Politecnico
di Bari

MANUALE TECNICO "MYEXAM"

Marco Rizzi
Potito Aghilar

14 MAGGIO 2019

Indice

MANUALE TECNICO “myExam”

1. Configurazione sistema

Affinché il sistema possa essere utilizzato, è necessario impostare i parametri di configurazione e impostare la base di dati.

a. Connessione al database

Affinché “myExam” possa funzionare è necessario che disponga di un database.
Per questo motivo, è stato definito il file database.php memorizzato in /www/api (o htdocs/api a seconda della configurazione del server) che contiene i parametri di connessione.
I principali parametri sono:

- **host**: indica l’indirizzo ip del database (generalmente localhost)
- **port**: indica la porta del database (generalmente 3306)
- **username**: indica il nome utente dell’utente del database
- **password**: indica la password dell’utente del database
- **database**: indica il database da utilizzare

```
const config = [
    'host' => 'localhost',
    'port' => '8889',
    'username' => 'root',
    'password' => 'root',
    'database' => 'esame',
];
```

b. Creazione Database

Attraverso la funzione di importazione del DBMS, è necessario importare il file database.sql contenuto in /setup/ che, consente di creare il database “esame” con la relativa struttura (le tabelle).

c. Inserimento titolo esame

All’interno della cartella /www/api/requires/, è possibile trovare il file config.php. All’interno di questo, è possibile trovare i seguenti parametri:

- **Questions**: permette di definire il numero delle domande;
- **Timequestion**: permette di definire il numero di secondi per ciascuna domanda;
- **Maxtime**: restituisce il tempo totale del quiz;
- **examTitle**: permette di definire il nome dell’esame.

d. Caricamento domande

Affinché sia possibile importare le domande ,con le relative risposte, è necessario inserire nella cartella setup il file questions.xml.
Il file dovrà avere la seguente struttura:

```
<?xml version="1.0" encoding="UTF-8"?>
<questions>
	<question>
		<text>Testo Domanda 1</text>
		<correctAnswer>Risposta 1</correctAnswer>
		<answer>Risposta2</answer>
		<answer>Risposta3</answer>
		<answer>Risposta4</answer>
		<answer>Risposta5</answer>
	</question>
…
<question>
		<text>Testo Domanda N</text>
		<correctAnswer>Risposta 1</correctAnswer>
		<answer>Risposta2</answer>
		<answer>Risposta3</answer>
		<answer>Risposta4</answer>
		<answer>Risposta5</answer>
	</question>
</questions>
```

2. Inizializzazione sistema (con importazione delle domande)

Recandosi all'indirizzo

http://127.0.0.1:numeroporta/admin/install.php

sarà possibile inizializzare il sistema.
L'inizializzazione sarà eseguita soltanto se all'interno della cartella /setup/ sarà presente il file questions.xml.
Al termina dell'inizializzazione, il file sarà eliminato.
N.B. La procedura di inizializzazione, troncherà il contenuto di tutte le tabelle (perdita dei dati)!
Nel momento in cui il file questions.xml non esiste, l'installer, interpreterà il sistema come già inizializzato e dunque non inizializzerà il database (l'inizializzazione non andrà a buon fine).

3. Funzionamento del sistema

Il sistema si basa su una logica basata su un frontend ed un backend.
L'obiettivo del frontend è quello di fornire una interfaccia all'utente mentre, il backend, ha l'obiettivo di fornire la logica di funzionamento.
L'interazione tra backend e frontend, avviane secondo uno schema comune alle api infatti, il client esegue le richieste al server non conoscendo la struttura delle varie componenti.
Per quanto riguarda l'implementazione del backend, esso fornisce i dati al frontend mediante delle API che hanno un comportamento da blackbox infatti, queste, ricevute dei dati in ingresso, restituiscono dei risultati formattati in JSON.

