Sistem de gestionare a cererilor de despăgubire pentru o societate de asigurări

Rareș-Florian Neagu

18 iunie 2017

Cuprins

1 Introducere **3**
- 1.1 Condițiile inițiale . . . 3
- 1.2 Accesul la aplicație . . . 4
- 1.3 Detalii despre dezvoltarea aplicației . . . 4
- 1.4 Soluțiile software adoptate . . . 5
- 1.5 Structura aplicației . . . 6
  - 1.5.1 Înregistrarea daunelor . . . 6
  - 1.5.2 Introducerea datelor vânzărilor săptămânale (import) . . . 6
  - 1.5.3 Sistemul de decizii . . . 6
  - 1.5.4 Sistemul de raportare . . . 6

2 Tehnologiile folosite **7**
- 2.1 PHP 5.6 . . . 7
- 2.2 Laravel 5.3 . . . 7
  - 2.2.1 Directive Blade . . . 8
  - 2.2.2 Laravel Forms / HTML . . . 9
  - 2.2.3 Consola Artisan . . . 9
  - 2.2.4 Eloquent ORM . . . 10
  - 2.2.5 Migrări . . . 10
- 2.3 Baza de date - MySQL . . . 11
  - 2.3.1 Relații . . . 11
  - 2.3.2 PhpMyAdmin . . . 12
- 2.4 Găzduire partajată . . . 13
- 2.5 GitHub . . . 14
- 2.6 Certificat SSL . . . 15
  - 2.6.1 StartCom . . . 15
  - 2.6.2 CloudFlare . . . 15
- 2.7 jQuery & jQuery UI . . . 16
- 2.8 Knockout.js . . . 17
- 2.9 Utilitare . . . 18
  - 2.9.1 Google Analytics . . . 18
  - 2.9.2 PhpStorm IDE . . . 18
  - 2.9.3 Heroku . . . 19
  - 2.9.4 Composer . . . 19
  - 2.9.5 Laravel Elixir . . . 20
  - 2.9.6 LaTeX . . . 21

3 Arhitectura software a aplicației **22**
- 3.1 Rutina de apelare a programului . . . 22
- 3.2 Baza de date . . . 23

3.3 Structura Model-View-Controller . . . 25
3.3.1 Controllers . . . 25
3.3.2 Models . . . 30
3.3.3 Views . . . 31
3.3.4 Funcții ajutătoare . . . 33
3.4 Rapoarte . . . 34
3.4.1 Implementarea platformei de rapoarte Excel . . . 35

4 Probleme apărute în timpul dezvoltării aplicației 37
4.1 Primul modul . . . 38
4.2 Problema datelor incomplete . . . 39
4.3 Modulul de rapoarte și decizii . . . 40
4.4 Decuplarea deciziilor de vânzări . . . 40
4.4.1 Modificări în viitorul apropiat . . . 41
4.5 După planul inițial de dezvoltare . . . 41
4.5.1 Introducerea datelor și rapoartele avansate . . . 41
4.5.2 În continuare . . . 41

5 Manual de utilizare 43
5.1 Înregistrarea unei cereri . . . 43
5.2 Secțiunea de mesaje din cadrul cererilor de despăgubire . . . 48
5.3 Interfața de administrare . . . 50
5.3.1 Modulul cererilor de despăgubire . . . 51
5.3.2 Modulul introducerii datelor în sistem . . . 53
5.3.3 Modulul de vânzări . . . 55
5.3.4 Crearea unei decizii . . . 58
5.3.5 Modulul de decizii . . . 58
5.3.6 Modulul de raportări . . . 61
5.4 Modificări în viitorul apropiat . . . 65
5.4.1 Introducerea câmpului de factură în decizie . . . 65
5.4.2 Modularizarea tabelelor . . . 65
5.4.3 Sistem modular de rapoarte . . . 65

Capitolul 1

Introducere

1.1 Condițiile inițiale

Societatea „Fandu Holdings Ltd." este o societate ce se ocupă cu cererile de despăgubire pentru Altex și Media Galaxy[1], ca asigurator în numele lui „AmTrust International Underwriters Limited", care gestionează cererile de despăgubire ale clienților unor lanțuri de magazine, conform asigurărilor încheiate pentru echipamente mobile achiziționate.

Inițial aplicația conținea un ecran de introducere a cererilor de despăgubire și salvare a pozelor cu documentele necesare finalizării cererii de despăgubire. Restul operațiilor de gestionare și raportare a daunelor se întocmeau manual prin fișiere Excel.

S-a dorit deci o automatizare a întregului sistem, pentru a asigura următoarele funcții:

1. Corectitudinea datelor.
2. Posibilitatea de modificare a datelor.
3. Trasabilitatea datelor.
4. Confidențialitatea datelor.
5. Preluarea datelor existente și omogenizarea lor, conform noii structuri.
6. Gestionarea datelor nestructurate prin digitalizare și introducere în sistem.
7. Asigurarea unui sistem flexibil de raportare.
8. Asigurarea continuității afacerii prin soluții de salvare a datelor în cloud.

După asimilarea cunoștințelor specifice domeniului de activitate – asigurări – și aprofundarea temei de proiectare, a fost întocmit, împreună cu reprezentanții societății, diagrama logică a aplicației, cu modulele aferente. De asemenea, s-au adoptat următoarele măsuri:

1. Aplicația va fi dezvoltată pe un mediu de test, folosind date reale depersonalizate.
2. Fiecare modul va fi testat inițial de către dezvoltator, urmărind și interdependența cu restul aplicației.
3. Modulul va fi testat de utilizatori și aprobat de către conducere.
4. Modulul va fi integrat în sistemul de producție.

Simultan cu dezvoltarea aplicației și implementarea modulară în producție, s-a păstrat sistemul anterior de gestionare și raportare în Excel, pentru a se asigura exactitatea rapoartelor scoase de sistem.

1.2 Accesul la aplicație

Conform procedurilor încheiate între „Fandu Holdings Ltd.” [1] și distribuitorul de produse mobile „Altex / Media Galaxy”, asiguratul primește la achiziția produsului polița de asigurare.

Regulamentul poliței „Mobile Protect” este valabil online la `https://mediagalaxy.ro/regulament-mobile-protect`[1].

În cazul unei defecțiuni, clientul trebuie să înregistreze cererea de despăgubire online la adresa `www.fandu.uk/claims`, fiind obligați să trimită și documentele justificative.

1.3 Detalii despre dezvoltarea aplicației

Dezvoltarea a început cu prima întâlnire între conducerea companiei, utilizatori și mine, unde s-au stabilit modulele aplicației:

1. Înregistrarea daunelor:
   (a) Înregistrarea cererii de despăgubire online.
   (b) Atașarea evidențelor – poze sau alte documente justificative.
2. Introducerea datelor vânzărilor săptămânale (import).
3. Sistemul de decizii:
   (a) Asocierea cererii de despăgubire cu contractul de asigurare întocmit la achiziția produsului.
   (b) Luarea deciziei de reparație sau înlocuire.
   (c) Actualizarea evoluției deciziei din punctul de vedere și al costului soluționării cererii de despăgubire.
4. Sistemul de raportare:
   (a) Statistici rapide (câte cereri de despăgubire, decizii au fost create în perioada respectivă de timp).
   (b) Raport cu toate deciziile plătite într-o anumită perioadă de timp.
   (c) Raport cu numărul de asigurări, grupate în funcție de tipul de asigurare și prețul ei, în perioada respectivă.
   (d) Raport cu numărul vânzărilor asigurărilor grupate în funcție de magazinul ce a făcut vânzarea, împreună cu perioada când s-a vândut primul și ultimul produs.
   (e) Raport detaliat despre produsele ce încă sunt asigurate, împreună cu posibila cerere de despăgubire asociată.

Accentul s-a pus pe dorința de a scoate rapoarte, unitare și flexibile, și micșorarea numărului de fișiere accesate.

Pe tot parcursul dezvoltării aplicației, am fost singurul decident și dezvoltator, referitor la:

- Arhitectura aplicației informatice.
- Construirea bazei de date.
- Împărțirea aplicației în module interconectate și flexibile.

Figura 1.1: Pagina principală a `https://fandu.info`

- Migrarea spre o soluție de găzduire partajată.
- Securitatea datelor.
- Interacțiunea utilizatorilor și a clienților cu aplicația.
- Soluții și strategii de backup.

Pe parcursul dezvoltării în mediul de test, rigiditatea legăturilor dintre module, impusă inițial, a fost întâmpinată cu rezistență, din cauza libertății oferite de Excel, dar exactitatea rapoartelor obținute au fost în favoarea soluției adoptate.

Interacțiunea cu clientul a fost gândită a fi cât mai rapidă și eficace. Atunci când intră pe pagina aplicației, el poate deja să completeze cererea de despăgubire, așa cum se poate observa în figura 1.1. Clientul este îndrumat pentru a completa cât mai detaliat detaliile despre cererea de despăgubire și este avertizat atunci când apare o eroare sau a uitat să completeze un câmp necesar.

Am ales să implementez baza de date și codul aplicației la o companie de încredere ce oferă soluții de găzduire partajată `https://xServers.ro`[2]. Dar pozele și spațiul de stocare a pdf-urilor încărcate de client sunt găzduite de Amazon Web Services[3], pentru a beneficia de prețul redus și valabilitatea extinsă a fișierelor pe „cloud”. Avantajul soluției alese este traficul de internet necontorizat și spațiu de stocare suficient.

De asemenea, au fost implementate sisteme de backup săptămânal a bazei de date, arhivate și encriptate în a terța locație.

Codul sursă este ținut într-un proiect „git”, un sistem de gestionare și versionare a codului de programare, încărcat și protejat pe GitHub.

1.4 Soluțiile software adoptate

Pentru a putea reduce costurile aplicației, am optat pentru o metodă tradițională de găzduire partajată, ce oferă PHP și o bază de date MySQL inclusă în pachet.

După prospectarea librăriilor de dezvoltare software în PHP, proiectul a fost conceput în Laravel[4] inspirată din arhitectura sistemului „Ruby on Rails”, care oferă cele mai ridicate standarde de programare.

Alt avantaj este încercarea minimizării complexității ascunse[5], ce face codul mult mai ușor de întreținut și permite dezvoltarea mai rapidă, fără a pierde mult timp înțelegând arhitectura sau codul scris anterior.

Sintaxa ușor de înțeles și ambiguitatea redusă fac din Laravel, în opinia mea, un bun provocator chiar și pentru „ASP.NET MVC"[6], liderul pe piață în momentul actual. Ambele platforme oferă soluții de la modele, la gestionarea bazei de date, la rularea migrațiilor, la salvarea stării și chiar și integrarea automată a unui sistem de utilizatori.

Dar spre diferență de soluția Microsoft, toată platforma necesară construirii și distribuirii aplicației nu este blocată pe sistemul de operare Windows (cu .NET Core situația se schimbă, dar în acest moment nu a fost adoptată la scară largă), ci poate să funcționeze pe orice sistem de operare.

1.5 Structura aplicației

- Înregistrarea daunelor.
- Introducerea datelor vânzărilor săptămânale (import).
- Sistemul de decizii.
- Sistemul de raportare.

Aici pun figura cu diagrama

Capitolul 2

Tehnologiile folosite

2.1 PHP 5.6

La apariția internet-ului, toate paginile erau statice. Nu puteai să interacționezi cu conținutul paginii. Dar apoi Netscape a revoluționat paginile, construind limbajul de programare LiveScript (redenumit ulterior JavaScript).

În puțini ani, Netscape a revoluționat și modul în care paginile sunt distribuite utilizatorilor prin conceptul de JavaScript pentru aplicațiile ce serveau paginile. Astfel, s-a construit primul scenariu de cod folosit de servere. [7]

În continuare, s-a definit „Common Gateway Interface" (CGI), ce oferă un protocol pentru serverele web de a executa programe pentru a genera pagini de web dinamice, actualizate în funcție de cererea utilizatorului. Serverul web permite administratorului să seteze ce URL-uri vor fi folosite de ce programe CGI prin setarea unui folder cu scripturi. În loc să trimită fișierul respectiv, serverul HTTP apelează script-ul și trimite orice ar fi afișat ca și rezultat pentru client. [8]

Un astfel de program a fost și invenția lui Rasmus Lerdorf. Acesta a extins programele ce gestionau pagina lui personală pentru a interacționa cu baze de date și formulare, dând numele implementării „Personal Home Page/Forms Interpreter" (sau pe scurt, PHP/FI). Însă acesta nu a avut o viziune clară asupra limbajului de programare, a continuat să adauge limbajului de programare ceea ce considera a fi următorul pas logic. [9]

Printre îmbunătățirile aduse limbajul se numără „superglobals" (o metodă ușoară de a accesa parametrii trimiși de formularele paginilor HTML) în versiunea 4.1, metode de gestiune a codului orientat pe obiecte și o interfață standardizată pentru a accesa bazele de date. Ulterior s-a adăugat pe parcursul dezvoltării limbajului suport pentru „namespace"-uri, generatoare, funcții anonime, extinderi. În 2014, a fost lansat PHP 5.6, ce va fi întreținut până pe 31 Decembrie 2018. [10]

2.2 Laravel 5.3

Laravel este o librărie gratuită de PHP. Pentru Laravel 5.3, minimul necesar librăriei este PHP 5.5.9 sau o versiune mai nouă. Librăria a fost concepută de Taylor Otwell, pentru a construi aplicații web bazate pe arhitectura „Model View Controller". Avantajele oferite de Laravel sunt[11]:

- Sistemul modular de pachete, cu propriul său sistem de gestionare.
- Moduri diferite de a accesa bazele de date relaționale.
- Utilitare pentru scrierea și menținerea aplicației.

- Orientarea spre expresivitatea codului.

Arhitectura „Model View Controller” se bazează pe cele trei componente inter-conectate „slab” ale unei aplicații: Model-ul, Controller-ul, respectiv View-ul. Model-ul este componenta centrală a șablonului, pentru că se ocupă de datele, logica și regulile aplicației. Controller-ul gestionează modelele. Acceptă datele de la utilizator și le transformă în comenzi pentru model și/sau View. View-ul este orice metodă de vizualizare a informației unui Model.[12]

Avantajele arhitecturii, mulțumită separării conceptuale, ajută dezvoltarea simultană și ușurința modificărilor. Astfel, mai multe persoane pot lucra simultan la o parte a aplicației. Adoptarea unei astfel de arhitecturi trebuie studiată în detaliu, pentru că nu este ușor de folosit pragmatic.

2.2.1 Directive Blade

Un mare avantaj al librăriei este un simplu, dar foarte puternic și eficient sistem de șablonare — Blade. Spre deosebire de alte sisteme de șablon, Blade nu inhibă dezvoltatorul a folosi cod PHP în View-uri. De fapt, toate View-urile Blade sunt traduse în cod simplu PHP și salvate până ce sunt modificate. Astfel reușește să adauge aproape zero logică în plus aplicației.

Pentru a afișa ceva pe ecran folosind PHP în mijlocul codului HTML, ar trebui să scrii:

```
<?= $variabila ?>
```

Dar asta te face vulnerabil la „XSS”.

XSS (Cross Side Scripting) este o vulnerabilitate a securității unei aplicații, des întâlnită în aplicațiile web, ce permite adăugarea oricărui cod, fie el malițios sau inocent, paginilor vizualizate de clienții normali.

Astfel, pentru a preveni exploatarea datelor trimise de la Controller, pentru a afișa valorile variabilelor trimise, directivele Blade indică folosirea sintagmei:

```
{{$variabila}}
```

Desigur, nu ești limitat la doar nume de variabile, ci la orice s-ar putea considera parametrul unei funcții, deoarece sunt trimise prin metoda
`htmlspecialchars` a PHP-ului.

Printre cele mai folositoare beneficii folosind Blade se află și construirea fișierelor ce definesc structura HTML-ului aplicației. Nu se definește practic o pagină principală, doar se specifică pentru paginile existente deja a se folosi structura găsită în pagina specificată prin comanda:

```
@extends('layouts.app')
```

unde în acest caz în folderul „layouts” găsim fișierul „app.blade.php”.

Pentru a introduce conținutul paginii ce extinde șablonul, Blade folosește secțiuni. Declarate în pagina principală, acestea indică locul unde se așteaptă conținut, putând fi denumite după nevoile șablonului. De asemenea, în cazul definirii unei secțiuni, se poate folosi și o valoare implicită. Cel mai des e folosit un șablon în definirea titlului, cu o valoare implicită a numelui proiectului. În cazul acestui proiect, am folosit ca sufix numele configurat în fișierul mediu, lăsând la dispoziția fiecărei pagini a seta titlul.

Folosind secțiuni, am descoperit o problemă la refolosirea unor componente vizuale. Blade adaugă precum PHP conținutul fișierului ce trebuie să existe unde ai scris instrucțiunea respectivă folosind comanda:

```
@include('decisions.includes.search')
// ce adaugă fișierul
// resources/views/decisions/includes/search.blade.php
```

Continuând șirul dezvoltării, descopăr o altă problemă la introducerea a mai multor segmente de cod JavaScript în aplicație, din fișiere Blade separate. Pentru a rezolva și această problemă, Blade se folosește de:

```
@push('scripts')
```

pentru a adăuga informații într-o stivă, ce va fi afișată în pagina ce definește structura HTML prin simpla comandă:

```
@stack('scripts')
```

Printre marile îmbunătățiri aduse de Blade, folosite în acest proiect, se numără și directivele. Acestea sunt propriile comenzi, definite de programator, ce extind limbajul. Sunt traduse în simple instrucțiuni de cod PHP. Am scris astfel propria directivă de a construi automat codul necesar afișării, folosind HTML și librăria vizuală „Bootstrap", a unui câmp dintr-un formular, cu nume, tip și validare proprie:

```
@input(['eveniment_strada', ['Strada', false]])
// unde 'eveniment_strada' - id-ul câmpului.
// unde 'Strada' - denumirea câmpului.
// unde 'false' - necesitatea completării câmpului.
```

2.2.2 Laravel Forms / HTML

O mare parte a oricărei aplicații web este interacțiunea cu clientul. Pentru asta, există formulare ce sunt trimise către calculatorul aplicației. O fostă librărie inclusă în Laravel, dar acum extinsă și desprinsă este librăria ajutătoare pentru formulare și HTML. Este întreținută de „Laravel Collective", o comunitate ce se ocupă de menținerea componentelor scoase din Laravel de „Taylor Otwell". Astfel, el are prilejul de a se concentra pe munca sa (Laravel) și comunitatea susține componentele îndrăgite de programatori în viață, independent.[13]

Un astfel de proiect este „Forms & Html" [14], ce simplifică codul ce trebuie scris de un programator Laravel. În loc de a scris de fiecare dată ce câmp are ca referință ce obiect din model, se definește în schimb în momentul construirii formularului modelul de referință. Când se construiește un câmp, trebuie doar să fie bine specificat identificatorul câmpului, iar când va fi trimis către client, va fi completat automat cu valoarea din model. De asemenea, un al treilea parametru lasă loc programatorului să completeze celelalte atribute ale nodului HTML printr-un vector unde cheia reprezintă numele atributului și valoarea reprezintă conținutul.

Nu doar metode ajutătoare, „Forms & Html" acceptă și componente specializate, definite de programator, ca șabloane Blade. Prin definirea lor într-un folder special, acestea pot fi folosite precum cele deja definite, pentru a ușura clasele și atributele necesare pentru dezvoltatorii ce folosesc „Bootstrap". Cum acest proiect se folosește de Bootstrap și „Forms & Html", vă pot da un exemplu de două componente des folosite în aplicație:

```
{{Html::panel_begin('Delete Assurance', 'in')}}
  // aici scriu cod...
{{Html::panel_end()}}
// unde 'Delete Assurance' - numele panoului
// unde 'in' - starea panoului: de a fi deja deschis
```

2.2.3 Consola Artisan

Degeaba poți avea o soluție elegantă a unei probleme, dacă pentru a crea un Controller nou ai nevoie de a scrie o grămadă de linii de cod. Este adevărat, poți să scurtezi mult din timpul

prețios folosind șabloanele unui mediu inteligent de programare. Dar poate uiți dacă trebuie să modifici în mai multe locuri.

Aici intervine consola „Artisan", o interfață prietenoasă pentru programatorii obișnuiți cu liniile de comandă din „Ruby on Rails". Ajută programatorul la a construi mai rapid aplicația, bazată pe arhitectura „Symfony". Printre cele mai des folosite opțiuni se numără:

- Construirea rapidă a mai multor tipuri de clase (Migrare, Controller, Model, Request, Event).
- Gestionarea migrărilor bazei de date.
- Curățarea fișierelor precompilate sau aflarea rutelor definite.
- Generarea codului necesar unui sistem rudimentar de înregistrare, intrare și resetare parolă, cât și filtrarea accesului utilizatorului în aplicație.

De menționat ar mai fi extensibilitatea consolei prin pachetele salvate, care pot adăuga funcții vitale pentru mediul de dezvoltare ales. Aici mă refer la „Laravel PhpStorm IDE Helper" (`barryvdh/laravel-ide-helper`), despre care voi discuta în subcapitolul despre PhpStorm.

2.2.4 Eloquent ORM

Un ORM („Object-relational mapping") este o tehnică de programare care traduce datele incompatibile (tabelele unei baze de date) în obiecte PHP, care pot fi folosite în cadrul aplicației. Spre diferență de tehnicile tradiționale de a transforma un obiect dintr-un limbaj orientat pe obiecte într-o linie dintr-un tabel al unei baze de date relaționale, ORM-ul adesea reduce numărul liniilor de cod scrise.[15]

Folosit în conjuncție cu un ORM, șablonul de linie activă („active record pattern") este folositor pentru a salva în memorie datele obiectului dintr-o bază de date relațională. Interfața obiectului conceput care se conformează șablonului va avea metode de a adăuga, modifica, șterge linii din baza de date, dar și câmpuri care corespund mai mult sau mai puțin coloanelor tabelei care stă la baza obiectului. Acest șablon arhitectural a fost denumit de „Martin Fowler" în 2003. [16]

Eloquent ORM, inclus cu Laravel, oferă programatorului o elegantă implementare a șablonului „Active Record". Fiecare tabel are un Model corespunzător, ce este folosit pentru a interacționa — a adăuga sau modifica — baza de date.

Laravel diferă de alte librării și se aseamănă cu ASP.NET cu puternica sa preferință a convențiilor împotriva configurației. Este recomandat deci de a denumi tabelele cu litere mici cu sublinie (e.g: `„test_cases"`), în cazul în care nu se specifică explicit tabelul folosit pentru model. Numele modelului ar trebui să urmeze ghidul de denumire a claselor conform standardului PHP PSR-1 [17] (i.e.: `„TestCase"`).

2.2.5 Migrări

Pentru a simplifica munca programatorilor, Laravel a ales să folosească un sistem de migrare a bazelor de date. Asigura la pornirea aplicației că versiunea bazei de date corespunde ultimelor modificări aduse codului prin tabela numită `„migrations"`. Avantajul gestionării incrementale și reversibile a schimbărilor a schemei unei baze de date relaționare asigură pierderi minime a datelor de pe calculatorul de producție. În avantajul programatorului mai există și ușurința de a reveni la varianta originală a bazei de date, când se dezvoltă o migrare nouă, în cazul în care aceasta are lipsuri sau se încearcă altă abordare.

Pentru a construi o migrare, se apelează comanda Artisan:

```
php artisan make:migration create_users_table --create=users
# sau, o variantă ce construiește și migrarea și modelul asociat
php artisan make:model --migration User
```

Comanda construiește un nou fișier în folder-ul special pentru migrări și scrie în interiorul lui o clasă nouă `CreateUsersTable`.

Se observă că această nouă clasă respectă principiul gestionării incrementale și reversibile a schimbărilor schemei bazei de date pentru că are două metode denumite „up", „down", pentru când se va aplica, respectiv scoate migrarea respectivă.

Clasa ce se ocupă de construirea tabelului are majoritatea funcțiilor ajutătoare pentru a construi orice schemă dorită de programator. Păstrează în spatele sintaxei ușor de folosit abilitatea de a se putea aplica asupra oricărui sistem relațional de baze de date. Avantajul programatorul este că nu trebuie să se mai complice cu duplicarea codului și testarea pe mai multe baze de date a migrării. Totul este asigurat de Laravel.

2.3 Baza de date - MySQL

Baza de date relațională cu sursa sa deschisă MySQL este un sistem de gestionare a unei baze de date relaționale. Numele provine de la numele fiicei fondatorului „Michael Widenius"[18] și abrevierii SQL („Structured Query Language"). Dezvoltarea sa, cu codul sub termenii licenței GNU GPL, a fost sponsorizată și proprietatea companiei suedeze „MySQL AB". În momentul actual, compania nu mai există, fiind cumpărată de corporația Oracle.

Această componentă vitală face parte din platforma cu codul deschis pentru dezvoltarea aplicațiilor web LAMP:

- Linux
- Apache
- MySQL
- PHP/Perl/Python

Multe aplicații, de la cele pentru amatorii paginilor personale, a buletinelor de știri sau a canalelor de discuție, se bazează pe MySQL. Printre marile companii ce folosesc MySQL, se numără:

- Google
- Facebook
- Twitter
- Flickr
- YouTube

2.3.1 Relații

În Octombrie 2005, corporația Oracle a achiziționat „Innobase OY", o companie finlandeză ce a dezvoltat InnoDB. [19] Sistemul dezvoltat de salvare permite bazei de date MySQL a oferi tranzacții și „chei străine". După achiziție, Oracle a confirmat prelungirea contractului ce permite folosirea lui de către „MySQL AB".

Cheile străine, în contextul bazelor de date relaționale, se referă la un câmp sau mai multe ce identifică unic o coloană sau o linie a unui tabel de alt tabel[20]. De asemenea, un tabel poate să aibă una sau mai multe referințe, ce se pot referi la alte tabele părinte, dar chiar și la el însuși (numindu-se „cheie străină recursivă"). Pentru a păstra legăturile între tabele, atunci când se modifică date sau se șterg, se oferă următoarele opțiuni:

- `CASCADE` - se șterge și referința.
- `RESTRICT` - nu se poate șterge linia atâta timp cât există o referință spre el.
- `NO ACTION` - asemănătoare `RESTRICT`, ce se verifică în schimb la finalizarea tranzacției.
- `SET NULL` - referința se va transforma în `NULL`.
- `SET DEFAULT` - referința se va transforma în variabila cu care este inițializat câmpul în mod implicit.

