbar Quiz. Controllare se vengono visualizzati i quiz e le date di creazione. | | | |
| Risultati attesi: | <ul style="list-style-type: none"> Visualizzazione corretta e formattata dei quiz e la data di creazione nel formato dd/mm/yyyy | | | |

| | | | | |
|--------------------------|---|--|--|--|
| Test Case: | TC-009 | Nome: | Risposta a quiz con più tipi di domande. | |
| Riferimento: | REQ-10. | Descrizione: Test della corretta generazione delle domande e dei risultati. | | |
| Prerequisiti: | <ul style="list-style-type: none"> Quiz già esistente con dentro risposte single e multiple. | | | |
| Procedura: | <ol style="list-style-type: none"> Dalla pagina della lista di quiz scegliere un quiz e premere il tasto Gioca del quiz scelto. Verificare che le domande singole abbiano I radio button e le domande multiple abbiano I checkbox. | | | |
| Risultati attesi: | <ul style="list-style-type: none"> Il Sistema ha caricato le domande con gli input corretti. | | | |

| | | | | |
|--------------------------|---|---|--------------------------|--|
| Test Case: | TC-010 | Nome: | Invio risposte del quiz. | |
| Riferimento: | REQ-13. | Descrizione: Test del corretto calcolo del punteggio delle risposte dei quiz | | |
| Prerequisiti: | <ul style="list-style-type: none"> Quiz già esistente con dentro risposte single e multiple. Pagina di gioco presente | | | |
| Procedura: | <ol style="list-style-type: none"> Scegliere un quiz da giocare Giocare il quiz Alla fine premere il tasto Invio. | | | |
| Risultati attesi: | <ul style="list-style-type: none"> Il Sistema ha calcolato bene il punteggio Mostra quante risposte giuste e quante sbagliate ci sono | | | |

| | | | |
|--------------------------|--|--------------|---|
| Test Case: | TC-011 | Nome: | Eliminazione domanda che c'è in un quiz |
| Riferimento: | REQ-05. | | |
| Descrizione: | Verificare che se elimino una domanda che si trova anche in un quiz viene eliminata anche dal quiz | | |
| Prerequisiti: | <ul style="list-style-type: none"> • Quiz contenente delle domande • Amministratore loggato nella dashboard | | |
| Procedura: | <ol style="list-style-type: none"> 1. Nella dashboard scegliere una domanda che si trova in un quiz. 2. Premere il pulsante elimina della domanda. 3. Confermare l'eliminazione. | | |
| Risultati attesi: | <ul style="list-style-type: none"> • Il Sistema rimuove la domanda sia dal DB che dal quiz a cui apparteneva | | |

| | | | |
|--------------------------|---|--------------|-------------------------|
| Test Case: | TC-012 | Nome: | Modifica di una domanda |
| Riferimento: | REQ-04. | | |
| Descrizione: | Verificare che la funzione di modifica delle domande funzioni | | |
| Prerequisiti: | <ul style="list-style-type: none"> • Una domanda già esistente • Amministratore loggato nella dashboard | | |
| Procedura: | <ol style="list-style-type: none"> 1. Nella dashboard scegliere una domanda che si trova in un quiz. 2. Premere il pulsante Modifica della suddetta domanda 3. Modificare un campo a piacimento 4. Salvare | | |
| Risultati attesi: | <ul style="list-style-type: none"> • Il Sistema salva la modifica effettuata alla domanda | | |

| | | | |
|--------------------------|---|--------------|---------------------|
| Test Case: | TC-013 | Nome: | Modifica di un quiz |
| Riferimento: | REQ-07. | | |
| Descrizione: | Verificare che la funzione di modifica dei quiz funzioni | | |
| Prerequisiti: | <ul style="list-style-type: none"> • Una quiz già esistente • Amministratore loggato nella dashboard | | |
| Procedura: | <ol style="list-style-type: none"> 1. Scegliere un quiz da modificare 2. Premere il pulsante Modifica del suddetto quiz 3. Cambiare un campo a piacimento 4. Salvare | | |
| Risultati attesi: | <ul style="list-style-type: none"> • Il Sistema salva la modifica effettuata alla domanda e non cambia la data di creazione del quiz. | | |

| | | | |
|--------------------------|--|--------------|-------------------------|
| Test Case: | TC-014 | Nome: | Eliminazione di un quiz |
| Riferimento: | REQ-08. | | |
| Descrizione: | Verificare che la funzione di eliminazione dei quiz funzioni | | |
| Prerequisiti: | <ul style="list-style-type: none">• Una quiz già esistente• Amministratore loggato nella dashboard | | |
| Procedura: | <ol style="list-style-type: none">1. Scegliere il quiz da eliminare2. Premere il pulsante Elimina del sudetto quiz3. Confermare l'eliminazione | | |
| Risultati attesi: | <ul style="list-style-type: none">• Il Sistema elimina il quiz e non deve più apparire nella dashboard e le domande associate ad esso non engono eliminate | | |

