UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

**Facultatea de Informatică**

****

Lucrare de licenţă

**Quizdom**

Propusă de

***Robert –Ilie Vicol***

**Sesiunea:** Iulie, 2018

Coordonator Ştiinţific:

***Lector, dr. Cosmin Vârlan***

UNIVERSITATEA „ALEXANDRU IOAN CUZA” IAŞI

**Facultatea de Informatică**

**Quizdom**

***Vicol Robert -Ilie***

**Sesiunea:** Iulie, 2018

Coordonator Ştiinţific:

***Lector, dr. Cosmin Vârlan***

**DECLARAŢIE PRIVIND ORIGINALITATE ŞI RESPECTAREA**

**DREPTURILOR DE AUTOR**

Prin prezenta declar că Lucrarea de licență cu titlul „Quizdom” este scrisă de mine și nu a mai fost prezentată niciodată la o altă facultate sau instituţie de învățământ superior din ţară sau străinătate. De asemenea, declar că toate sursele utilizate, inclusiv cele preluate de pe Internet, sunt indicate în lucrare, cu respectarea regulilor de evitare a plagiatului:

− toate fragmentele de text reproduse exact, chiar și în traducere proprie din altă limbă,

sunt scrise între ghilimele și deţin referinţa precisă a sursei;

− reformularea în cuvinte proprii a textelor scrise de către alţi autori deţine referinţa

precisă;

− codul sursă, imagini etc. preluate din proiecte open source sau alte surse sunt utilizate

cu respectarea drepturilor de autor și deţin referinţe precise;

− rezumarea ideilor altor autori precizează referinţa precisă la textul original.

Iaşi,

Absolvent Vicol Robert -Ilie

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(semnătura în original)

**DECLARAŢIE DE CONSIMŢĂMÂNT**

Prin prezenta declar că sunt de acord ca Lucrarea de licență cu titlul „Quizdom”, codul sursă al programelor și celelalte conţinuturi (grafice, multimedia, date de test etc.) care însoţesc această lucrare să fie utilizate în cadrul Facultăţii de Informatică. De asemenea, sunt de acord ca Facultatea de Informatică de la Universitatea „Alexandru Ioan Cuza” Iași să utilizeze, modifice, reproducă şi să distribuie în scopuri necomerciale programele-calculator, format executabil şi sursă, realizate de mine în cadrul prezentei lucrări de licenţă.

Iaşi,

Absolvent Vicol Robert -Ilie

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (semnătura în original)

# Introducere

Această lucrare are scopul de a prezenta etapele dezvoltării aplicației web „Quizdom”. Rolul principal al acesteia este de a facilita acumularea de cunoștințe tehnice de către studenți, în vederea obținerii unor rezultate mai bune la materiile studiate la Facultatea de Informatică Iași.

Am ales să realizez această aplicație întrucât consider că pot oferi o modalitate ușoară de a ințelege noțiuni abstracte și de a verifica nivelul de cunoștințe acumulat de utilizator, fiind cunoscut faptul că învățarea prin joc este foarte eficientă.

## Context

Pe parcursul celor trei ani de studiu, am remarcat faptul că multe dintre materiile predate conțin metode asemănătoare de testare a cunoștințelor acumulate: examene cu cerințe tip grilă, în care sunt solicitate, după caz, explicații adiționale ce demonstrează înțelegerea conceptelor verificate. Drept exemple, putem aminti discipline precum Structuri de Date, Programare Orientată Obiect, Baze de date, Ingineria Programării și Programare Avansată. Pentru a se pregăti mai bine, în afara parcurgerii materiei și a lecțiilor, studenții căutau materiale adiționale, precum subiecte sau cerințe formulate în anii trecuți la aceleași materii. Această nevoie, de a avea un model, un șablon, constituie una din situațiile care m-au îndemnat să caut o soluție.