2.3.2 PhpMyAdmin

PhpMyAdmin este un sistem gratuit și cu codul liber de administrare a bazelor de date MySQL și MariaDB. Ca o aplicație portabilă scrisă majoritar în PHP, a devenit una dintre cele mai populare sisteme de administrare, mai ales pentru serviciile de găzduire, precum xServers și nu numai.

A început dezvoltarea „Tobias Ratschiller", un consultant IT și mai târziu fondator a companiei Maguma, a unei interfețe bazate pe PHP a bazei de date MySQL în 1998. În anul 2000, din cauza lipsei de timp, a renunțat la proiect, dar până în acel moment, devenise deja una dintre cele mai populare aplicații administrative. Având o comunitate mare, un grup de trei dezvoltatori, pentru a coordona mai bine numărul care tot creștea de modificări, înregistrează la SourceForce „The phpMyAdmin Project" și preiau dezvoltarea în 2001. [21]

În continuare, principalul website a migrat spre un sistem distribuit și au mutat ulterior urmărirea problemelor pe GitHub. Înainte de versiunea 4, ce folosește apeluri asincrone pentru a optimiza folosința, se foloseau cadre HTML.

În afară de a gestiona bazele de date și a oferi o interfață web, mai poate să:

- importe datele din CSV și SQL.
- să salveze într-o multitudine de formate, printre care și CSV, SQL, XML, Excel.
- caute orice date dintr-un context global sau local.
- transforme orice „blob" într-un format predefinit.
- ofere grafice actualizate în timp real pentru monitorizarea:
  - conexiunilor
  - memoriei
  - consumului procesorului

2.4 Găzduire partajată

Găzduirea partajată se folosește pentru a pune un web server conectat la internet valabil pentru mai multe persoane. Partajându-se mai multor clienți, se amortizează costul îngrijirii și astfel prețurile sunt reduse.

Am ales pentru Fandu soluția oferită de xServers[2] pentru certificatele ce asigură calitatea (ISO 9001) și securitatea (ISO 27001) informației, suport tehnic non-stop, infrastructură și echipamente profesionale, cu o disponibilitate de 99.9% lunară, viteza aplicației fiind asigurată și de către soluția de stocare adoptată — SSD („Solid State Drive").

Un mare avantaj, dar în același timp și dezavantaj este oprirea comenzii de interpretare și executare a codului PHP, `eval`. Din păcate, este problematic pentru consola Artisan, pentru că nu poate să aplice în momentul actualizării bazei de date tot codul de migrare.

Soluția, la care s-a ajuns drept compromis în urma dezbaterilor cu departamentul tehnic, ce migrează codul și baza de date a server-ului, este:

1. Se salvează toate datele existente, folosind utilitarul oferit de xServers.
2. Se introduce un fișier nou în directorul aplicației la:

   ```
   storage/framework/down
   ```

   Se închide astfel accesul la server cât timp se migrează baza de date și conținutul aplicației în sine.
3. Se află codul necesar migrării bazei de date prin comanda:

   ```
   php artisan migrate --pretend -vvv
   ```

   Ce reiese va fi cod SQL ce va fi copiat cu ușurință în phpMyAdmin-ul de pe server.
4. După ce s-a migrat cu succes baza de date, se copiază următoarele foldere:
   - `app` - tot codul important al aplicației.
   - `bootstrap` - sistemul de încărcare a platformei Laravel.
   - `config` - setările implicite Laravel, în lipsa valorilor în fișierul mediului curent (`.env`).
   - `database` - migrările bazei de date
   - `public` - pentru că folosim „Laravel Elixir", ce se ocupă de gestiunea variantelor codului JavaScript și CSS, se vor copia și în calea publică `public_html`, pentru că acestea vor fi servite clienților și trebuie să poată să fie accesibile.
   - `resources` - directivele Blade, codul JavaScript și SASS.
   - `routes` - rutele ce determină Controller-ul apelat.
   - `storage` - doar structura, pentru că va conține fișierele temporare traduse din Blade în PHP.
   - `tests` - se poate sări, pentru că nu afectează rezultatul final.
   - `vendor` - dacă s-a modificat `composer.json`, adăugând sau modificând sau ștergând pachete.
5. Se va șterge fișierul nou creat:

`storage/framework/down`

și astfel migrația va fi completă.

În cazul unor migrări minore, ce nu implică baza de date, se pot suprascrie direct fișierele modificate în cadrul aplicației. PhpStorm ajută aici prin opțiunea de a încărca prin FTP (encriptat) fișierele modificate.

Observație: Se copiază directorul `public` de două ori, când se fac modificări la stil sau la codul JavaScript: o dată în cadrul directorului ce conține întregul proiect Laravel și o dată în directorul `public_html`.

„Laravel Elixir" funcționează bazându-se pe un fișier JSON, ce spune ce versiune de stil și cod să folosească, aflat la:

`/public/build/rev-manifest.json`

Această cale nu este valabilă clienților, pentru că aplicația nu se află în directorul public. S-au făcut deci mici modificări asupra fișierului `index.php`, aflat tot în directorul public, pentru a arăta calea corectă.

Trebuie să se copieze, în concluzie, codul și stilul și în acest director.

2.5 GitHub

Trebuie să ne asigurăm că păstrăm undeva codul sursă al aplicației, pentru a fi siguri că nu se vor corupe datele. De asta am ales să folosesc un program de gestionare a codului.[22]

Sistemul de gestiune a codului se ocupă cu organizarea și aflarea modificărilor fișierelor cu cod, construind revizii pentru fiecare modificare în parte, ușoare de identificat. De asemenea, comunică cu un server ce păstrează codul sursă și lasă programatorii să aibă o variantă ce funcționează pe mașina locală.

Funcționând pe un modelul tranzacțional, ajută enorm lucrului în echipă pentru că păstrează pentru cele mai des folosite operații (salvare, vizualizare istoric, revenind la modificări anterioare) sunt optimizare și deci rapide. Modelul tranzacțional asigură și că nu vor apărea probleme la modificări concurente de două sau mai multe persoane asupra aceluiași fișier.

Un astfel de sistem de gestiune a versiunii (VCS), pentru a urmări schimbările fișierelor și coordonarea lucrului în echipă, este „Git".[23] A fost conceput de „Linus Torvalds" în 2005 împreună cu alți dezvoltatori ai kernel-ului LInux pentru dezvoltarea kernel-ului. Se concentrează asupra vitezei, integrității datelor și susținerea unui mod de lucru distribuit și neliniar. Astfel, fiecare folder „Git" pe fiecare calculator conține istoricul complet al proiectului, independent de accesul la rețea. Este cel mai folosit sistem, mulțumită multiplelor garanții împotriva coruperii datelor, iar conform unui studiu făcut în 2015, 69.3% dezvoltatori folosesc activ „Git".

Pentru că un sistem de gestiune a versiunii se folosește de un server, am ales varianta online și bazată pe web „GitHub", ce folosește „Git" și adaugă și funcționalitate pentru lucrul în echipă, precum [24] :

- „bug tracking" - gestionarea problemelor în cod.
- „feature requests" - gestionarea propunerilor pentru funcționalitate nouă.
- „task management" - cine, ce, când are de modificat în cadrul proiectului.
- „wiki" - un loc comun despre tot ce ține de documentația proiectului.

GitHub valorifică securitatea informațiilor confidențiale și asigură de furtul intelectual sau de majoritatea problemelor de natură fizică, distrugerea laptop-ului, spre exemplu.

Fandu folosește de la prima variantă a codului de programare „git" și „GitHub", în modul privat. Sunt singurul ce are acces la codul sursă pentru că sunt și singurul dezvoltator. Pentru a fi extra precaut, tot codul este salvat pe un disc virtual encriptat cu VeraCrypt. Pentru a fi pus pe serverul de producție, se folosește doar FTP encriptat. La fel și pentru mediul de testare. Consider că astfel șansa de furt intelectual este infimă.

2.6 Certificat SSL

Certificatele SSL sunt fișiere mici ce leagă digital o cheie criptografică de detaliile unei organizații. Atunci când este instalat pe un server web, activează protocolul HTTPS și permite comunicare criptată între server și clienți. De obicei, SSL este folosit pentru a securiza tranzacțiile cu cărțile de credit și transferul de date sau de înregistrări. Dar mai nou, este folosit la orice activitate pe internet, de la media la paginile personale și/sau pagini cu date personale.

Certificatele SSL leagă împreună:

- Un nume de domeniu sau server.
- O identitate organizațională (i.e: numele companiei) și locația sa.

O autoritate de certificate este o entitate ce distribuie certificate digitale SSL. Acesta acreditează deținătorul asupra unei chei publice cu numele său. Astfel se asigură prin semnături digitale anumite presupuneri despre cheia privată. Autoritatea de certificate are aici rolul de terță de încredere, pentru deținător și nu numai.

O astfel de autoritate de certificate este StartSSL. StartSSL face partea din compania StartCom, fondată de „Eddu Nigg" în Israel. Aceasta oferă gratuit certificate SSL clasa 1, ce funcționează pentru un singur domeniu al unui website.

2.6.1 StartCom

La începutul dezvoltării aplicației, o mare necesitate era o puternică securitate a paginilor. Pentru că mai folosisem înainte StartCom și aveam o oarecare încredere în ei, am ales, pentru a activa rapid modulul HTTPS pe gazda de producție, un certificat de la ei.

StartCom a fost achiziționat în secret de „WoSign Limited" (Shenzen, China), o altă autoritate de certificate. Din cauza numeroaselor probleme cu ei, Mozilla, Apple și Google au anunțat retragerea încrederii în certificatele după 21 octombrie 2016 (Mozilla, Google), 30 septembrie 2016 (Apple). În continuare, Google Chrome 57 va avea încredere în certificatele website-urilor în topul Alexa 1M, iar Chrome 58 va avea încredere în cele în topul 500000. [25]

Din păcate, acest proiect de gestionare a cererilor de despăgubiri a fost prins în cadrul acestei conjuncturi, folosind un certificat semnat de StartCom. Apăruse astfel problema că după o perioadă de timp nu o să mai funcționeze sistemul HTTPS.

2.6.2 CloudFlare

După multe dezbateri cu personalul de la Fandu, a intrat în funcțiune cât de curând soluția unui certificat CloudFlare.

CloudFlare este o companie în Statele Unite ale Americii. Oferă un sistem de livrare al conținutului și un sistem distribuit de nume de domeniu. Sistemul distribuit de nume de domeniu stă între serverul administratorului aplicației și aplicația mobilă a clientului. Astfel, nu numai că poate să prevină orice atac malițios, dar poate să și optimizeze paginile trimise.

Un alt mare avantaj pentru CloudFlare, introdus recent, este un sistem de gestiune a certificatelor SSL [26]. Astfel, am putut folosi certificatul ce nu mai putea fi folosit pentru a face o legătură „Full SSL”:

- Se asigură legătura client – CloudFlare prin certificatul lor. Apare iconița ce indică transmisia encriptată a datelor către server (lacătul verde).
- Se asigură legătura CloudFlare – server prin certificatul deja existent pe Fandu. Nu se trimite nicio informație nesecurizată prin Internet.

De asemenea, mai oferă și redirectarea automată a website-ului către domeniul securizat, dar deja se ocupă de asta Laravel.

Principalul motiv pentru care CloudFlare este în momentul actual indispensabil pentru Fandu este protecția de atacurile DDoS. Un atac DDoS („Distributed Denial of Service”) poate să aducă în genunchi întreaga arhitectură. Cum CloudFlare oferă o opțiune de „sunt sub asediu”, ce poate fi activată cât mai repede, poate să blocheze orice om frustrat.

Mulțumită juxtapoziției între server și client, se mai poate optimiza orice resursă trimisă. Se poate renunța la spațiile inutile din stilul sau codul interactiv JavaScript. Dar nu este nevoie de optimizarea la nivel de CloudFlare, deoarece se face la nivelul dezvoltării aplicației de Laravel.

Mulțumită „Laravel Mix”, se optimizează, concatenează și minimizează tot codul și stilul afișat. Cu cât mai puține dependențe externe, cu atât mai bine pentru securitatea aplicației.

2.7 jQuery & jQuery UI

jQuery este o librărie de JavaScript ce ajută la: [27]

- Navigarea DOM-ului („Document Object Model” — toate componentele ce formează pagina)
- Găsirea ușoară a elementelor relative, precum a părinților sau a unor copii a unor noduri, folosind sintaxa similară CSS („Cascading Style Sheet”).
- Construirea ușoară a animațiilor și gestionarea ușoară a anunțurilor rezultate (și nu numai)
- Ușoara integrare a elementelor preluate dinamice din altă parte, folosind metodologia asincronă de a trimite și recepționa informații.

De asemenea, mulțumită arhitecturii, jQuery permite dezvoltatorilor a construi „plug-in”-uri, pentru a extinde funcționalitatea. O astfel de librărie ce extinde jQuery este „jQuery UI” [28].

jQuery UI este o colecție de elemente vizuale, animate, testate, ce permit dezvoltatorilor un timp mai scurt de a pune produsul pe piață. Oferă efecte profesionale și „widget”-uri (elemente vizuale predefinite), precum un modul de a putea selecta data dintr-un calendar vizual.

Pentru sistemul de gestiune a cererilor de despăgubire, am folosit „jQuery” și „jQuery UI” pentru majoritatea interacțiunilor specifice paginilor. Am și extins platforma, prin trei metode, pentru a trimite și primi mai ușor informațiile de la Laravel.

Spre exemplu, logica de a încărca vânzările săptămânale se bazează pe această platformă. Utilizatorul o dată ce adaugă fișierul dorit de a fi încărcat în sistem, nu navighează de la pagina curentă, ci este întâmpinat cu o bară de progres. În acest moment, fiecare fișier este încărcat separat și preluat de sistem. Se salvează rezultatul pentru fiecare fișier și la final se combină toate rezultate într-o interfață ce ajută utilizatorul sistemului a înțelege ce a funcționat și ce a greșit și ce trebuie corectat.

Mai este folosit jQuery exhaustiv pentru a construi legătura dintre o vânzare și o cerere de despăgubire. Sistemul astfel refolosește codul deja existent prin intermediul cadrelor HTML („iframe") pentru a selecta vânzarea. La final, reies mai multe câmpuri obligatorii, ce se completează automat dacă a fost ales o cerere de vânzare. Din cauza problemelor rapoartelor de vânzare, această abordare oferea cea mai mare flexibilitate și un număr minim de linii de cod scrise.

O altă bună integrare cu „jQuery" vine din partea librăriei „Bootstrap Validator", un „plug-in" simplu de folosit pentru a adăuga elemente de validare pentru formulare. [29] Mulțumită directivelor Blade, am integrat ușor construirea unor câmpuri cu validare necesară, pentru a ajuta clientul să completeze cât mai corect formularul de despăgubire.

Am reușit astfel de a scurta timpul de dezvoltare și de a asigura o interfață unificată în modul de abordare a înfățișării aplicației, mulțumită acestor trei platforme.

2.8 Knockout.js

Knockout este o implementare de sine stătătoare a structurii șablon de „Model View View-Model". Structura se referă la o separare clară a datelor primite din exterior, a componentele și a datelor ce vor fi folosite pentru a fi afișate, dar și la prezența unui cod specializat de a gestiona relațiile dintre cele două (componente și interfață). [30]

A fost conceput de „Steve Sanderson", un angajat Microsoft, drept un proiect cu codul distribuit gratuit. [31] Mulțumită arhitecturii, Knockout simplifică relațiile complexe dintre componentele vizuale, ce ajută aplicația a fi mai echilibrată și rapidă.

Am folosit Knockout pentru a rezolva problema codului duplicat. În momentul actual, este folosit pentru a gestiona pozele încărcate de client și de a salva extern baza de date. Am ales să folosesc librăria această zonă unde pot apărea probleme, pentru a asigura performanța și eliminarea erorilor ce pot apărea în timpul procesului de programare. Codul este astfel modular și folosește șabloane pentru a lega componentele vizuale cu structuri de date. Knockout asigură astfel funcționalitatea identică pentru toate modulele folosite.

Sistemul de gestiune a pozelor este împărțit în cele patru categorii:

1. Facturi.
2. Pozele cu produsul avariat.
3. Foile cu termenii și condiții.
4. Raportul poliției, în caz de nevoie.

Pentru fiecare element, se construiește o funcție nouă, pentru că JavaScript nu are clase. Funcția respectivă gestionează identic modelul și vizualizarea acestuia prin păstrarea detașată de celelalte modele a datelor proprii.

Pentru a putea încărca asincron, câte o poză pe rând, în mediul distribuit de stocare a datelor (cloud), am combinat Knockout.js cu jQuery. Astfel, datele sunt primite de la server, în format JSON, folosind o cale specială, determinată la momentul construirii rutelor.

Knockout fiind o librărie de sine stătătoare, nu oferă o legătură strânsă între datele primite de la server și reprezentarea internă a structurii, dar permite extensibilitatea cu orice alt limbaj, pentru că se folosește doar de spațiul de nume („namespace") `ko`. Dar mulțumită librăriei ajutătoare jQuery și a metodelor scrise de mine, se asigură o performanță și un număr redus de probleme ce pot apărea, pentru orice sistem al oricărui client, ce poate vizita website-ul.

Un alt exemplu unde s-a folosit Knockout este la salvarea baza de date din cadrul aplicației. Knockout primește de la server o listă cu toate tabelele bazei de date, care se modifică dinamic.

Soluția de backup este accesibilă numai administratorului de sistem. Administratorul poate salva separat un anumit tabel sau integral baza de date.

2.9 Utilitare

Am folosit utilitare pentru a scurta din timpul de dezvoltare. Dacă pentru Laravel există consola Artisan, pentru gestionarea proiectului am folosit PhpStorm, un mediu de dezvoltare integrat oferit de cei de la JetBrains.

Composer împreună cu npm au fost folosite pentru interfețe cu formatul Excel și asigurarea completarea câmpurilor obligatorii de client.

Iar pentru lucrarea de licență am folosit Sublime Text 3 și LaTeX.

2.9.1 Google Analytics

Pentru a înțelege traficul venind, am ales să folosesc analizatorul gratuit oferit de Google. Astfel, aceasta oferă detalii despre interacțiunea paginilor web, de la cât timp petrece un client web pe o anumită pagină, la cele mai des folosite butoane sau cele mai des folosite drumuri (legături între mai multe pagini). [32] De asemenea, poți vedea ce a declanșat o eroare, ceea ce pentru un dezvoltator ajută la identificarea și rezolvarea problemelor din momentul în care apar, nu după ce primești un apel furios de la unul dintre fondatorii companiei.

Urmărind traficul pentru o săptămână, că punctul sensibil și cel mai des folosit era în momentul în care se căuta o cerere de despăgubire. De cele mai multe ori, se știa deja numărul cererii, trebuia doar să se acceseze panoul de căutare și să se introducă numărul în căsuță. Astfel am modificat panoul de navigare principal, regăsit pe fiecare pagină, prin introducerea unui câmp „ID Claim", ce oferă aceeași funcție regăsită în panoul de căutare a modulului cererilor de despăgubire.

Urmărind traficul, s-a mai observat durata lungă petrecută de clienți în momentul introducerii datelor cererii de despăgubire. Laravel a fost conceput cu un sistem avansat de gestionare a sesiunii, pentru a preveni abuzurile, dar aceasta interfera cu durata lungă în care clienții își completau probabil informațiile și apărea des o eroare a expirării sesiunii. Astfel am rezolvat problema apelând constant, prin jQuery, un punct al aplicației cu singurul rol de a actualiza sesiunea Laravel (declarând sesiunea activă). Clientul va putea în final să completeze toate datele, sesiunea rămânând activă.

2.9.2 PhpStorm IDE

PhpStorm este un mediu de dezvoltare comercial, oferit de JetBrains. Suportă completarea avansată a codului PHP 5.6 (și mai nou 7.0), dar și analiza sintactică, prevenirea erorilor. Suportă oficial și sistemul relațional de bază de date MySQL, stilul CSS, JavaScript și are puternice metode de restructurare a codului PHP. Integrarea cu modul de depanare „XDebug" îl ajută să ajungă în fața multor simple editoare de text.

Laravel a fost gândit cu cât mai puține referințe, cu cât mai mult cod ajutător pentru dezvoltator și o grămadă de resurse valabile pe diverse site-uri. Directivele Blade, introduse de platforma Laravel, sunt înțelese de analizatorul sintactic al mediului de dezvoltare PhpStorm.

Pentru a profita din plin de cunoștințele sintactice acestui mediu de dezvoltare, un pachet des întâlnit în PhpStorm când se construiește o aplicație folosind platforma Laravel este:
„Laravel PhpStorm IDE Helper" (`barryvdh/laravel-ide-helper`)

„Eloquent ORM" de cele mai multe ori PhpStorm nu poate înțelege de ce a fost apelată o metodă ce nu există - pentru că se va interpreta codul scris în metoda magică `__callStatic`, ce va traduce numele metodei statice într-un text, iar apoi într-o clasă ce va construi o cerere

SQL. Dar aici intervine „phpDoc”, sistemul preferat de adnotare a funcțiilor, metodelor și variabilelor PHP. Pachetul astfel introduce trei funcții ce pot fi apelate de dezvoltator prin intermediul consolei:

- `php artisan ide-helper:generate` - generare cod „phpDoc” pentru fațade.
- `php artisan ide-helper:models` - generare cod „phpDoc” pentru modele.
- `php artisan ide-helper:meta` - generare cod „phpDoc” pentru structura aplicației.

De asemenea, se integrează cu sistemul de gestiune a versiunii codului. Poate să și încarce automat modificările făcute pe server-ul de producție, folosind protocolul FTP (encriptat, desigur).

Asigură deci rapiditatea și fluența necesară reparării unor probleme critice și a dezvoltării codului concis, după ghidul dorit de stil.

2.9.3 Heroku

Pentru a minimiza impactul unei migrări, s-a folosit în cazul acestui proiect Heroku.

Heroku este o platformă distribuită ce oferă platforma sa ca un serviciu („PaaS”). Această platformă oferă următoarele avantaje:

- Servicii pentru mai multe limbaje de programare, printre care și PHP.
- Implementarea ultimelor modificări preluate de pe GitHub, sistemul de gestiune a codului ales în cadrul proiectului.
- Acces în cadrul aplicației la consola Artisan, pentru a rula migrările bazei de date, dar și a testelor automate.
- Testele automate pot fi rulate înainte de a se schimba versiunea activă a mediului de testare pe ultima versiune, pentru a se asigura că nu s-a schimbat o parte critică a aplicației.

Toate aceste motive fac din Heroku mediul meu de testare favorit, ce și-a văzut folosința în cadrul acestui proiect de gestionare a cererilor de despăgubire. Mulțumită ușurinței de a schimba variabilele de mediu, am putut să testez de fiecare dată când modificam codul sursă al aplicației schimbările.

Fiecare modul al aplicației a fost implementată și testată mai întâi în Heroku, iar apoi migrată în producție.

2.9.4 Composer

Composer este un utilitar pentru a gestiona referințele externe într-un proiect PHP. Permite dezvoltatorului să declare librăriile de care depinde proiectul, pentru a le gestiona (instala sau actualiza) automat. Spre diferență de un sistem care gestionează pachetele pentru un sistem de operare bazat pe Linux (precum „Yum” sau „Apt”), acesta instalează pachetele în folderul proiectului numit `„vendor"`, dar permite restrâns instalarea într-un spațiu `„global"`, a unor referințe. [33]

Laravel depinde de Composer pentru a gestiona referințele. Pentru a porni un nou proiect de Laravel, trebuie să-l instalezi folosind comanda:

```
composer global require "laravel/installer"
```

Am vorbit despre extensia mediului de dezvoltare PhpStorm prin pachetul „Laravel PhpStorm IDE Helper”. Lista completă a pachetelor folosite, în detaliu, în afară de Laravel, a acestui proiect este:

- `barryvdh/laravel-ide-helper` - extensia mediului de dezvoltare, vezi secțiunea în care prezint mediul de dezvoltare PhpStorm.
- `doctrine/dbal` - extensie ajutătoare pentru „Laravel PhpStorm IDE Helper”, pentru a afla toate interacțiunile cu modelele Eloquent.
- `laravelcollective/html` - extensia „Laravel - Forms & Html”, întreținută de „Laravel Collective”.
- `phpoffice/phpexcel` - extensie ce ajută citirea și scrierea fișierelor în Excel.
- `guzzlehttp/guzzle` - extensie ajutătoare pentru AWS S3
- `league/flysystem-aws-s3-v3` - principala extensie pentru integrarea Laravel – AWS S3
- `golonka/bbcodeparser` - librărie ce se ocupă de traducerea „BBCode – HTML” și viceversa.

După ce se actualizează lista și proiectele folosite în aplicație, Composer poate să ruleze script-uri, prin adăugarea următorului câmp obiect în `composer.json`:

```
"post-update-cmd": [
        "Illuminate\\Foundation\\ComposerScripts::postUpdate",
        "php artisan ide-helper:generate",
        "php artisan ide-helper:meta",
        "php artisan ide-helper:models -W"
]
```

Exemplul de mai sus generează codul ajutător pentru mediul de dezvoltare PhpStorm.

2.9.5 Laravel Elixir

„npm” este un utilitar de gestionare a librăriilor pentru limbajul de programare JavaScript. De asemenea, este utilitarul folosit implicit de mediul de dezvoltare „Node.js”. Este compus dintr-un client accesibil prin linie de comandă și o bază de date pe internet formată din pachete publice, numit „npm registry”. Registrul este accesat de client, iar pachetele valabile pot fi găsite și accesate prin intermediul website-ului npm.[34]

„gulp” este un utilitar găsit pe platforma „npm” ce are ca scop automatizarea elementelor ce necesită mult timp, precum compilarea și mutarea fișierelor dintr-o parte în alta. Acesta preferă codul peste configurație, ce asigură un sistem rapid de dezvoltare, fără a scrie în fișiere intermediare pe disc.

Laravel Elixir a extins interfața sa ușoară de folosit și atunci când vine vorba despre automatizarea cu „gulp”. Astfel, acesta știe despre cele mai des folosite procesoare de „CSS” (i.e: „SASS”) și JavaScript (i.e: „Webpack”). Folosind sintaxa de a concatena metode una după alta, Laravel Elixir ajută dezvoltatorul a defini propria structură.

Laravel Elixir mai este folosit și pentru gestionarea foarte ușor versionare fișierelor. Asigura de fiecare dată când un client vizitează orice pagină că fișierele de stil și cod sunt actualizate.