4. Spiegazione pannello di amministrazione

Per implementare il pannello di amministrazione, si sono utilizzati i seguenti files:

- **Index.php**: è l'interfaccia principale e, permette all'amministratore di visualizzare l'elenco degli studenti. Da questa, è inoltre possibile gestire i punteggi bonus/malus degli studenti ed è possibile visualizzare in dettaglio le prove degli stessi;
- **studente.php**: è l'interfaccia che permette la stampa della copia dell'esame dello studente selezionato dall'interfaccia principale e, permette anch'esso di gestire i punti bonus/malus;
- **Lista.php**: permette di stampare gli elaborati di tutti gli studenti;
- **updatePoints.php**: è un file di supporto che consente l'aggiornamento dei punteggi bonus/malus degli studenti;
- **requires.php**: è un file di supporto che permette l'importazione dei parametri di configurazione in ciascun script.php;
- **head.php**: è un file di supporto che consente l'importazione degli stili e delle librerie (file css e file js);
- **security.php**: è un file di supporto che permette di gestire gli accessi all'interfaccia dell'esaminatore;
- **install.php**: consente l'inizializzazione del sistema (con relativa importazione delle domande).

5. Spiegazione backend

I files del backend, implementano la logica di funzionamento, sono localizzati all'interno della directory /htdocs/api (o /www/api) e sono:

- **examTitle.php**: permette di restituire al client il nome dell'esame;
- **login.php**: permette al client di eseguire l'autenticazione dell'utente;
- **publish.php**: permette di memorizzare le risposte date dagli utenti;
- **questions.php**: fornisce al client l'elenco delle domande;
- **submit.php**: permette al client di fornire l'interfaccia di chiusura delle esame da parte dell'utente mediante la pressione del pulsante "termina esame";
- **time.php**: fornisce al client il tempo residuo
- **forcedSubmit.php**: permette la chiusura della sessione in modo forzato allo scadere del tempo.

a. Spiegazione file examTitle.php

Il file al suo interno contiene il seguente codice scritto in php:

```php
require 'requires.php';

// Return the title of exam
echo json_encode([
    'title' => $examTitle,
]);
```

L'istruzione "require 'requires.php';", permette di importare i parametri di configurazione del sistema.

L'istruzione "echo json_encode(['title'] => $examTitle,]);", permette di restituire al client il file json che permette la stampa del nome dell'esame.

(1) Che funzione ha un file json?

JSON (JavaScript Object Notation) è un semplice formato adatto all'interscambio di dati fra applicazioni client/server. Per le persone è facile da leggere e scrivere, mentre per le macchine, risulta facile da generare e analizzarne la sintassi.

a. Spiegazione file login.php

Il file login.php come detto precedentemente, permette all'utente di eseguire l'autenticazione.
Prima di procedere con l'autenticazione, lo script ha il compito di verificare se il client ha inviato dei campi validi o non validi e, per farlo, si utilizzano le seguenti istruzioni:

```
if(empty($matricola) || empty($nome)
|| empty($cognome)) {
    $message->status = "error";
    $message->message = "I dati
    inseriti non sono validi.";
    echo json_encode($message);
    return;
}
```

Verificata la correttezza dei dati, viene eseguita una query che, permette di prelevare dal db i dati dell'utente

```
$user = mysqli_fetch_object($connect->query("select *, TIME_TO_SEC(TIMEDIFF(NOW(), start)) as deltaTime from
users where matricola = $matricola limit 1"));
```

Se la query non restituisce nessun risultato, il sistema inserirà i dati dell'utente nella tabella users.

```
// Check if user already exists
if(!$user) {

    // Insert user in users
    $insertuserdata = $connect->query("Insert into users
    (matricola,nome,cognome,start) values
    ('".$matricola."','".$nome."','".$cognome."',NOW())");

    // Return response to client: the exam can start
    $message->status = "success";
    $message->message = [];

} else {
```

Viceversa, se la query ha restituito un risultato, si verifica se l'utente può continuare o meno l'esame e, se può farlo, si restituiscono le risposte date.

b. Spiegazione forcedSubmit.php

Lo script ha il compito di terminare la sessione in modo forzato nel momento in cui il tempo scade.
La corretta esecuzione dello script php restituisce al client l'esito: studente idoneo o non idoneo.

c. Spiegazione Submit.php

Ha la stessa funzione dello script precedente ma, termina la sessione nel momento in cui l'utente chiede di chiuderla.
Nel momento in cui l'utente decide di chiudere la sessione, possono verificarsi due scenari:

1) Lo studente non ha risposto a tutte le domande: lo script, non permette la chiusura della sessione e restituisce il numero delle risposte non date;
2) Lo studente ha risposto a tutte le domande: la sessione viene terminata e, al client, viene restituito l'esito: idoneo/non idoneo.

