

UNIVERSITATEA "ALEXANDRU IOAN CUZA" DIN IAȘI
FACULTATEA DE INFORMATICĂ



LUCRARE DE LICENȚĂ

PL/SQLKit

propusă de

Andreea Asaftei

Sesiunea: *iulie, 2018*

Coordonator științific

Lect. Dr. Cosmin Vârlan

UNIVERSITATEA “ALEXANDRU IOAN CUZA” DIN IAȘI
FACULTATEA DE INFORMATICĂ

PL/SQLKit

Andreea Asaftei

Sesiunea: *iulie, 2018*

Coordonator științific

Lect. Dr. Cosmin Vârlan

Avizat,

Îndrumător Lucrare de Licență

Titlul, Numele și prenumele _____

Data _____ Semnătura _____

DECLARAȚIE privind originalitatea conținutului lucrării de licență

Subsemnatul(a)

domiciliul în

născut(ă) la data de, identificat prin CNP,
absolvent(a) al(a) Universității „Alexandru Ioan Cuza” din Iași, Facultatea de
..... specializarea, promoția
....., declar pe propria răspundere, cunoscând consecințele falsului în
declarații în sensul art. 326 din Noul Cod Penal și dispozițiile Legii Educației Naționale nr.
1/2011 art.143 al. 4 si 5 referitoare la plagiat, că lucrarea de licență cu titlul:

_____elaborată sub îndrumarea dl. / d-na
_____, pe care urmează să o susțină în fața
comisiei este originală, îmi aparține și îmi asum conținutul său în întregime.

De asemenea, declar că sunt de acord ca lucrarea mea de licență să fie verificată prin
orice modalitate legală pentru confirmarea originalității, consimțind inclusiv la
introducerea conținutului său într-o bază de date în acest scop.

Am luat la cunoștință despre faptul că este interzisă comercializarea de lucrări
științifice în vederea facilitării fașificării de către cumpărător a calității de autor al unei
lucrări de licență, de diploma sau de disertație și în acest sens, declar pe proprie

răspundere că lucrarea de față nu a fost copiată ci reprezintă rodul cercetării pe care am întreprins-o.

Data azi,

Semnătură student

DECLARAȚIE DE CONSIMȚĂMÂNT

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „*PL/SQLKit*”, codul sursă al programelor și celelalte conținuturi (grafice, multimedia, date de test etc.) care însoțesc această lucrare să fie utilizate în cadrul Facultății de Informatică.

De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” din Iași, să utilizeze, modifice, reproducă și să distribuie în scopuri necomerciale programele-calculator, format executabil și sursă, realizate de mine în cadrul prezentei lucrări de licență.

Iași, *data*

Absolvent *Andreea Asaftei*

Cuprins

Introducere și motivație	7
Contribuții	8
1 Tehnologii utilizate	9
1.1 Introducere	9
1.2 Scurt cuprins al tehnologiilor utilizate	9
2 Prezentarea aplicației	11
2.1 Introducere	11
2.2 Capabilitățile aplicației	11
2.3 Structura aplicației. Diagrame UML	24
2.4 Arhitectura aplicației	25
2.4.1 Detalii arhitecturale	25
2.4.2 Structura bazei de date în Firebase	28
2.5 Platforma de management folosită	30
2.6 Găzduirea aplicației. DigitalOcean	32
2.7 Wireframe-urile inițiale	34
2.8 Lecțiile PL/SQL	38
2.9 Testarea aplicației	38
Concluzii și provocări	40
Bibliografie	42
Anexă	43

Introducere și motivație

Tema de licență pe care am ales-o a fost propusă pentru a veni în ajutorul studenților din anul 2, care urmează disciplina PSGBD (Practica Sisteme Gestione Baze de Date). Prin “*PL/SQLKit*” propun un mod intuitiv și interesant de învățare a limbajului PL/SQL, dar și de aprofundare, constituind o resursă adițională pentru îmbunătățirea abilității de lucru cu acest limbaj. Scopul aplicației este atât de a oferi o bază de cunoștințe, cât și de a oferi provocări prin exerciții cu un grad ridicat de dificultate.

Motivația alegerii acestei teme a venit din dorința pe care o aveam ca și student la această disciplină, de a exersa și de a găsi toate resursele într-o singur loc. În online, există astfel de aplicații care oferă începătorilor o privire de ansamblu a unor limbaje de programare. Un astfel de exemplu ar fi site-ul <https://codecademy.com>. În schimb, aceștia nu oferă un modul de exersare pentru PL/SQL explicit. Astfel, am vrut ca prin aplicația mea, să vin în sprijin următorilor studenți care vor să învețe într-o manieră interactivă, conversațională.

Un alt motiv care m-a îndreptat spre conceperea acestei lucrări a fost faptul că am găsit un punct comun în lucrul cu tehnologiile web, iar partea de simulator de compilator online mi s-a părut o idee interesantă de integrat în aplicație.

Aplicația propusă stă la dispoziția studenților prin parcurgerea unor lecții enunțate astfel încât să mențină interesul. Un factor motivant este obținerea de puncte după parcurgerea cu succes a unei lecții.

Această lucrare este realizată cu ajutorul următoarelor instrumente de dezvoltare Web: PHP 7.1, Apache Web Server 2.4, Oracle Database 11g Express Edition Release 11.2.0.2.0, Firebase, HTML5, CSS3 și JavaScript.

Lucrarea urmează structura:

- specificații funcționale;
- detalii arhitecturale;
- wireframe-uri;
- diagrame UML;
- testarea aplicației;
- concluzii și provocări.

Contribuții

Tematica acestei lucrări a fost propusă de mine, fiind inspirată de ideea de aplicație tip tutorial de pe site-ul *codeacademy.com*. Propunerea a fost acceptată de domnul profesor Vârlan Cosmin (coordonatorul meu) considerând că este o modalitate bună la care studenții să poată avea acces, un mediu în care să poată urmări fundamentele PL/SQL.

Codeacademy.com este o platformă de învățare ce pune la dispoziție utilizatorilor săi posibilitatea de a parcurge tutoriale pentru diferite limbaje de programare. Utilizatorii au acces la un compilator unde pot introduce și rula instrucțiuni. Pe baza corectitudinii instrucțiunilor rulate utilizatorii pot avea acces la următoarea lecție.

„*PL/SQLKit*” urmărește aceeași structură, iar scopul său principal este de a pune la dispoziție un modul de exersare pentru limbajul PL/SQL pentru studenți.

Principalele contribuții personale sunt:

- Crearea unui compilator PL/SQL care validează textul scris;
- Formularea lecțiilor PL/SQL;
- Îmbinarea acestor funcționalități într-o aplicație web.

Elementele teoretice și practice folosite în dezvoltarea lucrării provin în proporție majoritară din experiența mea în lucrul cu *Tehnologiile Web, Ingineria Programării, Dezvoltarea aplicațiilor Web la nivel de client, Baze de date și Practica Sisteme Gestione Baze de Date*.

Mai multe despre procesul de creare al acestei lucrări de licență se găsesc în paginile ce urmează.

1 Tehnologii utilizate

1.1 Introducere

În acest capitol voi explica în detaliu partea teoretică a aplicației “*PL/SQLKit*”, mai exact constrângerile tehnice, librăriile, framework-urile și tehnicile folosite, partea de cercetare, partea de implementare și integrare a acestor tehnologii în aplicație. De asemenea, voi menționa pentru fiecare dintre acestea avantajele pe care mi le-au oferit, cât și dificultățile întâlnite în folosirea lor.

1.2 Scurt cuprins al tehnologiilor utilizate

Aplicația este de tip Web, ea fiind accesată prin intermediul internetului. În cele ce urmează voi detalia tehnologiile folosite și o scurtă motivație pentru care am ales să le folosesc:

- *HTML5* care a furnizat structura paginii web;
- *CSS3* care a ajutat la oferirea unei interfețe cât mai plăcute;

Alte tehnici CSS folosite:

- *CSS Media Queries*, cu ajutorul căreia am reușit să creez o aplicație responsive, prin aplicarea anumitor stiluri, în funcție de rezoluția ecranului;
- *CSS @keyframes*, care m-a ajutat în a controla apariția unei animații acolo unde a fost cazul. Animația a fost creată schimbându-se gradual de la un stil CSS la altul;
- *Bootstrap* framework a ajutat la structurarea paginii principale prin oferirea unui sistem grid.
- *JavaScript*, limbaj care a fost folosit împreună cu următoarele framework-uri:
 - Pe partea de FrontEnd am folosit o librărie flexibilă prin intermediul JavaScript, și anume, *ProgressBar.js*, care se bazează pe fișiere SVG¹ (progresul este afișat de-a lungul unei căi SVG). Prin intermediul acesteia, am putut crea un progress circle responsive și animat, care să informeze utilizatorul asupra evoluției făcute în cadrul aplicației.
 - Pe partea de BackEnd am folosit framework-ul *Firebase*, care este un API puternic pentru stocarea și sincronizarea datelor în timp real.

¹ SVG este definit ca Scalable Vector Graphics și sunt vectori grafici bazați pe formatul XML.

- De asemenea, tot cu ajutorul limbajului JavaScript am folosit pentru componenta de compilator, un text editor de la *CodeMirror*. De la acesta am utilizat atât o tematică CSS pentru customizarea aplicației, cât și funcționalități care sunt disponibile prin API-ul său.
- *PHP 7.1*, rulat de *Apache Web Server 2.4*. Aceste două tehnologii m-au ajutat să aduc partea de compilator la viață. Dacă prin intermediul *CodeMirror*, reușisem să creez o interacțiune pe partea aceasta, era necesar ca procedurile rulate să fie și executate într-un server Oracle. Împreună cu aceste două tehnologii, am reușit să conectez Apache la PHP și acesta din urmă la Oracle server.
- *Oracle Database 11g Express Edition Release 11.2.0.2.0* reprezintă pentru aplicația mea serverul Oracle pe care se rulează fiecare procedură scrisă de către utilizator. Rezultatul trimis de către acesta este preluat prin AJAX.
- *AJAX* a fost folosit în cadrul aplicației pentru a interschimba date cu un server, în cazul meu server-ul de la Oracle, fără ca pagina sa fie reîncărcată. Am folosit *jQuery* deoarece acesta încorporează cererile de tip AJAX, *jQuery* fiind o modalitate mai concisă de a scrie metode în JavaScript.
- *TFS* sau *Team Foundation Server* constituie pentru proiectul meu o platformă de management care m-a ajutat atât pe partea de organizare a sarcinilor care trebuiau îndeplinite, prin accesul la un *Backlog*, cât și pe partea de integrare a codului scris, prin faptul că ei mi-au oferit posibilitatea de depozitare a codului. Acesta m-a ajutat și să lucrez *Agile* prin faptul că dezvoltarea proiectului s-a făcut pe sprinturi a câte trei săptămâni.
- *Visual Studio* a fost folosit împreună cu TFS. Din acest program am putut să fac commit-uri și astfel ultimul cod să fie prezent pe platforma de management.
- *WinSCP* reprezintă tehnologia folosită pentru transferul de fișiere între calculatorul personal și serverul pe care este găzduită aplicația.
- *Figma* a fost folosit în cadrul proiectului pentru a elabora wireframe-urile inițiale ale aplicației. Acest instrument m-a ajutat în a clarifica structura paginilor care au fost ulterior implementate în aplicație.
- *ArgoUML* a fost folosit pentru a crea diagramele UML, ce au furnizat o structură care să ajute la înțelegerea funcționalităților aplicației.

2 Prezentarea aplicației

2.1 Introducere

Scopul principal al acestei lucrări este de a facilita învățarea limbajului PL/SQL. Această aplicație oferă utilizatorilor săi motivația de a rezolva exercițiile disponibile și de a avea un progres al nivelului de cunoștințe cât mai înalt.

Proiectul oferă pe lângă lecțiile enunțate cât mai descriptiv și o parte de compilator în care utilizatorul poate testa diferite interogări, blocuri anonime, proceduri și alte structuri PL/SQL ce pot ajuta înțelegerea conceptelor prezentate.

Lecțiile puse la dispoziție sunt ușor de urmărit, fiind formulate în instrucțiuni elaborate, conținând cuvinte cheie ce pot ajuta utilizatorul să rezolve cu ușurință exercițiul.

Utilizatorilor nu li se cer cunoștințe anterioare, ci doar voință de învățare și exersare.

Printre funcționalitățile pe care acest proiect le oferă, se enumeră vizualizarea lecțiilor, exersarea în permanență într-un compilator, oferirea de sugestii pentru completări în cod, resetarea progresului făcut la o anumită lecție, strângerea de puncte și vizualizarea acestui progres într-o pagină separată.

2.2 Capabilitățile aplicației

În acest capitol voi prezenta funcționalitățile pe care le pune la dispoziție aplicația „*PL/SQLKit*”. Acest capitol va cuprinde descrierea tuturor funcționalităților, însoțite de capturi de ecran din aplicație care să vină în ajutorul unei bune înțelegeri și clarificări a ceea ce poate să facă aplicația dezvoltată de mine.