Există, în prezent, aplicații asemănătoare, care ajută utilizatorii să capete fie cunoștințe generale, fie cunoștințe din domenii tehnice. Dintre acestea amintim:

**1) Aplicația din cadrul disciplinei Baze de Date** – folosită de studenți. Utilizatorii pot învăța sintaxa operațiilor SQL pe tabele (*select, insert, update, delete*). Aceasta este exclusivă disciplinei Baze de Date și nu are rolul și funcționalitățile unui joc.

**2) QuizUp**[[1]](#footnote-1)– joc trivia pentru Android / iOS. Este cea mai populară aplicație de acest tip, având zeci de milioane de utilizatori, însă categoriile din care sunt întrebările nu vizează domenii tehnice. Mai mult decât atât, aplicația este doar pentru mobil, fiind necesară așadar instalarea acesteia.

Spre deosebire de aceste aplicații, soluția propusă suportă orice platformă, poate fi extinsă ușor, pentru orice materie / domeniu tehnic și adaugă funcționalități noi care îmbunătățesc experiența utilizatorului.

## Funcționalități

*Crearea și administrarea unui cont* – necesar pentru a accesa lista întrebărilor „cumpărate”, precum și pentru salvarea unor statistici.

*Accesul la informații despre alți utilizatori* – aplicația salvează date publice despre fiecare utilizator (poziția în clasament, numărul de jocuri câștigate / pierdute, domeniile de interes).

*Accesul la setul de întrebări existent* – totodată cu revederea întrebărilor „cumpărate”, există posibilitatea de a contribui la lista de probleme, orice utilizator având șansa de a genera conținut nou, în cadrul aplicației.

*Raportarea unei probleme* – pentru cazul în care domeniul abordat de grila respectivă nu este ales corect, sau pentru formulări, răspunsuri greșite

*Accesul la lista de utilizatori activi* – cei înregistrați pot vizualiza lista utilizatorilor care sunt activi[[2]](#footnote-2), pentru a-i provoca la joc.

*Invitația la joc* – aplicația permite utilizatorilor să își trimită unii altora provocări la joc în timp real. Dacă oponentul acceptă invitația, jocul între doi utilizatori va începe.

*Jocul propriu-zis* – pentru un singur utilizator sau între 2 utilizatori. Reprezintă partea de bază a aplicației. Deciziile luate (în cazul jocului intre 2 utilizatori) sunt văzute în timp real de către oponent.

*Premierea câștigătorilor* – este realizată prin atribuirea de token-uri, care pot fi folosiți în joc pentru acumularea de probleme sau propunerea acestora. De asemenea utilizatorii cu numărul cel mai mare de victorii se află în fruntea unui clasament public.

*Sistem de notificări* – utilizatoriilogați beneficiază de un sistem de notificări, prezent atât în interiorul, cât și în exteriorul aplicației. Notificările sunt afișate în timp real și sunt declanșate de anumite acțiuni[[3]](#footnote-3) ale utilizatorului.

*Păstrarea stării aplicației* – reprezintă una dintre cele mai importante funcționalități ale aplicației. Permite reluarea activității[[4]](#footnote-4) în orice moment, chiar și în timpul jocului.

## Descrierea soluției

Aplicația se constituie din două părți:

1. REST API la nivel de server, cu o interfață minimalistă ce permite unui administrator să modifice resurse[[5]](#footnote-5).
2. Aplicație Vue la nivel de client, unde sunt disponibile funcționalitățile principale.

Implementarea pornește de la un set de întrebări de tip grilă, stocate într-o bază de date de tip **MySQL**. Utilizatorii logați pot folosi token-uri pentru a “cumpăra” întrebări; în urma acestui proces, întrebările respective vor deveni disponibile pentru răspuns. În urma acumulării unui număr de răspunsuri corecte, utilizatorii pot să contribuie la setul de întrebări existent. Există mai multe moduri prin care utilizatorii pot obține token-uri, dintre care amintim: răspunsuri corecte la intrebări,”contribuția” la setul de întrebări existent, “cumpărarea” întrebărilor proprii de către alți utilizatori.