În cadrul aplicației, am transformat simplul fișier dat implicit de Laravel „`gulpfile.js`" într-o platformă elegantă. Algoritmul constă în următorii pași:

1. Citesc toate fișierele aflate la calea:

   `resources/assets/js/pages/*`

2. Construiesc pentru fiecare fișier în parte o variantă a sa minimizată și cu propriul nume (i.e: numele concatenat cu versiunea). De asemenea, construiesc pentru principalul stil (`app.scss`) și librăria de cod a aplicației (`app.js`) la fel varianta minimizată cu propriul nume.

3. La final, Laravel Elixir se ocupă de mutarea lor în fișierul public și modificarea referințelor prin fișierul JSON aflat la calea:

   `/public/build/rev-manifest.json`

Folosesc „Webpack", un utilitar ce combină mai multe fișiere JavaScript într-unul singur. Prin intermediul „npm", am instalat componentele necesare validării timpului, a afișării soluțiilor posibile și a diagramelor prin simplele comenzi:

```
// pentru sistemul vizual
require('./bootstrap');

// pentru validarea timpului
require('./components/timepicker.min');

// pentru afișarea soluțiilor posibile
require('./components/typeahead.min');

// pentru diagrame
require('chart.js');

// extensiile proprii
require('./extensions');
```

2.9.6 LaTeX

TeX este un utilitar pentru a redacta documente, conceput de „Donald Knuth". Programul preia un fișier pregătit și-l transformă într-un format ce poate fi tipărit în diferite moduri, cu diferite imprimante.

LaTeX este un set de macro-uri pentru TeX, conceput pentru a reduce din textul necesar formatării în pagină [35].

Folosind conceptul de la programarea „DRY" („Don't Repeat Yourself" - „Nu Te Repeta"), am împărțit fiecare capitol în fișierul său separat, cu fiecare fișier având la nevoie sub-directoare cu secțiunile importante.

Pentru a edita documentația licenței, am ales să folosesc „Sublime Text 3", pentru că este unul dintre cele mai versatile editoare de text în momentul actual. Folosind o extensie, m-a ajutat să pot compila întregul proiect, oriunde m-aș afla în cadrul lui, în formatul PDF („Portable Document Format"), ce poate fi deschis pe orice calculator, cu orice sistem de operare.

Folosind gândirea de proiect, am setat în „Sublime Text 3" folosirea unui font cu diacritice. De asemenea, am activat și descărcat un dicționar pentru Română.

Capitolul 3

Arhitectura software a aplicației

3.1 Rutina de apelare a programului

Pentru a recapitula tehnologiile folosite și pentru a putea intra în detalii despre cum funcționează aplicația per ansamblu, voi detalia cum interacționează un client cu acest sistem de gestiune a cererilor de despăgubire, bazat pe Laravel. Cererea pornește de la premiza accesării paginii scrise în regulamentul contractului „Mobile Protect” și parcurge pașii unui client web (ceea ce nu se observă când se încarcă pagina).

1. Clientul navighează spre `www.fandu.uk/claims`.
2. El este redirectat spre `https://fandu.info`, prin intermediul „header”-ului, pagina livrată fiind goală.
3. Folosind un sistem distribuit de nume de domeniu, presupunând că domeniul nu a fost descoperit până acum de client, se face cererea către CloudFlare. Presupunem că nu este un atac malițios, altfel ar fi fost oprit în acest punct și ar fi fost posibil obligat să rezolve o serie de întrebări pentru a continua. Depinde de severitatea atacului.
4. Aplicația Laravel se încarcă. Își identifică astfel mediul de lucru și variabilele definite.
5. Se conectează la baza de date MySQL și încearcă să identifice sesiunea clientului, în funcție de „cookie”-ul denumit „laravel_session”. Pentru că presupunem că este un client nou, nu va găsi o sesiune deja existentă. Nu este un client înregistrat.
6. Laravel observă rutele predefinite și încearcă să găsească o rută ce se potrivește căii, reținând contextul.
7. În acest moment, Laravel deține numele Controller-ului, numele metodei ce va fi apelate și toate denumirile scurte ale claselor intermediare ce trebuiesc validate pentru a ajunge cererea la metoda clasei, aceste clase mijlocii fiind denumite „middleware”.

   Un exemplu bun al unei clase middleware este „auth”, ce asigură un utilizator înregistrat, cu o sesiune activă.
8. Se construiește Controller-ul folosind calea determinată la pasul anterior și se actualizează (în cazul în care apare cod de adăugare unei clase „middleware” în interiorul metodei ce construiește obiectul) lista de „middleware”.
9. Se construiește și se validează fiecare clasă „middleware”, iar în cazul unui eșec, nu se va continua mai departe.

10. Se apelează metoda clasei asigurându-se prin intermediul unui obiect ce rezolvă legăturile dintre obiectele dorite și cele existente trimiși ca parametrii metodei Controller-ului. Se va primi un răspuns.

11. În cazul în care răspunsul nu este o directivă Blade, se va afișa sau va redirecționa clientul web. Altfel, se va compila directiva Blade dacă nu există fișierul deja compilat, sau dacă timpul ultimei modificări directivei e mai recent decât timpul ultimului rezultat al compilării.

3.2 Baza de date

Tabelele din baza de date sunt denumite cu pluralul obiectului la care se referă, folosind în loc de spații sublinia „_”. Puteți observa în detaliu diagrama bazei de date la figura 3.1

Aceasta este salvată o dată pe săptămână și conține datele despre:

1. `claims` - Cererile de despăgubire.
2. `sales` - Vânzări, cu legăturile de cheie primară (una-la-multe) la următoarele tabele:
   (a) `products` - Produse
   (b) `assurances` - Asigurări
3. `decisions` - Decizii, cu legătura circulară folosind tabelul `decisions_decisions`.
4. `chat_claims` - Mesajele trimise în cadrul cererilor între utilizatorii sistemului, ce depinde de tabelul de cereri.
5. `messages` - Mesajele trimise în cadrul cererilor între client și utilizatorii sistemului, ce depinde de tabelul de cereri.
6. `migrations` - Migrările schemei bazei de date, menținut de Laravel.
7. `users` - Utilizatorii aplicației, cu parola encriptată.
8. `photos` - Legăturile între locația pozelor și cererea, respectiv categoriilor lor, ce depinde de tabelul de cereri.

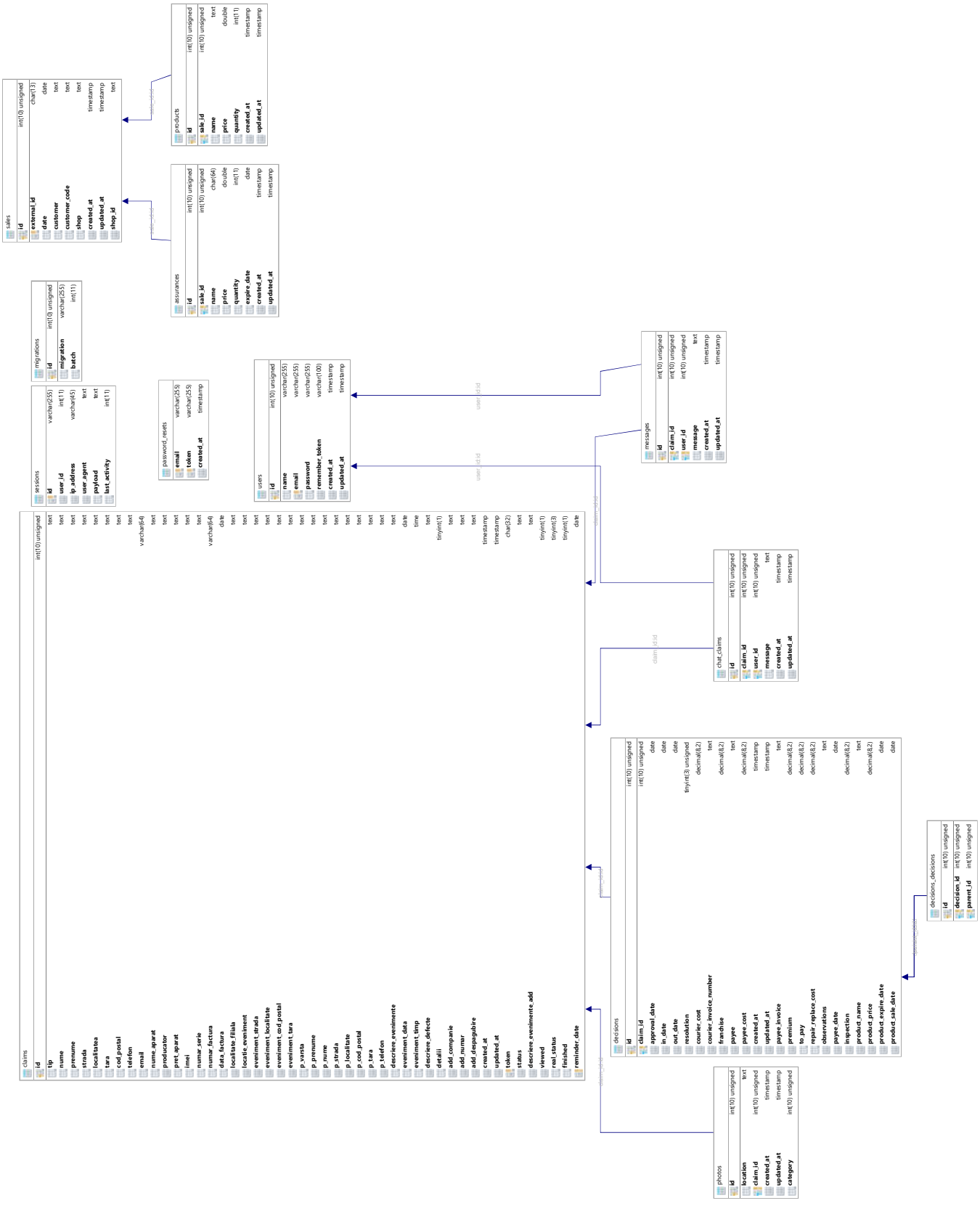


Figura 3.1: Diagrama bazei de date

3.3 Structura Model-View-Controller

Revenind la definiția arhitecturii, „Model-View-Controller” se bazează pe separarea puterii celor trei componente definitorii pentru o aplicație: Modelul, View-ul, respectiv Controller-ul.[12]

Modelul este componenta centrală a șablonului. Ea se ocupă de datele aplicației, iar mulțumită Laravel Eloquent, face legătura dintre baza de date și aplicație.

Modelele sunt gestionate de Controller, ce acceptă datele de la utilizator și le transformă în comenzi pentru model și/sau View.

View-ul este orice metodă de vizualizare a informației unui Model - în cazul nostru directivele Blade.

Laravel construiește astfel aplicații web bazate pe arhitectura „Model-View-Controller”. Deci și această aplicație respectă arhitectura.

În cadrul aplicației următoarele căi sunt notabile:

- Model - `/app/`
- View - `/resources/views/`
- Controller - `/app/Http/Controllers/`

3.3.1 Controllers

În afară de separarea conceptuală a arhitecturii MVC, Laravel se folosește și de un sistem de gestionare a rutelor aplicației. [37]

Spre deosebire de alte librării precum CakePHP, Laravel nu pune mult accent pe numele Controller-ului. Ci mai degrabă se folosește de un fișier principal unde se specifică legătura dintre un URL și un Controller, ce se regăsește în cadrul proiectului Laravel la:

`/routes/web.php`

În cazul proiectului de gestionare a cererilor de despăgubire, se pune foarte mult accent pe securitatea datelor. Din acest motiv, printre primele linii ale fișierului, regăsim

```
URL::forceSchema("https");
```

ce indică schemei de rutare Laravel rescrierea oricărui link a folosi HTTPS, pentru a menține legătura encriptată cu clientul. În cazul în care clientul cere varianta normală (HTTP) a unei pagini, el este redirectat automat spre varianta HTTPS. Din acest motiv nu am activat în cadrul „CloudFlare” opțiunea de „Automatic HTTPS rewrites” (rescrieri automate HTTPS), pentru că deja se ocupă de asta librăria Laravel. În cazul în care aș fi activat opțiunea respectivă, s-ar fi bătut cap în cap cu implementarea Laravel. Nu am descoperit cauza, dar încă investighez.

Un mare avantaj al sistemului de rutare este gruparea rutelor prin includerea automată a unor „middleware”, precum sistemul de asigurare a unui utilizator cu o sesiune activă, dar și scurtarea căilor Controller-lor sau căilor. Se poate observa prin intermediul codului:

```
Route::group([
        'middleware' => 'auth',
        'prefix' => 'admin',
        'namespace' => 'Admin',
], function () {
        // funcții aici
});
```

- prefixarea căilor cu „admin”
- în numele spațiului claselor „Admin”
- ce sigur va conține „middleware”-ul „auth”

Laravel în afară de cererile simple poate combina mai multe dintre cererile simple într-o „resursă”. Cererile simple pot fi:

- GET - citește una sau mai multe informații.
- POST - salvează (de obicei prin adăugare) o informație (de obicei nouă).
- PUT / PATCH - modifică de obicei o informație deja existentă.
- DELETE - șterge informația.

O „resursă” în Laravel atribuie rutele „CRUD” (Create / Read / Update / Delete) unui Controller cu o singură linie de cod:

```
Route::resource('obiecte', 'ObiectController');
```

Și mulțumită consolei Artisan, se poate construi la fel de rapid și codul necesar unui astfel de Controller, prin simpla comandă:

```
php artisan make:controller ObiectController --resource
```

Următoarele acțiuni sunt definite și gata de a fi modificate de dezvoltator pentru a fi gestionate:

Verb	URI	Acțiune	Nume rută
GET	`/obiecte`	index	obiecte.index
GET	`/obiecte/create`	create	obiecte.create
POST	`/obiecte`	store	obiecte.store
GET	`/obiecte/{obiect}`	show	obiecte.show
GET	`/obiecte/{obiect}/edit`	edit	obiecte.edit
PUT/PATCH	`/obiecte/{obiect}`	update	obiecte.update
DELETE	`/obiecte/{obiect}`	destroy	obiecte.destroy

În cadrul proiectului, am următoarea structură a directoarelor controalelor:

- Folderul implicit:
  - `MessagesController.php` - gestionează mesajele trimise între client și operatorul cererii de despăgubire.
  - `HelperController.php` - ajută la testarea sistemului de mail și menținerea sesiunii clientului.
  - `ClaimsController.php` - construiește cererea de despăgubire, afișează pagina principală și clauza de confidențialitate.
- Folderul `Json`:
  - `ImagesController.php` - gestionează încărcarea pozelor asincron de către client sau operator spre soluția distribuită de stocare a datelor — Amazon Web Services S3.
- Folderul `Admin`:

– Folderul `Json`:
  * `SearchController.php` - gestionează încărcarea rezultatelor parțiale asincron de către operator.
– `ClaimsController.php` - gestionează vizualizarea, modificarea și finalizarea cererilor de despăgubire. În cazul în care se găsește în parametrul GET (ce urmează dupa „?" în cadrul URL-ului) `id`, se va încerca redirecționarea spre cererea de despăgubire cu acel `id`. Dacă este aplicată o căutare, afișează toate rezultatele, altfel paginează câte 100 cereri pe pagină folosind Eloquent.

  În momentul în care se salvează o cerere noua de despăgubire, se trimite un mail cu datele create către căsuța principală a operatorilor sistemului și către client. El este rugat să-și verifice și căsuța de mail, iar cât mai curând să încarce toate pozele necesare completării dosarului său.

  Sistemul de căutare poate căuta în:
  * Număr de telefon.
  * Nume.
  * Prenume.
  * Model telefon.
  * Email.
  * O dată de început a facturii.
  * O dată de sfârșit a facturii.
– `DecisionController.php` - gestionează vizualizarea, modificarea și posibila șterge a deciziilor cererilor de despăgubire. În cazul în care există o căutare, acesta caută și în cerere, și în decizie, folosind următoarele câmpuri:
  * Număr de telefon.
  * Nume.
  * Prenume.
  * Model telefon.
  * Email.
  * O dată de început a înregistrării cererii.
  * O dată de sfârșit a înregistrării cererii.
  * Numele produsului conform deciziei.

  În cazul creării unei noi decizii în funcție de o cerere de despăgubire, se încearcă mai întâi găsirea automată a vânzării asociate după numărul facturii. Dacă căutarea nu reușește, atunci se va indica utilizatorului necesitatea unei construirii unei legături manuale.

  Pentru a salva o legătură între o cerere de despăgubire și o decizie, trebuie să existe următoarele câmpuri completate valid, ce nu trebuie să fie vide:
  * `claim_id` - id-ul cererii de despăgubire.
  * `product_name` - numele produsului de pe factură.
  * `product_price` - prețul produsului de pe factură.
  * `product_premium_price` - prețul asigurării.
  * `product_sale_date` - data vânzării produsului.
  * `product_expire_date` - data expirării asigurării.

  Dacă se salvează cu succes decizia, ea va apărea asincron, integrată în pagina de vizualizării a cererii de despăgubire, în partea de jos a paginii.

– `HomeController.php` - gestionează o interfață simplă cu două rapoarte, împreună datele de contact personale, în cazul în care apare vreo problemă.

  Sistemul arată precum în figura 5.7.

– `ImportController.php` - gestionează sistemul de introducere de date prin fișierele Excel în sistem.

  Acesta construiește automat un adaptor ce preia fișierul Excel și afișează un răspuns, de obicei printr-un View. Este împărțit în două categorii de fișiere — cele standarde, despre care s-au discutat la momentul definitivării modulelor și cele personalizate pentru fișierele apărute pe parcursul dezvoltării.

  Această diferențiere apare mulțumită notificării trimise pe email în momentul salvării cu succes a datelor din fișierele standard.

  Sistemul arată precum în figura 5.12(a).

– `ReportsController.php` - gestionează sistemul de rapoarte. Inițial scris astfel încât dintr-o singură metodă să reiasă un raport, acum funcționează modular și necesită cod asincron JavaScript pentru a scoate un raport.

  Sistemul a fost conceput astfel din cauza numărului imens de date ce trebuia să treacă prin modulul ce transformă datele în fișierul de format Excel. Fiecare raport are o metodă ce decide dacă mai are nevoie de pași suplimentari și deci de încă o cerere asincronă, pentru a folosi apoi la final, la pasul de combinare, toate datele din fișiere pentru a scoate raportul în formatul dorit. Avantajul folosirii fișierelor temporare și a mai multor apeluri asincrone este de a face interfața cu utilizatorul mai plăcută și în același timp de a reduce din consumul de memorie.

  Pentru a arăta interactiv că fișierul s-a descărcat cu succes, setez un „cookie” atunci când descarcă fișierul utilizatorul, iar în pagina de rapoarte aștept apariția acestui „cookie” pentru a semnala finalizarea descărcării. După ce l-am identificat, îl șterg setându-l să expire pe „1 Ianuarie 1970 00:00:01 GMT”.

  Sistemul arată precum în figura 5.25.

– `SalesController.php` - gestionează existența, modificarea, afișarea și căutarea vânzărilor introduse în sistem.

  Funcția de căutare, prezentă în figura 5.14(b) se referă la:

  * `Product` - Numele produsului căutat.
  * `Customer` - Numele clientului ce a cumpărat produsul.
  * `Date Start` - Data de început de când s-ar fi putut vinde produsul.
  * `Date End` - Data de sfârșit de când s-ar fi putut vinde produsul.
  * `Date Added` - Data când a fost importat fișierul (acest câmp, conform statisticilor Google Analytics, este degeaba și va fi scos într-o variantă ulterioară, cel mai probabil).
  * `External ID` - Numărul facturii.

– `ProductsController.php` și `AssurancesController.php` - gestionează doar construirea și ștergea produselor, respectiv asigurărilor, din interfața ce modifică vânzările. Nu au niciun alt rol.

– `ExportController.php` - gestionează compunerea unor fișiere .CSV cu toate datele, fără a fi trecute printr-un filtru, a unui tabel din baza de date.

  A fost folosit într-un moment de „criză”, unde era nevoie obligatoriu de a putea scoate datele din sistemul informatic pentru a trimite un raport rapid, când nu era gata sistemul de raportare.

  Nu are interfață grafică.

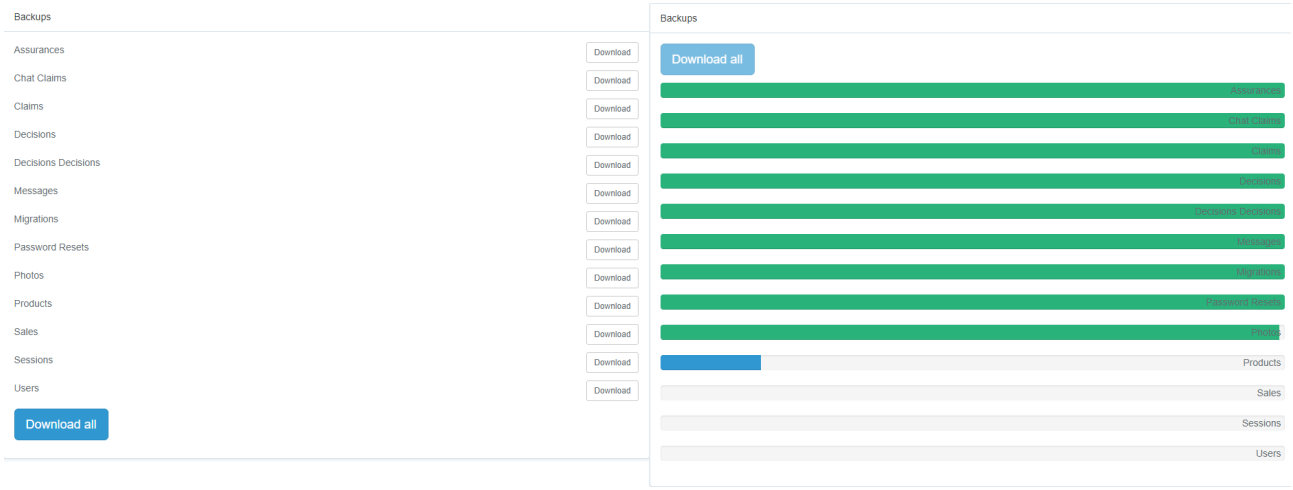


Figura 3.2: Sistemul de salvare a bazei de date

– `BackupController.php` - gestionează sistemul actual de salvare a bazei de date de orice operator al companiei. Acest sistem necesită din cauza acelorași probleme de memorie mai multe apeluri, pentru a construi arhiva cu toate datele tabelelor bazei de date. Se folosește de punctul de colectare:

`storage_path("app/*.sql")`

și comprimă tot ce găsește obținut din apelurile asincrone din JavaScript anterioare pentru a compune o arhiva finală „.tar.gz”.

Pe serverul de producție este foarte lentă compunerea arhivei finale, așa că voi considera scoaterea comprimării arhivei și transmiterea folosind unei arhive „.zip”. Nu pot să folosesc și algoritmi de comprimare a datelor pentru arhivele „.zip” deoarece s-a adăugat abia în PHP 7 opțiunea.

În prima parte a figurii 3.2 este interfața ce permite salvarea individuală a tabelelor. În momentul apăsării butonului de „Download All”, o să arate precum în a doua parte a figurii.

– `SettingsController.php` - gestionează

- Folderul `Auth` este cel construit implicit de Laravel atunci când construiește codul de a autentifica utilizatorii.

  O mică modificare ce s-ar putea să treacă neobservată este schimbarea controller-ului `RegisterController` de a fi folosit împreună cu grupul algoritmului 3.3.1, ceea ce indică puterea de a adăuga utilizatori aparține doar utilizatorilor de sistem.

  De asemenea, singurul mod prin care se poate înregistra un utilizator nou este prin cunoașterea adresei specifice Controller-ului:

```
Route::get('register', [
        'middleware' => 'auth',
        'as' => 'register',
        'uses' => 'Auth\RegisterController@showRegistrationForm
']);
Route::post('register', [
        'middleware' => 'auth',
        'as' => 'register.post',
```

```
        'uses' => 'Auth\RegisterController@register'
    ]);
```

Pentru a nu fi blocați pe dinafară, se adaugă mereu un utilizator în baza de date printr-o migrare.

3.3.2 Models

Baza noastră de date relațională conține multe relații. Spre exemplu, cererea de despăgubire poate că este legată la o decizie. Eloquent ORM ajută gestionarea și lucrul cu aceste legături foarte ușor și suportă majoritatea legăturilor, printre care:

1. One To One
2. One To Many
3. Many To Many

Relațiile Eloquent sunt definite ca metode în clasele noastre. Precum modelele în sine, acestea sunt folosite precum generatoare de sistem. Definind relațiile precum funcții ajută la propagarea metodelor și concatenarea mai multor constrângeri, precum pozele facturii determinate de această metodă aflată în clasa cererii:

```
public function factura_photos()
{
        return $this->hasMany('App\Photo')
                ->where('category', Photo::FACTURA);
}
```

În acest caz, pentru că fotografie se poate afla în oricare dintre cele patru categorii, se pune constrângerea ca ea să se afle în categoria facturii. Punându-se în legătură mai multe poze la un singur obiect, această relație „One To Many” are nevoie de câmpul `claims_id` să fie prezent în coloana tabelei de poze, conform convenției Laravel. [38]

Acum că putem să accesăm toate pozele unei cereri, putem defini și relația inversă. Schimbând registrul de la poze la decizii, putem accesa având decizia cererea din care provine. Pentru asta, se folosește următoarea metodă în clasa deciziei:

```
public function claim()
{
        return $this->belongsTo('App\Claim');
}
```

La fel, pentru că respectă convenției Laravel – se regăsește câmpul `claims_id` – se va completa și încărca modelul Eloquent a cererii, atunci când va fi apelată metoda.

De asemenea, toate metodele folosite sunt „puturoase”. Ele nu sunt interpretate și nu rețin rezultatul până când nu au fost chemate cel puțin o dată.

Presupunând că am dori să avem un număr rotund în loc de cel rotunjit cu virgulă mobilă cu o simplă precizie, am putea să definim un accesor. Un accesor cu numele `getFirstNameAttribute` va fi apelat în momentul apelării aflării valorii proprietății `first_name`. Cel mai bun exemplu este calcularea valorii de plată nete din modelul „Decision”:

```
public function getNetPayableAttribute()
{
    if ($this->resolution == Decision::REPAIRED) {
```

```
        return $this->repair_replace_cost - $this->franchise
            + $this->courier_cost;
    }
    if ($this->resolution == Decision::REPLACED) {
        return $this->payee_cost - $this->franchise
            + $this->courier_cost;
    }

    return "N/A";
}
```

O altă problemă delicată ce ar putea apărea într-un sistem de legare relațională între obiecte și reprezentările bazei de date constă în câmpurile ce pot fi atribuite „la grămadă”. O vulnerabilitate „la grămadă” apare atunci când un utilizator mai adaugă să spunem un câmp „Administrator” când se înregistrează și-l setează pe adevărat, ce este trimis apoi în metoda de creare a relației tabel – obiect, escaladându-și astfel drepturile.