Deoarece în capitolul 2.7 *Wireframe-urile inițiale*, menționez wireframe-urile folosite de la care am pornit în construirea paginilor, doresc ca în acest capitol să furnizez o captură de ecran pentru fiecare dintre paginile prezente în aplicație. Astfel se poate face o comparație între ceea ce mi-am propus și rezultatul final obținut. Menționez acestea deoarece unele pagini cuprind puține elemente, dar necesită a fi evidențiate pentru motivul prezentat mai devreme.

Pentru început, voi menționa prima pagină cu care utilizatorul interacționează. Aceasta este pagina *Welcome* și funcționalitatea pe care o înglobează este de a redirecționa utilizatorul către pagina de *Login* sau de *Register*, în funcție de preferințele acestuia. În această pagină am folosit *CSS @keyframes* pentru a crea un efect de typewriting de a introduce utilizatorul în aplicație.

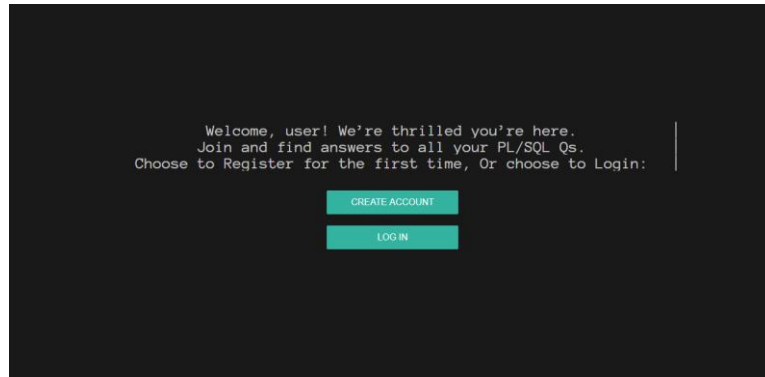


Fig. 1 Prima pagină a aplicației

Evidențiez prin Fig. 1 pagina intitulată *Welcome* a aplicației. Ce am vrut să exemplific mai mult prin această captură de ecran este modul în care sunt afișate enunțurile în urma typewriting-ului.

Voi continua prin a exemplifica structura paginii *Register*. Așa cum am menționat și în introducerea acestui capitol, doresc să prezint capturi de ecran a fiecărei pagini din aplicație, pentru a putea face o comparație între funcționalitățile propuse (wireframes) și rezultatul final obținut.

Deoarece pagina *Register* este asemănătoare cu pagina *Login* din punctul de vedere al câmpurilor cerute a fi completate de către utilizator (email, parolă), voi atașa o captură doar a uneia dintre pagini. Funcționalitatea acestei pagini este una de bază, și anume, utilizatorul trebuie să introducă date de autentificare care nu au mai fost anterior create. Pentru pagina de *Login*, acesta trebuie să introducă, bineînțeles, date de autentificare care au fost anterior create.

Există și o parte de validări a datelor introduse de utilizator, și anume, pentru pagina *Register*, cazurile în care utilizatorul introduce parole diferite pentru situația în care se cere aceeași parolă, cât și atenționarea sa atunci când introduce date de autentificare deja existente. Un student, viitor utilizator se poate înregistra în aplicație și cu contul său de *fenrir*, dacă îi este mai ușor. Atât timp cât utilizatorul introduce un email valid, contul acestuia poate fi creat cu succes. De asemenea, o altă validare care merită menționată este aceea în care utilizatorul nu introduce nimic și vrea să aibă acces, sau când parola aleasă nu are suficiente caractere.

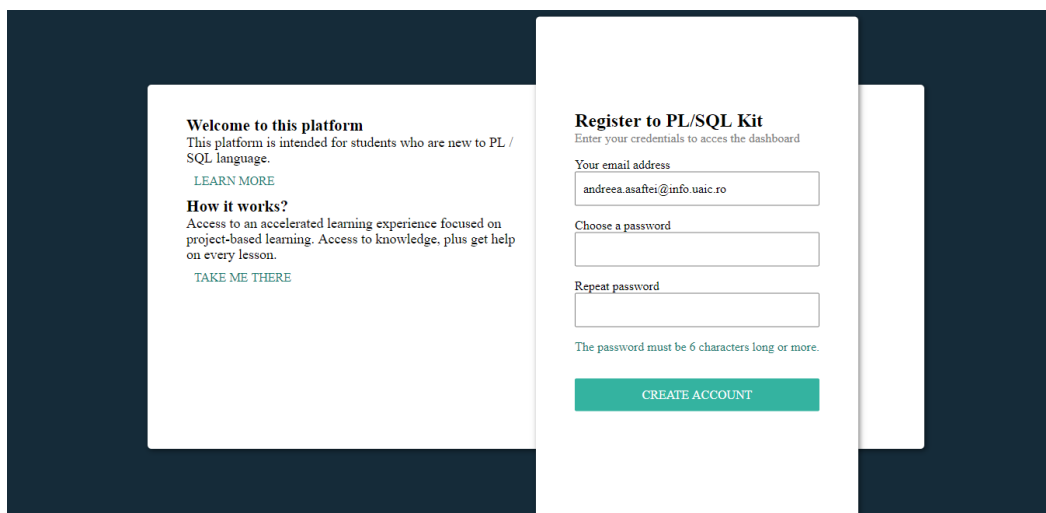


Fig. 2 Captură de ecran a paginii de *Register*

Prin Fig. 2 evidențiez o validare existentă pentru această pagină în momentul în care utilizatorul nu a introdus nicio parolă, dar dorește să își creeze un cont. Așa cum am specificat și anterior, aceasta este una dintre validările existente pentru pagina *Register*, cât și pentru pagina *Login*.

Pentru cele două pagini, *Register* și *Login*, există o serie de redirectări în cazul în care utilizatorul nu este autentificat, dar dorește accesul la anumite pagini. De exemplu, dacă utilizatorul se află pe pagina din Fig. 2, și schimbă URL-ul pentru a merge pe pagina principală, acesta va fi redirectat automat către pagina inițială. Un alt exemplu ar fi atunci când utilizatorul este deja autentificat, și dorește accesul la pagina *Login*. Acesta va fi automat redirectat către pagina principală, fără a îl pune în situația de a își reintroduce credențialele. Pentru fiecare acțiune executată în aplicație se verifică dacă utilizatorul încă mai este autentificat.

Pentru partea de autentificare am beneficiat de sistemul *Firebase Auth*, de la *Firebase* care oferă o autentificare pe bază de parolă și email, pe care am încorporat-o ulterior în aplicație.

Firebase Auth se integrează direct în baza de date *Firebase*, astfel încât am putut să o folosesc pentru a controla accesul la fiecare dintre utilizatorii aplicației mele, putând să personalizez aplicația în funcție de contul utilizatorului.

Un alt lucru de care am ținut cont a fost faptul că pentru fiecare cont nou creat, era necesară inițializarea punctajului său ca fiind zero, urmând ulterior a crește după parcurgerea cu succes a fiecărei lecții. Codul pentru această funcție poate fi vizualizat în Anexă în Fig. 23.

Am ales să utilizez *Firebase* în proiectul de licență deoarece lucrasem și înainte cu acesta în cadrul altui proiect din facultate și am considerat că am o mică experiență cu el. *Firebase* este un

BaaS² în cadrul Google Cloud Platform. Firebase este atât un server, cât și API, și bază de date. Este scris generic astfel încât am putut porni de la funcționalitățile pe care acesta le oferă și să construiesc în continuare ceea ce am avut nevoie în cadrul aplicației.

Din momentul în care mi-am conectat aplicația la Firebase, nu m-am conectat printr-un HTTP normal. M-am conectat printr-un WebSocket, fiind mult, mult mai rapid față de HTTP. Toate datele sunt sincronizate printr-un singur WebSocket, atât de rapid cât permite rețeaua utilizatorului.

Ca prim pas ce merită menționat în utilizarea acestei tehnologii în cadrul proiectului, este acela că imediat ce mi-am creat un proiect Firebase, aceștia mi-au pus la dispoziție o serie de configurări necesare accesului, asociate doar proiectului meu. În Anexă, în figura Fig. 22 se poate vizualiza conexiunea în Firebase în cadrul aplicației.

Mi-am dorit de la început să creez o aplicație *responsive* care să poate fi accesată de pe orice fel de dispozitiv. De asemenea țin să menționez că în momentul de față cam orice aplicație trebuie să fie capabilă să se adapteze la diferite rezoluții pentru a oferi o bună experiență utilizatorului.

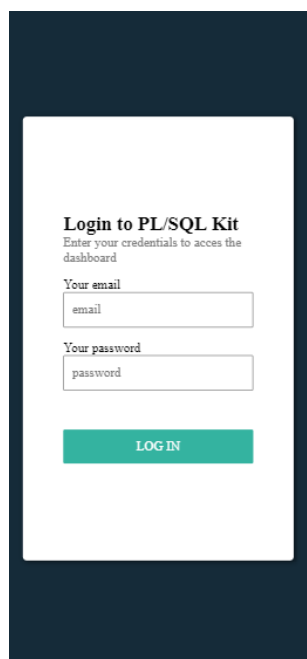


Fig. 3 Captură de ecran a paginii de Login la rezoluție de mobil

Prin Fig. 3 se poate observa adaptarea aplicației la rezoluția 375 x 812, rezoluție specifică pentru un iPhoneX.

Partea de noutăți care era dispusă în partea stângă a fost ascunsă, tocmai pentru a introduce utilizatorul în esența acestei pagini, și anume, procesul de autentificare.

Aplicația are ca și limbă, engleza, având scop versatil, pentru a putea fi folosită și de către studenții de la grupa de Engleză.

² BaaS sau Backend-as-a-Service oferă servicii bazate pe Cloud pentru procesarea din cadrul BackEnd-ului.

În cele ce urmează voi enunța principalele funcționalități ale paginii principale ale aplicației “PL/SQLKit”. Pagina principală a aplicației reprezintă de fapt nucleul întregului proiect.

Pagina principală este împărțită în trei coloane. Prima coloană este destinată instrucțiunilor sau lecțiilor pe care utilizatorul le va urmări și pe care va trebui să le rezolve. Acesta poate exersa în partea de compilator, care este de fapt a doua coloană. Pentru a putea vedea orice fel de rezultat, acesta trebuie să ruleze, sau să apese pe butonul *Run*.

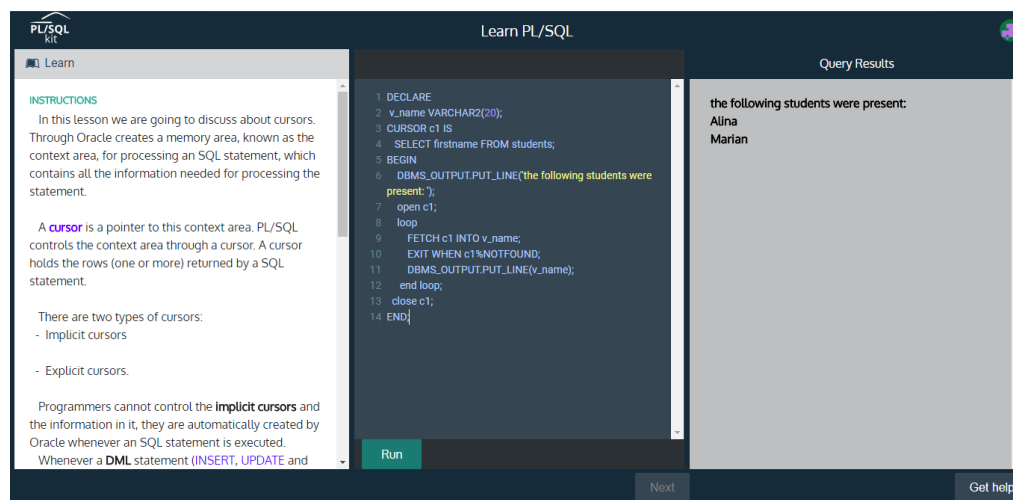


Fig. 4 Captură de ecran a paginii principale

Prin Fig. 4 se poate observa pagina principală a aplicației cu prima coloană aferentă unei lecții PL/SQL, a doua coloană fiind destinată compilatorului PL/SQL și a treia coloană conținând rezultatele aduse în urma executării procedurii din partea de compilator. De asemenea, se poate observa tematica în partea de compilator, cât și faptul că butonul de *Next* este dezactivat deoarece procedura introdusă nu este suficientă pentru a trece la următoarea lecție.

Utilizatorul are parte de o experiență foarte bună deoarece lecțiile din prima coloană sunt *up to date*, fiindu-i aduse cele mai noi enunțuri datorită bazei de date folosite, și anume, *Firestore*.