Aplicația oferă, de asemenea, funcționalitatea unui joc multiplayer de tip quiz, în care câștigătorul este cel care acumulează un număr mai mare de puncte, răspunzând corect la un număr de întrebări în timp cât mai scurt. Miza jocului este stabilită anterior de comun acord (un număr fix de token-uri).

Printre tehnologiile folosite amintim framework-ul  **Laravel** pe partea de server, respectiv **VueJS** pe partea de client. Comunicarea este realizată prin protocolul HTTP, aderând la modelul arhitectural REST.

În cadrul componentei de joc a aplicației, informațiile se transmit în timp real prin intermediul modelului Publish/Subscribe. Este folosită, în acest sens, librăria **Pusher Channels**, care folosește Websocketi.

Partea internă a sistemului de notificări e construită pe același model Publish / Subscribe amintit, în timp ce partea externă are la bază serviciul Push Notifications, apelat prin API-ul de la OneSignal de pe partea de server.

## Structura Lucrării

În continuare voi menționa contribuțiile mele în realizarea proiectului, urmând ca restul paginilor să detalieze pașii dezvoltării aplicației web (cap 1), să prezinte un studiu mai detaliat asupra funcționalităților (cap 2), precum și unele optimizări posibile și dificultăți întâmpinate (cap 3)

# Contribuții

Pe parcursul dezvoltării acestei aplicații, contribuțiile mele au fost, pe de o parte teoretice și pe de altă parte, practice:

* Alegerea și configurarea serverului Apache, precum și a framework-ului Laravel
* Studiu pentru găsirea unei soluții optime pentru transmiterea de date și comunicarea în timp real (necesare jocului între doi utilizatori)
* Dezvoltarea unei modalități de a asigura persistența datelor în cazul pierderii conexiunii.
* Dezvoltarea unui sistem de notificări care să asigure primirea informațiilor chiar și în cazul în care utilizatorul are aplicația închisă.
* Implementarea aplicației pe parte de client, unde s-a încercat construirea unei interfețe intuitive, compatibile cu majoritatea ecranelor din punct de vedere al dimensiunii, care să permită accesarea tuturor funcționalităților printr-un număr cât mai mic de click-uri / atingeri.
* Implementarea aplicației pe parte de server, unde a fost necesară aprofundarea modelului MVC și găsirea unei metode prin care inițierea comunicării dintre client si server să fie făcută de cel din urmă.
* Testarea aplicației și a funcționalităților.
* Optimizarea aplicației, din punct de vedere al timpului de răspuns al serverului, precum și al tipului de stocare folosit.

Totodată, poate fi considerată contribuție publicarea unor componente de front-end în managerul de pachete **npm**. Vue fiind un framework pentru front-end care încurajează în mod special organizarea codului astfel încât bucăți din acesta să fie reutilizabile (componente), oricine are sistemul de dependințe cerut în proiect poate importa și folosi componentele publicate de mine. Printre acestea, se numără următoarele:

* vueAnimatedBar : bară de lungime / lățime variabilă, ce simbolizează modificarea dimensiunii, scurgerea timpului.
* vueTimer : componentă cu logica necesară unui cronometru sau a unei număratoare inverse.
* vueCube : componentă de meniu cu aspectul unui cub.

Acestea dovedesc aportul meu pentru comunitatea de front-end din jurul frameworkului **Vue,js**.

# Dezvoltarea aplicației web

În acest capitol voi detalia tehnologiile folosite în realizarea proiectului, precizând în același timp motivele pentru care am optat pentru aceste tehnologii.

## Laravel 5

Laravel este un framework open-source construit în jurul modelului arhitectural **MVC** (Model-View-Controller) dezvoltat în php, care a căpătat o popularitate imensă pe parcursul a câtorva ani, având atât un manager de dependințe dedicat, cât și o documentație bine structurată ce permite implementarea cu ușurință a unor funcționalități generale, care servesc oricărui proiect.