Nel momento in cui la chiusura della sessione viene accettata, il server memorizza il timestamp e blocca il recupero della sessione.

d. Spiegazione publish.php

Lo script php ha il compito di memorizzare le risposte degli utenti.
In ingresso, richiede la matricola dello studente, l'id della domanda e l'id della risposta data.

e. Spiegazione time.php

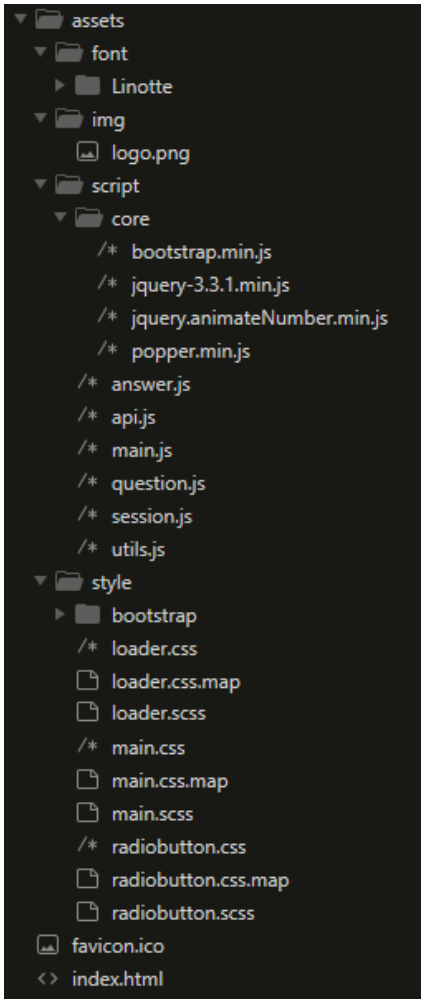
Lo script php ha il compito di restituire al client il tempo residuo

```
$getTime = $connect->query("SELECT TIME_TO_SEC(TIMEDIFF(NOW(), start)) seconds from users
where matricola=" . $matricola);
```

6. Spiegazione Frontend

a) Struttura progetto

La struttura del progetto per quanto concerne il frontend è la seguente:



La directory **assets** contiente tutte le risorse necessarie al corretto funzionamento della parte grafica.

All'interno troviamo diverse sottodirectory:

- **font:** contiene il font principale utilizzato per tutta la web app, in questo caso **Linotte**
- **img:** contiene il logo del Politecnico di Bari
- **script:** contiene tutti i javascript che gestino la logica della pagina
- **style:** contiene tutti i fogli di style css necessari alla corretta visulizzazione della pagina web

E' presente anche un file **favicon.ico** utile a mostrare l'icona nella finestra del browser.

L'ultimo file, ovvero **index.html**, contiente la struttura principale della web app, che verrà modificata dinamicamente di volta in volta durante l'esecuzione del codice JavaScript.

Esaminaniamo nel dettaglio i singoli file.

Index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Politecnico di Bari - Esame</title>

    <link rel="stylesheet" href="assets/style/loader.css">
    <link rel="stylesheet" href="assets/style/radiobutton.css">
    <link rel="stylesheet" href="assets/style/main.css">
</head>
<body>
```

E’ presente la classica struttura di HTML5 per la creazione di pagine web.
I due tag principale sono **head** e **body**.

In **head**, attraverso i tag **link**, vengono importati i fogli di stile quali:

- **loader:** ci permette di mostrare correttamente lo spinner di caricamento delle domande
- **radiobutton:** per il rendering degli input di tipo **radio** per la scelta delle risposte
- **main:** è il file di stile principale

Nel **body** è presente:

- Una **section** per mostrare nome, cognome, matricola e tempo rimanente durante la prova:

```
<section>
     <div id="user_details" class="float-left">
        <div id="nameSurname_value"></div>
         <div id="matricola_value"></div>
    </div>
    <div id="time" class="time float-right text-center">
        <div>Tempo rimanente:</div>
        <span class="value"><span class="mins">00</span>:<span class="secs">00</span></span>
    </div>
</section>
```

- Un tag **img** per mostrare il logo e dei tag **div** e **span** per la visualizzazione del numero della domanda attuale rispetto il numero totale di domande:

```
<div class="col-5 my-5 mx-auto">
    <div class="row">
        <div class="col-9 mx-auto logo">
            <img class="main-logo" src="assets/img/logo.png">
        </div>
        <div class="my-auto questions" style="width: 0; opacity: 0;">
            <div>Domanda:</div>
            <span class="questions-done">0</span> / <span class="questions-total">0</span>
        </div>
    </div>
</div>
```

- Un tag **h3** per la visualizzazione del titolo esame caricato attraverso le API, il loader (ovvero lo spinner) e il container principale della sessione di esame:

```
<!-- Exam title -->
<h3 id="exam-title" class="mb-5 exam-title">

</h3>

<!-- Loader -->
<div id="loader" class="mt-5" style="display: none;">
    <div class="loader">Caricamento</div>
</div>

<!-- Main session container -->
<div id="session" class="mx-auto w-75" style="display: none;"></div>
```

- Un form di input per l'inserimento di nome, cognome e matricola e relativo pulsante per avviare l'esame:

```
<!-- Input -->
<div id="login-form">
    <input id="nome" type="text" placeholder="Nome" />
    <input id="cognome" type="text" placeholder="Cognome" />
    <input id="matricola" type="text" placeholder="Matricola" />
    <!-- New Exam Button -->
    <div class="col text-center">
        <button id="newSession" type="button" class="btn btn-primary mainButton px-4 py-2">
        Avvia esame</button>
    </div>
</div>
```

- Un **div** con id **error** da mostrare in caso di errore di qualsiasi tipo richiamato dal backend:

```
<div class="modal fade" id="error" tabindex="-1" role="dialog">
    <div class="modal-dialog" role="document">
        <div class="modal-content">
            <div class="modal-header">
                <h5 class="modal-title">Errore</h5>
                <button type="button" class="close" data-dismiss="modal">
                    <span aria-hidden="true">&times;</span>
                </button>
            </div>
            <div class="modal-body">

            </div>
            <div class="modal-footer">
                <button type="button" class="btn btn-secondary" data-dismiss="modal">Chiudi</
                button>
            </div>
        </div>
    </div>
</div>
```

- In chiusura è presente un tag **footer** ed i relativi script da caricare solo al termine del documento (motivo per cui sono collocati in basso al documento, prima della chiusura del tag **body**):

```
<footer>
    <small>
        Politecnico di Bari
    </small>
</footer>

<script src="assets/script/core/jquery-3.3.1.min.js"></script>
<script src="assets/script/core/popper.min.js"></script>
<script src="assets/script/core/bootstrap.min.js"></script>
<script src="assets/script/core/jquery.animateNumber.min.js"></script>

<script src="assets/script/api.js"></script>
<script src="assets/script/answer.js"></script>
<script src="assets/script/question.js"></script>
<script src="assets/script/utils.js"></script>
<script src="assets/script/session.js"></script>
<script src="assets/script/main.js"></script>
```

b) SCSS e CSS (Cascading Style Sheets)

Per il progetto sono state utilizzare diverse soluzioni per uno styling efficace ed efficiente. E' stato preferibile utilizzare file SCSS per tutti i vantaggi che essi offrono rispetto al semplice CSS (dichiarazione di variabili globali, chiamate a funzioni quali **mix** per il mescolamento dei colori ecc...), poi compilati attraverso un apposito compilatore in file CSS interpretabili dal browser. Altri file CSS, invece, sono stati importati dall'esterno favorendo il riuso del codice.

Inoltre, è stato utilizzato il framework per frontend **Bootstrap** per la definizione di spazi quali righe e colonne per una migliore impostazione della pagina.
Il framework è presente nella directory sotto il nome di style/bootstrap ed è importato nel file **main.scss**: unico file fondamentale per la modifica stilistica dei vari elementi nella pagina.

> **N.B.:** gli altri file presenti nella directory **style** sono codici riutilizzati e compilati, quindi l'unico file necessario per la modifica degli stili della webapp è main.scss

```
$primary: #188886;
$secondary: #136967;
//$tertiary: #c7eb00;

@import "bootstrap/bootstrap";

html body {
  font-family: Linotte, serif;
  font-weight: bold;
  font-size: 18px;
  margin-bottom: 40px;
}

...
```

In **main.scss** troviamo la dichiarazione dei colori primario e secondario dell'intera webapp, l'importazione di bootstrap è la definizione degli stili custom definiti da noi per ogni elemento della pagina.

Al termine del file è presente anche l'importazione del font **Linotte**.

In questo caso sono stati importati sia la variante **Bold** che **Regular**