Doresc să menționez tot aici și modul în care evaluez un răspuns al utilizatorului ca fiind corect sau greșit și câte puncte atribui, în funcție de enunțul rezolvat. O lecție este rezolvată corect de utilizator dacă acesta nu are erori de execuție, mai exact, dacă răspunsul adus de pe serverul Oracle conține ceea ce era așteptat a fi afișat pentru enunțul dat. Într-o tabelă din baza de date *Firestore* țin cont de răspunsul care trebuie primit de la server, dar și de cuvintele cheie care

trebuie să se regăsească în răspunsul utilizatorului. Dacă toate aceste condiții sunt îndeplinite, atunci butonul care îi permite utilizatorului să meargă la următoarea lecție se va activa și acesta va putea accesa următoarea lecție. Porțiunea de cod care se ocupă de aceste acțiuni poate fi vizualizată în Fig. 24 din Anexă.

De asemenea, întrebările au trei grade de dificultate: ușor, mediu și greu. Punctajele sunt distribuite astfel: o lecție cu grad ușor va avea asignate cinci puncte, o lecție cu grad mediu va avea asignate zece puncte, iar o lecție de nivel greu, va avea asignate douăzeci de puncte. Porțiunea de cod care se ocupă de asignarea de puncte poate fi vizualizată în Fig. 25 din Anexă.

Nu printre ultimele lucruri care merită menționate și la care am adus o contribuție este situația în care am ținut cont ca un utilizator să nu primească aceleași enunțuri (la care a răspuns deja). Țin cont de lecțiile parcurse prin marcarea într-o tabelă destinată managementului enunțurilor, id-ului utilizatorului, id-ului lecției și numărul de răspunsuri date acesteia. Astfel probabilitatea ca o lecție la care s-a răspuns deja să fie reafișată utilizatorului este zero. Ce merită aici menționat este faptul că în momentul în care un utilizator a terminat de răspuns la toate întrebările existente, va fi atenționat asupra acestui fapt și enunțurile se vor repeta. Acesta este singurul caz în care probabilitatea discutată mai devreme va fi maximă. Porțiunea de cod care se ocupă de afișarea întrebărilor care nu au primit încă un răspuns cu prioritate mai mare față de cele la care s-a dat deja un răspuns poate fi vizualizată în Anexă în Fig. 26.

Dificultatea în folosirea acestei tehnologii a venit în momentul în care am construit modelul de evaluare a răspunsurilor unui utilizator, dar și în momentul în care căutam o metodă de afișare a enunțurilor la care s-a răspuns deja, față de cele care nu au fost încă parcurse. Firebase este folosit cu ajutorul JavaScript, iar sursele de cercetare au fost suficiente pentru a ieși din impas atunci când a fost cazul.

Pentru partea de compilator sau coloana din mijloc a paginii, utilizatorul are parte de o tematică și de numerotarea liniilor. Așa cum se poate observa în Fig. 4, linia de cod cu numărul 6 fiind suficient de lungă, nu a determinat apariția unui scrollbar orizontal, ci aceasta aparține în continuare aceleiași linii. Această parte de compilator dispune și de o listă de autosugestii care poate fi accesată la combinația tastelor *Ctrl+Space*. În cele ce urmeză voi expune o captură de ecran ce va surprinde această funcționalitate. De asemenea, și această pagină este *responsive*, coloanele paginii adaptându-se corespunzător în funcție de rezoluția ecranului dispozitivului folosit.

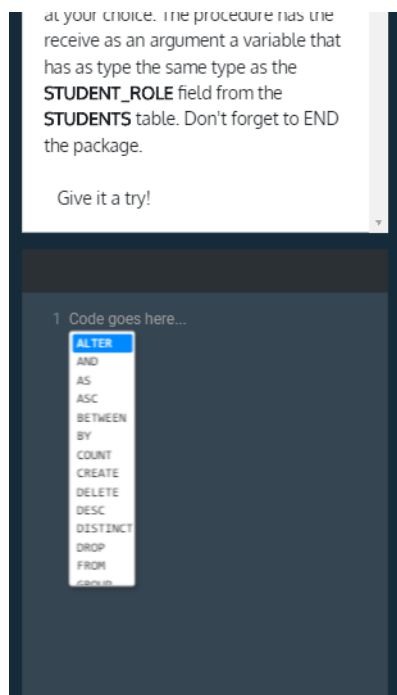


Fig. 5 Captură de ecran a paginii principale la rezoluție de mobil

Evidențiez prin Fig. 5 cum arată lista de autosugestii pentru partea de compilator (lista albă din partea de compilator). De asemenea, captura de ecran este la rezoluție de mobil pentru a se putea observa acest aspect tot aici.

Țin să menționez că orice fel de procedură SQL sau PL/SQL este rulată în partea de compilator este executată într-un server de Oracle. Răspunsul pe care utilizatorul îl primește este adus, de asemenea, din serverul Oracle. Utilizatorul are nevoie ca experiența sa în cadrul limbajului PL/SQL să fie cât mai autentică, de aceea un răspuns predefinit nu ar fi fost o soluție.

Pentru acest proiect am ales să configurez separat serverul de Apache împreună cu limbajul de scripting PHP, nefolosind o variantă de XAMPP. Motivația vine din faptul că XAMPP poate veni cu versiuni mai vechi și mai slabe ale acestor pachete, iar astfel mi-am ales cele mai noi versiuni, cu diferențe majore în performanță.

Așa cum am menționat și în capitolul 1.2 *Scurt cuprins al tehnologiilor utilizate*, dacă prin intermediul CodeMirror, am reușit să conturez ideea de code editor, era nevoie ca procedurile introduse și rulate de către utilizator, să fie executate într-un server Oracle, și să întoarcă, bineînțeles, un rezultat.

După ce am conectat Apache la PHP și PHP la Oracle server, a început o altă parte complexă din cadrul proiectului de licență. A fost necesar să găesc o abordare pentru fiecare dintre tipurile de instrucțiuni care ar putea fi cerute de către utilizator. Specific acest lucru deoarece nu primeam rezultate pentru toate tipurile de proceduri, sau chiar primeam erori pentru

instrucțiuni care erau mai complexe decât o simplă interogare, cu funcțiile clasice oferite de PHP care urmau pașii:

- *oci_connect* , se făcea conexiunea la baza de date;
- *oci_parse*, se parsea instrucțiunea scrisă de utilizator la baza de date;
- *oci_execute*, se executa instrucțiunea în baza de date;
- *oci_fetch_array*, se aduceau rezultatele obținute, care erau ulterior afișate utilizatorului.

Consider acesta primul caz din cele ce urmează. În cadrul acestui scenariu, folosesc doar aceste funcții și îl consider a fi cel mai simplu, urmând un *happy flow*³, în care se execută simple interogări sau se aduc avertismente utilizatorului când acesta vrea să execute formulări invalide. Un exemplu de folosire pentru returnarea rezultatelor poate fi vizualizat în Fig. 27 din Anexă.

Urmează să relatez cel de al doilea caz, în care utilizatorul dorește să execute proceduri, funcții, sau blocuri PL/SQL. Aici a fost necesară tratarea lor într-o manieră diferită. Aceste tipuri de date sunt considerate de către PHP resurse și în loc de obișnuitul *oci_fetch_array*, care nu a funcționat deloc în acest caz, a fost necesară introducerea funcției *oci_get_implicit_resultset*. Aceasta m-a ajutat să preiau seturi consecutive de rezultate de interogare, după executarea unui bloc Oracle PL/SQL stocat sau anonim. Un exemplu de folosire al acestei funcții poate fi vizualizat în Fig. 28 din Anexă.

Printre cele mai dificile cazuri a fost tratarea scenariului în care utilizatorul dorea să obțină printr-o procedură Oracle, *dbms_output.put_line*, anumite date, rezultate. După ce am făcut cercetări am înțeles că acest lucru nu este posibil cu PHP, chiar dacă setam variabila de mediu, *set serveroutput*, ca fiind *on*. Aceasta este în mod obișnuit folosită pentru printarea argumentelor sau valorilor din *dbms_output.put_line* în consolă în serverul de Oracle.

Soluția găsită pentru acest ultim caz o voi descrie în cele ce urmează.

Am folosit o *funcție pipeline*⁴ PL/SQL, ceea ce a adus în final și un aport de performanță, executată inițial în serverul Oracle. În fișierul PHP prin intermediul unei funcții am folosit clauza *CALL* din PL/SQL pentru a executa funcția *dbms_output.put_line* de sine stătătoare. Această funcție a urmat a fi executată pentru fiecare apariție a unei instrucțiuni de tipul *dbms_output.put_line* scrisă de utilizator. Porțiunea de cod care se ocupă de tratarea cazurilor de

³ Happy flow este considerat a fi un caz fericit în cazul utilizării unei aplicații, un caz fără condiții excepționale sau de eroare.

⁴ O funcție pipeline returnează rezultatele imediat ce sunt produse.

execuție a instrucțiunilor de tip *dbms_output.put_line* în aplicație poate fi vizualizată în Fig. 30 din Anexă.

Următoarea funcție lucrează împreună cu situația descrisă mai sus. Cele două au reprezentat soluția pentru ultimul scenariu posibil. Aceasta este o funcție care preia fiecare rând din tabela creată de funcția pipeline. Se construiește un vector de argumente din *dbms_output.put_line* care este ulterior returnat și afișat utilizatorului. Porțiunea de cod care tratează acest caz poate fi vizualizată în Fig. 29 din Anexă.

De asemenea, doresc să menționez faptul că am întâmpinat probleme atunci când a fost cazul de afișare a rezultatelor ce proveneau dintr-un cursor sau procedură PL/SQL. Funcția *oci_fetch_all* din PHP mi-a permis să preiau mai multe rânduri dintr-un anumit tip de interogare, care să fie afișate ulterior.

După cazurile prezentate și descrierea aplicării funcțiilor de mai sus, doresc ca în cele ce urmează să evidențiez prin descrierea unor scenarii, modul în care se execută interogările la baza de date în funcție de datele introduse de utilizator în compilator. Prin aceste scenarii vreau să clarific mai bine cazurile tratate în lucrare.

- Utilizatorul nu introduce nimic în compilator și rulează ceea ce a introdus.

În acest caz, nu se va trimite nimic la serverul Oracle, anunțând utilizatorul asupra faptului că trebuie să fie introdus măcar un caracter.

- Utilizatorul a introdus un șir aleatoriu de caractere și rulează ceea ce a introdus.

Datele se vor trimite și executa în serverul de Oracle, iar rezultatul afișat utilizatorului va fi unul tot de atenționare, primind o eroare sau un avertisment pentru caracterele invalide introduse.

- Utilizatorul a introdus o interogare și rulează ceea ce a introdus.

Datele vor fi afișate în funcție de tipul interogării. De exemplu, dacă este un SELECT pentru afișarea datelor dintr-un tabel, acestea vor fi afișate tot în format de tabel. Pentru interogări care nu au motiv să returneze un rezultat anume, nu se va afișa nimic, cum ar fi în cazul unui UPDATE.

- Utilizatorul a introdus un bloc anonim/procedură/funcție fără a conține *dbms_output.put_line* și rulează ceea ce a introdus.

Utilizatorul primește ca și rezultat fraza *"PL/SQL procedure successfully completed."*, dacă procedura a fost scrisă corect, sau un mesaj de eroare, în funcție de corectitudinea codului său.

- Utilizatorul a introdus o structură care conține *dbms_output.put_line* și rulează ceea ce a introdus.

Indiferent de câte astfel de linii și indiferent de tipul argumentelor, utilizatorul va putea vedea pentru fiecare un rezultat afișat. Dacă el dorește afișarea unui set de caractere, a unui simplu număr sau a valorii unei variabile asignate altundeva în cod, acest lucru este posibil.

- Utilizatorul a introdus un cursor sau o instrucțiune iterativă care conține o singură instrucțiune de tip *dbms_output.put_line*, care trebuie să returneze mai multe rezultate.

Lucrul cu aceste tehnologii a constituit o provocare, dar rezultatul este de fapt baza acestui proiect. Gradul de dificultate întâmpinat a fost mare atunci când a trebuit să separ instrucțiunile pe cazuri, dar mai ales faptul că PHP nu a oferit suport pentru funcția *dbms_output.put_line* din Oracle. Deși am mai lucrat cu PHP și la alte proiecte din facultate, prin proiectul acesta s-a conturat o mai mare experiență.

Pentru ca utilizatorul să treacă la următoarea lecție, butonul *Next* trebuie să fie disponibil. În mod obișnuit acest buton este dezactivat pentru a îndemna utilizatorul să fie perseverent în rezolvarea fiecărei lecții în parte. Butonul devine activ în momentul în care toate instrucțiunile au fost atinse și executate cu succes. Accesarea sa îi va pune la dispoziție următoarea lecție. Faptul că o lecție a fost rezolvată cu succes va fi notată prin creșterea punctajului utilizatorului în funcție de dificultatea lecției. Acest progres poate fi vizualizat în pagina de profil, ale cărei funcționalități le-am descris în paginile ce urmează.