Pentru a nu fi vulnerabili, putem specifica doar variabilele ce pot fi atribuite „la grămadă”. Asta împiedică atribuirea altor câmpuri existente, asupra modelului Eloquent. Funcționează incluzând următoarea linie:

```
protected $fillable = ['nume',  'elemente', 'tabel'];
```

Revenind la proiectul nostru, am mai definit o metodă numită `getReadable`, pentru a afla ce câmpuri sunt obligatorii a fi completate, pentru a construi un nou model.

Următoarele modele au cod propriu interesant:

- `Decision` - Se folosește de o metodă „Factory” (ce produce obiecte gata pregătite) pentru a construi obiecte din vânzări sau din elemente generice.

  De asemenea se mai calculează prin intermediul metodelor `„remaining"` și `„updateToPay"` sumele rămase, respectiv de plătit.

- `Photo` - Se folosește o metodă „Factory” preia extensia și construiește un șir aleator din 32 de caractere, imposibil de ghicit.

  După aceea se atribuie modelului categoria și cererea din care face parte.

  În final, se încarcă pe S3 și se salvează în baza de date.

  De asemenea, este extinsă metoda prin care se șterge modelul, asigurându-se ștergerea sa și de pe AWS S3.

- `Sale` - În afară de metoda ce compune un nou generic sale, ce nu mai e folosită, acest model mai oferă și două metode pentru calcularea prețului total al produselor și cantității lor.

  De asemenea, mai oferă încă două metode pentru calcularea prețului total al asigurărilor și a cantităților lor.

3.3.3 Views

Gândirea pe care am urmat-o dezvoltând aplicația de gestiune a cereri de despăgubiri, când vine vorba de interfața grafică, a fost bazată pe module.

De menționat că multe directoare conțin un director numit „includes”. Acest director conține directive Blade parțiale ce sunt utilizate pe parcursul afișării, modificării sau interacțiunii cu paginile principale.

Astfel, interfața e împărțită în mai multe directoare:

- `vendor` - elementele de paginație extrase din context, ce permit păstrarea parametrilor „GET” în momentul folosirii paginației.

  Am ales să fac modificarea implicită de a adăuga doar câmpurile folosite la momentul respectiv pentru că era singurul scenariu în care se puteau pagina rezultatele.

- `settings` - elementele vizuale ce afișează setările și interfața de a salva datele aplicației.

- `sales` - elementele ce permit vizualizarea, crearea, modificarea și ștergea produselor și asigurărilor, în cazul coruperii datelor.

- `reports` - elementele de paginație ce permit construirea rapoartelor.

  Se folosesc de Laravel Elixir pentru a adăuga codul scris în fișierul JavaScript `reports.js` în pagină.

  Astfel se asigură viteza optimă de distribuire a resurselor și funcționalitate cu tipurile de raport.

  Integrarea cu Knockout se face implicit prin declararea includerea din componenta vizuală de șablon a aplicației a fișierului `app.js`.

- `newclaim` - elementele de paginație ce sunt vizualizate prima oară de clienții aplicației.

  Directorul `includes` conține împărțit pe componente toate marile arii acoperite de cererea de despăgubire, printre care și un parțial inclus în mail-ul trimis mai departe. Acest parțial Blade se referă la actele necesare a fi încărcate în a doua parte a aplicației. (partea de încărcare imagini)

- `messages` - elementele de paginație ce sunt incluse cu fiecare tranzacție, ce asigură un canal de comunicare între client și operatorul aplicației.

- `mail` - Înăuntrul acestui director se află fișierul `layout.blade.php`, ce conține stilul general valabil tuturor email-urilor trimise.

  De asemenea include pentru fiecare tip de obiect o acțiune desăvârșită ca nume.

- `layouts` - conține singurul fișier `app.blade.php`, de care depinde toate celelalte View-uri.

  În cazul în care utilizatorul nu este înregistrat, nu îi se arată bara de navigație în cadrul aplicației de administrare.

  De asemenea, conține referințe la Laravel Elixir, mai ales la stilul și codul precompilat și optimizat de „gulp” (pentru că până la urmă, gulp lucrează cel mai mult).

  În afară de aceste mici modificări, împreună cu Google Analytics și metoda specială de a picura câte o cerere de actualizare a sesiunii la câteva minute, tranformă această pagină a aplicației într-un adevărat utilitar și arată funcționalitatea pe deplină a directivelor Blade.

- `import` - elementele de paginație ce sunt vizualizate de cei ce doresc să introducerii date în sistem.

  Directorul `steps` conține împărțit pe componente toate rezultatele posibile introducerii standard de date, dar și celor modificate.

  Directorul `types` conține rezultatele introducerii datelor în sistem, în formatul standard de date.

- `errors` - elementele de paginație ce vor fi afișate în caz de erori cu numărul respectiv.

- `decisions` - elementele de paginație extrase pentru decizii.

  Conține și o metodă eficientă și rapidă, folosind jQuery , de a sorta informațiile prin apăsarea butoanelor colorate din capul tabelului.

- `components` - elementele de paginație extrase din context, ce-mi permite să scriu mai puțin cod pentru a folosi panourile din Bootstrap.

- `claims` - elementele de paginație extrase pentru cereri.

  Conține și o metodă eficientă și rapidă, folosind jQuery , de a sorta informațiile prin apăsarea butoanelor colorate din capul tabelului.

  Directorul `includes` conține componenta de căutare și componenta de modificare a cererii de despăguipre dacă persoana era autorizată (înregistrată și cu o sesiune activă).

- `auth` - elementele originale de înregistrare, logare și resetare parolă.

- `home.blade.php` - Pagina de introducere când intri în sistem.

.

3.3.4 Funcții ajutătoare

Funcțiile ajutătoare sunt pentru a minimiza timpul dezvoltatorului de lucru. Fișierele astfel folositoare sunt:

1. `FormHelper` - Utilitar ce înregistrează o directivă Blade de compunere, ce sunt traduse în simple instrucțiuni de cod PHP. Am scris astfel propria directivă de a construi automat codul necesar afișării, folosind HTML și librăria vizuală „Bootstrap", a unui câmp dintr-un formular, cu nume, tip și validare proprie:

   ```
   @input(['p_varsta', ['Varsta', true]])
   // unde 'p_varsta' - id-ul câmpului.
   // unde 'Varsta' - denumirea câmpului.
   // unde 'true' - necesitatea completării câmpului.
   ```

2. `Helpers` - Funcțiile ce traduc o variabilă în ceva mai util, precum un șir ce poate fi citit mult mai ușor, dar ce și traduc starea actuală a unei cereri sau decizii în clasa specială CSS.

3. `TemporaryFiles` - O clasă ce se ocupă de gestiunea fișierelor temporare.

   Este necesară pentru împărțirea în mai multe fișiere a unor cereri mari de date, pentru a nu intra în probleme de memorie.

3.3.4.1 Request-uri

O cerere, presupunând că a fost completată corect pe partea clientului, trebuie să fie verificată și pe partea server-ului.

În loc să se verifice în fiecare metodă de fiecare dată validitatea cererii, se poate abstractiza și a se face referire la o clasă de tipul Request din Laravel.

Următoarele clase de tip Request apar, nu sunt goale și sunt folosite în proiect:

1. `ClaimRegisteredRequest` - verificarea existenței unui IMEI înregistrat în ultimele 24 ore.

2. `StoreClaim` - asigură toate câmpurile unei cereri respectă anumite reguli, declarate folosind sintaxă specială Laravel.

3.3.4.2 Mediul de dezvoltare

Pentru a folosi o aplicație bazată pe Laravel, trebuie să ai configurate anumite servicii, precum tipul de conexiune la o bază de date, ce sistem de mail folosești sau locație aplicației tale pe internet.

Mulțumită fișierului „`.env`", nu mai trebuie a te complica cu setarea variabilelor globale sau a variabilelor de mediu, acestea vor fi automat preluate în cazul în care nu există variabila de mediu, din acest fișier.

O importantă precizare este necesitatea unei chei din acest fișier, „`APP_KEY`", ce trebuie să existe.

Această cheie este folosită într-un algoritm de încriptare pentru a face sesiunea clienților și a utilizatorilor cât mai sigură.

3.4 Rapoarte

Pentru a asigura compatibilitatea cu sistemele folosite de personalul companiei, s-a dorit un ușor acces pentru rapoartele extrase. S-a decis la începutul dezvoltării folosirii sistemului „.csv", a valorilor delimitate de virgulă. Mulțumită suportului nativ a PHP-ului pentru aceste valori, s-a construit rapid o clasă ajutătoare ce prin intermediul platformei Laravel salvează rapoartele.

Mulțumită pachetului de dezvoltare `phpoffice/phpexcel`, am putut integra, la finalul construirii modulelor, un sistem ce salvează direct în formatul Excel. Apăruse, folosind o implementare naivă a algoritmului de salvare în Excel, o problemă de timp. Consuma din ce în ce mai multe resurse și pentru că numărului de linii urma să crească, sistemul ar fi rămas fără memorie.

Până la urmă am ales să-l optimizez pentru a reduce din complexitatea programului, făcând următoarele modificări:

1. Majoritatea vectorilor în cadrul aplicației ce interacționează cu sistemele de raportare au fost înlocuite cu `SplFixedArray`. Astfel se reduce memoria și se optimizează accesul la elementele vectorului, prin declararea de la începutul construirii obiectului a dimensiunii vectorului.

2. S-a activat folosirea unei baze de date auxiliare pentru „PhpExcel", pentru a reduce consumul memorie.

3. Atunci când se trimit informațiile către librăria de „PhpExcel", se trimite tot vectorul deodată. Înainte, se trimitea fiecare element, rând cu rând, coloană cu coloană.

4. Împărțirea memoriei ce trebuia gestionată prin apeluri secvențiale și la final concatenarea lor. Această metodă este folosită des în sistemele distribuite – „Map-Reduce":

   (a) Se aplică metoda de calcul dorită informației împărțită în mod egal — „Map".

   (b) Se combină la final toate rezultatele parțiale în cel final — „Reduce".

Soluția finală astfel se poate încadra în 256MB RAM, pentru un tabel Excel cu peste 33000 linii, iar pentru sistemul găzduit pe xServers, ce are limita de memorie 128MB, se poate calcula un întreg an.

3.4.1 Implementarea platformei de rapoarte Excel

Pentru gestionarea platformei, am construit următoarele clase:

- `Facade` - Clasă ce conține doar metode statice, ce ajută introducerea unui raport în sistemul de gestiune a cererilor și despăgubirilor. Se folosește de două „trait"-uri, `AdaptorFacade` și `OutputFacade`, ce conțin metodele statice pentru a scoate, respectiv a introduce rapoartele din sau în sistem.

  De menționat ar fi că în funcție de șablonul dorit, clasa `Facade` se ocupă de construirea clasei respective, prin următorul cod:

  ```
  $adaptorName = 'App\\Excel\\Adaptors\\' . ucfirst(
          camel_case($shorthand)
  ) . 'Adaptor';

  return new $adaptorName;
  ```

- `FileData` - Clasă scrisă inițial ce traduce un vector de vectori, unde prima linie reprezintă capul de tabel, într-un fișier „.csv" sau în ââcazul în care se dorește o reprezentare `readable`, într-un tabel format din caractere ASCII.

- `FileDataExcelAdaptor` - Clasă ce primește ca un constructor o altă clasă `FileData`, ce traduce atunci când este folosită ca un șir de date ce reprezintă echivalentul fișierului Excel formatat.

- Directorul `Adaptors`, ce conține următoarele adaptoare, în funcție de nevoia fișierului de importat

  - `DailyAdaptor` - Importul datelor din fostul principal Excel SpreadSheet. S-a dovedit o provocare construirea unui algoritm ce ține cont de toate posibilitățile câmpurilor, dar mulțumită clasei ajutătoare `ExcelHelper`, o dată ce am definitivat datele ce trebuiesc scoase din acest fișier cu operatorii sistemului, am reușit să trec peste și acest obstacol.

    Această clasă construiește și cereri și deciziile asociate cererilor. Se asigură astfel continuitatea aplicației din punctul de vedere al rapoartelor și existența datelor în cazul în care va fi nevoie de o căutare în anii anteriori.
  - `ExcelAdaptor` - Interfață implementată de toate adaptoarele.
  - `ExcelHelper` - clasă utilitară ce spune coloana unde se află un text, dat de prima linie a raportului de introdus. Astfel se putea definitiva o structură aleatoare când vine vorba de organizarea coloanelor, dar ce mereu va putea fi introdusă în sistem mulțumită căutării după text.

    S-a aplicat o normalizare a valorilor numelor coloanelor primei linii a raportului, prin scoatere spațiilor și a semnelor de punctuație des folosite, dar și transformarea șirului pentru a conține doar litere mari.
  - `AltexGalaxyAdaptor` - Principalul import de rapoarte.

    Acesta se folosește exhaustiv de clasa ajutătoare `ExcelHelper` pentru a asigura datele vitale sunt introduse corect din raport în aplicație.
  - `SalesImportAdaptor` - Clasă cu rol secundar, ce introduce datele din rapoartele existente până în momentul actual a vânzărilor produselor.
  - `OldAdaptor` - Clasă cu rol secundar, ce introduce doar IMEI-ul din vechile rapoarte trimise de soluția anterioară în cadrul cererilor de despăgubire.

- Directorul `Outputers`, ce conține următoarele rapoarte:
  - `StatisticsOutputer` - Pentru statistici rapide (câte cereri de despăgubire, decizii au fost create în perioada respectivă de timp.)
  - `DecisionsOutputer` - Pentru rapoartele cu toate deciziile plătite în perioada de timp specificată.
  - `ProductsOutputer` - Pentru numărul de asigurări, grupate în funcție de tipul de asigurare și prețul ei, în perioada respectivă.
  - `SalesOutputer` - Pentru numărul vânzărilor asigurărilor grupate în funcție de magazinul ce a făcut vânzarea.

    Se mai adaugă și perioada când s-a vândut primul și ultimul produs.
  - `AbstractDateOutputer` - Clasă abstractă ce toate rapoartele momentan „sub-clasează”, pentru că toate rapoartele sunt delimitate de data de început și sfârșit a raportului.
  - `ExcelOutputer` - Interfața implementată de toate clasele din acest director.
  - `YearlyOutputer`, împreună cu clasa suport `NegativeSale`, scoate un raport detaliat despre produsele ce încă sunt asigurate, împreună cu posibila cerere de despăgubire asociată.

    Momentan toate asocierile cu cererile de despăgubire trebuie verificate manual, deoarece nu mai sunt corelate vânzările de cererile de despăgubire, iar algoritmul euristic nu are valabil numărul facturii.
  - `MonthlyOutputer` - Generează un raport asemănător clasei `YearlyOutputer` adăugând câmpul lunii.

Capitolul 4

Probleme apărute în timpul dezvoltării aplicației

Conform metodologiei de lucru „Agile”, ce se concentrează pe modificări incrementale a cerințelor și a livrării rapide unei baze funcționale [36],

.. Chiar dacă am fost singurul dezvoltator al aplicației și nu am avut un lider de echipă sau o echipă, am vrut să câștig experiența lucrului rapid și să mă folosesc de un canal deschis de discuții cu persoanele cărora le voi dezvolta aplicația.

Doresc să fac acum distincția între cele trei obiecte în jurul cărora a fost gândită baza de date, înainte de a vorbi despre structura aplicației.

Baza de date a fost gândită în jurul a trei entități: cererea, decizia și vânzarea.

Cererea este scrisă de clientul ce are produsul avariat și îl dorește înlocuit sau reparat. În cerere trebuiesc completate următoarele date:

- Date personale.
- Descrierea avariei și a evenimentului ce a produs avaria.
- Existența sau nu a altei polițe de asigurare.

Este legată relațional de imagini încărcate pentru:

- Paginile contractului.
- Raportul poliției în caz de furt agravat.
- Copia bonului cu ștampilă.
- Copia actului de identitate.
- Pozele produsului.

Aceasta poate să fie ori respinsă, ori acceptată, construindu-se astfel o decizie.

Decizia este construită în momentul acceptării cererii de despăgubire. Conține datele relevante pentru rapoartele de vânzare, precum:

- Prețul reparației.
- Fostele decizii asociate. (i.e: foste reparații)
- Franciza.
- Data livrării produsului în service.

- Costul inspecției și a curierului.

Aceasta, pentru o bună perioadă de timp, era legată obligatoriu de o decizie și de o vânzare.

De la un punct, când trebuia să se importe toate vechile vânzări din raportul păstrat în Excel, s-a luat decizia de a se decupla aceste două entități. Se construiau până în acel moment vânzări generice, ce nu existau. Practic, existau într-un raport vânzările respective, dar nu puteau fi găsite din cauza lipsei informațiilor din partea rapoartelor de la Altex / Media Galaxy. Nu se putea lega IMEI-ul sau numărul facturii, din cauza modului de organizare a organizației. Aveau nevoie urgentă de rapoartele de vânzare, așa că am fost anevoios de acord să decuplez legătura dintre decizie și vânzare.

Mai nou decizia conține numele produsului, prețul, data achiziționării și perioada asigurată, deși informațiile tot se citesc din tabelul de vânzări.

Vânzarea conține:

- Numele produsului.
- Prețul produsului.
- Data achiziționării.
- Perioada de asigurare.
- Numele clientului (ce poate lipsi)

O vânzare poate conține una sau mai multe asigurări, precum unul sau mai multe produse.

4.1 Primul modul

Primul modul și cel mai important era cel de înregistrarea a cererilor de despăgubire. Am discutat despre soluția actuală, ce probleme are, ce câmpuri ar trebui să fie obligatorii și unde am putea optimiza interacțiunea clientului cu pagina web.

Un bun exemplu de modificări iterative asupra primului concept este la descrierea evenimentelor, după cum puteți observa în figura 4.1.

Inițial fusese vorba despre un simplu câmp text ce va fi la latitudinea utilizatorului spre a fi completat, dar mulțumită experienței în domeniu, cele mai des folosite motive au fost puse în schimb în această listă. În cazul în care s-a selectat „Altceva", detaliile despre descrierea evenimentului devine obligatorie, pentru a ajuta clientul să identifice câmpurile ce trebuiesc completate.

De asemenea, data și ora au un format ușor de înțeles și respectat, iar atunci când se selectează câmpul respectiv, apare un element vizual ajutător pentru a introduce corect data și ora.

Pentru acest modul, trebuia să fie funcțională partea din spate de încărcare poze și gestionare a cererilor, dar și mai important, trebuia să fie gândită deja structura bazei de date, pentru modulele ulterioare.

Am gândit ca fiecărui utilizator să-i atribui un identificator unic, format din 64 caractere, pentru a nu putea fi abuzat sistemul de încărcare a pozelor. Avantajul clar abordării generării unui „token" unic asigură și metoda ulterioară de comunicare client - corespondent al companiei prin construirea unei platforme de a trimite și primi mesaje în Octombrie 2016.

Secțiunea de rapoarte nu exista, dar puteai să cauți în sistemul de cereri în funcție de:

- Nume.
- Prenume.

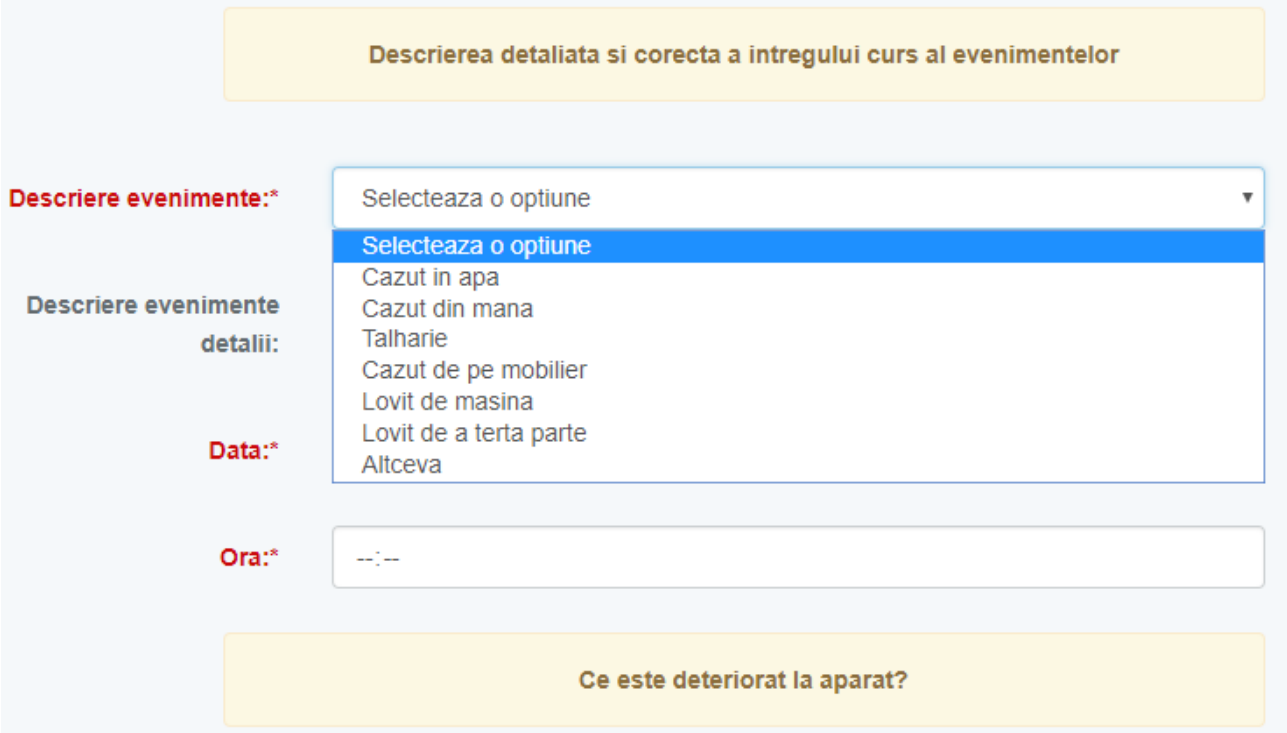


Figura 4.1: Descriere evenimente

- Număr de telefon.
- Numele aparatului.
- Email.
- Id-ul cererii de despăgubire.

4.2 Problema datelor incomplete

Sistemul de importare al datelor săptămânale, primite de la Altex / Media Galaxy se face printr-un fișier Excel. Din păcate, nu știam și nu aveam experiența să întreb despre ce format de raport trebuia aplicația mea să suporte, deci din punctul lor de vedere îmi luasem angajamentul de a importa cu succes toate tipurile de rapoarte existente de-a lungul timpului.

Când am început dezvoltarea, aveam un singur format ce trebuia respectat și am gândit baza de date astfel încât să nu mai păstrez câmpurile redundante, precum categoria și codul de bare al produsului. Din păcate, trebuia să revin asupra algoritmului de a introduce datele aproape pentru fiecare săptămână, când am primit toate vânzările anterioare. Acest modul a fost în dezvoltare continuă, inclusiv pe perioada dezvoltării altor module, din cauza discrepanțelor datelor rapoartelor săptămânale.

În aceeași perioadă se termină sistemul de încărcare al pozelor spre Amazon Web Services și trimiterea clienților un email de confirmare în momentul înregistrării cererii de despăgubire.

Algoritmul final, finalizat în Ianuarie 2017, ținea cont nu de liniile și de coloanele prestabilite, ci de ce câmpuri se aflau pe prima linie în raport. Se normalizau și se foloseau pentru a determina ce coloane apăreau sau nu în raport.

Au fost puține rapoarte ce nu s-au putut importa din cauza lipsei unei coloane obligatorii — a numelui clientului. Au încercat șefii companiei să discute cu Altex / Media Galaxy și au primit confirmarea schimbării rapoartelor pe modelul vechi, dar nu au primit rapoartele

modificate. S-a ajuns în final la un compromis de a pune numele clientului „GENERIC”, pentru că existența lor ca o vânzare în perioada respectivă era mai importantă decât legătura (ce în momentul compromisului nu mai exista) între decizie și vânzare.

4.3 Modulul de rapoarte și decizii

O dată ce a fost implementat modulul de vânzări săptămânale, într-o stare incipientă (după cum s-a explicat în secțiunea anterioară), s-a trecut la dezvoltarea modulului de decizii.

A fost conceput să se lege obligatoriu de o cerere de despăgubire și de o vânzare.

Inițial, s-au construit rapoarte în format „.csv” (valori delimitate prin virgule). Ulterior s-a adăugat opțiunea de a putea vedea într-un tabel, în partea de jos a paginii, a rezultatelor rapoartelor, precum în figura 5.24.

Gândirea pentru legătura dintre o vânzare și o decizie se învârte în jurul imposibilității existenței unei cereri de despăgubire, fără a fi fost vândut, al unui produs.

Avantajul acestei abordări era ușurința aflării deciziilor multiple. Se putea afla ușor pentru că era aceeași vânzare, dar avea cereri multiple legate.

În această perioadă de timp s-au mai adăugat și culori pentru starea curentă a unei cereri de despăgubiri, după cum se poate observa în figura 5.8(a).

După ce s-a terminat modulul de rapoarte, s-a revenit asupra modului de decizii pentru repararea micilor probleme numerice. Nu înțelesesem cum funcționează franciza, sau a sumei totale de plată.

În acest moment s-a lucrat la adăugarea unei verificări a IMEI-ului clienților ce salvau o cerere de despăgubire de a limita depunerea unei cereri la un interval de 24 de ore pentru același IMEI. O problemă des întâlnită de operatorii aplicației erau cererile multiple de la aceeași persoană, cu puține modificări între ele. Prin limita de 24 de ore pentru același IMEI se împiedicau abuzurile. A fost scrisă astfel încât să nu se afle mai multe informații decât necesare.

4.4 Decuplarea deciziilor de vânzări

La scurt timp după ce a intrat modulul de legare a deciziilor de vânzări în funcțiune, structura bazei de date s-a izbit de prima problemă a lumii reale, datele incomplete sau lipsă.

Utilizatorii sistemului nu găseau vânzările asociate cererilor, din cauza incompatibilităților datelor săptămânale trimise de Altex / Media Galaxy. Problema aceasta a fost accentuată de adăugarea obligativității de a asocia decizia cu cererea de despăgubire. Înainte de a putea modifica orice câmp ce ținea de cerere, mulțumită dezvoltării treptate, trebuia astfel găsită asocierea, pentru a reduce din numărul cererilor fără decizie, pentru a recupera din datoria tehnică.