Odată instalat framework-ul, putem crea un nou proiect ce va avea următoarea structură:

* Dosarul *app* – aici se regăsește logica aplicației, dosarul fiind încărcat automat sub spațiul de nume *App*. În interiorul acestuia se află:
* Dosarul *Console* conține toate comenzile de tip **Artisan**[[6]](#footnote-6).
* Dosarul *Http* cuprinde entități legate de serverul Http. Printre acestea numărăm: **Controllers** și **Middlewares**.
* Dosarul *Exceptions* conține un fișier denumit *Handler*, în acesta regăsindu-se o clasă care descrie comportamentul aplicației în cazul în care sunt aruncate excepții.
* Dosarul *Providers* cuprinde clase care furnizează diferite servicii utile precum autentificare - *AuthServiceProvider*, rutare – *RouteServiceProvider* sau transmitere de evenimente – *BroadcastServiceProvider*.

În dosarul *app* se regăsesc, de asemenea, modelele din structura MVC, care moștenesc clasa *Model* din **Eloquent**[[7]](#footnote-7)**.**

* Dosarul *bootstrap* – conține fișierul *app.php*, folosit pentru a încărca framework-ul. Regăsim totodată dosarul Cache, ce cuprinde fișiere de tip cache generate de framework pentru creșterea performanței, cum ar fi fișiere cache pentru rutare și pentru servicii.
* Dosarul *config* – conține fișiere utile dezvoltatorului atât pentru configurarea serverului, cât și a framework-ului. Reglarea acestora determină comportamentul unor secțiuni diferite din aplicație, cum ar fi baza de date, serviciul de transmitere a fișierelor, serviciul de transmitere a evenimentelor (broadcasting) sau implementarea unei cozi pentru sarcinile consumatoare de timp.
* Dosarul *database* – conține fișierele de tip migrare și alte fișiere necesare pentru popularea bazei de date cu înregistrări pentru testare. Migrările le folosim pentru a crea tabelele, pentru a efectua schimbări în structura bazei de date și a tabelelor, cât și pentru a reveni la o stare anterioară a bazei de date (dacă este nevoie).
* Dosarul *public* – aici regăsim fișierul *index.php*, acesta fiind punctul de intrare pentru toate cererile din aplicație. Este necesar de asemenea, pentru încărcarea automată a fișierelor (autoloading). Aici se află, totodată, fișierele servite către client (html, css, imagini, etc).
* Dosarul *resources* – cuprinde dosarul cu view-uri, precum și fișierele necompilate javascript (**EcmaScript 5**) sau css (**sass**, **scss**).
* Dosarul *routes* – aici găsim fișierele necesare rutării. Putem defini rute pentru aplicația noastră, sau putem activa metode numite **middlewares** care modifică structura răspunsului sau filtrează cererile către server. Funcțiile middleware pot fi efectuate înainte sau după generarea răspunsului.
* Dosarul *storage* – acesta include:
* Dosarul *app*, unde se află fișiere generate de aplicația nostră. Spre acest dosar putem crea o legătură simbolică in dosarul *public*, astfel încât fișierele aflate aici să accesibile doar grupurilor de utilizatori alese de noi.
* Dosarul *framework* conține fișiere generate de framework și fișiere de tip cache.
* Dosarul *logs* cuprinde fișiere unde sunt înregistrate răspunsuri date de server sau posibile erori întâmpinate. Pentru ca aceste fișiere să poată fi scrise, dosarul trebuie să aibă permisiuni de scriere.
* Dosarul *tests* – cuprinde teste automate. Orice clasă trebuie să fie denumită cu sufixul ”Test”. Pentru a rula testele este folosit framework-ul **PHPunit**.
* Dosarul vendor – aici regăsim conținutul dependințelor **Composer**.