Pentru moment voi descrie o altă funcționalitate a paginii principale, și anume curățarea spațiului de lucru al utilizatorului printr-un simplu click. Această funcționalitate poate fi accesată prin butonul *Get Help* din pagina principală. Așa cum se putea observa în Fig. 4, acest buton este dispus în partea dreaptă jos a paginii. În cele ce urmează voi evidenția modul în care arată această funcționalitate în aplicație.

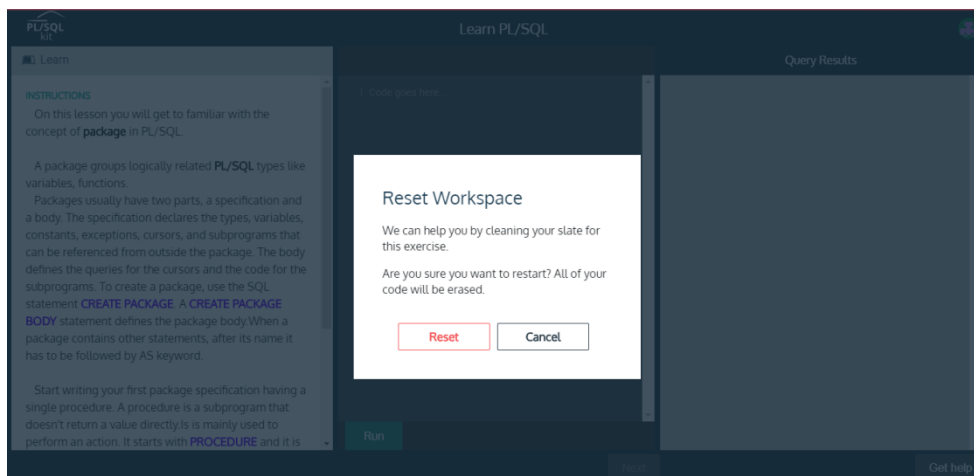


Fig. 6 Captură de ecran ce surprinde funcționalitatea de Reset

Prin Fig. 6 se poate observa cum arată aplicația după accesarea funcționalității de *Get Help*. Accesarea butonului de *Reset* va reseta spațiul de lucru prin ștergerea atât a procedurilor scrise în partea de compilator, cât și a rezultatelor primite, dacă au fost afișate.

Tot în această pagină poate fi observată în partea dreaptă sus, imaginea de profil a utilizatorului. Prin accesarea imaginii, utilizatorul va fi redirectionat către pagina sa de profil despre care am menționat anterior. Pagina de profil este destinată accesării funcționalității de *LogOut*, sau mai important, pentru vizualizarea progresului făcut în aplicație ori opțiunea de se reîntoarce la partea de compilator.

În rândurile ce urmează este atașată o captură de ecran a acestei pagini, pentru a înțelege mai ușor structura sa.

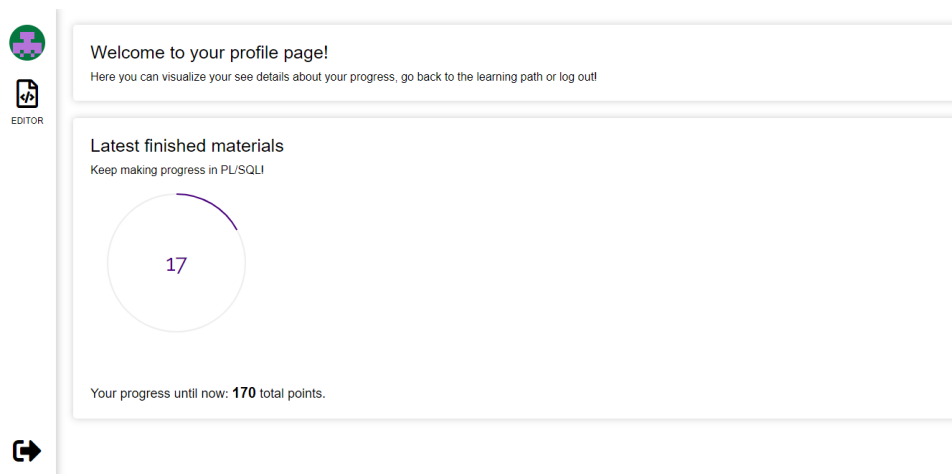


Fig. 7 Captură de ecran a paginii de profil

Evidențiez prin Fig. 7 o captură de ecran ce surprinde pagina de profil a utilizatorului. Aceasta înglobează funcționalitatea de *LogOut*, vizualizarea progresului făcut prin lecțiile parcurse cu succes, dar și opțiunea pentru utilizator de a își relua activitatea prin întoarcerea la partea de editor.

Deși la o primă vedere această pagină poate părea simplă, ea face parte din structura esențială a unei aplicații. Menționez că cercul de progres este dinamic și este o evidență a punctajului înregistrat în baza de date. Cu *ProgressBar.js*, a fost ușor să creez un cerc elegant și *responsive* pentru pagina de profil a utilizatorului. Încă de la prima schiță a paginii mi-am dorit o astfel de animație care să aducă la cunoștință utilizatorului progresul făcut.

ProgressBar.js vine cu câteva forme gata construite, cum ar fi linie, cerc sau semicerc, dar ei oferă și posibilitatea de a crea și stiliza o formă în modul propriu. Este un proiect licențiat de MIT, la fel ca și librăria *CodeMirror*. Ei dispun de suficiente exemple de cod, de la care am și pornit în primă etapă.

ProgressBar.js folosește la rândul său o bibliotecă pentru a anima traseul stabilit. Animația se face cu JavaScript și astfel se oferă mai mult control asupra ei. Despre cum am integrat mai exact acest progress circle în lucrarea mea și ce valoare aduce aplicației, voi specifica în următoarele rânduri.

Mi-am dorit să îi ofer utilizatorului o experiență plăcută atunci când dorește să își vizualizeze progresul pe care l-a făcut sau mai specific, câte punctaje a acumulat până în acel moment. Progresul poate fi vizualizat în pagina de profil.

Procentajul afișat în progress circle este dinamic, el modificându-se în funcție de punctajul obținut de utilizator. Am reușit să obțin acest lucru prin folosirea JavaScript, deoarece atât

ProgressBar.js, cât și *Firebase*, unde sunt reținute datele fiecărui utilizator, se prelucrează cu acest limbaj.

Astfel închei capitolul destinat prezentării funcționalităților principale ale aplicației care poate fi considerat și o sursă pentru manualul de utilizare al aplicației.

Pot afirma faptul că rezultatul final se află la egalitate cu așteptările setate inițial. Dificultățile întâlnite au fost numeroase pe parcursul dezvoltării. Printre cele mai importante pot enumera faptul că PHP nu a oferit un suport pentru funcția de afișare din PL/SQL, *dbms_output.put_line*, pentru care a trebuit să folosesc o funcție pipeline din Oracle care să aducă rezultatele de acest tip. Această soluție a adus în final și un raport de performanță aplicației prin faptul că nu s-a folosit structura clasică de apelare a acestei funcții. Rezultatele obținute au fost următoarele:

- Pentru primul apel la baza de date:
 - Preluarea datelor folosind *dbms_output.get_line()*: 14.674 secunde.
 - Preluarea datelor din funcția pipeline folosită: 1.162 secunde.
- Pentru al doilea apel la baza de date:
 - Preluarea datelor folosind *dbms_output.get_line()*: 14.714 secunde
 - Preluarea datelor din funcția pipeline folosită: 1.44 secunde
- Pentru al treilea apel la baza de date:
 - Preluarea datelor folosind *dbms_output.get_line()*: 14.385 secunde
 - Preluarea datelor din funcția pipeline folosită: 1.131 secunde

Concluzia este aceea că funcția pipeline este semnificativ mai rapidă și diferența este vizibilă pentru un număr mare de rânduri care sunt preluate din baza de date.

Faptul că am pornit având ca și sursă de inspirație pe cei de la <https://codecademy.com>, m-a ajutat prin a crea o un kit cât mai practic pentru utilizator, și pot afirma cu mulțumire că am reușit prin aplicația mea să ofer ceea ce ei nu pun la dispoziție, și anume un mod de exersare PL/SQL. Chiar și tehnologia folosită pentru a crea un text editor nu a pus la dispoziție un modul pentru PL/SQL, ci doar SQL, de la care a trebuit să dezvolt.

Aplicația “*PL/SQLKit*” semnifică, cel puțin personal, un kit foarte bun de exersare (de la care vine și numele aplicației) pentru studenți. Acest proiect înglobează lecții, instrucțiuni detaliate, parte de exersare și are și o parte motivantă prin a găsi cât mai multe răspunsuri corecte pentru a ajunge la un progres cât mai însemnat.

În capitolele ce urmează voi intra mai în detaliu pe partea de structură a soluției, modul în care am gândit arhitectura aplicației, relațiile dintre tabele, și nu numai.

2.3 Structura aplicației. Diagrame UML

Pentru a ajuta la formarea unei perspective asupra modului în care este structurată aplicația, am ales să evidențiez prin acest subcapitol, o serie de diagrame UML. Aceste diagrame au fost realizate personal prin folosirea aplicației *ArgoUML*. ArgoUML mi-a dat posibilitatea să surprind secvențele de acțiuni care pot fi executate de către entitățile din afara programului denumiți uzual actori.

În cele ce urmează voi reprezenta prin Fig. 8 modul în care obiectele comunică în aplicație din punctul de vedere al acțiunilor care pot fi făcute, dacă sunt îndeplinite anumite criterii (autentificare cu succes), printr-o diagramă use-case.

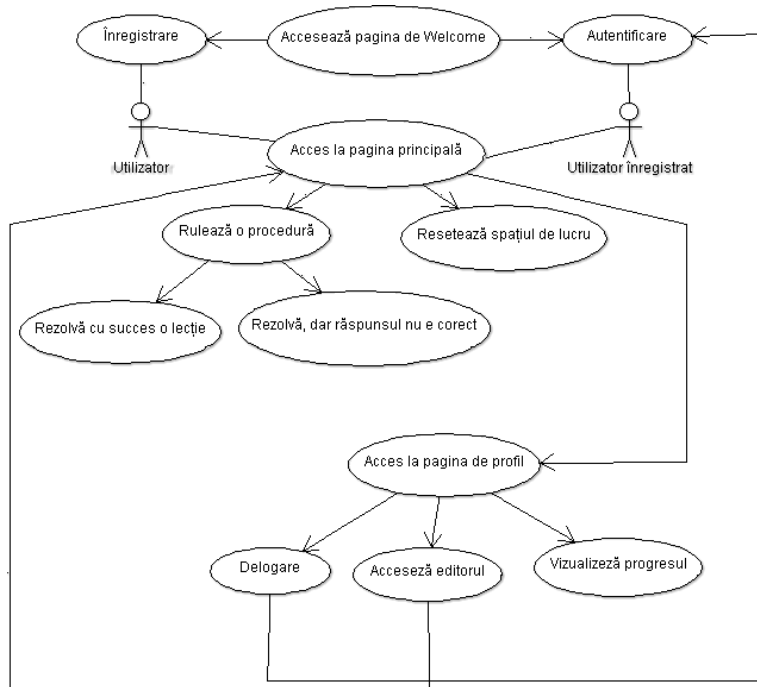


Fig. 8 Diagrama use-case a aplicației

În următoarele rânduri voi evidenția o diagramă de activitate pentru a reda o execuție secvențială a unor acțiuni în aplicație.

În Fig. 8 se poate observa structura întregii aplicații și acțiunile care pot fi făcute în aceasta. Așa cum se poate observa, procesul de autentificare este necesar pentru a avea acces la orice fel de funcționalitate din aplicație. Ulterior, utilizatorul poate avea acces la celelalte pagini, care sunt cea principală, și cea de profil.

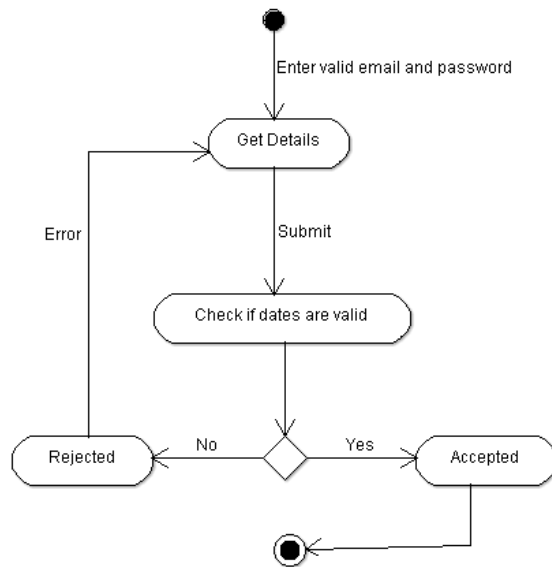


Fig. 9 Diagramă de activitate a aplicației

Evidențiez prin Fig. 9 o diagramă de activitate a aplicației ce surprinde aspectele dinamice ale sistemului.