```
@font-face {
  font-family: Linotte;
  font-weight: normal;
  src: url("../font/Linotte/Linotte Regular.otf");
}

@font-face {
  font-family: Linotte;
  font-weight: bold;
  src: url("../font/Linotte/Linotte Bold.otf");
}
```

c) JavaScript e jQuery

Per la realizzazione del codice che gestile la logica dell'applicazione è stato utilizzato codice JavaScript insieme all'uso di un framework per frontend **jQuery**, il quale permette di gestire situazioni più o meno complesse con poche righe di codice grazie alle sue API.

All'interno della directory **core** troviamo:

- jQuery versione 3.3.1
- Bootstrap
- Popper (un plugin necessario a mostrare correttamente determinati componenti della UI)
- AnimateNumber (un plugin di jQuery utile ad animare in maniera incrementale dei valori numerici sulla User Interface)

Elenco dei file presenti nella directory **script**:

- **answer.js:** contiene la classe che definisce la risposta da dare ad ogni domanda
- **api.js:** contiene l'interfaccia che collega il frontend con il backend
- **main.js:** è il codice di controllo principale (l'entry point della webapp)

- **question.js:** contiene la classe che definisce ogni domanda dell'esame
- **session.js:** è la classe che contiene la logica per gestire l'intera sessione d'esame
- **utils.js:** contiene dei metodi ausiliari richiamabili globalmente nell'applicazione

Procedendo per ordine logico analizziamo i singoli file:

answer.js

Il file contiene la classe che definisce il modello di ogni risposta che lo studente può dare ad ogni domanda.

I campi che contiene sono:

- **id** (intero)
- **risposta** (stringa)

```
/**
 * Model of a single Answer
 */

class Answer {

    constructor(id, answer) {
        this.id = id;
        this.answer = answer;
    }

}
```

question.js

```
class Question {

    constructor(id, question, answers) {
        this.id = id;
        this.question = question;
        this.answers = answers;
    }

}
```

Contiene la classe che definisce il modello di ogni domanda che arriva dal backend.

I campi che contiene sono:

- **id** (intero)
- **domanda** (stringa)
- **risposte** (array di oggetti *risposta*)

utils.js

Definisce tutti metodi definiti *"utili"* all'interno dell'intera applicazione, richiamabili globalmente.

Questi sono:

- **swap** – inverte l'ordine di due elementi
- **shuffle** – randomizza l'ordine degli elementi
- **pad** – aggiunge degli zeri per formare una stringa di lunghezza fissa
- **sleep** – per mettere in pausa l'esecuzione per un tempo definito

api.js

E' la classe contenente l'interfaccia di comunicazione tra backend e frontend.
All'inizio viene definito un baseEndpoint per indirizzare le chiamate HTTP al giusto URL della REST API.

Vengono definiti gli endpoint che sono:

- **LOGIN:** per eseguire il login dello studente
- **TITLE:** per ottenere il nome dell'esame in corso
- **TIME:** per ottenere il tempo residuo utile al completamento dell'esame
- **QUESTIONS:** per ottenere la lista delle domande
- **SUBMIT:** per notificare il server della chiusura dell'esame
- **FORCEDSUBMIT:** per forzare la chiusura dell'esame in corso
- **ANSWER:** per inviare una risposta data da uno studente al server

Mentre i metodi statici (autoesplicativi) definiti all'interno della classe sono:

- **login (nome, cognome, matricola)**
- **getExamTitle ()**
- **getTime (matricola)**
- **getQuestions ()**
- **submit (matricola)**
- **submitForced (matricola)**
- **postAnswer (matricola, question_id, answer_id)**

Cio che questi metodi fanno internamente è mandare una richiesta di tipo **GET** o **POST**, a seconda della situazione, al web server che elaborerà una risposta e che verrà restituita in formato **JSON** (in generale) all'oggetto chiamante.

main.js

E' il file principale che inizzializza l'intera applicazione.

Non appena il documento è pronto, ovvero è stato totalmente caricato nel browser, viene utilizzata l'API per mostrare il titolo esame sullo schermo e subito dopo viene collegata un metodo alla pressione del pulsante **Avvia Esame**.