(Directory Structure, 2018)

În rădăcina proiectului există, de asemenea, fișiere importante:

* *.env*, unde putem configura constante precum numele aplicației, tipul de bază de date, numele de utilizator și parola pentru conectare la baza de date sau adresa aplicației.
* *composer.lock*, unde sunt precizate dependințele aplicației.

O interacțiune cu serverul, inițiată de client, trece prin următorii pași:

* Utilizatorul face o cerere spre server de tip Get, Post, Put, Delete la o adresă corespunzătoare.
* Punctul de intrare al oricărei cereri către o aplicație Laravel este fișierul *public/index.php*. Ulterior este executat scriptul din *bootstrap/app.php*, care creează o instanță a furnizorului de servicii Laravel.
* În continuare clasa din fișierul *app/Http/Kernel.php* încarcă componentele severului (baza de date, validări, rutare). Dacă cererea trece de filtrarea făcută de middleware-uri și dacă aceasta accesează un link valid (aflat în fișierele de rutare), procesul continuă.
* În cele din urmă, aceeași clasă conține o metodă care este apelată, primind cererea (**Request**) și returnând un răspuns (**Response**). Comportamentul acesteia este determinat de metoda corespunzătoare adresei accesate (legătura fiind definită în fișierul de rutare). Metoda respectivă poate fi scrisă direct în fișierul de rutare, însă este recomandat ca aceasta să se regăsească într-un **controller**.
* Metoda apelată din **controller** poate accesa, dacă este nevoie, baza de date, prin intermediul modelului: fiecare tabel din baza de date are câte un model care îi corespunde, **Eloquent** având metode pentru selectarea, inserarea, editarea și ștergerea înregistrărilor din tabele.

Următoarea figură ilustrează, într-un mod simplificat, un exemplu de cerere http către server. Cererea este una de tip GET, resursa cerută fiind ”cats”. Modelul primește comanda *all()*, care este tradusă în baza de date prin ”select \* from cats”.

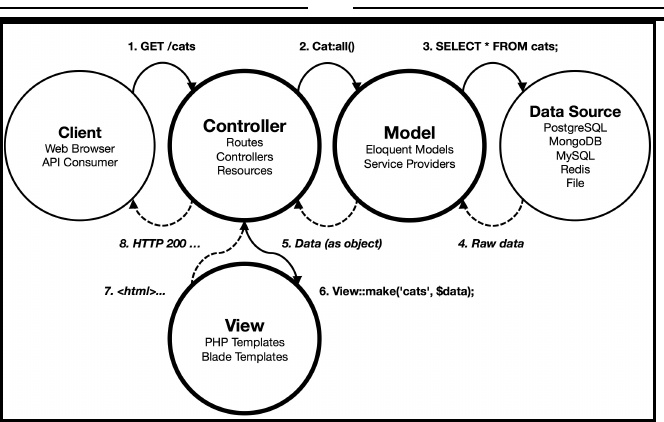


Figura 1 – Cerere HTTP către server (Bean, 2015)

Metodele unui **controller** returnează în general, un **view**, care este completat cu informațiile procesate. Fișierele de tip **view** au terminația *blade.php*, sugerând existența șabloanelor de tip **Blade** în componența paginilor web. Aceste șabloane permit utilizarea datelor returnate odată cu **view-ul** respectiv. Șablonul **Blade** permite, de asemenea, dezvoltatorului, să scrie cod php în interiorul paginilor web într-un mod organizat și propune o soluție pentru extinderea paginilor web. Spre exemplu, toate paginile pot moșteni un șablon de tip **Blade** ce reprezintă un meniu de navigare. Astfel, prin cuvinte cheie precum *@section* sau *@yield* , este încurajată refolosirea codului într-un mod sugestiv.