De la un punct, când trebuia să se importe toate vechile vânzări din raportul păstrat în Excel, s-a luat decizia de a se decupla decizia de vânzare.

Am încercat pentru un scurt moment de a modifica interfața de asociere a cererii și a vânzării pentru a forma o despăgubire, adăugând opțiunea de creere a unor vânzări generice. Problema apare când se dorește scoaterea unui raport de vânzare într-o anumită perioadă, deoarece nu se știe magazinul ce a vândut produsul.

Deci, într-un final, am decuplat decizia de vânzare. Am renunțat la structura strâns legată a cererilor de vânzări, în favoarea căutării manuale. Fiecare decizie acum conține detalii și despre produs și despre asigurare.

Altă soluție mai elegantă nu s-ar fi putut găsi, deoarece lipsa de informații ce n-ar putea să difere din cadrul cererii de despăgubire și a datelor primite nu există.

4.4.1 Modificări în viitorul apropiat

Doresc să reintroduc în viitorul apropiat descoperirea automată a deciziilor vechi înapoi prin adăugarea unui câmp de factură la asocierea unei noi cereri de o vânzare.

Diferența acum, față de sistemul decuplat, este că pot să aflu ce decizii au fost legate înainte, mulțumită serviciului de backup, dar și a modului actual de legare a informațiilor, pentru că informațiile legate de vânzare nu au fost modificate până acum de mână.

Avantajul existenței opționale a câmpului de factură ajută la funcționarea algoritmului elegant de găsire a deciziilor, pentru că ar fi același număr de factură, adică aceeași vânzare. Normal că ar trebui să fie trecut prin prisma utilizatorului sistemului, deoarece s-ar putea referi factura la un alt produs și o altă asigurare. De aceea sistemul ar fi opțional, precum sistemul actual de IMEI duplicat.

4.5 După planul inițial de dezvoltare

În urma discuțiilor și finalizării modulelor inițiale, s-a finalizat dezvoltarea modulelor pe orizontală, dar s-a mutat pe verticală. A dezvolta pe orizontală reprezintă dezvoltarea modulelor și a legăturii dintre ele, iar a dezvolta pe verticală reprezintă aprofundarea unui domeniu (în cazul acesta, a construirii rapoartelor și a introducerii datelor).

După o întâlnire, s-a decis introducerea fostelor facturi plătite în sistem și ulterior posibile planuri pentru un sistem modular de rapoarte.

4.5.1 Introducerea datelor și rapoartele avansate

La fel ca în cazul datelor primite de la Altex / Media Galaxy, rapoartele primite (de data asta complete) au determinat o parcurgere completă a lor și modificarea coloanelor ce existau într-un raport, dar lipseau dintr-altul.

Mulțumită numărului redus de rapoarte dorite s-a reușit parcurgerea lor și asigurarea unui algoritm mai simplu din punctul de vedere al condițiilor posibile.

S-au mai descoperit probleme în momentul introducerii lor de două ori in baza de date – operație ce n-ar fi trebuit să mai arate diferențe între datele introduse și cele deja existente. Unele câmpuri se modificau de două ori. De vină a fost greșeala umană și după ce s-a discutat cu persoanele responsabile de rapoarte s-au mutat pe server-ul de producție.

4.5.2 În continuare

Chiar dacă s-a terminat dezvoltarea și pe verticală, mai există o mică problemă nu-mi dă pace: problema tabelelor prea mari. Tabela cu cele mai multe informații, a cererii, are în momentul actual 48 coloane.

Doresc să construiesc o migrare cu următoarele tabele:

- Persoană
- Eveniment
- Factură
- Produs
- Garanție veche

Aceste tabele vor avea o relație unu-la-unu cu tabela cererii. Se vor încărca din baza de date doar valorile necesare. Se va putea implementa mult mai ușor și statistici pentru fiecare persoană, eveniment sau factură.

Legat de statistici, se dorește după o scurtă pauză de dezvoltare implementarea unui sistem modular de rapoarte. S-ar defini prin intermediul unei interfețe grafice ce câmpuri să fie scoase, din ce tabele, cu ce legături între ele. Rapoartele vor fi salvate în Excel și se vor putea reutiliza șabloane pentru cele mai des folosite.

Practic, se dorește înlocuirea sistemului actual de rapoarte cu unul nou, modular.

Capitolul 5

Manual de utilizare

Se vor folosi următoarele denumiri și presupuneri:

1. `Utilizatorul` - Cel ce gestionează cererile de despăgubire și folosește interfața de administrare.
2. `Clientul` - Cel ce dorește înlocuirea sau repararea produsului.
3. `Calea implicită a serverului` - `https://fandu.info`

   Orice cale scrisă *nu* va conține adresa, ci doar calea relativă.

   Exemplu: calea pentru a accesa interfața de administrare, se găsește la:

   `/admin/`
4. Orice câmp cu steluță și / sau roșu trebuie completat înainte de a se putea continua mai departe.

5.1 Înregistrarea unei cereri

Pentru a înregistra o cerere cât mai ușor și fără probleme, aplicația se împarte modular în mai multe categorii, observabile în figurile 5.2(a), 5.2(b), 5.2(d), 5.2(e).

Cea mai mare categorie și cea mai importantă este despre informațiile generale legate de cererea de despăgubire, ce se poate observa în figura 5.1.

După ce vor fi completate câmpurile obligatorii, înainte de a se trimite cererea, clientul este avertizat cu un dialog:

```
Asigurați-vă că ați completat formularul corespuzător!
Următoarea solicitare pentru cererea curentă poate fi
înregistrată abia în următorul interval de 24 de ore.
```

După ce se înregistrează cererea, se trimite un mail către client și utilizatorii sistemului, pentru a certifica crearea unei noi cereri de despăgubire cu toate datele completate.

În figura 5.3 se poate observa structura mesajului.

În același timp, utilizatorul este redirecționat către pagina de încărcare poze, vizibilă în figura 5.4.

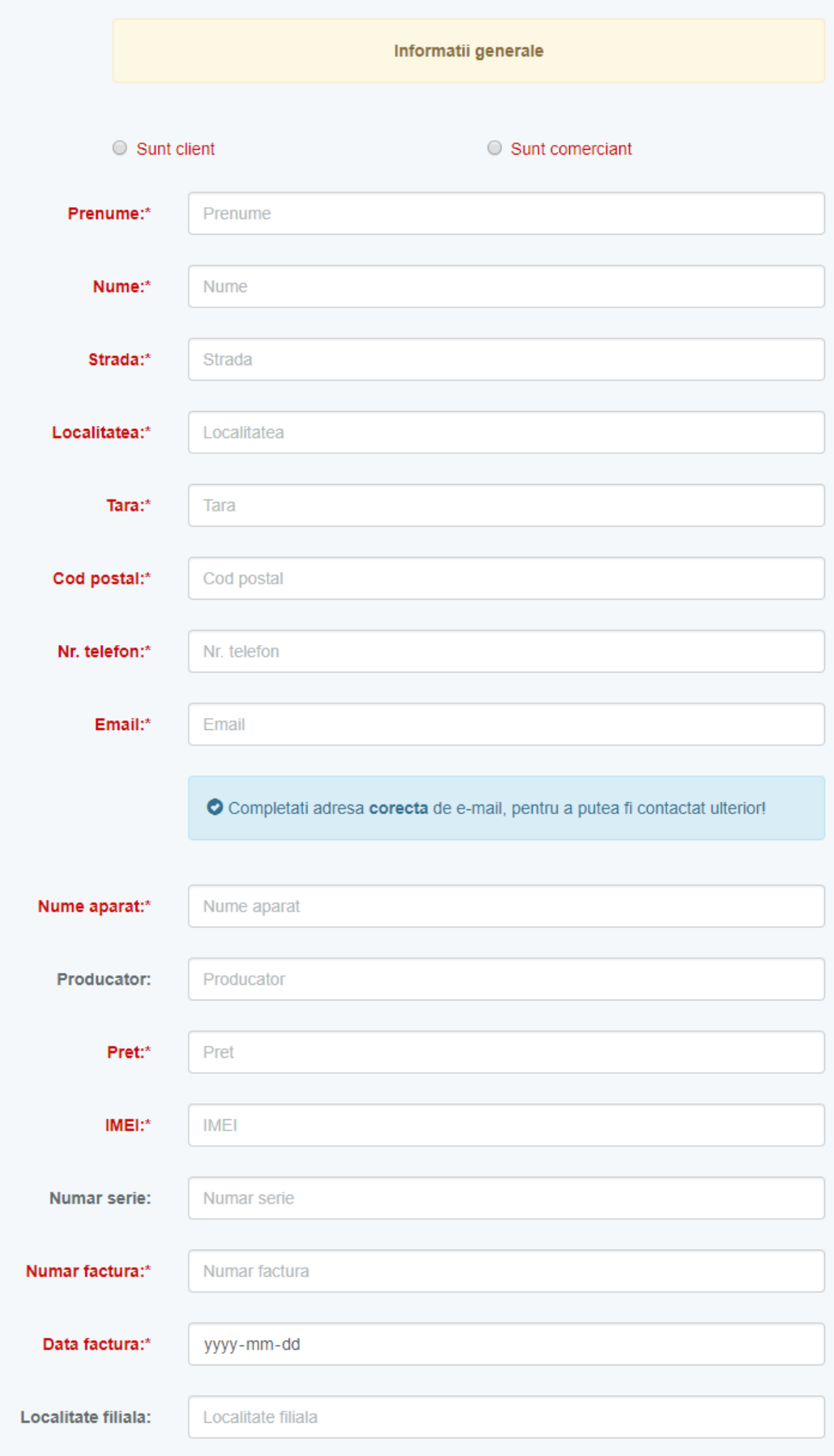
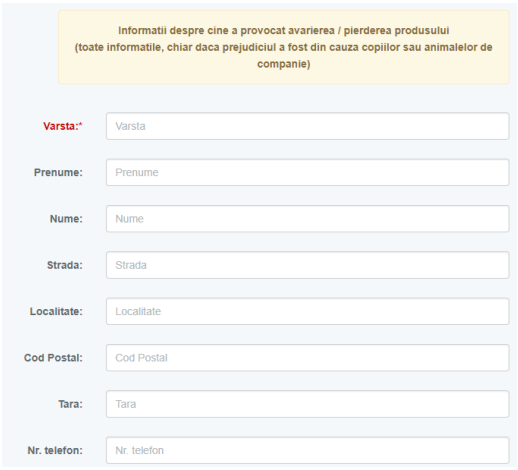
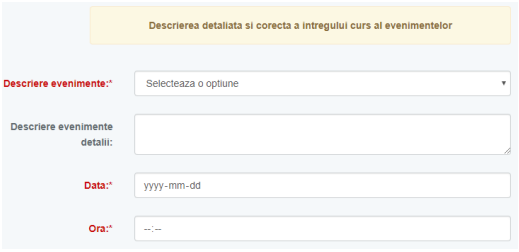


Figura 5.1: Informații generale

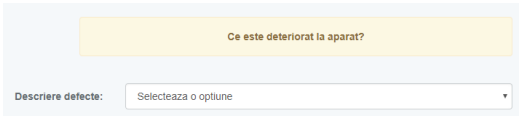


(a) Informații despre cine a provocat avarierea produsului

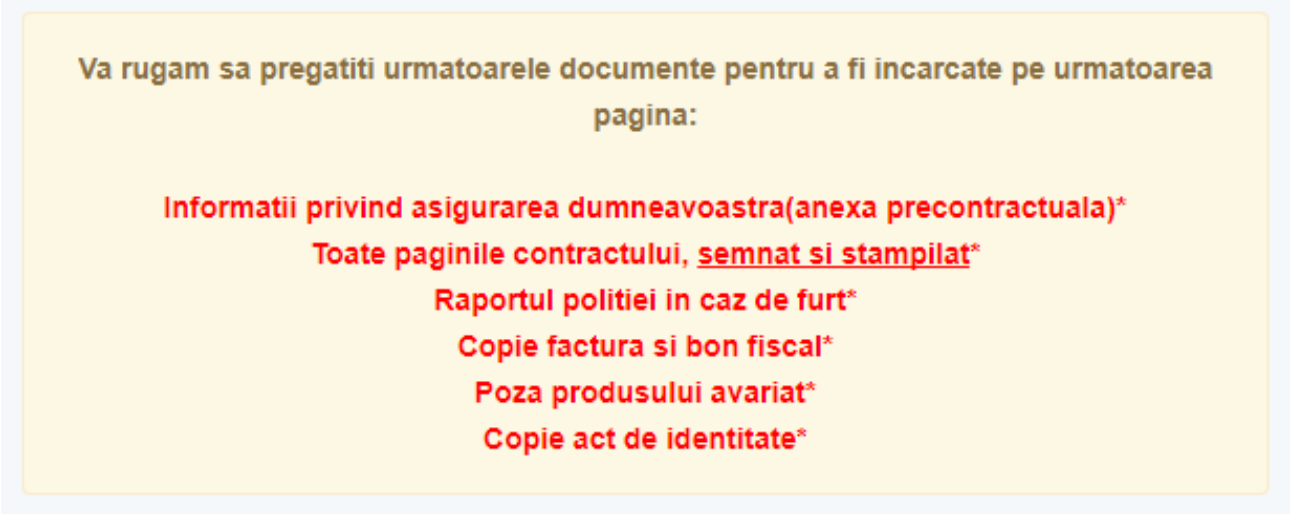
(b) Informații despre cererea de despăgubire



(c) Descrierea evenimentelor

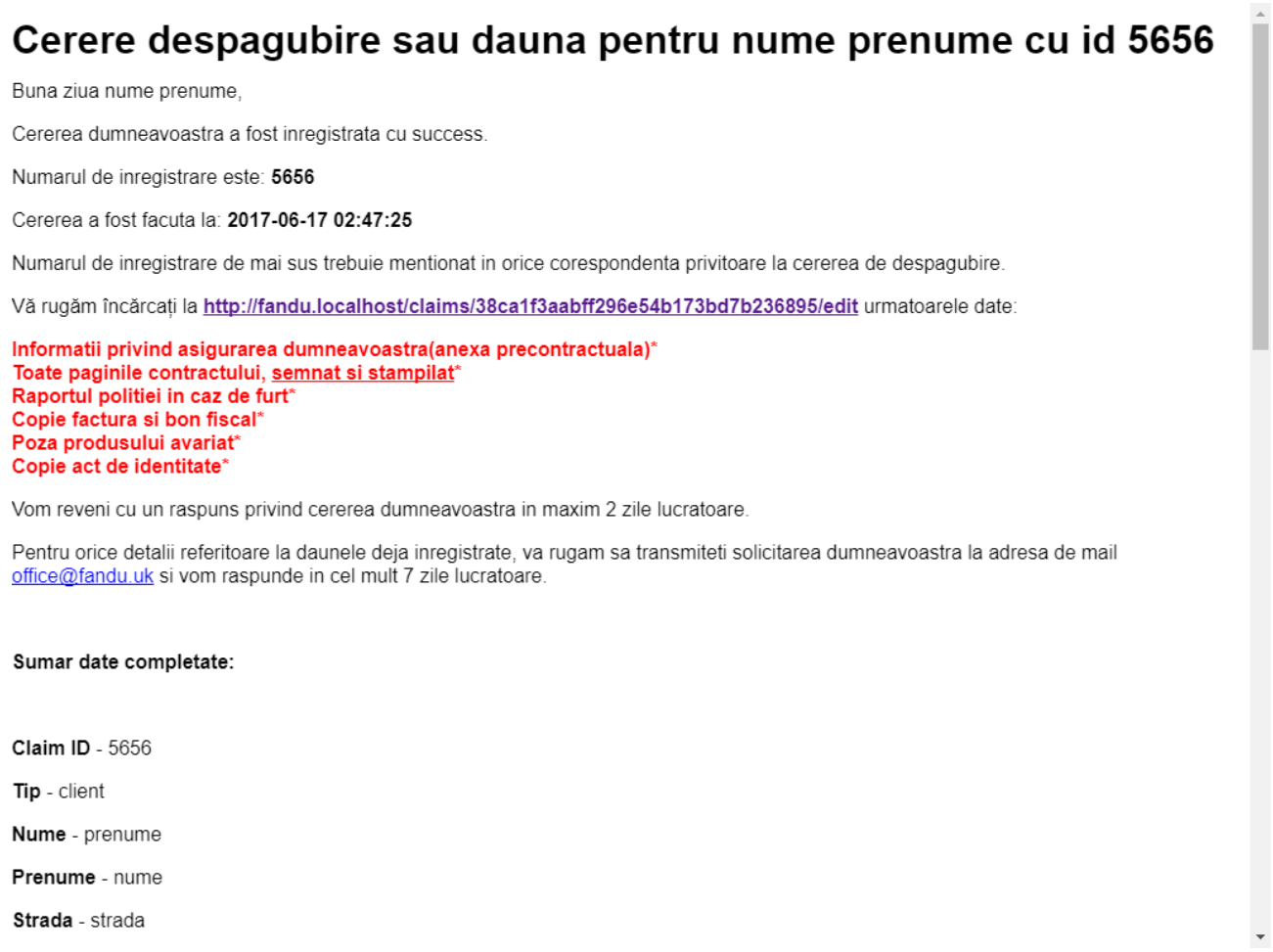


(d) Problemele dispozitivului



(e) Avertizare încărcare poze

Figura 5.2: Câmpurile de completat în cererea de despăgubire



Cerere despagubire sau dauna pentru nume prenume cu id 5656

Buna ziua nume prenume,

Cererea dumneavoastra a fost inregistrata cu success.

Numarul de inregistrare este: **5656**

Cererea a fost facuta la: **2017-06-17 02:47:25**

Numarul de inregistrare de mai sus trebuie mentionat in orice corespondenta privitoare la cererea de despagubire.

Vă rugăm încărcați la **http://fandu.localhost/claims/38ca1f3aabff296e54b173bd7b236895/edit** urmatoarele date:

Informatii privind asigurarea dumneavoastra(anexa precontractuala)*
Toate paginile contractului, semnat si stampilat*
Raportul politiei in caz de furt*
Copie factura si bon fiscal*
Poza produsului avariat*
Copie act de identitate*

Vom reveni cu un raspuns privind cererea dumneavoastra in maxim 2 zile lucratoare.

Pentru orice detalii referitoare la daunele deja inregistrate, va rugam sa transmiteti solicitarea dumneavoastra la adresa de mail office@fandu.uk si vom raspunde in cel mult 7 zile lucratoare.

Sumar date completate:

Claim ID - 5656

Tip - client

Nume - prenume

Prenume - nume

Strada - strada

Figura 5.3: Mesajul trimis de sistem către client și utilizatorii sistemului

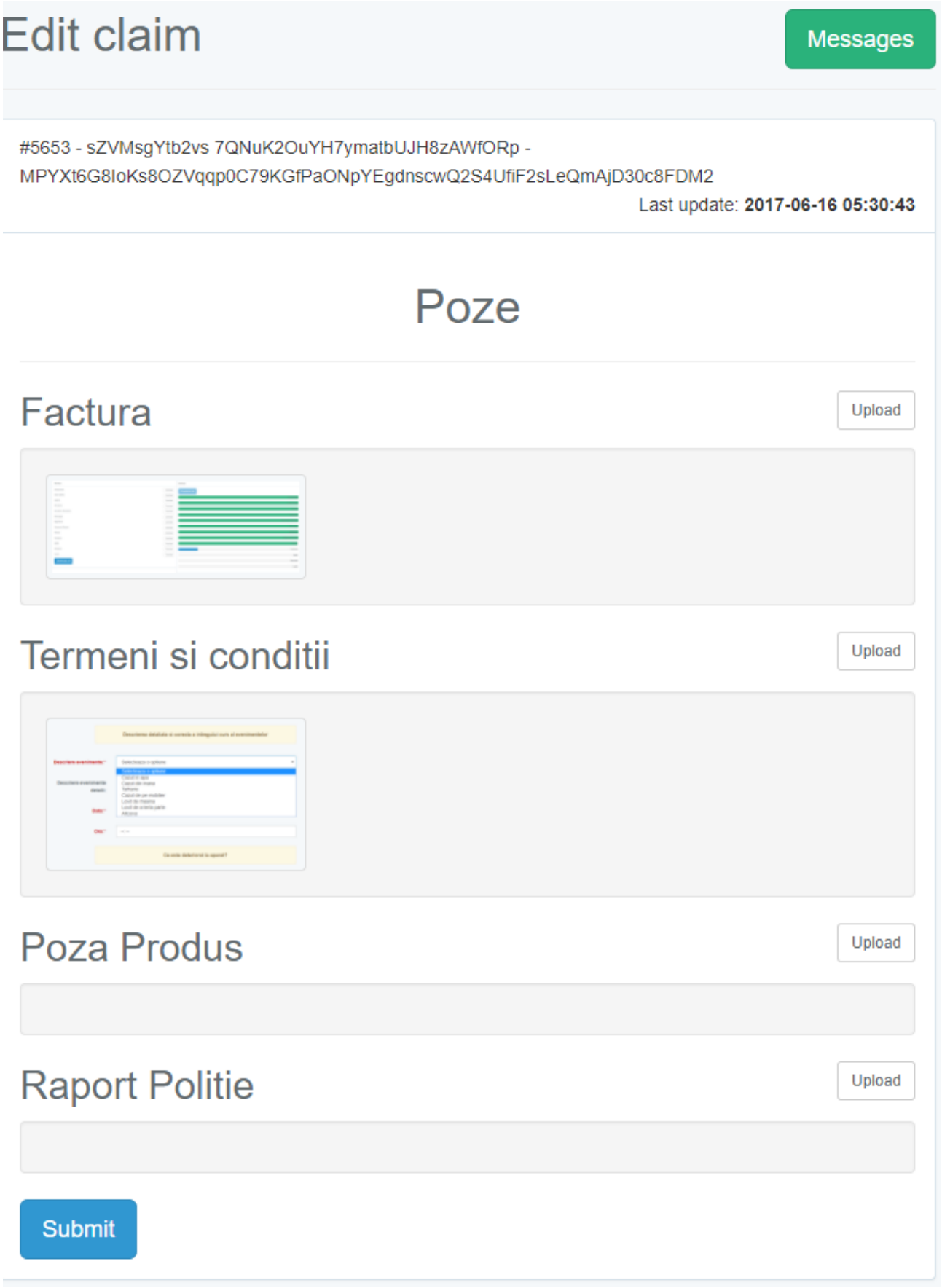


Figura 5.4: Formularul de încărcare a pozelor

5.2 Secțiunea de mesaje din cadrul cererilor de despăgubire

În caz de probleme sau nelămuriri, în cadrul link-ului primit pe mail, o dată ce se navighează la pagina respectivă, se poate apăsa butonul „Messages” pentru a trimite într-un dialog mesaje de la client la utilizator, cum se poate observa în figurile 5.5(a), 5.5(b), 5.5(c).

De asemenea, același sistem poate fi folosit și de utilizator, pentru a trimite un mesaj clientului, fiecare fiind anunțat pe mail în cazul unui nou mesaj.

5653 - New message from sZVMsgYtb2vs 7QNuK2OuYH7ymatbUJH8zAWfORp

Good day,

sZVMsgYtb2vs 7QNuK2OuYH7ymatbUJH8zAWfORp has added a new message

Please click on the link below to view it:
http://fandu.localhost/claims/250d65c6c6564bb629db7264d01dbcbb/messages

Thank you!

Va rugam NU raspundeti la el.

(a) Email trimis când clientul trimite un mesaj

5653 - Mesaj nou cerere de la Office

Buna ziua sZVMsgYtb2vs 7QNuK2OuYH7ymatbUJH8zAWfORp,

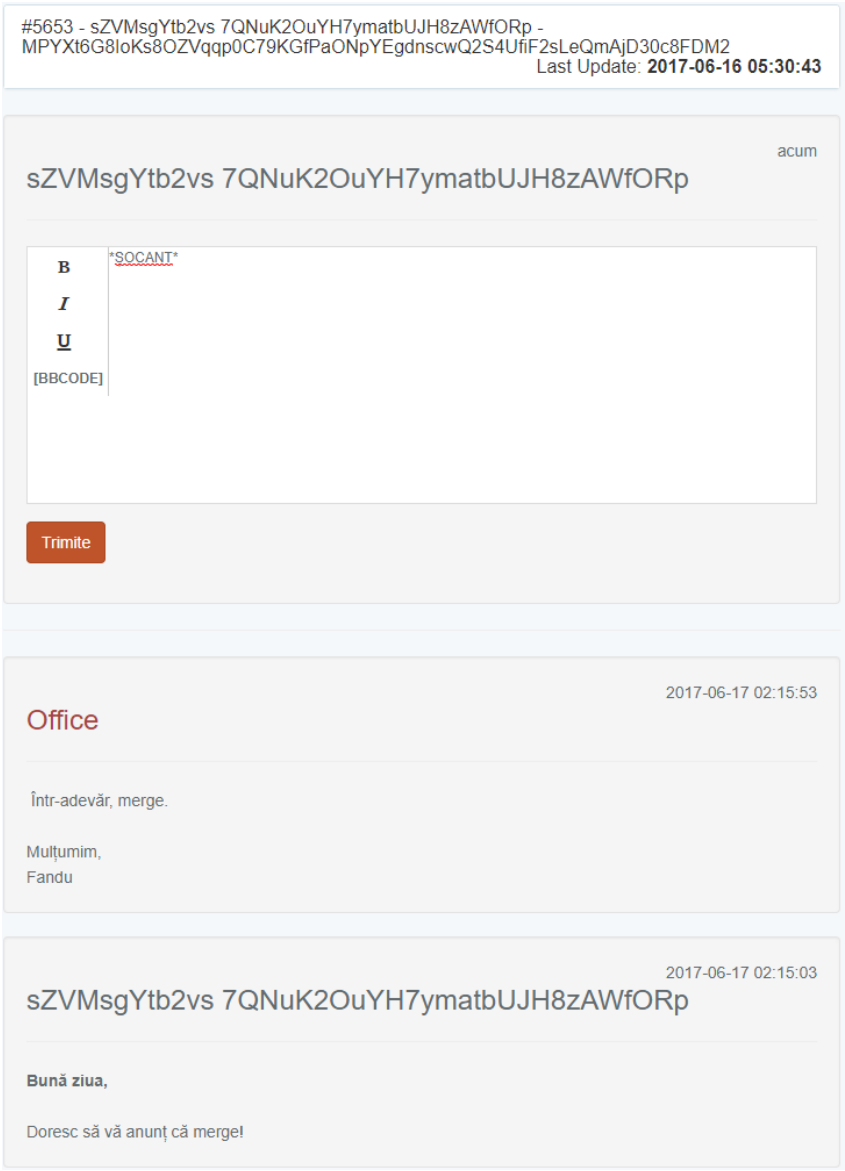
Office a actualizat cererea cu un nou mesaj

Va rugam sa intrati pe urmatorul link pentru a-l vedea:
http://fandu.localhost/claims/250d65c6c6564bb629db7264d01dbcbb/messages

Va multumim!