5.2 Risultati test

| Test Case | Risultato | Note |
|-----------|-----------|--|
| TC-001 | Superato | Il servizio è attivo |
| TC-002 | Superato | Il login riesce |
| TC-003 | Superato | La dashboard viene visualizzata |
| TC-004 | Superato | Gli oggetti nella navbar cambiano |
| TC-005 | Superato | Un utente normale non riesce ad entrare nelle pagine riservate |
| TC-006 | Superato | La domanda viene creata |
| TC-007 | Superato | Il quiz viene creato con le domande associate |
| TC-008 | Superato | I quiz vengono visualizzati correttamente e la data è nel formato corretto |
| TC-009 | Superato | Le domande vengono generate nel modo corretto |
| TC-010 | Superato | Il sistema calcola il punteggio nel modo corretto |
| TC-011 | Superato | La domanda viene eliminata anche dai quiz a cui è associata |
| TC-012 | Superato | La domanda viene modificata |
| TC-013 | Superato | Il quiz viene modificato mantenendo la data di creazione |
| TC-014 | Superato | Il quiz viene eliminato e le domande rimangono intatte |

5.3 Mancanze/limitazioni conosciute

Le seguenti caratteristiche sono attualmente mancanti o in parte limitate nel progetto:

- **Design mobile non completamente ottimizzato:** Il design delle pagine visualizzate da mobile non sono completamente ottimizzate e potrebbero creare problemi di layout nella visualizzazione.
- **Mancanza di revisione del quiz:** Mancanza di un sistema per visualizzare il proprio quiz fatto vedendo cosa si è sbagliato concretamente.