De asemenea, această diagramă de activitate redă pașii pentru execuția secvențială a unor acțiuni în aplicație.

Astfel închei acest capitol prin a concluziona faptul că structura aplicației poate fi înțeleasă prin urmărirea diagramelor de mai sus care surprind principalele acțiuni ale aplicației și cum se realizează accesul pentru acestea.

2.4 Arhitectura aplicației

În acest capitol voi descrie detaliile arhitecturale ale aplicației, mai exact, voi oferi o perspectivă, în primul subcapitol, a cum este organizată soluția și motivația care suportă această alegere. De asemenea, în cel de al doilea subcapitol voi exemplifica structura bazei de date în Firebase, însoțită de motivația aferentă.

Mi-am dorit să evidențiez această parte de structură într-un capitol separat deoarece sunt mult mai ușor de înțeles nivelele care fac parte din aplicație și oferă și o perspectivă a cum m-am organizat în dezvoltarea acestui proiect. Menționez acestea deoarece o structură bine aleasă reprezintă jumătate dintr-o lucrare bine făcută.

2.4.1 Detalii arhitecturale

În acest subcapitol vreau să motivez alegerea structurării soluției, cât și motivația pentru care este organizată astfel.

Funcționalitățile componentelor sunt împărțite în secțiuni, și anume marcate ca și foldere în Visual Studio, dar și în soluția locală. În cele ce urmează voi motiva de ce sunt mai multe foldere în loc de unul singur, de exemplu, și de ce este o bună practică să fie structurate astfel.

În aplicația mea am ales să creez foldere în funcție de funcționalitatea pe care o înglobează cât și pentru structurarea în arii de interes. Denumirile folderelor sunt compuse din numele funcționalităților pe care le conțin și o să evidențiez printr-un exemplu în figura de mai jos.

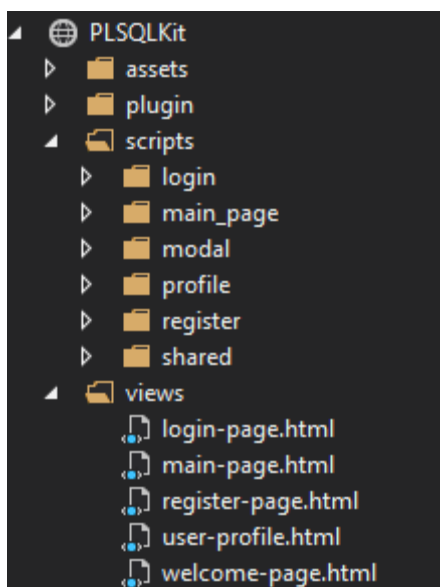


Fig. 10 Imagine ce surprinde structurarea soluției

În Fig. 10 se poate observa structura proiectului.

Așa cum am specificat mai sus, folderele sunt structurate pe arii de interes și pe funcționalități.

O astfel de structurare ajută la o organizare optimă și la o înțelegere ușoară.

Motivația alegerii unei astfel de structurări vine din faptul că ajută la o organizare optimă a soluției. Atunci când caut o funcționalitate anume, trebuie să știu în ce modul se găsește. În cele ce urmează voi exemplifica modulele create.

Folderul *assets* este gândit astfel încât să conțină fișierele *css* de stilizare ale aplicației, cât și imaginile care se găsesc în aplicație cum ar fi imaginea logo-ului aplicației care se găsește în pagina principală a aplicației, favicon-ul prezent în fiecare pagină, cât și imaginea de profil a utilizatorului prezentă pe pagina de profil, cât și pe pagina principală.

Folderul *plugin* conține versiunea 5.38.0 a text editorului *Codemirror*. Am păstrat versiunea pe care ei o pun la dispoziție pentru descărcare, în care se găsește o altă structurare, pe fișiere ce țin de stilizare, tematică și documentație oficială.

CodeMirror este un code-editor care se poate încorpora ușor într-o pagină Web. De când am pornit de la ideea acestei aplicații, mi-am dorit ca partea aceasta de compiler să fie punctul de

maxim interes. Componenta aceasta reprezintă spațiul de lucru al utilizatorului, unde el poate exercita și își poate evalua structurile scrise în funcție de rezultatul primit. Fiind un proiect open-source, editorul pe care cei de la *CodeMirror* îl oferă, este folosit de către diverse instrumente de dezvoltare cum ar fi Chrome, Adobe Brackets, Bitbucket și multe altele. Cercetând, am descoperit că și cei de la *codecademy.com*, sursa mea principală de inspirație în construirea propriei aplicații, folosesc același code editor.

Ceea ce mi-a oferit de fapt *CodeMirror* în aplicația mea a fost transformarea unui simplu container `div`⁵ într-o porțiune de code editor, o componentă care să fie interactivă pentru utilizator. Partea aceasta dispune de numerotarea liniilor, sugestii de autocompletare și o tematică de colorare a cuvintelor cheie din codul scris de către utilizator.

CodeMirror mi-a oferit accesul la aceste funcții menționate anterior. Nucleul acestei biblioteci oferă doar componenta de text editor. Alte funcționalități de la care se poate porni și implementa ulterior sunt disponibile doar prin intermediul API-ului său.

S-a depus un efort destul de mare pentru acesta componentă. Gradul de dificultate pe care l-am întâlnit în folosirea ei a depins de faptul că nu aveam experiență cu ea. A contat foarte mult partea de cercetare pentru funcțiile pe care le-am folosit ulterior.

Această librărie are foarte multe de oferit, mai ales pentru multe alte limbaje. Pentru PL/SQL, ceea ce aveam eu nevoie, nu are nimic predefinit, de aceea a trebuit să îmbin ceea ce avea disponibil, cu ceea ce îmi doream ca și rezultat final. De asemenea, a fost dificil și integrarea unui *textarea* după ce făcusem inițial interfața paginii. Am avut probleme în a o stiliza și a o integra în tot containerul în care îmi doream să fie compilatorul.

Folderul *scripts* este structurat la rândul său pe alte subfoldere destinate fiecărei pagini sau funcționalități. Fișierele care se găsesc în aceste foldere sunt de tip JavaScript, iar local, aici se găsesc și fișierele de scripting PHP. Subfolderul *shared* conține fișierele care nu sunt destinate unei singure pagini, ci sunt necesare mai multor fișiere, cum ar fi cele care țin de inițializarea Firebase. De asemenea, am ales să folosesc un fișier de rutare pentru redirectionarea între pagini la acțiunea unui buton apăsat de către utilizator. Motivația vine din faptul că a fost mai ușor de stilizat și de manageriat decât obișnuitul *href*⁶.

⁵ Un `<div>` definește o secțiune din documentul HTML. Acest element este deseori folosit ca și container pentru alte elemente HTML pentru a îl putea stiliza și executa anumite sarcini împreună cu JavaScript.

⁶ Atributul `html`, *href*, este folosit pentru a trimite utilizatorul de pe o pagină pe alta, la acțiunea de click a unui buton (pentru exemplu).

```
function redirectToLoginPage() {
    document.location.href = "../views/login-page.html";
}

function redirectToRegisterPage() {
    document.location.href = "../views/register-page.html";
}
```

Fig. 11 Exemplu de funcții de rutare

Am ales să exemplific prin Fig. 11 două funcții de rutare care sunt apelate la acțiunea de click a unui buton.

Folderul *views* conține toate fișierele *html* din aplicație care asigură structurarea paginilor. Aprin urmare, aceasta este structurarea folosită pentru această lucrare. Așa cum am mai menționat, motivația alegerii unei astfel de structuri vine din faptul că am mai fost obișnuită cu o astfel de structurare și la alte proiecte din facultate, dar și faptul că este ușor de înțeles structura aplicației astfel.

2.4.2 Structura bazei de date în Firebase

Firebase este una dintre tehnologiile de bază care stau la dezvoltarea acestei lucrări. Firebase este un Backend-as-a-Service și oferă servicii bazate pe Cloud pentru procesarea din cadrul Backend-ului.

Ca prim pas pe care a trebuit să îl fac înainte de a începe să lucrez cu *Firebase Realtime Database*, a fost să schițez cum anume vor fi datele salvate, ca apoi mai târziu, să îmi fie ușor să preiau și să prelucrez aceste date.

Firebase are datele stocate ca obiecte JSON. Mai exact, un arbore JSON găzduit în Cloud. Spre deosebire de o bază de date SQL, nu există tabele sau înregistrări. Când adaug date în arborele JSON, acesta este de fapt un alt nod al arborelui cu o cheie asociată. Deși Firebase permite o structură care să conțină date *nested*⁷ până la un nivel egal cu 32, trebuia ținut în vedere acest lucru. Menționez acestea deoarece în momentul în care se aduc anumite date din arbore, pot fi aduse și toate nodurile copii aferente aceluia nod. În aceste cazuri pot apărea probleme de performanță. De aceea, în baza mea de date, structura este cât mai simplă posibilă.

Firebase pune la dispoziție o interfață din care poți vizualiza ca și deținător al proiectului baza de date. Pentru proiectul meu, aceasta poate fi accesată la adresa: <https://console.firebase.google.com/project/plsqlkit/database/plsqlkit/data>.

În continuare voi detalia modul în care este proiectată structura bazei de date în Firebase și care este legătura dintre nodurile principale.

⁷ Date *nested* sunt acele date care aparțin unui grup evidențiind relația părinte-copil sau rădăcină-descendenți.

Pentru necesitățile proiectului meu, am structurat baza de date în felul următor:

- Un nod principal este cel care ține evidența enunțurilor întrebărilor, nodul numindu-se *questions*. Acesta are ca și copil câte un nod de tip lecție. Fiecare lecție are la rândul său alte noduri copii destinate managementului lecțiilor, cum ar fi enunțul sau descrierea, care sunt afișate utilizatorului și care trebuie să fie parcurse de către acesta, un nod ce conține id-ul întrebării și un alt nod care conține id-ul răspunsului. Este necesar a ține în evidență id-ul răspunsului deoarece astfel facem legătura între nodul *questions* și nodul *answers*.
- Un nod *answers* care la rândul său conține un id unic, id-ul întrebării care permite legătura cu nodul *questions* și o parte de descriere. Descrierea conține de fapt cuvinte cheie care trebuie să se regăsească în răspunsul utilizatorului astfel încât să se evite posibilitatea în care utilizatorul ar fraudă un răspuns.
- Nodul *userQuestions* ține evidența răspunsurilor date la lecții de către utilizator. Acest lucru ajută la îmbunătățirea procesului de evaluare al utilizatorului prin faptul că acestuia nu îi va fi afișată o lecție de două ori. Incrementarea în subnodul *answers* se face în momentul în care s-a răspuns cu succes la o lecție și astfel această întrebare va fi marcată, și nu va mai fi afișată utilizatorului decât după ce a parcurs toate celelalte lecții și va fi momentul să fie reluate. Pentru a ști la care dintre întrebări s-a răspuns, am făcut legătura dintre acest nod principal și cel de *questions*, prin *questionID*.
- Nodul *userScores* permite managementul utilizatorilor în aplicație. Pentru fiecare nou cont creat se va crea automat un alt nod copil ce va ține evidența, în mare a progresului făcut în aplicație.

Structura bazei de date în Firebase și mai exact cum sunt dispuse aceste noduri se poate vizualiza în Fig. 12 de mai jos. Am dorit să atașez o captură de ecran cu aceasta deoarece am considerat că este necesară înțelegerea modului în care se face managementul lecțiilor, a răspunsurilor și a informațiilor utilizatorilor în aplicație.

Rapiditatea cu care se crează noduri în baza de date din Firebase după rularea unui script, depinde de viteza rețelei pe care o are utilizatorul în acel moment. De aceea pot menționa că am întâlnit mici impedimente în momentul în care puterea rețelei în care lucram era scăzută și nodurile nu erau create pe moment.

Ștergerea și recrearea bazei de date se poate face foarte simplu, deoarece după ce s-au șters toate nodurile care aparțin acestui proiect, sau doar o parte, din interfața pe care Firebase o pune la dispoziție, se poate repopula baza de date prin reîncărcarea scriptului din soluție. Mai multe detalii despre partea importantă a acestui script care este de fapt conținutul lecțiilor din aplicație, voi vorbi în capitolul 2.8 *Lecțiile PL/SQL*.



Fig. 12 Captură de ecran ce evidențiază structura bazei de date în Firebase

Prin Fig. 12 doresc să clarific structura bazei de date în Firebase din cadrul proiectului meu.

După cum se poate observa, nodurile principale ce asigură managementul aplicației sunt nodurile *questions*, *answers*, *userQuestions* și *userScores*.

Mai multe detalii despre cum se face legătura între ele și de ce sunt necesare, este exemplificat mai sus.

Astfel închei acest subcapitol destinat structurii de baze de date în *Firebase* prin a menționa, nu în cele din urmă, că este relativ ușor de a lucra cu *Firebase*, iar impedimentele pe care le-am întâlnit în lucrul cu aceasta au fost legate mai mult de faptul că sincronizarea datelor a depins foarte mult de viteza rețelei.