Va rugam NU raspundeti la el.

(b) Email trimis când utilizatorul trimite un mesaj



(c) Interfața de messages, împreună cu istoricul mesajelor

Figura 5.5: Interacțiunea mesajelor de mesaje

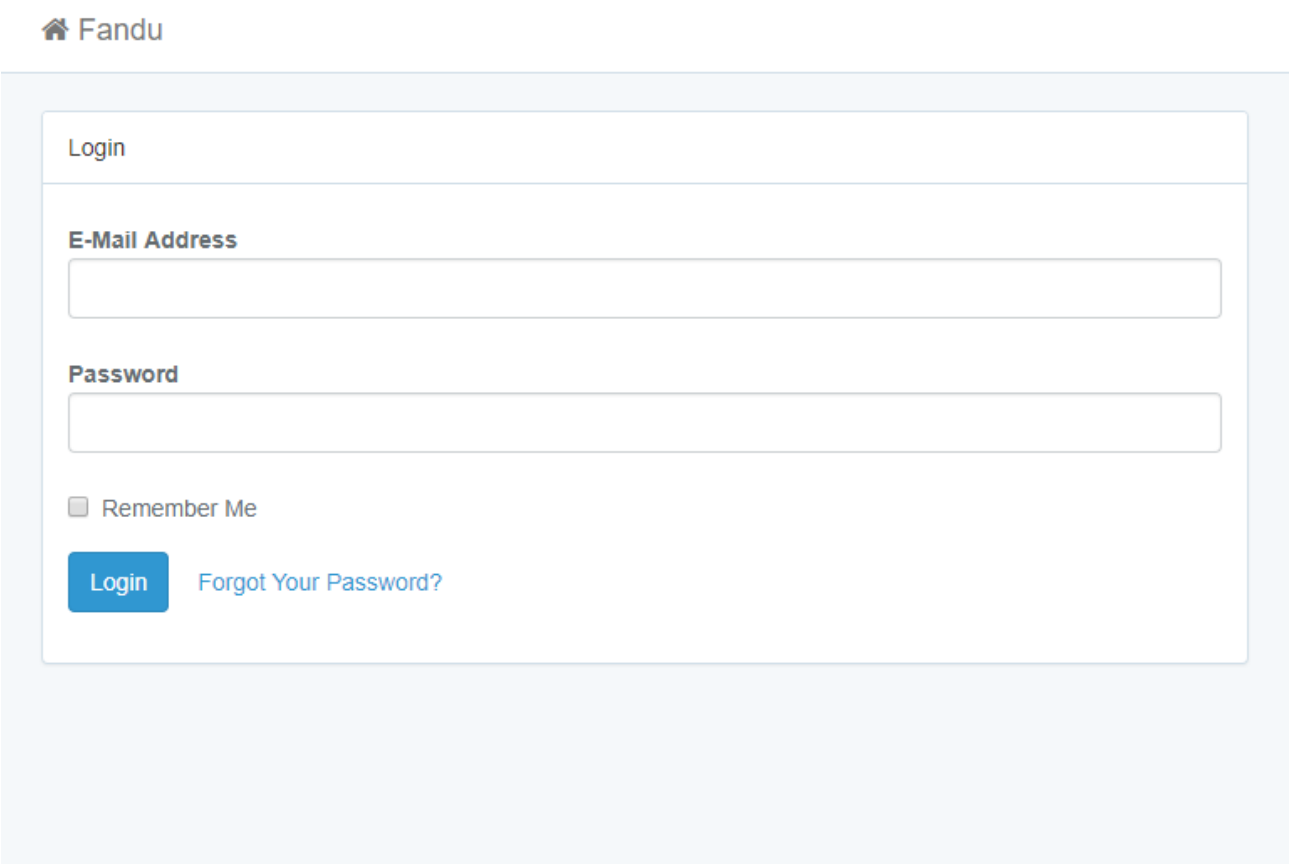


Figura 5.6: Pagina de înregistrare pentru interfața de administrare

5.3 Interfața de administrare

Interfața de administrare nu poate fi accesată dacă nu se cunoaște ori calea, ori dacă nu se navighează din cadrul vizualizării cererii (pentru client, acest pas reprezintă încărcarea pozelor) pe pagina principală.

Pagina principală a interfeței de administrare se găsește la:

```
/admin/
```

În momentul în care utilizatorul navighează la *orice* parte a aplicației de administrare și nu este înregistrat, pagina apare ca în figura 5.6.

După ce utilizatorul introduce email-ul și parola sa corect în sistem, sistemul va naviga la pagina ce utilizatorul încerca să acceseze.

Pagina principală a interfeței de administrare arată ca în figura 5.7. În cazul oricărei probleme, sunt lăsate datele mele de contact pentru a le remedia cât mai rapid. De aici, utilizatorul poate introduce rapid un număr de cerere de despăgubire folosind câmpul din bara de navigație.

În pagina principală, se găsesc două grafice:

1. Primul se referă la cererile de despăgubire noi în această săptămână, raportat la numărul lor pe luna respectivă.

2. Al doilea se referă la numărul total de decizii terminate, active și refuzate.

Numerele folosite în aceste grafice apar atunci când se navighează cu cursorul deasupra lor. Bara de navigație conține toate modulele aplicației.

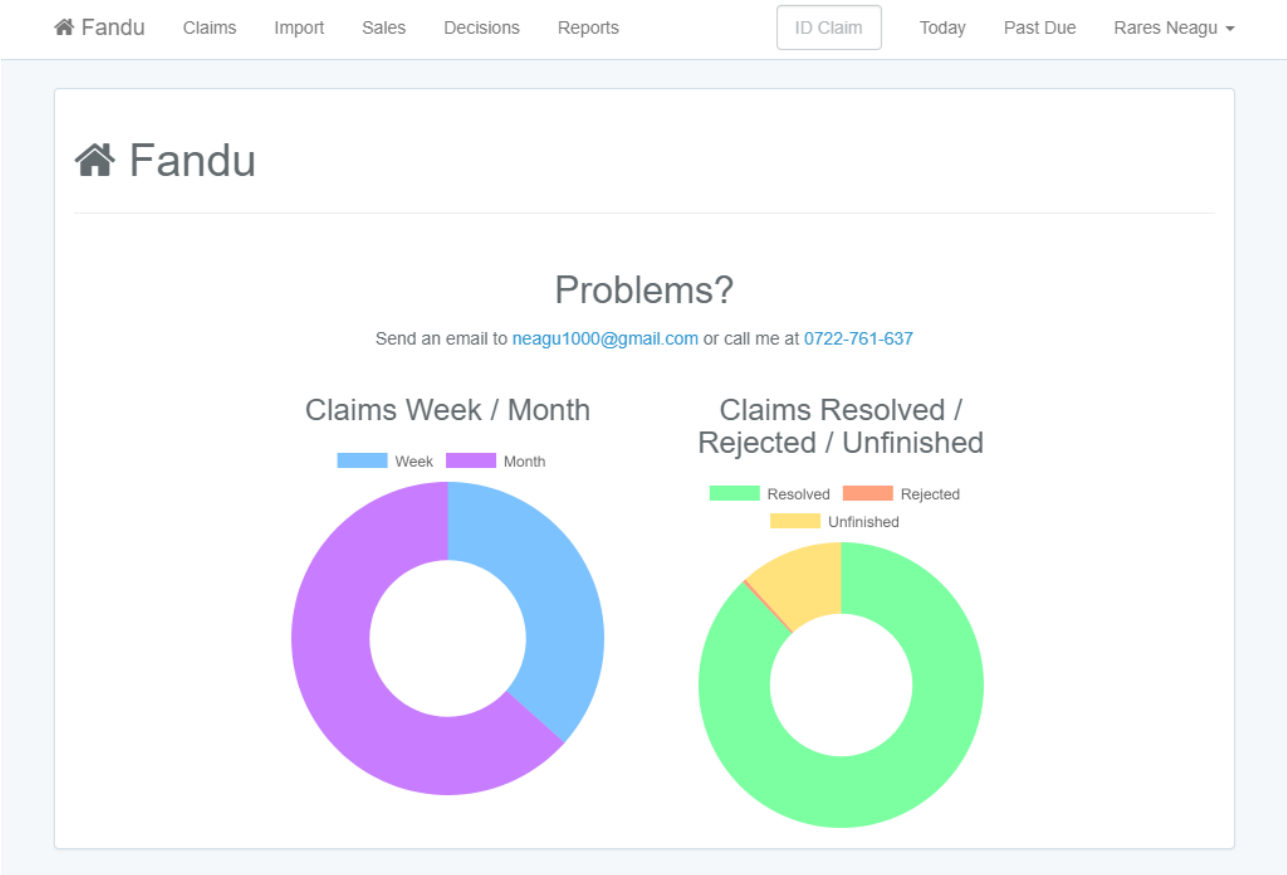


Figura 5.7: Pagina principală a interfeței de administrare

5.3.1 Modulul cererilor de despăgubire

Modulul cererilor de despăgubire se poate accesa prin butonul „Claims” din bara de meniu.

Pe pagina principală a modului se vizualizează ultimele 100 cereri de despăgubire în funcție data sosirii lor în sistem. Fiecare cerere are patru butoane:

1. `Decide` - pentru a naviga rapid la decizia cererii de despăgubire. Vezi asdasdasd
2. `View` - vizualizarea cererii de despăgubire scrise de client.
3. `Edit` - modificarea cererii de despăgubire
4. `Delete` - iconița de gunoi reprezintă ascunderea cererii din interfața principală.

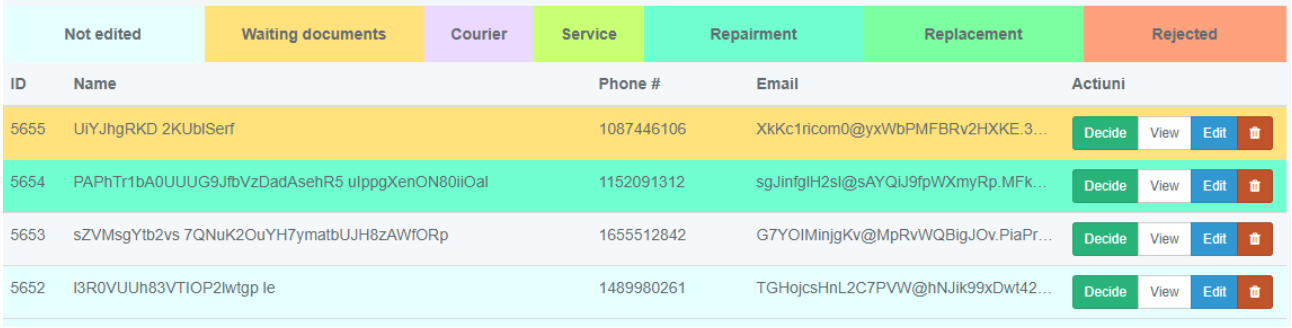
Pentru a ajuta utilizatorul să identifice mai ușor starea cererilor, se reprezintă fiecare linie din tabel cu un sistem de culori, regăsit deasupra capului de tabel. Dacă se apasă un element din acesta, se filtrează cererile de pe pagina curentă după elementul respectiv.

Figura 5.8(a) reprezintă un exemplu a datelor din tabel, figura 5.8(b) reprezintă cererile filtrate după „Repairment” (reparare), figura 5.8(c) arată sistemul de căutare în acțiune, iar figura 5.8(d) prezintă o cerere cu informații adiționale.

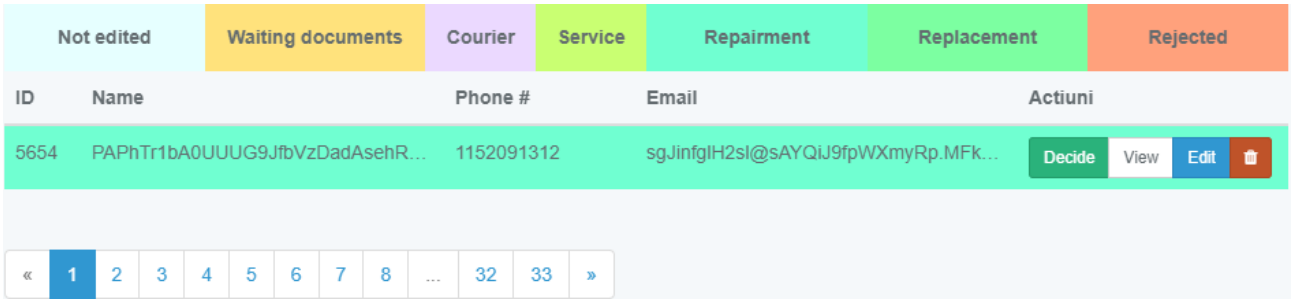
5.3.1.1 Vizualizarea și modificarea cererilor

Când se vizualizează o cerere, apar date folositoare utilizatorului, conform figurii 5.9, precum:

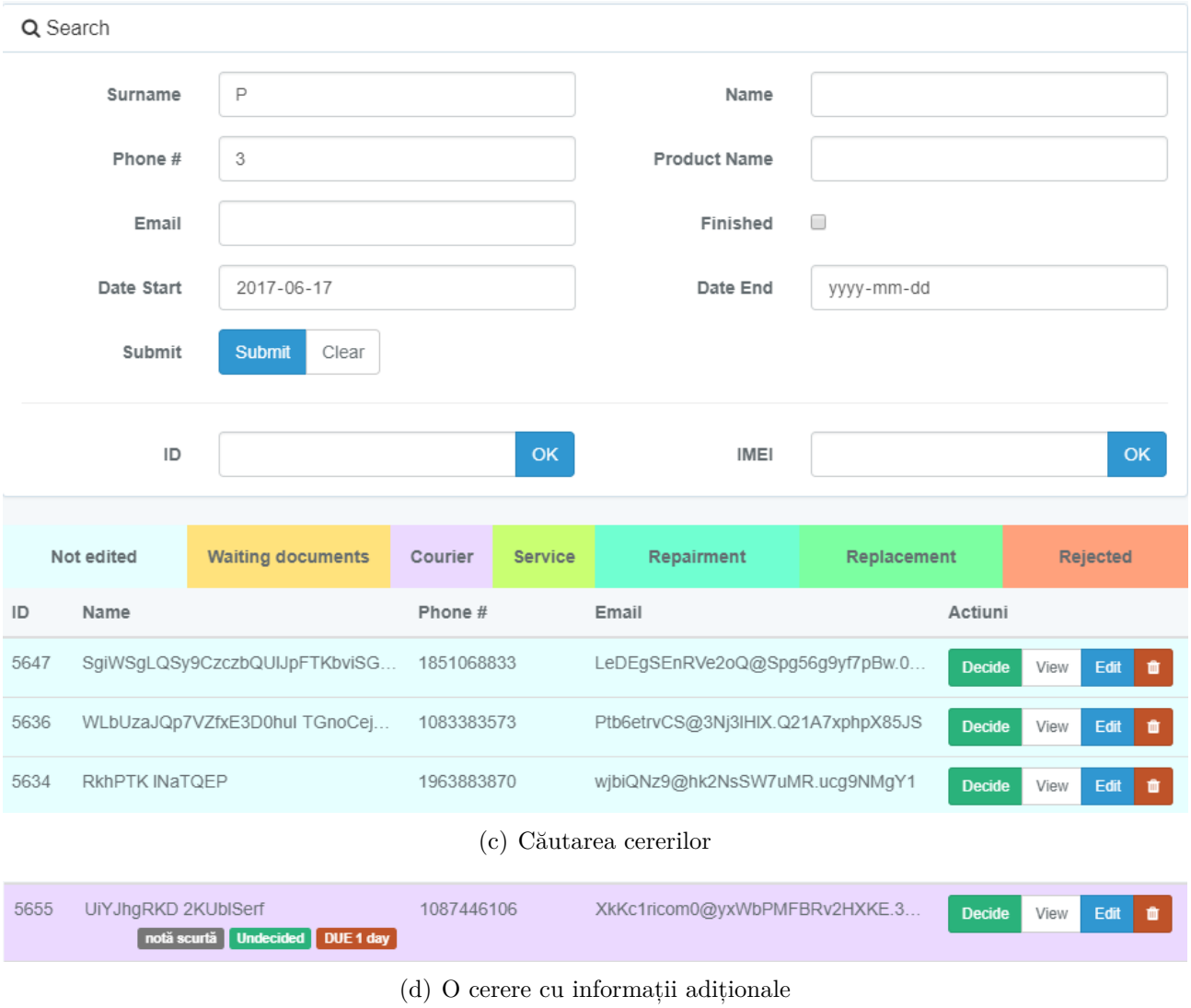
1. Dacă cererea de despăgubire are un IMEI duplicat cu o altă cerere anterioară de despăgubire.



(a) Structura culorilor



(b) Filtrarea prin culori



(c) Căutarea cererilor

(d) O cerere cu informații adiționale

Figura 5.8: Interfața administrării cererilor

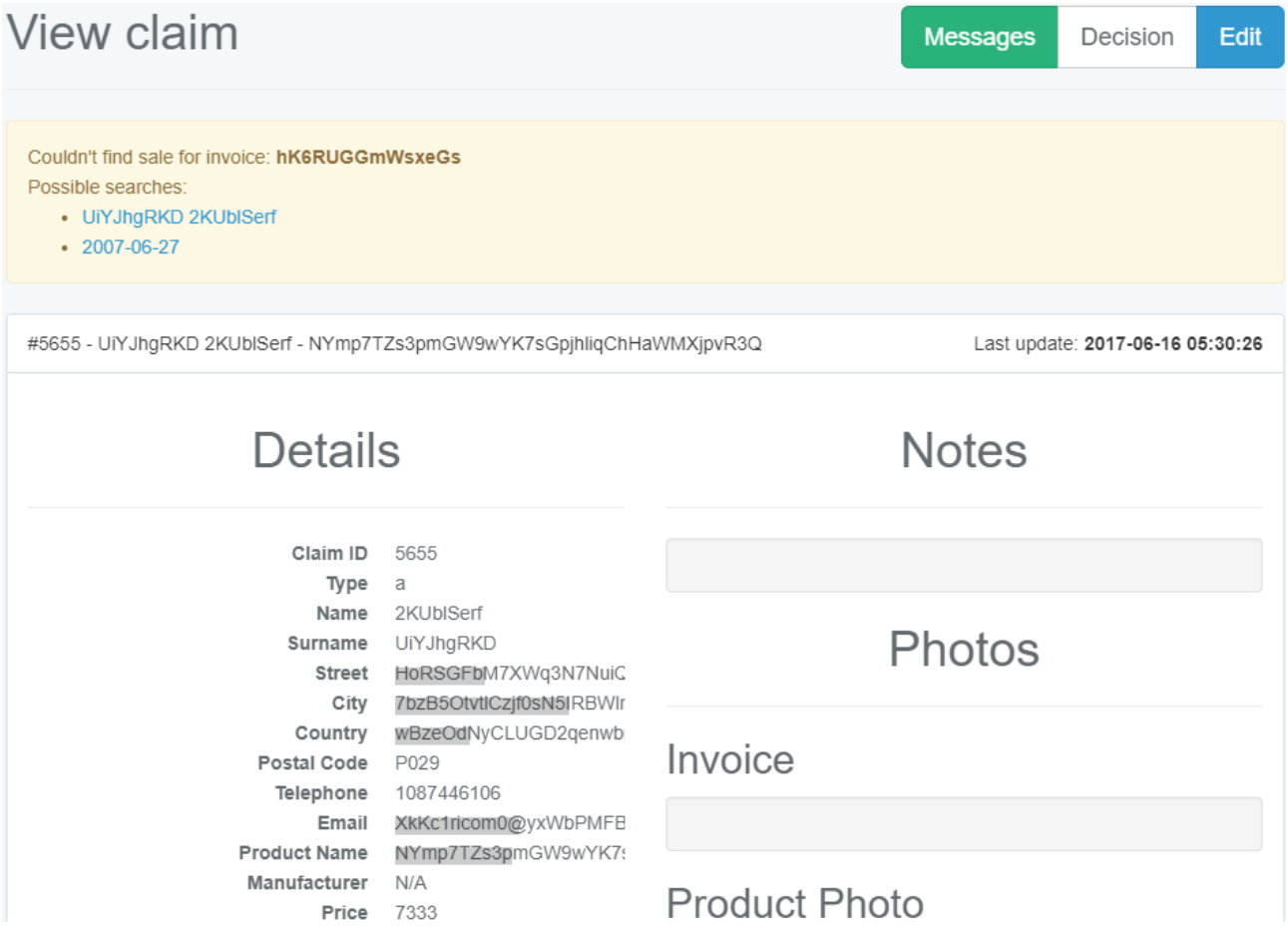


Figura 5.9: Date suplimentare pentru utilizator în cadrul vizualizării unei cereri

2. Dacă nu s-a putut identifica vânzarea, după numărul facturii.

Pentru a modifica cererea, este obligatorie o decizie. Utilizatorul nu poate salva niciun câmp, după cum se vede în figura 5.10(a).

După ce se adaugă decizia respectivă, utilizatorul poate scrie un mic detaliu despre cerere, a-i modifica starea, ce implicit modifică și starea deciziei (câmpul „Resolution") și de a specifica când să-i aducă aminte aplicația despre cerere – „Reminder" – pentru a putea fi actualizate. Se pot observa aceste câmpuri în figura 5.10(b).

5.3.1.2 Reminders

Deoarece clienții au o perioadă de grație pentru a încărca pozele și pentru că multă lume revine ulterior cu ele, sistemul în meniul principal are două butoane:

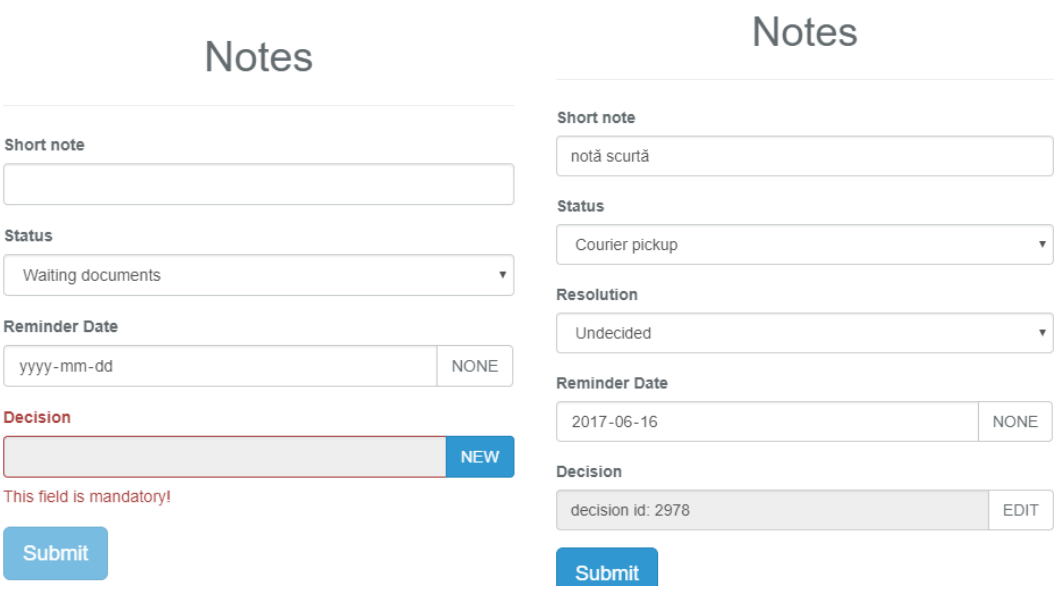
1. `Today` - Cererile ce trebuiesc actualizate exclusiv azi.

2. `Past Due` - Cererile ce trebuiau actualizate, aranjate după când trebuiau actualizate.

Ambele secțiuni arată precum figura 5.11. S-a adăugat și butonul de șterge automată a câmpului „Reminder".