6 Consuntivo

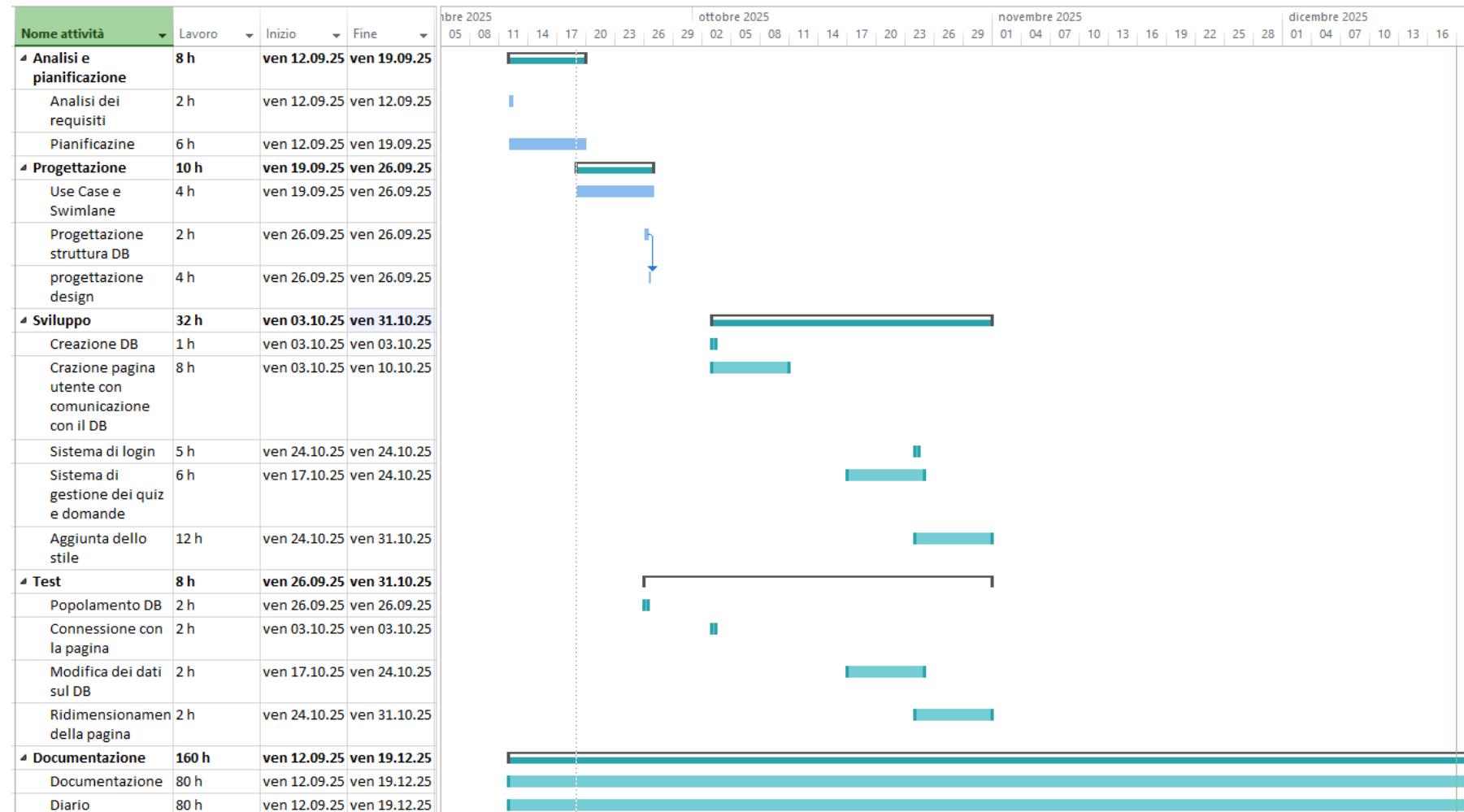


Figura 35 - Gantt Consuntivo

7 Conclusioni

Il progetto realizzato ha portato allo sviluppo di una piattaforma web per la gestione e compilazione di quiz con una chiara separazione tra area pubblica per gli utenti normali e l'area amministrativa. L'obiettivo principale era quello di creare un sistema semplice ed intuitivo, capace di permettere agli utenti di svolgere quiz senza doversi autenticare e agli amministratori di gestire i contenuti in modo controllato e sicuro.

Dal punto di vista funzionale, il progetto soddisfa tutti i requisiti iniziali: consente la creazione, la modifica e l'eliminazione di domande e quiz da parte dell'amministratore. La compilazione dei quiz da parte degli utenti e il calcolo automatico dei risultati.

L'impatto del progetto non è rivoluzionario né destinato a "cambiare il mondo" siccome esistono molte piattaforme online per giocare a quiz di vario tipo, ma rappresenta un prodotto completo, coerente e funzionante.

Nel complesso, il lavoro svolto non è stata una perdita di tempo, ma mi ha permesso di consolidare conoscenze già apprese e apprendere nuove conoscenze nell'ambito dello sviluppo web moderno.

7.1 Sviluppi futuri

Il progetto, nella sua versione attuale, può essere esteso e migliorato in diversi modi.

Tra i possibili sviluppi futuri sono possibili:

- Introduzione di un **sistema facoltativo di registrazione utenti** con il salvataggio delle loro statistiche e di molti altri dati.
- Implementazione di **statistiche avanzate** utilizzando la prima implementazione si potrebbero mostrare la media dei punteggi dei quiz, i quiz più giocati, ecc.
- Implementazione di **un'interfaccia grafica mobile completamente funzionante**.
- Introduzione di un sistema di **categorie o tag** per i quiz.

Questi sviluppi permetterebbero di trasformare il progetto in un prodotto più completo e pronto per un utilizzo reale.

7.2 Considerazioni personali

Questo progetto è stato molto formativo per me, poiché ho potuto apprendere lo sviluppo di un applicazione web e come funziona l'intero ciclo di un progetto: dall'analisi dei requisiti alla progettazione, dall'implementazione ai test e la documentazione finale.

In concreto ho imparato dal progetto:

- A strutturare correttamente un progetto Node.js;
- Ad utilizzare MongoDB per la gestione dei dati;
- Ad implementare sistemi di autenticazione e middleware;
- A documentare in modo chiaro ed ordinato.

Nel complesso, l'esperienza è stata positiva e utile per imparare la metodologia di sviluppo di un progetto, rappresentando un importante passo avanti nel mio percorso di crescita come informatico.