2.5 Platforma de management folosită

Din punctul de vedere al organizării soluției am folosit o platformă de management numită pe scurt TFS. TFS sau Team Foundation Server, m-a ajutat atât pe partea de organizare a sarcinilor care trebuiau îndeplinite, cât și pe partea de integrare a codului scris, prin faptul că oferă posibilitatea de depozitare a codului.

Pentru a putea conecta un proiect la TFS, sau mai bine zis, pentru a depozita soluția pe această platformă de management, soluția trebuie construită în *Visual Studio*. De aceea pot enumera altă tehnologie pe care am folosit-o, și anume, aceasta din urmă menționată. Pentru

început am creat o soluție în Visual Studio pe care am conectat-o la proiectul creat în TFS. Ulterior, această soluție a crescut, fiind structurată cât mai optim, pentru a putea găsi ușor fișierele de care aveam nevoie. Despre structurarea soluției voi vorbi în subcapitolul 2.4.1 *Detalii arhitecturale*.

TFS este o platformă creată de Microsoft care oferă accesul la un backlog, ce conține cerințe ce trebuiesc îndeplinite pentru a dezvolta aplicația, denumite pe scurt PBI-uri. Cerințele au fost scrise de către mine în totalitate având în minte structura la care voiam ca aplicația să ajungă. Acronimul de PBI se traduce prin *Product Backlog Item* și poate conține atât cerințe funcționale, cât și nonfuncționale. Cerințele funcționale se referă la schimbări vizibile de către utilizator, prin adăugarea sau ștergerea unor elemente din interfața grafică. Cerințele nonfuncționale se referă, bineînțeles, la schimbările neobservabile de către utilizator, cum ar fi schimbările din structura codului, dar care păstrează funcționalitatea.

TFS mi-a oferit șansa să mă organizez optim prin faptul că la început de sprint îmi scriam PBI-urile pe care doream să le îndeplinesc și cu ajutorul task-urilor destinate fiecărui PBI, puteam să fac și o parte de management al timpului care urma să fie destinat implementării.

De asemenea, prin faptul că am avut acces la un *board* vizual, am putut să analizez progresul făcut prin actualizarea stărilor task-urilor la care lucram din *New*, în *Active*, care mai apoi să fie *Resolved*, și în final să ajungă în starea de *Closed*. De asemenea, atunci când terminam funcționalitatea unui PBI, starea acestuia se schimba din *Active* în *Closed*, așa cum se poate observa în figura de mai jos.



Fig. 13 Previzualizare a Board-ului din TFS

Am ales să exemplific prin Fig. 13 cum arată Board-ul în TFS dintr-un sprint. Se poate observa starea unui PBI care a fost dus la îndeplinire și căruia i-am schimbat starea din *Active*, în *Closed*.

Țin să menționez faptul că în dezvoltarea acestui proiect mi-am dorit să lucrez Agile în sprint-uri a câte trei săptămâni. Prin acest fapt demonstrez că am folosit elemente de *Ingineria programării*. Beneficiul folosirii acestei platforme de management a fost acela că proiectul s-a putut descompune în unități administrabile, dar și că s-a putut urmări progresul făcut prin vizualizarea commit-urilor făcute. Commit-urile sunt datate, dar și însoțite de un comentariu, pentru a se putea urmări ușor ce s-a implementat. De asemenea este și o metodă bună de a ști în ce fișiere s-au făcut schimbări, fiind o soluție perfectă de backup deoarece TFS oferă posibilitatea de a depozita codul.

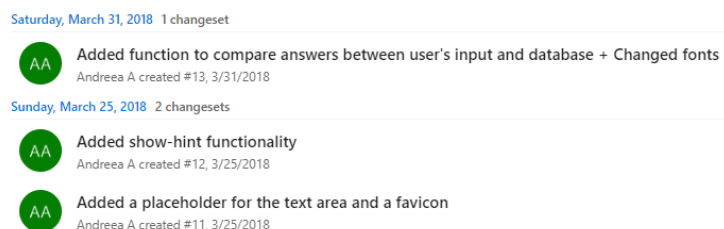


Fig. 14 Exemplu de commit-uri făcute în soluția TFS

Am ales să evidențiez prin Fig. 14 cum arată commit-urile făcute în soluția din TFS. Faptul că acestea sunt datate și sunt însoțite de un comentariu, constituie o evidență a progresului în dezvoltarea aplicației.

Merită menționat faptul că am lucrat în *Visual Studio* încă de la început pentru că focusul meu principal în prima etapă a fost să construiesc tot ce ține de interfața grafică, cu tot ceea ce utilizatorul avea să interacționeze. Ulterior, când am început să lucrez la partea de *BackEnd*, și am ales să folosesc *PHP*, am observat că *Visual Studio* nu oferă instrumente gratuite pentru folosirea acestui limbaj. De aceea, dezvoltarea a continuat separat de *Visual Studio*, însă am continuat să folosesc *TFS* pentru urmărirea progresului și logarea cerințelor pe care le-am avut de făcut.

Nu am avut dificultăți în a folosi această platformă de management deoarece am folosit-o și înainte, la alte proiecte din facultate. Apreciez foarte mult această platformă prin faptul că se poate vizualiza foarte ușor progresul făcut în dezvoltarea unei aplicații.

2.6 Găzduirea aplicației. DigitalOcean

Pentru ca aplicația să fie la îndemâna studenților, aceasta a fost găzduită pe una dintre mașinile virtuale a celor de la DigitalOcean. Aceștia oferă o platformă de cloud computing, în care îți poți gestiona infrastructura aplicației într-un mod cât mai eficient. Aceștia mi-au oferit

mai mult de o mașină virtuală, mi-au oferit posibilitatea de a crea propriul *droplet*⁸ care reprezintă în momentul curent locul în care este găzduită aplicația. Un droplet are și capacități suplimentare de securitate și monitorizare, pe lângă cele de stocare, pentru ca aplicația să poată rula. Ceea ce merită menționat este faptul că accesul la oricare dintre servicii se face conform unor subscripții plătite, lunar.

Configurarea proiectului pe mașina virtuală la care am primit acces, cât și setarea și instalarea tuturor programelor necesare rulării proiectului de licență s-a făcut doar din terminal, fiind o mașină virtuală Ubuntu, la a cărei interfață nu am putut avea acces. Am întâmpinat dificultăți în procesul de configurare, în special la setarea bazei de date Oracle. Ce pot însă menționa pozitiv despre această platformă de domenii este faptul că au o documentație foarte elaborată care m-a ajutat să trec peste impedimentele întâlnite. De asemenea, o altă problemă întâlnită este aceea de a avea versiuni diferite de PHP configurate față de versiunea locală a aplicației, ce determină un comportament diferit față de cel așteptat al aplicației.

Versiunea PHP de pe server este PHP 7.0.30 Ubuntu, iar versiunea de Apache este Ubuntu Apache 2.0.

Din punctul de vedere al încărcării fișierelor ce alcătuiesc proiectul, am folosit programul WinSCP⁹. Acesta mi-a permis transferul de fișiere între calculatorul local și mașina virtuală la care am acces. Astfel, proiectul se poate sincroniza ușor cu ultima versiune locală.

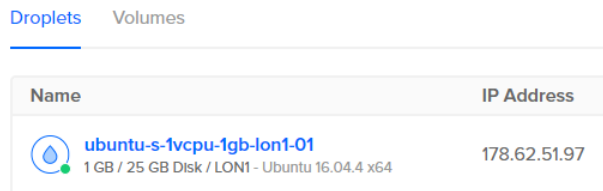
Adresa la care poate fi găsită aplicația în momentul de față este: <http://178.62.51.97/PLSQLKit/views/welcome-page.html#>, unde 178.62.51.97 reprezintă IP-ul mașinii virtuale.


Prin figura Fig. 15, de mai jos, evidențiez principalele informații ale mașinii virtuale ce găzduiește aplicația “*PL/SQLKit*”.

⁸ Cei de la DigitalOcean își numesc serverele de tip cloud droplet. Fiecare droplet creat reprezintă un server nou, pentru uz personal.

⁹ Windows Secure CoPy este un utilitar ce permite transferul de fișiere între calculatorul local și alte calculatoare.

Droplets



Droplets		Volumes
Name		IP Address
 ubuntu-s-1vcpu-1gb-lon1-01 1 GB / 25 GB Disk / LON1 - Ubuntu 16.04.4 x64		178.62.51.97

Prin Fig. 15 se poate observa mașina virtuală a droplet-ului creat pe platforma celor de la DigitalOcean. De asemenea, se poate observa și IP-ul corespunzător acestei mașini.

Fig. 15 Captură de ecran a mașinii virtuale pe care este găzduită aplicația

Acest proces a însemnat pentru mine o experiență cu totul nouă și necunoscută. Pot menționa ca și concluzii ale acestui subcapitol faptul că găzduirea aplicației a fost o provocare prin prisma faptului că toată configurarea proiectului a fost făcută doar din terminal, dar și prin faptul că versiunile diferite de PHP și Apache de pe server, față de versiunea locală, pot aduce un comportament diferit față de cel așteptat al aplicației.

2.7 Wireframe-urile inițiale

Încă dinainte de a mă apuca de implementarea interfeței aplicației, mi-am dorit să clarific cum aș vrea să arate paginile în aplicație. Astfel, am proiectat câteva schițe care conțin structura principală a unor pagini. Opțiunile inițiale de culoare sau font s-au schimbat ulterior, pentru a avea un aspect omogen în aplicație, dar structura furnizată prin schițele construite inițial s-au păstrat până la rezultatul final.

În mod normal astfel de schițe se numesc *wireframes*, ele fiind simple machete alb-negru, care conturează dimensiunea și plasarea unor elemente pe paginile din aplicație. Deși wireframe-urile inițial construite de către mine conțin alegeri cromatice și de font, acestea tot m-au ajutat în a rămâne concentrată pe structura aplicației.

Voi începe cu prima pagină din aplicație, și anume pagina de întâmpinare a utilizatorului. De la această pagina el poate fi redirecționat către pagina de creare de cont nou sau de logare. Mai multe despre aceste funcționalități sunt discutate în capitolul 2.2 *Capabilitățile aplicației*. Wireframe-ul pentru această pagină a fost inspirat de pe site-ul celor de la <https://codecademy.com>, deoarece, așa cum am mai menționat și în partea de *Introducere și motivație* de la începutul acestei lucrări, ei au fost sursa mea principală de inspirație încă de la

început. Aceștia nu dețin aceeași funcționalitate, însă au un efect de typewriting într-o pagină destinată unui chestionar.

Voi continua cu wireframe-ul paginii *Login*. Acesta a fost făcută cu ajutorul instrumentului de proiectare *Figma*. Experiența mea cu *Figma* nu este atât de avansată, dar este un instrument foarte bun, cu care se poate lucra pentru a adăuga cadre, forme, text și editarea lor.

Acest wireframe a folosit drept model și pentru pagina *Register*, pentru că datele cerute de la utilizator sunt asemănătoare, un email și o parolă.

Nu am proiectat wireframe-uri pentru mobil sau tabletă deoarece am pornit de la structura paginii pentru Desktop, pe care o aveam deja și am adaptat-o prin ascunderea unor elemente. De exemplu, pentru formularul de Login, așa cum se poate observa în Fig. 17, atunci când ecranul are o rezoluție mai mică, partea de noutăți (partea stângă a ecranului) este ascunsă, și doar formularul de înregistrare/logare mai rămâne vizibil.

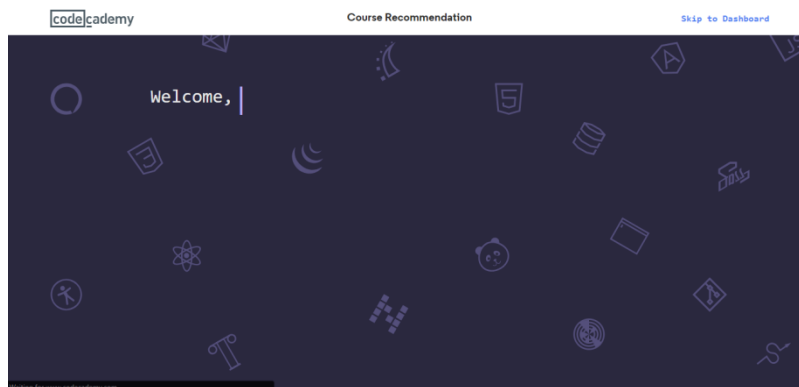


Fig. 16 Wireframe pentru pagina de Welcome

Prin Fig. 16 exemplific wireframe-ul folosit pentru prima pagină din aplicație. Acesta este de fapt o captură a unei pagini din aplicația <https://codecademy.com>.

Deoarece am vrut să aduc un efect de typewriting în aplicație, această captură de ecran a servit drept sursă de inspirație pentru pagina ulterior implementată.