5.3.2 Modulul introducerii datelor în sistem

Introducerea datelor în sistem se face prin butonul „Import" din bara de meniu. Apar astfel două panouri, conform figurii 5.12(a):



(a) Fără o decizie - imposibil de modificat (b) Cu decizie - ușor de modificat

Figura 5.10: Modificarea unei cereri

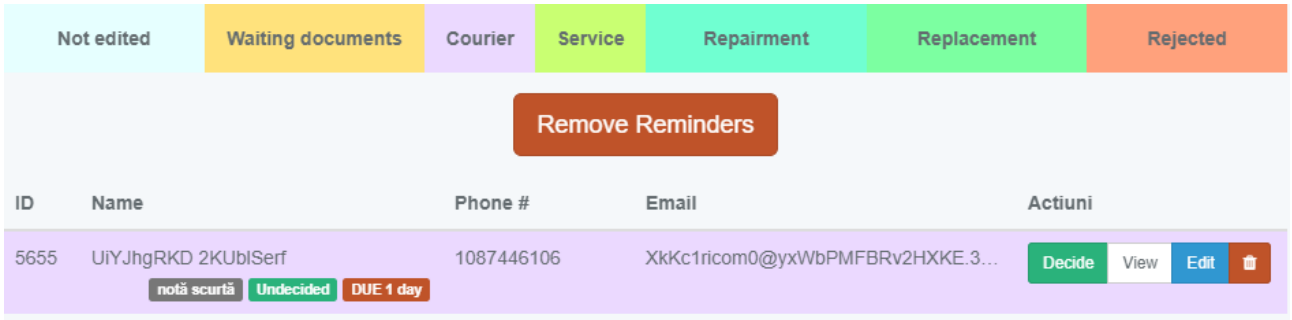
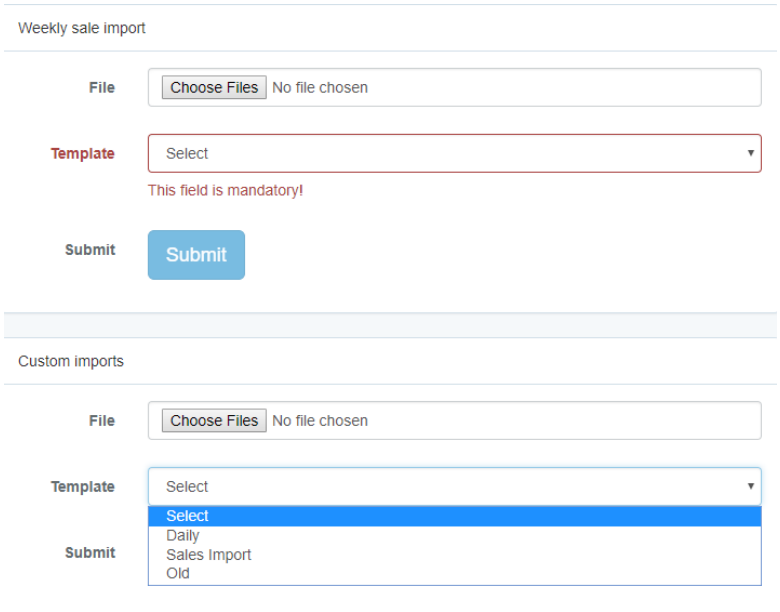


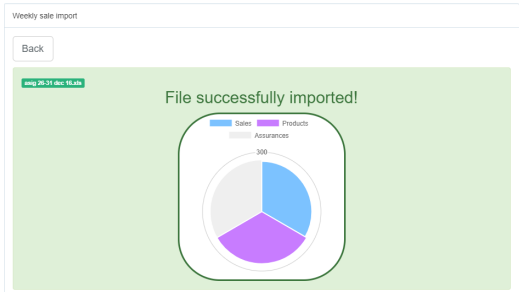
Figura 5.11: Sistemul de aducere aminte a cererilor de despăgubire



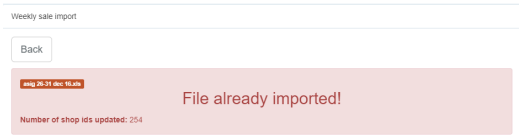
(a) Sistemul de introducere a datelor

(b) Mesaj trimis utilizatorilor sistemului în momentul adăugării datelor în sistem

Figura 5.12: Introducerea datelor



(a) Exemplu la introducerea cu succes a datelor



(b) Exemplu la încărcarea unui duplicat

Figura 5.13: Rezultate posibile la introducerea datelor în sistem

1. `Weekly sale import` - Introducerea raportului principal aplicației. Se va trimite email de confirmare utilizatorilor aplicației, precum cel din 5.12(b).

2. `Custom imports` - Introducerea datelor dinaintea aplicării sistemului în funcțiune pentru compania de asigurări.

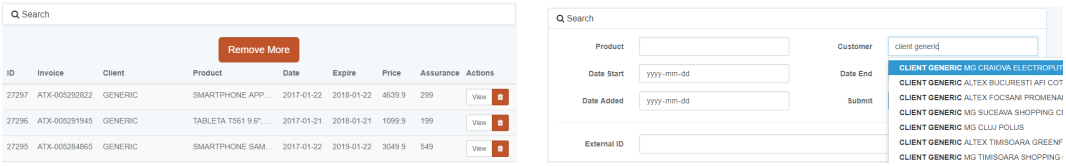
Se vor dezvolta propriile șabloane, ajustate nevoilor fiecărei companii ce folosește acest sistem de gestiune.

După momentul încărcării, utilizatorul trebuie să aștepte finalizarea introducerii datelor.

Sistemul are grijă în cazul introducerii unui fișier duplicat, precum se vede în figura 5.13(b), pentru că s-a încercat re-introducerea, din greșeală, aceluiași raport introdus cu succes în figura 5.13(a).

5.3.3 Modulul de vânzări

Modulul de vânzări se poate accesa prin butonul „Sales” din bara de meniu. Conform figurii 5.14(a), sunt afișate într-un tabel vânzările, cu opțiunea de a intra în detalii apăsând



(a) Exemplu vânzări (b) Exemplu căutare vânzări

Figura 5.14: Exemplu modul vânzări

pe butonul „View” și de a șterge vânzarea (sau vânzările, dacă se apasă pe butonul „Remove More”) prin apăsarea iconiței în forma unui coș de gunoi.

Căutarea după numele produsului „Product” sau numele clientului „Customer”, cât și intervalul vânzării produsului se pot vedea în figura 5.14(b). Funcționalitatea este identică pentru momentul legării unei decizii de o vânzare.

5.3.3.1 Vizualizarea vânzării

Dacă se apasă pe butonul de vizualizare a unei vânzări, vor apărea din baza de date detaliile vânzării, precum în figura 5.15.

Din această perspectivă, se pot apăsa următoarele butoane:

1. `Details` - Vizualizarea detaliilor vânzării 5.15.
2. `Delete Product` - Ștergerea unui produs 5.16(d)
3. `Delete Assurance` - Ștergerea unei asigurări 5.16(c)
4. `Add Product` - Adăugarea unui produs 5.16(b)
5. `Add Assurance` - Adăugarea unei asigurări 5.16(a)

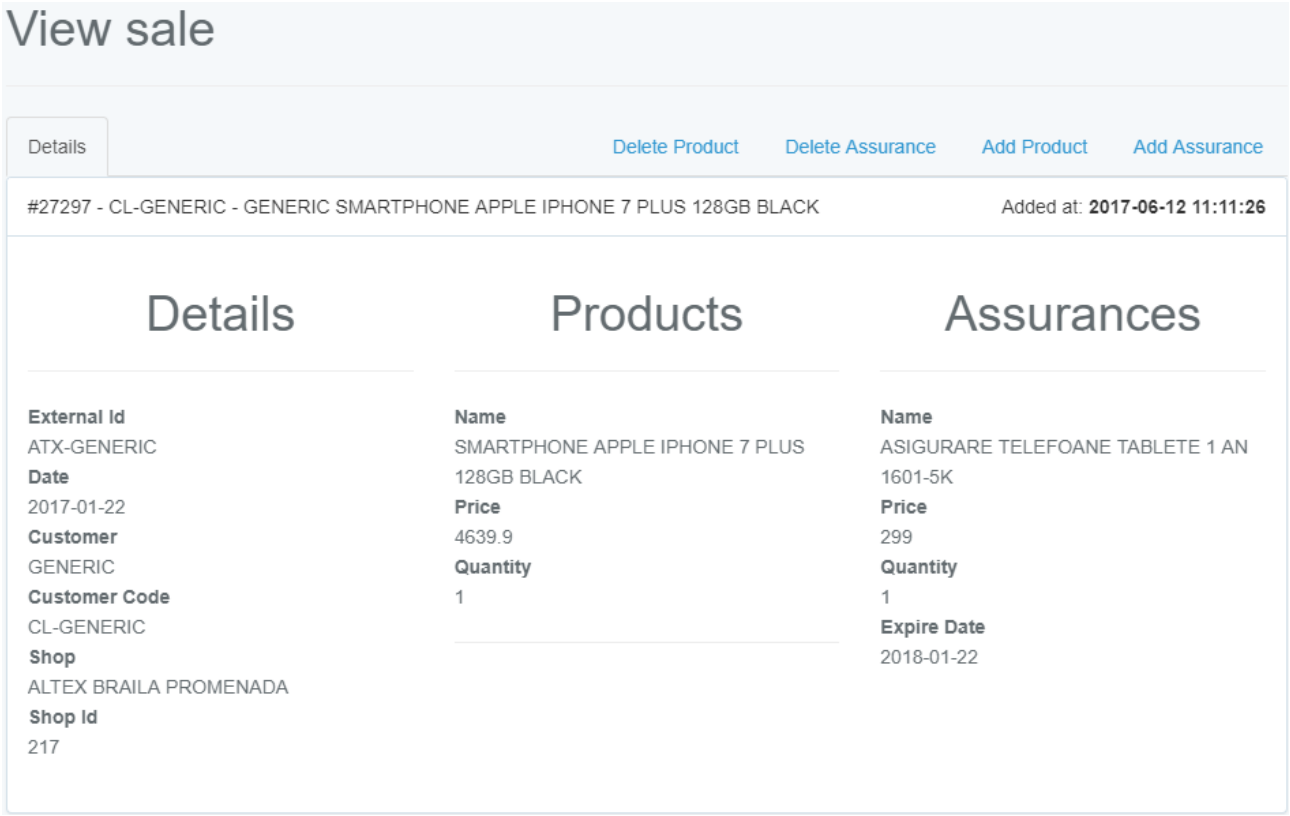
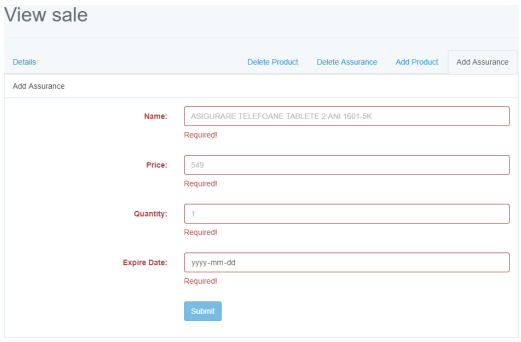
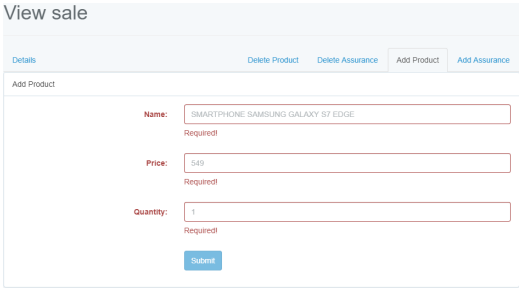


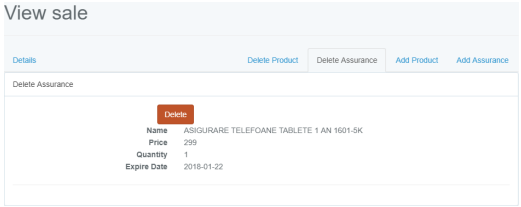
Figura 5.15: Detaliile vânzării



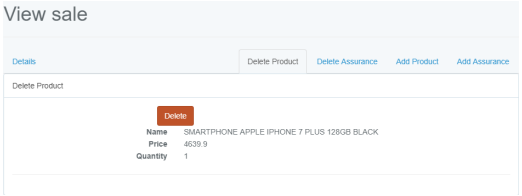
(a) Interfața de adăugare a unei asigurări



(b) Interfața de adăugare a unui produs



(c) Interfața de ștergere a unei asigurări



(d) Interfața de ștergere a unui produs

Figura 5.16: Detalii vânzare

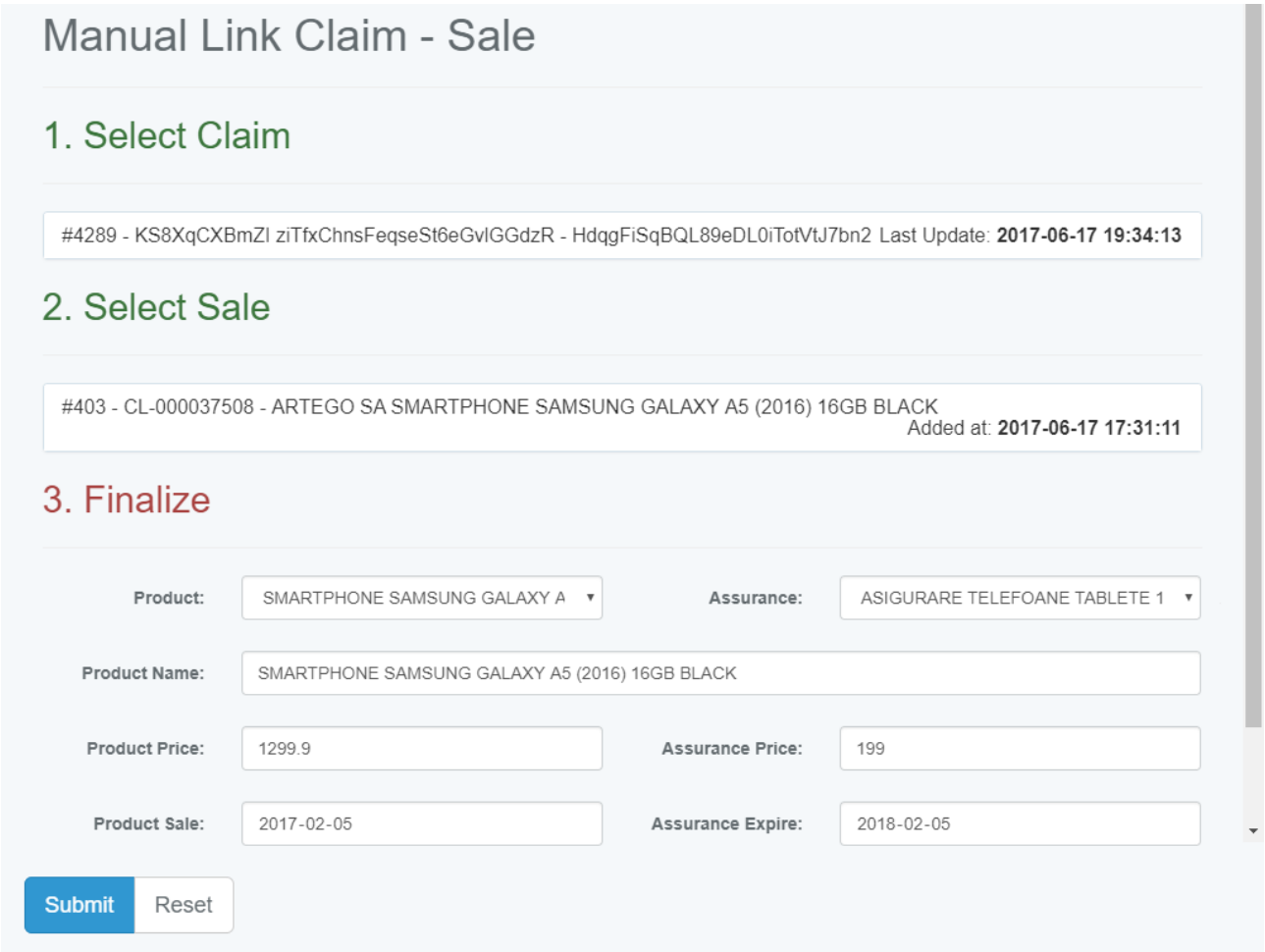


Figura 5.17: Confirmarea legăturii între cerere și vânzare

5.3.4 Crearea unei decizii

Crearea unei decizii se face apăsând butonul „Decision” în cadrul vizualizării tabelului cererilor sau vizualizării / modificării unei cereri.

În care nu există, se va crea o nouă decizie urmărind următorii pași:

1. Dacă se poate găsi automat o vânzare în funcție de numărul facturii, se va sări pasul 2 și se va trece la pasul 4.

2. Dacă nu se poate găsi automat o vânzare, se va afișa pagina de asociere manuală a vânzării, precum în figura 5.18.

3. Dacă nu se poate găsi vânzarea, se poate apăsa butonul „Manual Sale”, de a introduce manual valorile necesari construirii deciziei, precum în figura 5.19.

   Acest pas este ultimul în cazul construirii legăturii manuale.

4. Ecranul va arăta precum în figura 5.17, pentru a valida produsul și asigurarea asociată vânzării.

5.3.5 Modulul de decizii

Modulul de decizii se poate accesa prin butonul „Decision” din bara de meniu. Conform figurii 5.20(a), sunt afișate într-un tabel deciziile, cu opțiunea de a intra în detalii apăsând pe

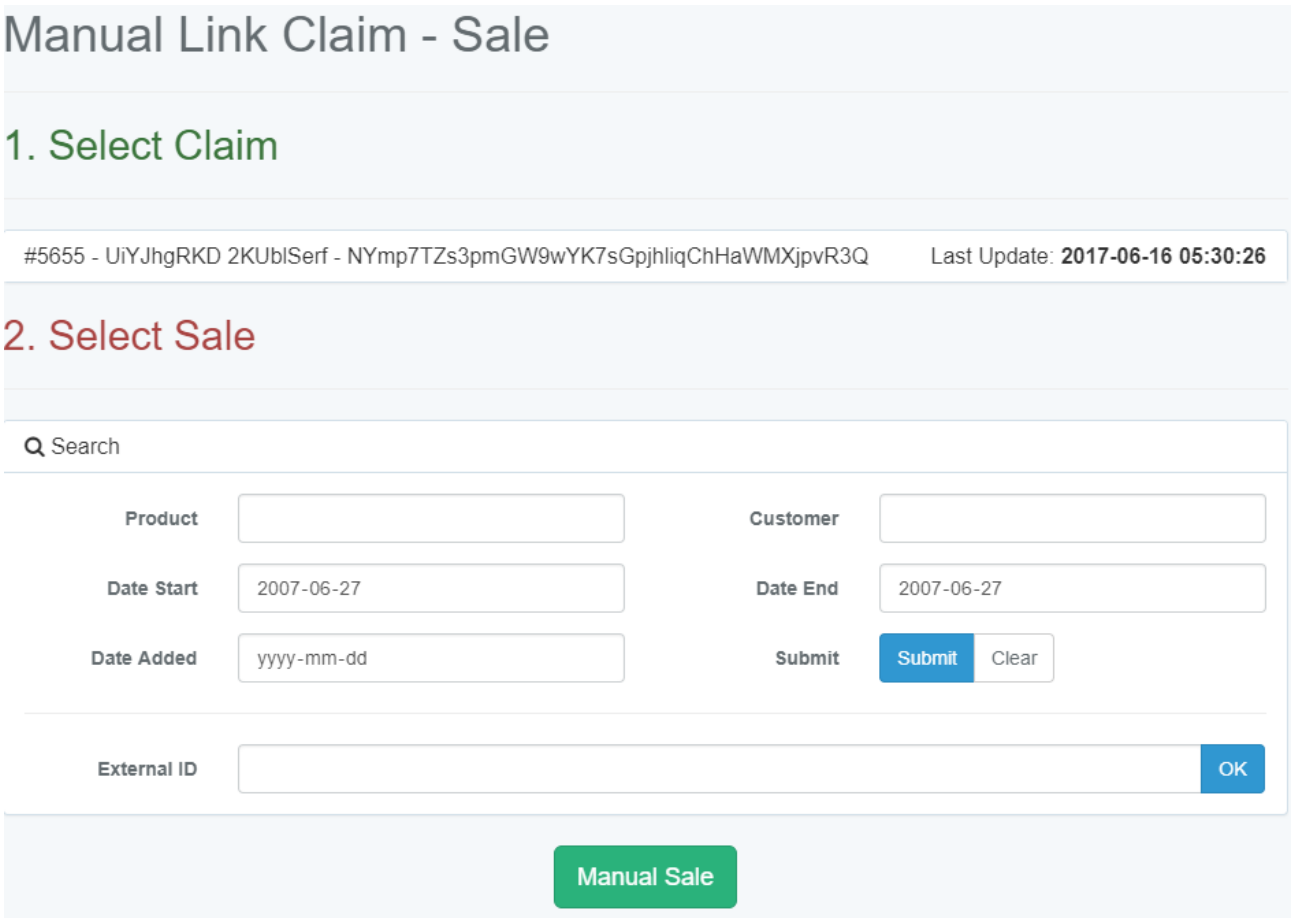


Figura 5.18: Selectarea manuală a vânzării

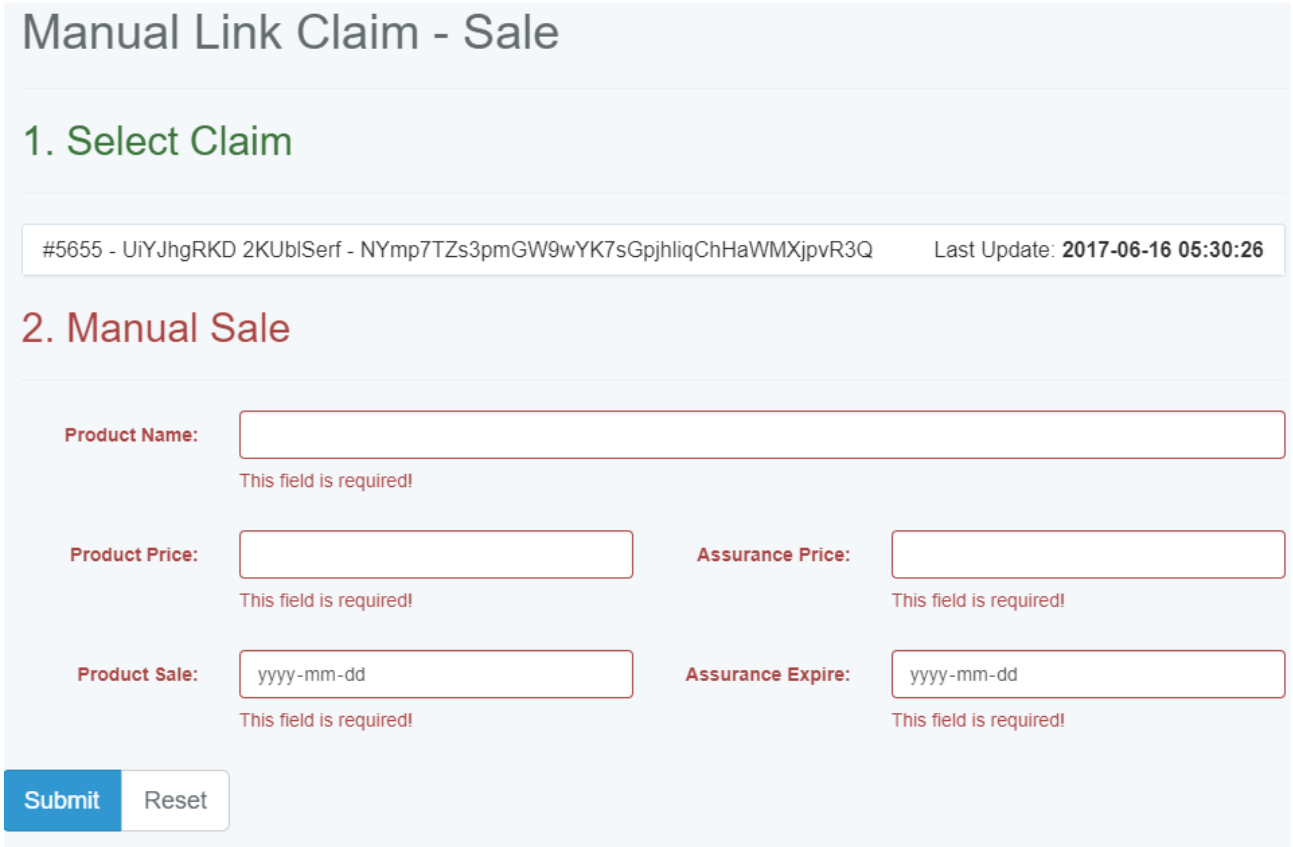
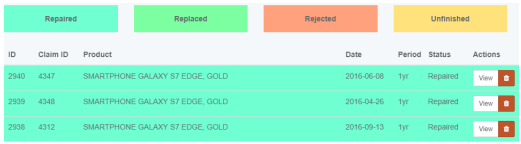
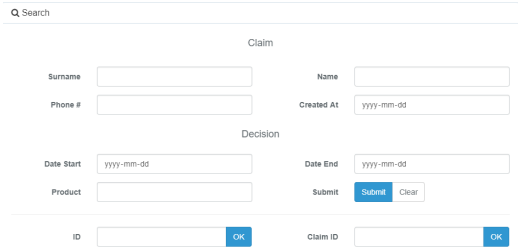


Figura 5.19: Construirea manuală a vânzării



(a) Exemplu decizii deja filtrate după condiția de a fi reparate



(b) Exemplu căutare decizii

Figura 5.20: Exemplu modul vânzări

butonul „View” și de a șterge decizia prin apăsarea iconiței în forma unui coș de gunoi. Opțiunea de a șterge decizia există pentru a se putea reface aceasta în cazul alegerii unei vânzări proaste.

De asemenea, la fel ca la modulul de vânzări, se reprezintă fiecare linie din tabel cu un sistem de culori, regăsit deasupra capului de tabel. Dacă se apasă un element din acesta, se filtrează cererile de pe pagina curentă după elementul respectiv.

Căutarea se pot vedea în figura 5.14(b). Este o combinație între căutarea vânzării și a datelor aflate în decizie. Căutarea după dată caută momentul construirii deciziei. Funcționalitatea este identică când se dorește adăugarea unei decizii soluționate anterior în cadrul „Add Old Decision”, când se modifică decizia vizualizată.

5.3.5.1 Modificarea deciziilor

Interfața de modificare a deciziilor se regăsește la figura 5.21. Se navighează la cerere, vânzare sau mesaje, dar se poate și adăuga o decizie soluționată deja, pentru a calcula automat ce valoare a rămas acoperită pentru repararea produsului pentru polița de asigurare respectivă, folosind următoarele butoane:

1. `Decision` - Navigarea la detaliile deciziei, vizibilă în momentul încărcării paginii.
2. `Claim` - Vizualizarea cererii de despăgubire.
3. `Messages` - Vizualizarea mesajelor interschimbate de utilizatorii aplicației și client.
4. `Add Old Decision` - Adăugarea deciziilor soluționate anterioare. Pentru această interfață se poate vizualiza figura 5.22. După ce se selectează decizia anterioară, trebuie confirmată conform figurii 5.23 înainte de a fi introdusă legătura în baza de date.
5. `Edit Claim` - Legătură rapidă către modificarea datelor cererii de despăgubire. Acest buton a fost adăugat în urma analizei funcționării programului de către utilizatori.

Următoarele câmpuri sunt calculate automat și nu pot fi modificate:

- `Client Price To Pay` - Valoarea de plătit de către client, în cazul care nu este acoperită total repararea sau înlocuirea telefonului de polița de asigurare.
- `Franchise` - În cazul înlocuirii telefonului, franciza se calculează automat în funcție de prețul telefonului.
- `Net Payable` - Suma de plătit către firma de asigurare de operatorii sistemului de gestiune a cererilor de despăgubire.

Dacă soluția este să se înlocuiască produsul (conform câmpului „Resolution”), se va ascunde câmpul
`Repair / Replace Cost`, pentru că suma de înlocuit produsul va fi prețul din câmpul `Payee Cost`.

În partea de jos a paginii se găsește un calculator interactiv, ce lasă utilizatorul să vadă estimarea câmpurilor calculate automat, conform listei de mai sus.

5.3.6 Modulul de raportări

Modulul de raportări se poate accesa prin butonul „Reports” din bara de meniu.

Fiecare raport poate fi descărcat sau vizualizat în cadrul paginii, alegând „Type” -> „Download” sau „View”.