Cu toate acestea, întrucât aplicația dezvoltată pe parte de server constituie doar un API, metodele din controllere nu vor returna view-uri, ci răspunsuri de tip JSON (Javascript Object Notation), care sunt interpretate în aplicația client de limbajul javascript. Paginile propriu-zise sunt realizate cu ajutorul framework-ului **Vue.JS**, care va fi descris în unul din capitolele următoare.

## MySQL

Stocarea datelor reprezintă un aspect important al oricărei aplicații web. Fără aceasta nu aș fi reușit să rețin informații despre utilizatori și statistici despre numărul de meciuri jucate sau numărul de întrebări contribuite la setul existent. Pentru acest tip de informație este utilă o bază de date relațională. După cum am văzut în Figura 1, un avantaj al folosirii framework-ului Laravel este acela că suportă mai multe metode de a stoca date: **PostgresSQL**, **MongoDB**, **MySQL**, **Redis**, **File**. Dintre acestea, am ales să folosesc **MySQL**, fiind un proiect dezvoltat în regim open-source, popular și folosit de multe companii (Facebook, Twitter, Youtube, Spotify, Netflix) tocmai datorită faptului că oferă performanță și stabilitate.

De asemenea, **MySQL** face parte deja din structura **LAMP**, alături de componentele **Linux**, **PHP** și **Apache**, ceea ce inlătură dificultățile procesului de integrare.

Avem opțiunea de a realiza configurarea bazei de date din fișierul *.env*. Printre setări amintim:

* DB\_CONNECTION (tipul bazei de date, în acest caz, așa cum am menționat, MySQL)
* DB\_HOST (adresa la care este disponibilă baza de date)
* DB\_PORT (portul deschis conectării)
* DB\_DATABASE (numele dat bazei de date)
* DB\_USERNAME
* DB\_PASSWORD

Manipularea datelor este realizată prin intermediul ORM-ului specific Laravel, denumit **Eloquent**, acesta fiind asemănător cu alte ORM-uri populare, precum **Entity Framework** (asp.net). Ambele elimină necesitatea de a scrie interogările direct către baza de date și la fel de important este faptul că ambele aderă la principiul ”code-first”: tabelele sunt create prin migrări, clasele respective având două metode (up, down), iar legăturile dintre tabele sunt definite pe baza modelelor. În continuare precizăm tipurile de legături dintre tabele și echivalentul lor în Eloquent:

* unu-la-unu: metoda *hasOne*, cu inversa *belongsTo*.
* unu-la-mai-mulți: metoda *hasMany*, cu inversa *belongsTo*.
* mai-mulți-la-mai-mulți: metoda *belongsToMany*.

Un dezavantaj al ORM-ului **Eloquent** față de **Entity Framework** ar fi acela că nu poate genera fișierele de migrare automat. Neavând opțiunea de a declara tipul de dată în PHP, migrările nu au de unde să știe tipul unui atribut doar din structura modelului.

În cazul acestei aplicații au fost necesare următoarele tabele, respectiv câmpuri:

* ***Users***: conține informații legate de utilizatorii aplicației.
  + *id*: constituie cheie primară.
  + *name*: numele utilizatorului.
  + *tokens*: monedă ce poate fi folosită în aplicație.
* ***Answers***: cuprinde date legate strict de răspunsurile grilă.
  + *id*: constituie cheie primară, deci acest câmp este unic și nu poate fi nul. În plus, se va auto-incrementa odată cu adăugarea unui nou răspuns.
  + *text*: reprezintă enunțul răspunsului, nu poate fi nul.
* ***Questions***: cuprinde date legate strict de întrebări.
  + *id*: constituie cheie primară, deci acest câmp este unic și nu poate fi nul. În plus, se va auto-incrementa odată cu adăugarea unei noi întrebări.
  + *text*: reprezintă textul întrebării, nu poate fi nul.
* ***Categories***: conține date legate despre domeniile din care pot face parte problemele.
  + *id*: constituie cheie primară, deci acest câmp este unic și nu poate fi nul. În plus, se va auto-incrementa odată cu adăugarea unei noi categorii.
  + *name*: numele categoriei.
* ***Problems***: reprezintă entitatea principală din aplicație.
  + *id:* constituie cheie primară.
  + *user\_id*: cheie străină ce referențiază tabela ***Users***. Constituie id-ul utilizatorului care a creat problema.
  + *category\_id*: cheie străină ce referențiază tabela ***Categories***. Constituie id-ul categoriei din care face parte problema.
  + *question\_id*: cheie străină ce referențiază tabela ***Questions***. Constituie id-ul întrebării corespunzătoare.
  + *answer\_id*: cheie străină ce referențiază tabela ***Answers***. Constituie id-ul răspunsului corect. Nu poate fi nul (o grilă trebuie sa aibă un răspuns corect).
  + *bad1\_id*: cheie străină ce referențiază tabela ***Answers***. Constituie id-ul primului răspuns greșit. Nu poate fi nul (o grilă trebuie să aibă cel puțin un răspuns greșit).
  + *bad2\_id*: cheie străină ce referențiază tabela ***Answers***. Constituie id-ul primului răspuns greșit. Poate fi nul.
  + *bad3\_id*: cheie străină ce referențiază tabela ***Answers***. Constituie id-ul primului răspuns greșit. Poate fi nul.
* ***Matches***: tabelă ce conține date despre jocurile între doi utilizatori. Poate fi considerat un istoric al meciurilor terminate.
  + *id*: constituie cheie primară, deci acest câmp este unic și nu poate fi nul. În plus, se va auto-incrementa odată cu adăugarea unei noi categorii.
  + *user1\_id*: cheie străină ce referențiază tabela ***Users***. Constituie id-ul utilizatorului care participă la joc.
  + *user2\_id*: cheie străină ce referențiază tabela ***Users***. Constituie id-ul celui de-al doilea utilizator care participă la joc.
* ***User-Problems***: tabelă pivot necesar legăturii mai-mulți-la-mai-mulți dintre utilizatori și probleme (în sensul în care o problemă poate fi cumpărată de mai mulți utilizatori, iar un utilizator poate să aibă mai multe probleme cumpărate).
* ***Match-Problems***: tabelă pivot necesar legăturii mai-mulți-la-mai-mulți dintre jocuri și probleme (în sensul în care o problemă să se regăsească în mai multe jocuri, iar un joc poate să conțină mai multe probleme).

În plus față de informațiile menționate, fiecare tabelă conține două câmpuri adăugate și editate de **Eloquent**: *created\_at* și *updated\_at*. Acestea se actualizează automat atunci când vom crea o nouă instanță a modelului corespunzător tabelului, sau atunci când vom modifica proprietățile unui obiect deja existent, apelând metoda *save().*

## Cache

În cazul proiectului descris, există situații în care salvarea informațiilor în baza de date relațională nu este o idee bună. Un exemplu concret este reprezentat de perioada în care doi utilizatori se află în joc. Fiecare răspuns dat de oricare dintre aceștia trebuie să ajungă la server, pentru a fi primit unul de la altul. Jocul arată în timp real deciziile adversarului, fie că acestea au fost corecte sau greșite. Cu alte cuvinte, informațiile suferă foarte multe actualizări într-un timp scurt, ceea ce ar însemna multe accesări spre baza de date relațională, pentru editări și inserări. Mai mult decât atât, în istoricul jocurilor păstrăm doar informații relevante precum ce probleme au picat, cine a câștigat jocul sau dacă acesta a fost încheiat cu remiză. Nu este necesară stocarea permanentă a datelor vehiculate în timpul jocului (cât timp mai are un utilizator să răspundă, câte puncte are un utilizator în prezent sau la ce întrebare se află cei doi jucători)