Modul cum arată pagina de Welcome în aplicația mea, poate fi vizualizată în capitolul 2.2 *Capabilitățile aplicației*.

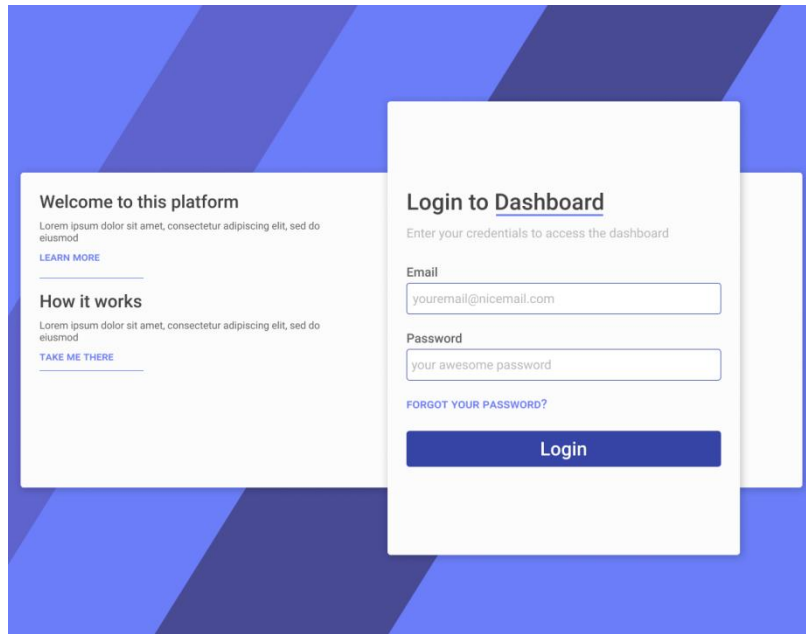


Fig. 17 Wireframe-ul pentru Login

Evidențiez prin Fig. 17 wireframe-ul inițial gândit pentru pagina Login. Acesta este asemănător cu cel al paginii Register, așa cum am menționat anterior.

Opțiunile cromatice și de font nu au rămas acestea din figură, însă structura a fost implementată.

Rezultatul obținut după implementare se poate vizualiza în capitolul 2.2 *Capabilitățile aplicației*, fiind introduse capturi de ecran din aplicația finală.

Voi continua cu pagina principală a aplicației. Structura acestei pagini a fost în totalitate inspirată, din nou, de pe site-ul celor de la <https://codecademy.com>. Structura pe care ei o oferă și pe care am implementat-o și eu este foarte sugestivă pentru utilizator și îl ajută să înțeleagă mai bine ce are de făcut și cum funcționează aplicația. Mai exact, pagina se împarte în trei coloane, primul conținând instrucțiuni, al doilea conținând partea de compilator, și ultimul conținând rezultatele obținute în urma unor rulări.

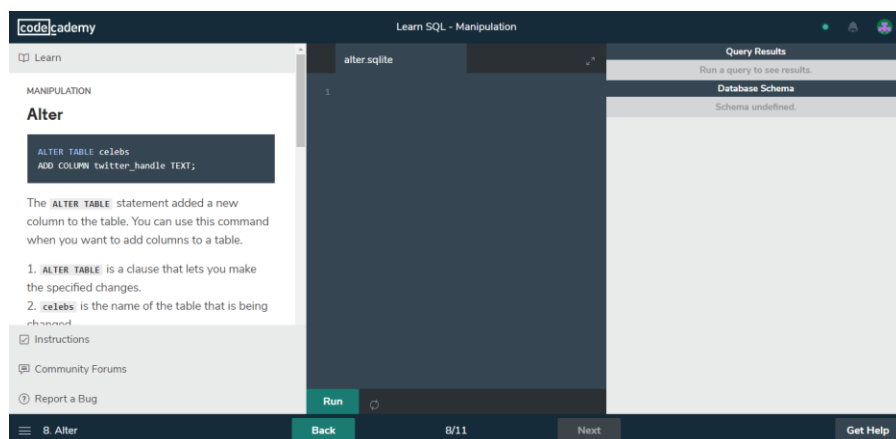


Fig. 18 Wireframe pentru pagina principală

Evidențiez prin Fig. 18 wireframe-ul folosit pentru pagina principală din aplicația mea. Acesta este de fapt o captură a unei pagini din aplicația <https://codecademy.com>. Deoarece aplicația mea se aseamănă în multe privințe cu aceasta, am dorit să folosesc aceeași structură.

Pentru pagina de profil în care utilizatorul își poate vizualiza progresul, am construit un wireframe separat în Figma, așa cum am făcut și cu pagina de Login.

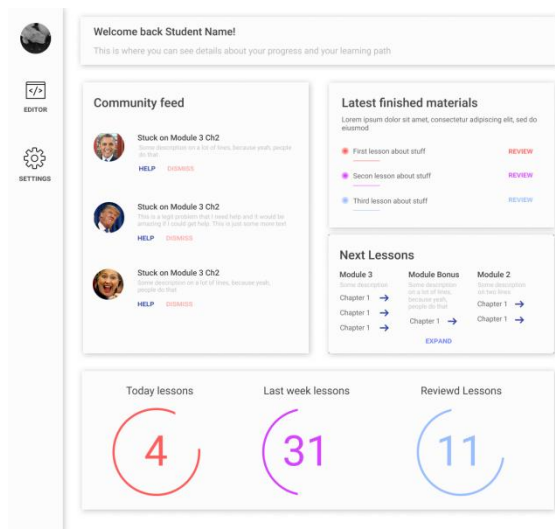


Fig. 19 Wireframe pentru pagina de profil

Prin Fig. 19 doresc să evidențiez wireframe-ul pe care l-am gândit inițial pentru pagina de profil a utilizatorului. Elementele gândite inițial nu au ajuns să fie toate implementate. Am considerat că vreau să ofer utilizatorului o experiență mai bună pe partea de exersare, având lecții cât mai bine formulate și de aceea, focusul meu a fost pe această direcție.

De asemenea, captura de ecran pentru pagina de profil actuală poate fi vizualizată în capitolul 2.2 *Capabilitățile aplicației*.

Astfel închei acest capitol prin a concluziona faptul că wireframe-urile proiectate înainte de a mă apuca de implementare m-au ajutat în a avea o imagine cât mai clară a rezultatului la care voiam să ajung. Pentru orice proiect, de altfel, wireframe-urile sunt foarte importante tocmai din acest aspect, dar și din faptul că din acest pas se poate contura o idee a tehnologiilor care o să fie ulterior folosite pentru implementarea funcționalităților.

2.8 Lecțiile PL/SQL

“*PL/SQLKit*” are un scop educativ și facilitează învățarea limbajului PL/SQL. Această aplicație se adresează în principal studenților din anul doi, care urmează disciplina PSGBD. Ea pune la dispoziție lecții ușor de urmărit, fiind formulate în instrucțiuni cât mai detaliate, conținând cuvinte cheie ce pot ajuta utilizatorul să rezolve cu ușurință lecțiile. De asemenea, proiectul oferă utilizatorilor săi motivația de a rezolva exercițiile disponibile și de a avea un progres al nivelului cunoștințelor cât mai ridicat.

Așa cum am menționat și în introducerea în unul dintre capitolele principale ale acestei lucrări, în capitolul 2 *Prezentarea aplicației*, utilizatorilor nu li se cer cunoștințe anterioare, ci doar voință de învățare și exersare.

Enunțurile lecțiilor au fost formulate în mare parte de către mine, deoarece ca și student care am urmat disciplina PSGBD, am observat care sunt pilonii care trebuie să stea la bază pentru a stăpâni limbajul PL/SQL. O sursă de inspirație care m-a ajutat în formularea lecțiilor a fost cartea “*PL/SQL. User’s Guide and Reference 10g Release*” de John Russell, care este menționată de altfel și în partea de *Bibliografie*. Această carte este foarte bine organizată și detaliată și cuprinde cunoștințe esențiale de PL/SQL. Eu am preluat din aceasta doar structuri de bază care vor fi folosite și se regăsesc în cerințele acestei discipline de la facultate.

Așa cum am menționat în capitolul 2.4.2 *Structura bazei de date în Firebase*, este necesar a trata în acest capitol modul în care este construit scriptul de populare al lecțiilor în baza de date din Firebase. După ce am stabilit care va fi topicul unei lecții, enunțul acesteia este scris în script. Răspunsul dat de utilizator trebuie să înglobeze anumite cuvinte cheie care se regăsesc ca și cerințe în lecția dată, iar aceste cuvinte cheie sunt salvate în baza de date de asemenea. Se pot observa trăsăturile menționate în Anexă.

2.9 Testarea aplicației

Testarea unei aplicații face parte dintre fundamentele dezvoltării unui proiect. O testare bună a aplicației mă poate asigura că aplicația funcționează conform așteptărilor, iar în caz de se găsesc anumite nereguli, denumite în mod obișnuit *bug-uri*, acestea să poată fi rezolvate din timpul dezvoltării componentei respective. Scrierea de test case-uri m-a ajutat la înțelegerea și la ținerea unui istoric legat de comportamentul așteptat al aplicației la inițierea diferitelor acțiuni care vin din partea unui utilizator.

TFS, platforma de management folosită pentru proiectul meu deține și această funcționalitate, și anume posibilitatea de a scrie și administra test case-uri, cât și de a le putea alocă iterațiilor. Test case-urile sunt structurate în mare după două elemente: descrierea unei acțiuni făcute de către un utilizator în cadrul aplicației (apăsarea unui buton, introducerea unor caractere) denumită în mod uzual *Action* și descrierea rezultatului așteptat după acțiunea inițiată, denumit *Expected result*. Această structură se poate observa în Fig. 20 de mai jos.

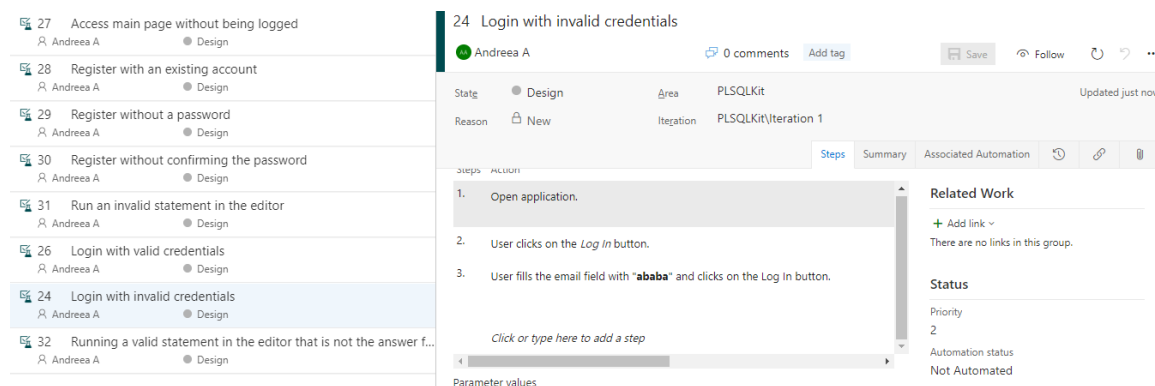


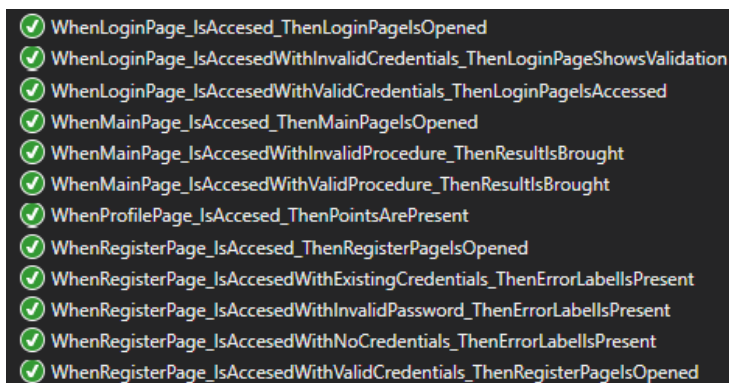
Fig. 20 Captură de ecran ce surprinde o parte din test case-urile destinate aplicației

În Fig. 20 se pot observa câteva test case-uri scrise de mine care acoperă mare parte dintre funcționalitățile aplicației. Așa cum am menționat mai devreme, acestea m-au ajutat în a gestiona mai ușor comportamentul așteptat pentru acțiunile executate în aplicație.

Pe lângă suita de test case-uri, ce m-a mai ajutat pe partea de testare a aplicației a fost o soluție de testare automată. Aceasta a fost creată în Visual Studio și este în sine un proiect Unit Test ce folosește framework-ul SpecDrill. Framework-ul SpecDrill este asemănător cu framework-ul Selenium, iar sursa se găsește pe GitHub.