Când se alege descărcarea raportului, apare câmpul „Download Type”. Sunt valabile următoarele opțiuni:

1. `Excel` - Se va descărca un fișier Excel.
2. `CSV` - Se va descărca un fișier în formatul `.csv`.

Pentru fiecare raport se poate specifica o dată de început și sfârșit a perioadei raportului.

În cazul în care lipsește data de început, raportul va începe cu prime elemente ale bazei de date până la data de sfârșit. În cazul în care lipsește data de sfârșit, se vor include în raport toate elemente de la data de început. În cazul lipsei ambelor perioadelor, se generează raportul cu toate datele existente în baza de date.

În momentul apăsării butonului „Submit”, generarea raportului se va vedea în bara de progres.

Conform figurii 5.25, sunt disponibile următoarele rapoarte:

1. `Statistics` - Statistici rapide (câte cereri de despăgubire, decizii au fost create în perioada respectivă de timp). Sunt implicit vizualizate în interfața web, precum în figura 5.24.
2. `Decisions` - Raport cu toate deciziile plătite într-o anumită perioada de timp.
3. `Products` - Raport cu numărul de asigurări, grupate în funcție de tipul de asigurare și prețul ei, în perioada respectivă.
4. `Sales` - Raport cu numărul vânzărilor asigurărilor grupate în funcție de magazinul ce a făcut vânzarea, împreună cu perioada când s-a vândut primul și ultimul produs.
5. `Yearly Statistics` - Raport detaliat despre produsele ce încă sunt asigurate, împreună cu posibila cerere de despăgubire asociată.

   Momentan toate asocierile cu cererile de despăgubire trebuie verificate manual, deoarece nu mai sunt corelate vânzările de cererile de despăgubire.
6. `Monthly Statistics` - Raportul asemănător generat de `Yearly Statistics`, cu câmpul lunii adăugat.

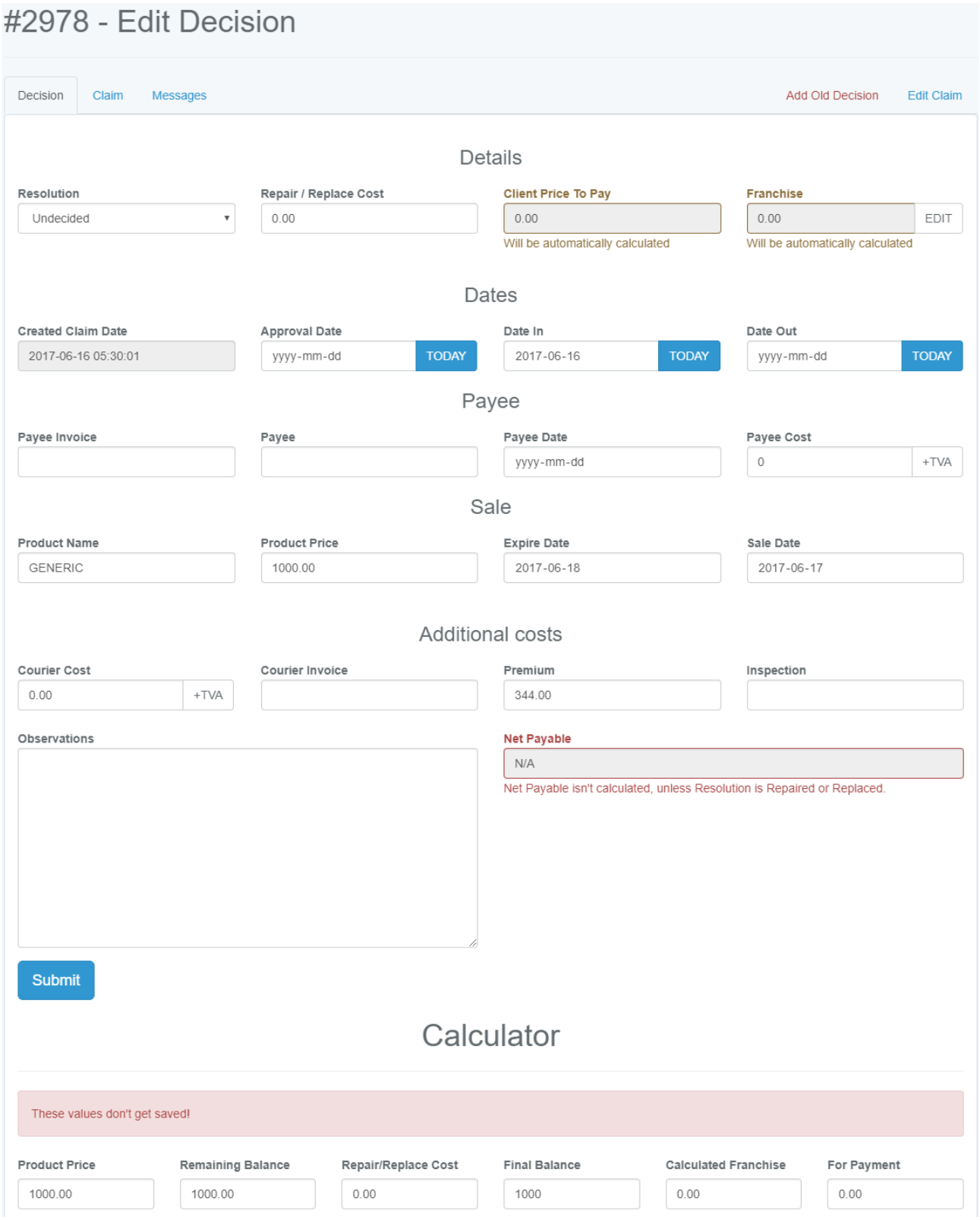


Figura 5.21: Interfața de modificare a deciziilor

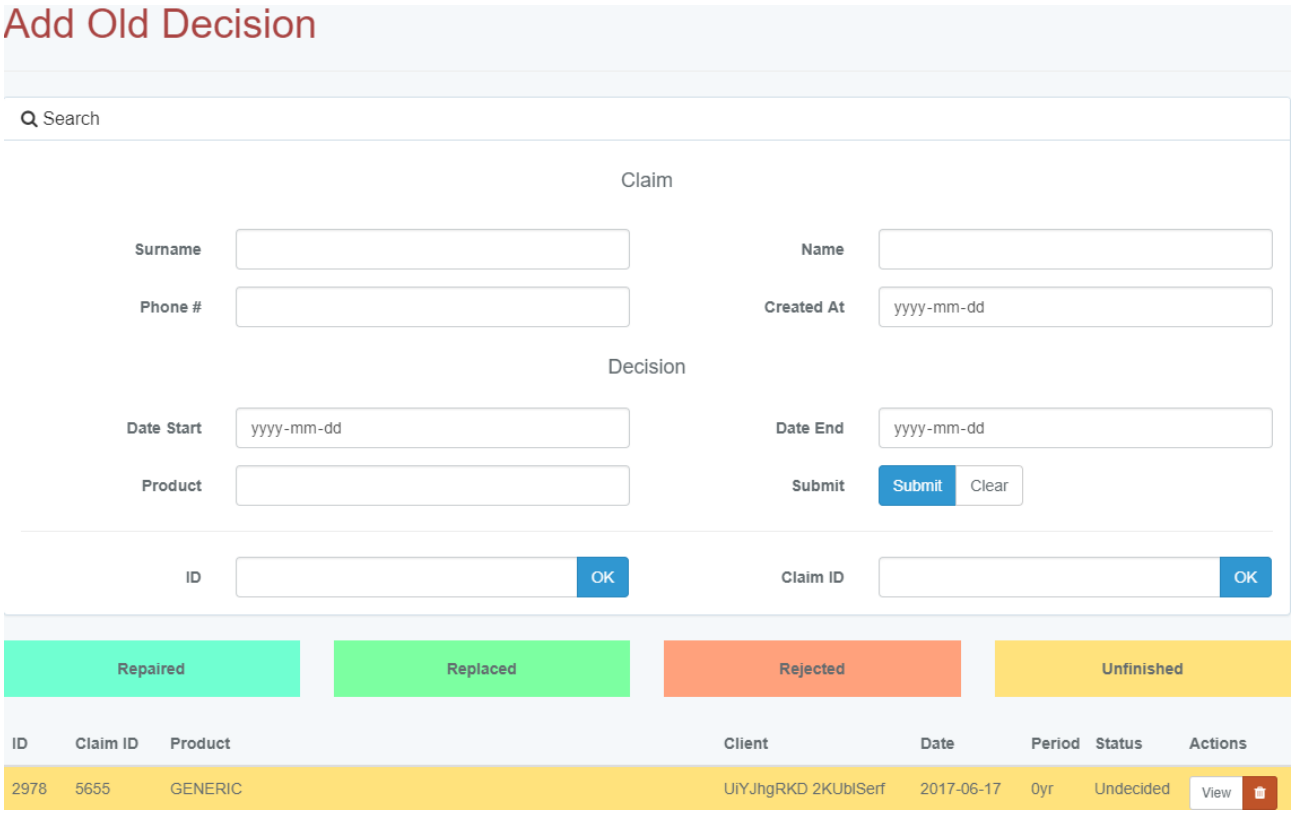


Figura 5.22: Interfața de adăugare a unei decizii soluționate anterior

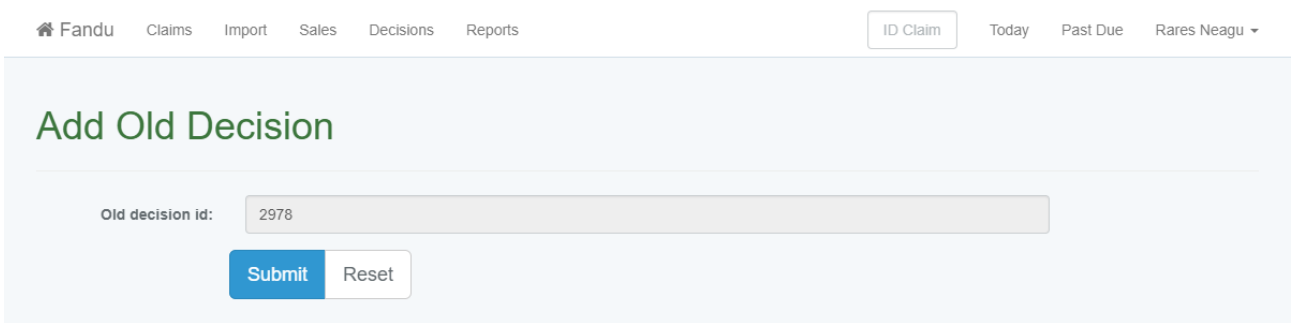


Figura 5.23: Confirmarea adăugării deciziei soluționate anterior

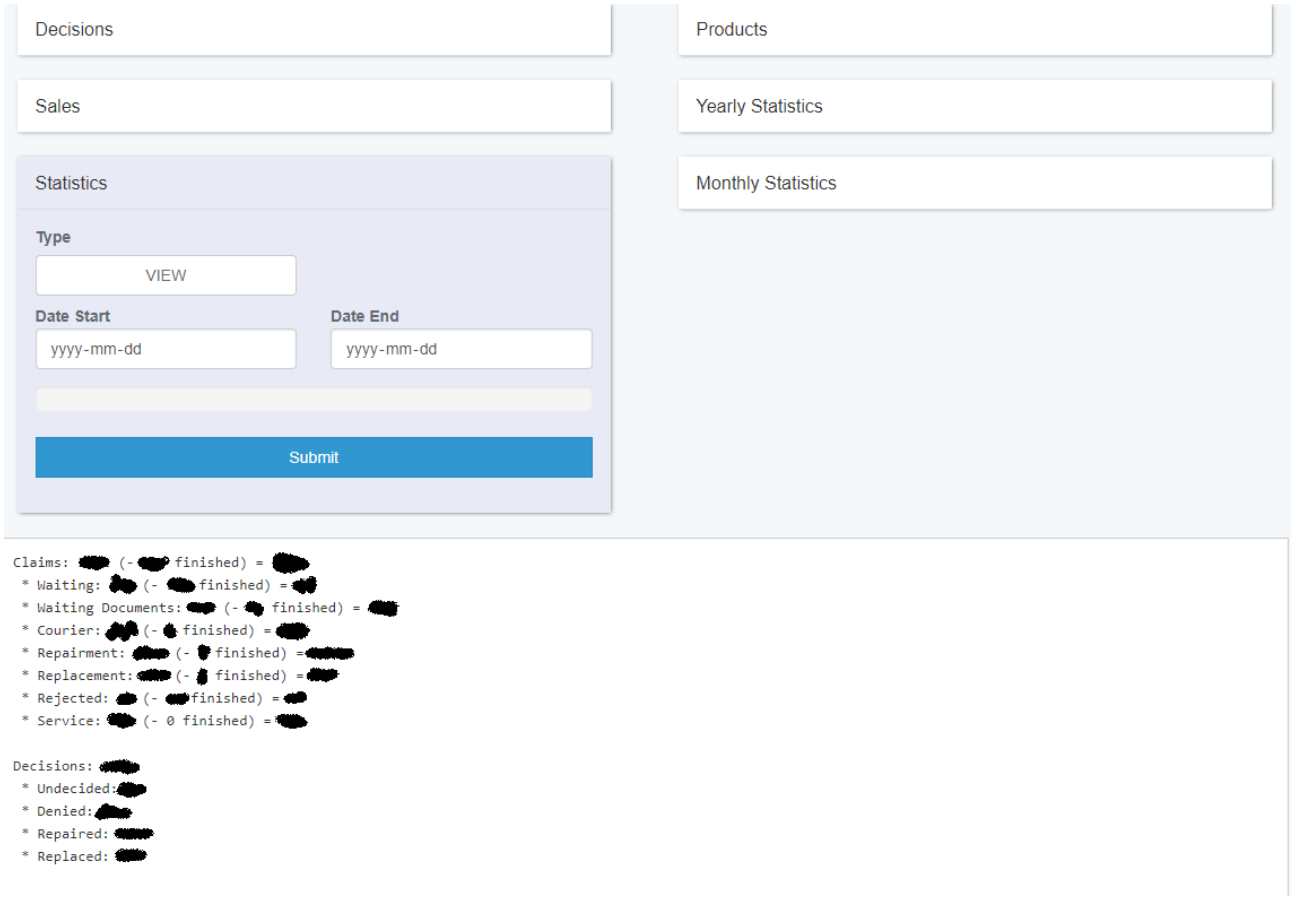


Figura 5.24: Statistici

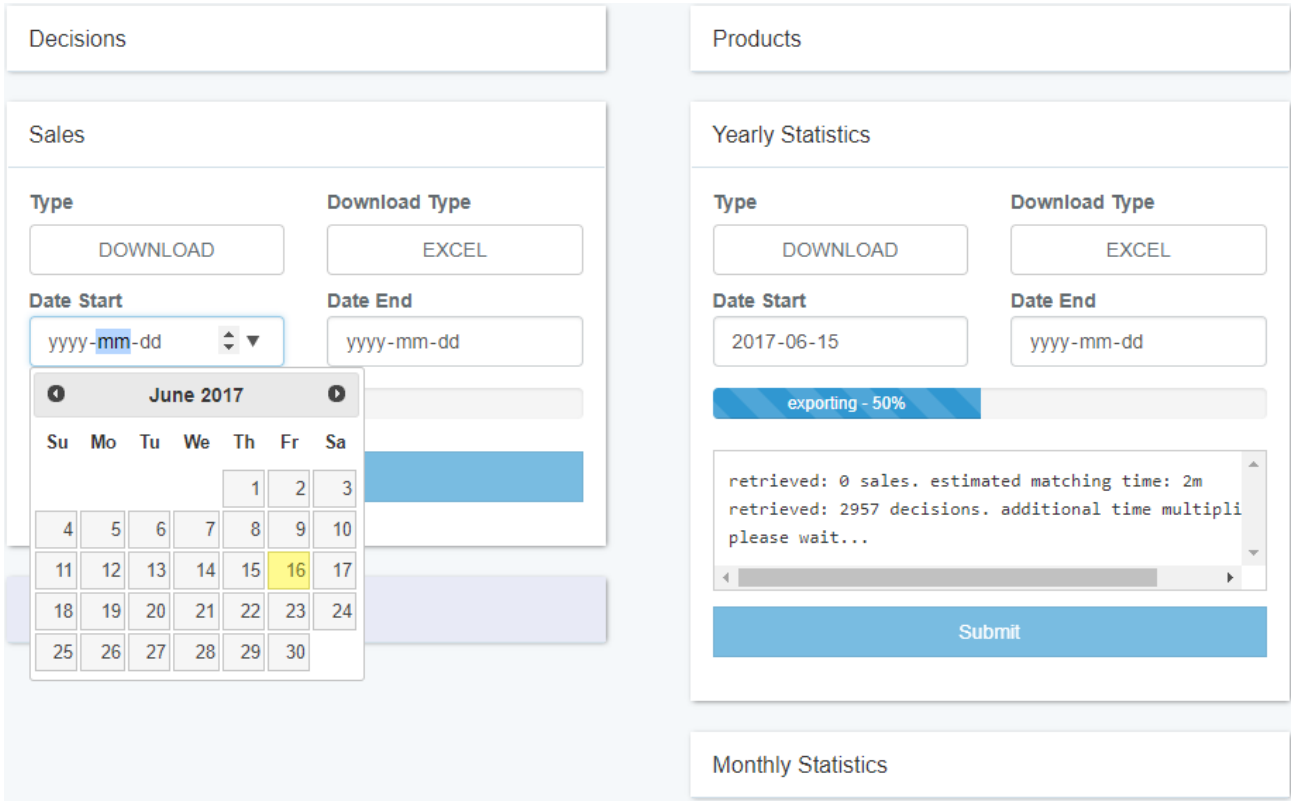


Figura 5.25: Sistemul de raportare

Concluzie

Business Continuity Plan

Teste automate

În lunile care vin, înainte de mă ocupa de sistemul modular de rapoarte, o să doresc să petrec o bună perioadă scriind teste automate.

Testele automate, mai ales „testele unitare”, sunt o metodă de verificare a modulelor individuale, asocierilor datelor, procedurilor de modificare a datelor. [39]

Acestea pornesc de la premiza că un modul detașat de modificările externe, odată testat, se va comporta corect și atunci când va fi folosit în conjuncție cu alte module, ce la rândul lor vor fi testate. Astfel, de la cele mai mici module, se asigură comportamentul corect al aplicației. De la nivelul microscopic de funcție, până la nivelul macroscopic de funcționalitate.

Doresc să extind aplicația să se folosească de codul gata scris de cadrul Laravel și de a obliga aplicația găzduită pe mediul de testare să se asigure că fiecare nouă linie de cod nu strică funcționalitatea deja existentă. Acest principiu se mai numește și „integrare continuă”, propusă prima oară de Grady Booch în 1991 [40] și folosită de majoritatea dezvoltatorilor moderni.

5.4 Modificări în viitorul apropiat

5.4.1 Introducerea câmpului de factură în decizie

Se va reintroduce în viitorul apropiat căutarea automată a deciziilor vechi, prin adăugarea unui câmp cu numărul facturii în cadrul deciziei.

Diferența acum, față de sistemul decuplat, este că pot să aflu ce decizii au fost legate înainte, mulțumită serviciului de backup, dar și a modului actual de legare a informațiilor, pentru că informațiile legate de vânzare nu au fost modificate până acum de mână.

Avantajul existenței opționale a câmpului de factură ajută la funcționarea algoritmului elegant de găsire a deciziilor, pentru că ar fi același număr de factură, adică aceeași vânzare. Trebuie să fie trecut prin prisma utilizatorului sistemului, deoarece s-ar putea referi factura la un alt produs și o altă asigurare. De aceea sistemul ar fi opțional, precum sistemul actual de IMEI duplicat.

5.4.2 Modularizarea tabelelor

5.4.3 Sistem modular de rapoarte

Bibliografie

[1] Altex Romania S.R.L. *Regulament Mobile Protect*. URL: https://mediagalaxy.ro/regulament-mobile-protect (visited on 06/14/2017).

[2] S.C. Clax Telecom SRL. *Hosting | Servere dedicate | VPS - xServers*. URL: https://www.xservers.ro/ (visited on 06/14/2017).

[3] Amazon. *Amazon Web Services S3*. URL: https://aws.amazon.com/s3/ (visited on 06/14/2017).

[4] Taylor Otwell. *Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/ (visited on 06/14/2017).

[5] Taylor Otwell. *Taylor Otwell on Twitter: "Average cyclomatic complexity per method.* URL: https://twitter.com/taylorotwell/status/817494232541839367 (visited on 01/07/2017).

[6] HotFrameworks. *Web framework rankings*. URL: https://hotframeworks.com/ (visited on 06/14/2017).

[7] J. R. Okin. *The Information Revolution*. Ed. Ironbound Press, 2005. ISBN: 0-9763857-4-0.

[8] Anne Nelson and William Nelson. *Building Electronic Commerce with Web-Driven Databases with Cdrom*. 1st. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2001. ISBN: 020174130X.

[9] Rasmus Lerdorf. *Rasmus Lerdorf, Senior Technical Yahoo: PHP, Behind the Mic*. URL: https://web.archive.org/web/20130728125152/http://itc.conversationsnetwork.org/shows/detail58.html (visited on 11/19/2003).

[10] The PHP Group. *PHP: Supported Versions*. URL: http://php.net/supported-versions.php.

[11] Martin Bean. *Laravel 5 Essentials*. books.google.com, 2015. ISBN: 978-1785283017.

[12] Frank Buschmann. *Pattern-Oriented Software Architecture*. Volume 1. Wiley, 1996. ISBN: 978-0471958697.

[13] Adam Engebretson. *About The Collective*. URL: https://laravelcollective.com/about (visited on 06/14/2017).

[14] Laravel Collective. *Forms & HTML*. URL: https://laravelcollective.com/docs/master/html (visited on 06/14/2017).

[15] Torsten Stanienda and Douglas Barry. "Solving the Java Object Storage Problem". In: *Computer* 31 (1998), pp. 33–40. ISSN: 0018-9162. DOI: doi.ieeecomputersociety.org/10.1109/2.730734.

[16] M. Fowler. *Patterns of Enterprise Application Architecture*. A Martin Fowler signature book. Addison-Wesley, 2003. ISBN: 9780321127426. URL: https://books.google.ro/books?id=FyWZt5DdvFkC.

[17] PHP Framework Interop Group. *PSR-1: Basic Coding Standard - PHP-FIG*. URL: http://www.php-fig.org/psr/psr-1/ (visited on 06/14/2017).

[18] Michael Widenius and Davis Axmark. *Mysql Reference Manual*. Ed. by Paul DuBois. 1st. Sebastopol, CA, USA: O'Reilly & Associates, Inc., 2002. ISBN: 0596002653.

[19] Oracle. *Oracle Announces the Acquisition of Open Source Software Company, Innobase*. Oct. 7, 2005. URL: http://www.oracle.com/us/corporate/press/016679_EN (visited on 09/16/2012).

[20] Carlos Coronel and Steven Morris. *Database Systems: Design, Implementation, & Management*. 11th ed. Course Technology, Feb. 2014. ISBN: 9781285196145.

[21] Marc Delisle. *Mastering phpMyAdmin 3.4 for Effective MySQL Management*. Packt Publishing, 2012. ISBN: 9781849517782.

[22] Bryan O'Sullivan. *Mercurial: the Definitive Guide*. Sebastopol: O'Reilly Media, Inc., 2009. ISBN: 9780596555474.

[23] Kathryn D. Scopatz Anthony; Huff. *Effective Computation in Physics*. O'Reilly Media, Inc., 2015. ISBN: 9781491901595.

[24] Alex William. *GitHub Pours Energies into Enterprise. Raises $100 Million From Power VC Andreessen Horowitz*. URL: http://techcrunch.com/2012/07/09/github-pours-energies-into-enterprise-raises-100-million-from-power-vc-andreesen-horowitz/ (visited on 07/09/2012).

[25] Chromium. *685826 - Restrict the set of domains for WoSign/StartCom certificates*. Jan. 26, 2017. URL: https://bugs.chromium.org/p/chromium/issues/detail?id=685826 (visited on 04/28/2017).

[26] CloudFlare. *Secure Socket Layer (SSL)/TLS*. URL: https://www.cloudflare.com/ssl/ (visited on 06/14/2017).

[27] Jon Duckett. *JavaScript & jQuery : interactive front-end web development*. Indianapolis, IN: Wiley, 2014. ISBN: 1118531647.

[28] Eric Sarrion. *JQuery UI*. Sebastopol, Calif: O'Reilly, 2012. ISBN: 1449316999.

[29] 1000hz. *Bootstrap Validator*. URL: http://1000hz.github.io/bootstrap-validator/ (visited on 06/14/2017).

[30] Timothy Moran. *Mastering KnockoutJS : use and extend Knockout to deliver feature-rich, modern web applications*. Birmingham, UK: Packt Publishing, 2014. ISBN: 1783981008.

[31] knockoutjs.com. *Knockout*. URL: http://knockoutjs.com/ (visited on 06/14/2017).

[32] Feras Alhlou. *Google analytics breakthrough : from zero to business impact*. Hoboken, New Jersey: John Wiley & Sons, Inc, 2016. ISBN: 1119144019.

[33] Jordi Boggiano Nils Adermann and many community contributions. *Composer*. URL: https://getcomposer.org/ (visited on 06/14/2017).

[34] Isaac Z. Schlueter. *Forget CommonJS. It's dead. **We are server side JavaScript.**.* Mar. 25, 2013. URL: https://github.com/joyent/node/issues/5132#issuecomment-15432598 (visited on 06/14/2017).

[35] Stefan Kottwitz. *LaTeX beginner's guide*. Birmingham, UK: Packt, 2011. ISBN: 1847199860.

[36] Robert Martin. *Agile software development, principles, patterns, and practices*. Harlow, Essex: Pearson, 2014. ISBN: 1292025948.

[37] Taylor Otwell. *Controllers - Laravel - The PHP Framework For Web Artisans*. URL: https://laravel.com/docs/5.3/controllers (visited on 06/14/2017).

[38] Taylor Otwell. *Eloquent: Relationships - Laravel - The PHP Framework For Web Artisans*. URL: `https://laravel.com/docs/5.3/eloquent-relationships` (visited on 06/14/2017).

[39] Dorota Huizinga. *Automated defect prevention : best practices in software management*. Hoboken, N.J: Wiley-Interscience IEEE Computer Society, 2007. ISBN: 0-470-04212-5.

[40] Grady Booch. *Object oriented design with applications*. Redwood City, Calif: Benjamin/Cummings Pub. Co, 1991. ISBN: 9780805300918.