Laravel propune o soluție pentru această problemă, având un serviciu de cache, util pentru stocarea temporară a informațiilor ce suferă multe actualizări într-un timp scurt. Acesta salvează datele sub forma unor perechi cheie-valoare, pentru un timp specificat. Nu avem nevoie de stocarea permanentă a acestor date, întrucât ele sunt relevante doar pe parcursul meciului dintre doi utilizatori. Datele respective pot fi șterse fie în cazul în care jocul s-a terminat, fie în cazul în care a trecut prea mult timp pentru ca jocul să poată fi reluat. (spre exemplu, dacă unul dintre jucători pierde conexiunea și nu revine în joc mai mult de 5 minute).

Serviciul cache este, de asemenea, folosit pentru stocarea (permanentă, de această dată) informațiilor primite de la un middleware customizat. Am făcut acest middleware astfel încât să preceadă toate cererile http venite de la client. Middleware-ul respectiv nu are rolul de a filtra cererile, ci de a înregistra data la care s-a făcut cererea. Salvând mereu în cache această dată, am căpătat pe partea de client funcționalitatea de a vedea, în timp real, ultima oră la care orice utilizator a fost activ în aplicație.

În continuare voi prezenta structura pe care am ales să o dau memoriei cache disponibile. Pentru fiecare utilizator este disponibilă o listă, identificată unic prin id-ul acestuia. Toate listele existente în cache conțin:

* O variabilă în care este păstrată data ultimei cereri făcute de utilizator către server.
* O listă de notificări față de care utilizatorul nu a acționat în niciun fel (Notificarea constituie un obiect ce conține date precum numele și id-ul utilizatorului de la care provine aceasta, sau data la care notificarea respectivă a fost trimisă)
* Un obiect ale cărui proprietăți sunt folosite pentru a stoca starea jocului, astfel încât în orice moment acesta să poate fi reluat. Astfel, asigurăm persistența datelor în cazul meciurilor dintre doi utilizatori. Un obiect de acest gen are următoarele proprietăți:
  + Id-ul utilizatorului logat.
  + Numele utilizatorului logat.
  + Scorul utilizatorului logat.
  + Id-ul oponentului.
  + Numele oponentului.
  + Scorul oponentului în prezent.
  + Statusul jocului (început, terminat, pe cale să înceapă, în stadiul de pregătire a jucătorilor)
  + Numărul problemei la care s-a ajuns, împreună cu o listă ce conține toate problemele care vor apărea pe parcursul jocului.
  + Timpul de la care pornește numărătoarea inversă.

Este necesar, ca pe parcursul jocului, utilizatorul să aibă acces către aceste informații, dar odată ce statusul meciului se transformă în ”Terminat”, datele pot fi salvate în baza de date, întrucât nu vor mai suferi modificări. De asemenea, un meci terminat nu va mai putea fi accesat nici de pe partea clientului, deci aceste date pot fi șterse și din cache-ul serverului.

## OneSignal

## Pusher Channels

# Bibliografie

Bean, M. (2015). *Laravel 5 Essentials.* Birmingham: Packt.

*Directory Structure*. (2018, May 2). Adaptat de pe Laravel Docs: https://laravel.com/docs/5.6/structure

1. https://www.quizup.com [↑](#footnote-ref-1)
2. Utilizatorii care au aplicația deschisă. [↑](#footnote-ref-2)
3. Invitația la joc adresată oponentului, respectiv acceptarea provocării de către oponent. [↑](#footnote-ref-3)
4. Orice informație este stocată temporar

   (în cazul în care este pierdută conexiunea la internet). [↑](#footnote-ref-4)
5. Întrebări eronate, probleme raportate sau cu răspunsuri greșite, probleme raportate pe nedrept, etc [↑](#footnote-ref-5)
6. Consolă în care sunt implementate comenzi care ajută dezvoltatorul în realizarea aplicației web. [↑](#footnote-ref-6)
7. ORM specific framework-ului Laravel. [↑](#footnote-ref-7)