```
$(document).ready(async function () {

    // Show exam title
    $('#exam-title').text('"' + (await API.getExamTitle()) + '"');

    // Start a new session when we click on "Avvia Esame"
    $('#newSession').click(newSession);

    function newSession() {
```

Alla presione del tasto per avviare l'esame vengono eseguiti dei controlli preventivi lato client per la verifica di inserimento di informazioni corrette.

Passate le quali viene mostrato lo spinner per il caricamento delle domande, vengono mostrati anche i dati utenti relativi al suo profilo (nome, cognome e matricola), utili alla sua identificazione in fase di esame.

Subito dopo viene instanziata una nuova sessione di esame con le informazioni fornite e viene richiamato il medoto per eseguire il login.

```
// Get login datas
const nome = $('#nome').val();
const cognome = $('#cognome').val();
const matricola = $('#matricola').val();

// Check if all fields are filled
if(nome === '' || cognome === '' || matricola === '') {
    $('#error').modal('show').find('.modal-body').text('Inserisci i dati richiesti per iniziare l\'esame');
    return;
}

// Check if matricola has 6 chars only
if(matricola.length != 6) {
    $('#error').modal('show').find('.modal-body').text('La matricola inserita deve contenere 6 caratteri');
    return;
}

// Prepare environment with some visual effects
$('.logo').removeClass('col-9').addClass('col-6');
$('.exam-title').fadeOut();
$('#login-form').fadeOut();

// Timer useful to wait the transitions triggered above
setTimeout(function () {

    // Show the Loading spinner
    $('#loader').fadeIn();

    // Show user profile on top left
    $('#matricola_value').text(matricola);
    $('#nameSurname_value').text(nome + " " + cognome);

    // Instantiate session
    const session = new Session(nome, cognome, matricola);

    // Check login
    session.login();

}, 1000);
```

session.js

Gestisce tutta la sessione d'esame.

Vengono definite delle costanti per poter facimente navigare tra le varie domande.

Il costruttore, richiamato al momento della creazione dell'oggetto relativo, inizzializza tutti parametri necessari al funzionamento della sessione.

I metodi presenti nella classe sono:

- **setTimer ()**
- **login ()**

- **fetchQuestions ()**
- **createQuestion (questionData)**
- **initSession ()**
- **setTimerTick ()**
- **changeQuestion (nextOrPrevious)**
- **closeSession ()**
- **forceCloseSession ()**
- **updateTimerUI ()**
- **saveAnswer (answerId)**
- **draw ()**
- **showQuestionsCounter ()**
- **showResults (apiResult)**

setTimer ()
Inizzializza il tempo residuo con quello ottenuto dal server master.

login ()
Esegue il login attraverso l'API e se la risposta è positiva vengono caricate le domande, altrimenti viene mostrato un messaggio di errore

fetchQuestions ()
Vengono caricate le domande attravero l'API e per ogni domanda restituita viene chiamato il metodo **createQuestion**.
Inoltre viene caricato il totale delle domande e avviata la sessione.

createQuestion (questionData)
Per la domanda che viene creata vengono istanziate tante risposte quante sono quelle provenienti dall'API, mescolate e inserite all'interno dell'oggetto domanda.

initSession ()
Viene aggiornato e mostrato il tempo residua sulla UI.
Viene nascosto lo spinner.
Viene mostrato il contatore di domande affianco al logo.
Viene mostrata la prima domanda.

setTimerTick ()
Decrementa il tempo residuo per completare la prova e viene chiusa non appena il tempo risulta scaduto.

changeQuestion (nextOrPrevious)
Cambia la domanda in precedente o successiva e richiama il metodo **draw ()**.

closeSession ()
Chiude la sessione dell'esame in corso dopo aver controllato da backend che tutte le risposte siano state date.

forceCloseSession ()
Mostra il risultato finale dopo aver forzato la chiusura dell'esame.

updateTimerUI ()
Aggiorna minuti e secondi residui sull'UI per completare la prova.

saveAnswer (answerId)
Invia, attraverso l'API, la risposta data dallo studente per una relativa domanda.

draw ()
Genera e rimpiazza il codice HTML sulla pagina per mostrare la domanda precedente o successiva e le relative risposte con i pulsanti **Precedente**, **Successiva**.
Viene mostrato il pulsante per chiudere l'esame al termine della prova.

showResults (apiResult)
Viene mostrata la schermata nella quale è presente l'esito dell'esame.