8 Glossario

| Termine | Descrizione |
|-------------------------|---|
| Admin | Utente con privilegi di amministrazione che può creare, modificare ed eliminare quiz e domande. |
| Architettura MVC | Modello architettonico che separa l'applicazione in Model, View e Controller, migliorando organizzazione e manutenibilità del codice. |
| Cookie | Piccola porzione di dati salvata nel browser dell'utente, utilizzata per mantenere informazioni di sessione. |
| cookie-session | Middleware di Express che gestisce le sessioni tramite cookie firmati. |
| CSS | Cascading Style Sheets: linguaggio utilizzato per definire lo stile e l'aspetto grafico delle pagine web. |
| Dashboard | Pannello di controllo riservato all'amministratore per la gestione dei contenuti del sito. |
| Express.js | Framework per Node.js utilizzato per la gestione delle rotte e delle richieste HTTP. |
| Handlebars (HBS) | Template engine che consente di generare HTML dinamico utilizzando variabili e helper. |
| Helper | Funzione personalizzata di Handlebars utilizzata per eseguire logica all'interno dei template. |
| HTML | HyperText Markup Language: linguaggio di markup utilizzato per strutturare le pagine web. |
| Middleware | Funzione che intercetta le richieste HTTP prima che raggiungano la route finale, utilizzata per controlli e trasformazioni dei dati. |
| MongoDB | Database NoSQL orientato ai documenti, utilizzato per salvare quiz, domande e utenti. |
| Mongoose | Libreria ODM per MongoDB che permette di definire schemi e modelli in Node.js. |
| Node.js | Ambiente di esecuzione JavaScript lato server. |
| Quiz | Insieme di domande che l'utente può compilare per ottenere un punteggio finale. |
| Route | Percorso che definisce come il server risponde a una specifica richiesta HTTP. |
| Sessione | Meccanismo che permette di mantenere lo stato di un utente tra più richieste. |
| Template Engine | Strumento che permette di generare HTML dinamico partendo da template e dati. |
| User | Utente registrato nel sistema, in questo progetto utilizzato solamente per l'amministratore. |
| POST / GET | Metodi HTTP utilizzati rispettivamente per inviare dati al server e per richiederli. |

9 Bibliografia

9.1 Sitografia

<https://mongomodeler.com/editor.html>, Mongo Modeler – Community Preview
<https://www.w3schools.com/nodejs/default.asp>, Node.js Tutorial
<https://www.w3schools.com/mongodb/index.php>, MongoDB Tutorial
<https://nodejs.org/en>, Node.js
<https://www.mongodb.com/>, MongoDB
<https://chatgpt.com/>, ChatGPT
<https://stackoverflow.com/questions/tagged/node.js?tab>Newest>, Stack Overflow

10 Indice delle immagini

| | |
|---|----|
| Figura 1 - Use Case..... | 9 |
| Figura 2: diagramma di Gantt..... | 10 |
| Figura 3 - Diagramma E-R del DB..... | 14 |
| Figura 4 - Mockup Home | 15 |
| Figura 5 - Mockup lista dei quiz | 16 |
| Figura 6 - Mockup pagina di gioco | 17 |
| Figura 7 - Mockup pagina risultati | 18 |
| Figura 8 - Mockup Dashboard | 19 |
| Figura 9 - Mockup form domande | 20 |
| Figura 10 - Mockup form quiz | 21 |
| Figura 11 - Mockup login | 22 |
| Figura 12 - Struttura progetto | 24 |
| Figura 13 - File .env | 24 |
| Figura 14 - Schema domande | 25 |
| Figura 15 - Funzione creazione domanda | 26 |
| Figura 16 - .hbs della dashboard, parte domande | 27 |
| Figura 17 - Funzione di caricamento della dashboard | 28 |
| Figura 18 - Focus file .hbs parte domande..... | 28 |
| Figura 19 - .hbs lista dei quiz..... | 29 |
| Figura 20 - Funzione per caricare i quiz | 29 |
| Figura 21 - Funzione per caricare le domande..... | 30 |
| Figura 22 - .hbs del Login | 31 |
| Figura 23 - funzioni per caricare il login..... | 31 |
| Figura 24 - Funzione middleware del Login | 32 |
| Figura 25 - Funzione per richiedere l'admin autenticato | 33 |
| Figura 26 - Esempio di funzione che richiede admin | 33 |
| Figura 27 - Come importare la funzione per richiedere l'admin | 33 |
| Figura 28 - Creazione helper eq..... | 34 |
| Figura 29 - Esempio di uso di eq..... | 34 |
| Figura 30 - Altro esempio di eq | 34 |
| Figura 31 - Creazione helper includes..... | 34 |
| Figura 32 - Esempio uso includes | 34 |
| Figura 33 - Creazione helper gte | 35 |
| Figura 34 - Esempio uso gte | 35 |
| Figura 35 - Gantt Consuntivo | 42 |