Consider că existența unei astfel de soluții pentru testare automată este un element pentru care în mod obișnuit nu se acordă suficientă importanță, dar care aduce multe beneficii printre care ușurință în verificarea funcționalităților implementate, cât și impactul pe care îl au componentele nou adăugate. Menținerea unei astfel de soluții necesită la rândul său, scrierea de cod.

Voi atașa în cele ce urmează o captură de ecran ce surprinde suita de teste automate scrise pentru această aplicație.



În Fig. 21 se pot observa o serie de teste automate care sunt marcate cu indicatorul *Passed*.

Fig. 21 Captură de ecran ce surprinde o serie de teste automate

Ca și concluzie a acestui subcapitol menționez că partea de testare a aplicației m-a asigurat asupra faptului că funcționalitatea nu s-a schimbat pe parcursul dezvoltării, iar neregulile întâlnite au putut fi reparate la timp.

Concluzii și provocări

“*PL/SQLKit*” reprezintă lucrarea de licență ce marchează apogeul a trei ani de Informatică. Pot spune că ceea ce a început cu o admirație față de site-ul <https://www.codecademy.com>, cât și curiozitatea de a reuși în a construi o astfel de replică, dar și gândul de a investi și ajuta mai departe viitorii studenți, s-a finalizat într-un produs finit și capabil de a fi folosit.

Așa cum am detaliat în lucrare, proiectul creat înglobează o serie de tehnologii, unele noi, unele cu care am avut o mică experiență, dar care împreună constituie un kit de învățare, atât prin scopul aplicației, cât și prin înțelegerea ecosistemul construit. Aplicația dezvoltată îmbină mai multe tehnologii printre care se enumeră PHP 7.1, Apache Web Server 2.4, Oracle Database 11g Express Edition Release 11.2.0.2.0, Firebase, HTML5, CSS3 și JavaScript.

Pe lângă partea de cod a proiectului, menționez și platforma de management *Team Foundation Server* ce a ajutat la buna organizare a proiectului, *Figma* ce a ajutat în proiectarea wireframe-urilor și *ArgoUML* ce a ajutat în proiectarea diagramelor aplicației, fiind descrise mai în detaliu în lucrare.

Printre dificultățile întâlnite în dezvoltarea acestei lucrări, menționez câteva dintre ele:

- am întâmpinat probleme în gestionarea răspunsului dat de către utilizator (executarea diferitelor tipuri de intrări în serverul Oracle, neavând suport pentru funcția *dbms_output.put_line* din PL/SQL);
- am avut probleme pentru programe care nu au fost gratuite (folosirea instrumentelor PHP în Visual Studio);
- am întâmpinat probleme în configurarea proiectului pe mașina virtuală pe care este găzduită aplicația în momentul de față (variante diferite de PHP și Apache pentru Linux).

Există și dependențe în aplicația dată, prin faptul că lecțiile puse la dispoziție necesită a fi înnoite la perioade de timp și prin faptul că găzduirea aplicației nu este gratuită pe o perioadă mai mare de o lună.

Am fost de părere că nu este necesară introducerea unei pagini în proiect care să ofere un ghid de utilizare al aplicației deoarece am considerat că persoanele care o vor folosi sunt persoane tehnice, studenți, care se pot adapta rapid la cerințele acesteia.

Pot menționa faptul că există și loc de îmbunătățiri în cadrul aplicației. În primul rând, s-ar putea introduce o pagină care să poată fi accesată doar de către administratorii aplicației, de unde aceștia să poată scrie enunțurile lecțiilor mult mai ușor, fără a fi necesară introducerea acestora într-un script. O altă sugestie la care m-am gândit pentru partea funcțională ar fi introducerea unui forum disponibil pe partea de profil a fiecărui utilizator, care să permită comunicarea între studenți pe subiecte legate de dificultățile întâlnite în rezolvarea unor enunțuri. Lista de îmbunătățiri poate continua făcând comparații între alte aplicații ce oferă diverse experiențe utilizatorului.

În concluzie, consider că am reușit prin aplicația “*PL/SQLKit*” să înglobez un kit ușor de folosit de către studenți care poate fi extins și pentru alte materii. Modul de distribuire al informațiilor se face fără efort, iar progresul poate fi vizualizat chiar și de către profesori. Consider că atât lucrarea de față cât și aplicația finală și-au atins scopul, acela de a demonstra posibilitatea construirii unei aplicații pentru facultate, îmbinând multiple tehnologii. Provocările întâlnite m-au ajutat în a îmi dezvolta cunoștințele. Am avut ocazia să lucrez pentru prima oară cu DigitalOcean (o platformă de cloud computing ce oferă servere virtuale pentru găzduirea unei aplicații), iar pentru a doua oară cu Firebase (bază de date realtime) și TFS (platformă de management).

Bibliografie

- [1] “*HTML and CSS Web Standards Solutions*” de Christopher Murphy , Nicklas Persson
- [2] “*PL/SQL. User’s Guide and Reference 10g Release*” de John Rusell
- [3] “*Oracle PL/SQL Programming: Covers Versions Through Oracle Database 11g Release 2*” de Steven Feuerstein, Bill Pribyl
- [4] Documentația oficială CodeMirror: <https://codemirror.net/doc/manual.html>
- [5] Documentația oficială PHP: <http://php.net/manual/ro/index.php>
- [6] Documentația oficială Firebase: <https://firebase.google.com/docs/database/web/start>
- [7] Documentația oficială ProgressBar.js: <http://progressbarjs.readthedocs.io/en/latest/>
- [8] Documentația oficială DigitalOcean: <https://www.digitalocean.com/community/tutorials>
- [9] Sursa GitHub pentru framework-ul SpecDrill: <https://github.com/CosminSontu/SpecDrill>

Anexă

```
function initializeFirebase() {  
  // Initialize Firebase  
  var config = {  
    apiKey: "AIzaSyDr6xDl8Wg3EXtDSyWZLyITWnFvUSJIjqk",  
    authDomain: "plsqlkit.firebaseio.com",  
    databaseURL: "https://plsqlkit.firebaseio.com",  
    projectId: "plsqlkit",  
    storageBucket: "plsqlkit.appspot.com",  
    messagingSenderId: "186503669104"  
  };  
  firebase.initializeApp(config);  
}
```

Fig. 22 Cod necesar accesului la Firebase în aplicație

```
function createUserScore(userUID, userEmail) {  
  var user = {  
    userUID: userUID,  
    score: 0,  
    email: userEmail  
  };  
  var updates = {};  
  updates['/userScores/userScore-' + userUID] = user;  
  firebase.database().ref().update(updates);  
}
```

Fig. 23 Exemplu de folosire a tabelii userScores

```
if (response.includes(snapshot.val()[key].description)  
{  
  updateUserQuestion(questionID);  
  document.getElementById("next-btn").disabled =  
    false;  
  document.getElementById("next-btn").style.color =  
    "White";  
  
  var userScore =  
    getQuestionDifficultyPoints(questionDifficulty);  
  setUserScore(userScore);  
}
```

Fig. 24 Porțiune de cod ce tratează cazul unui răspuns corect

```
function getQuestionDifficultyPoints(difficulty) {  
  if (difficulty.toLowerCase() === 'hard') {  
    return 20;  
  } else if (difficulty.toLowerCase() === 'medium') {  
    return 10;  
  } else if (difficulty.toLowerCase() === 'easy') {  
    return 5;  
  }  
  return 10;  
}
```

Fig. 25 Distribuirea punctajelor în funcție de dificultate

Prin Fig. 22 se pot observa configurările folosite pentru a avea acces la Firebase în cadrul aplicației. Aceste configurări sunt unice și sunt puse la dispoziție după crearea unui nou.

Am ales să evidențiez prin Fig. 23 codul în care tratez cazul menționat mai sus, și anume, pasul de inițializare cu punctajul zero a unui cont nou creat.

Am ales să evidențiez prin Fig. 24 porțiunea de cod care se ocupă cu activarea butonului ce îi permite utilizatorului să meargă la următorul enunț, care conține și actualizarea punctajului acestuia în tabela aferentă.

Fig. 25 conține porțiunea de cod ce se ocupă cu distribuirea punctajelor în funcție de dificultatea setată acestora.

```

for (var key in usersQuestions) {
    if (max < usersQuestions[key].answers
        &&
        chapterIds.indexOf(usersQuestions[key].questionID
        ) > -1) {
        max = parseInt(usersQuestions[key].answers);
    }
}

```

Fig. 26 Căutarea numărului de răspunsuri date întrebărilor disponibile

Fig. 26 reprezintă un exemplu vizual de cod ce execută cele specificate mai sus, și anume, întrebările care nu au primit încă un răspuns vor fi afișate utilizatorului cu prioritate mai mare față de cele la care s-a dat deja un răspuns.

```

while ($row = oci_fetch_array($s, OCI_ASSOC+OCI_RETURN_LOBS))
{
    echo "<tr>\n";
    foreach ($row as $item) {
        echo "    <td>" . ($item !== null ?
            htmlentities($item, ENT_QUOTES) : "") . "</td>\n";
    }
    echo "</tr>\n";
}

```

Fig. 27 Exemplu de folosire a funcției *oci_fetch_array*

Evidențiez prin Fig. 27, un exemplu de folosire pentru returnarea rezultatelor în cadrul primului caz prezentat mai sus.

```

while ($stmt_c = oci_get_implicit_resultset($s)) {
    while ($row = oci_fetch_array($stmt_c,
        OCI_ASSOC+OCI_RETURN_NULLS)) {
        foreach ($row as $item) {
            echo $item . "    ";
        }
        echo "<br/>";
    }
}

```

Fig. 28 Exemplu de folosire al funcției *oci_get_implicit_resultset*

Prin Fig. 28, evidențiez folosirea funcției descrise în cadrul celui de-al doilea caz întâmpinat.

```

function createoutput($c,$statement)
{
    $s = oci_parse($c,
        "call dbms_output.put_line($statement)");
    for ($i = 0; $i < 1; ++$i) {
        oci_execute($s);
    }
}

```

Fig. 29 Exemplu de utilizare a declarației *CALL* din Oracle

Am ales să exemplific prin Fig. 29 cele descrise mai sus, și anume abordarea găsită pentru tratarea cazurilor de execuție a instrucțiunilor de tip *dbms_output.put_line*.

```
function getdbmsoutput_pl($c)
{
    $s = oci_parse($c, "select * from table(mydofetch())");
    oci_execute($s);
    $res = false;
    oci_fetch_all($s, $res);
    foreach ($res['COLUMN_VALUE'] as $key => $value){
        echo nl2br("$value \n");
    }
}
```

Fig. 30 Exemplificarea construirii unui vector de *dbms_output.put_line-uri*

```
<div class="widget-compiler-body" id="codeeditor">
  <textarea id="codemirror-textarea"
    placeholder="Code goes here..." name="name">
  </textarea>
  <script>
    var editor =
      CodeMirror.fromTextArea(document.getElementById(
        Id("codemirror-textarea"), {
          lineNumbers: true,
          extraKeys: { "Ctrl-Space": "autocomplete"
        }
      });
  </script>
</div>
```

Fig. 31 Crearea unei instanțe de CodeMirror editor

```
function writeQuestions() {
    var id1 = writeQuestion(
        1,
        "Welcome! First of all, an anonymus PL/SQL is an
        executable statement that can be compiled and executed
        by the data server. It consist of up two main section:
        an optional declaration section and a mandatory
        executable section.The optional declaration section
        starts with DECLARE keyword where you usually declare
        the variables,cursors you are going to use by
        statements.The variables are prefixed by the <b>v</b>
        followed by the '_' character and the name of the
        variable. After the name of the variable, you have to
        detail the type of the variable. <br/>The mandatory
        keyword that introduces the executable section is
        BEGIN. The END keyword ends the mandatory block, of
        course.Start now with your first example! Declare a
        NUMBER variable x, and in the BEGIN END block, assign
        the value 100. That's all. Give it a try!",
        "Easy"
    );
    writeAnswer(id1, "PL/SQL procedure successfully
    completed.", "DECLARE", "BEGIN", "END", "100");
}
```

Fig. 32 Captură de ecran ce surprinde scriptul de populare al unei lecții

Prin figura Fig. 30 exemplific modul în care am folosit funcția descrisă anterior, și anume construirea unui vector de *dbms_output.put_line-uri* fiind cu ușurință, ulterior, returnat utilizatorului.

Am ales să vă exemplific prin Fig. 31 , ceea ce am menționat anterior, și anume declararea unui CodeMirror editor într-un textarea din fișierul html. De asemenea se pot observa și proprietățile de numerotare a liniilor și deschiderea listei de autocompletare la combinația tastelor Ctrl-Space.

În Fig. 32 se poate observa scriptul de populare care conține o lecție. După ce enunțul acesteia este formulat cât mai detaliat, țin cont și de cuvintele cheie care trebuie să se regăsească în răspunsul dat de către utilizator. Aceste cuvinte cheie se pot observa în funcția *writeAnswer*.