Universitatea din București

Facultatea de Matematic**ă** și Informatic**ă**

LUCRARE DE LICENȚĂ

Cassandra Interface – Aplicație Web

COORDONATOR ȘTIINȚIFIC STUDENT

Conf. dr. Alin Ștefănescu Lina Luca-Cristian

București 2019

Curpins

[Introducere 3](#_Toc10819612)

[Tehnologiile si arhitecturile folosite 5](#_Toc10819613)

[Apache Cassandra 3.11.3 5](#_Toc10819614)

[Java 8 8](#_Toc10819615)

[Spring, Spring Boot 2.0.5 si Spring Security 9](#_Toc10819616)

[Thymeleaf 14](#_Toc10819617)

[HTML si HTML5 15](#_Toc10819618)

[CSS si Bootstrap 16](#_Toc10819619)

[Javascript si Jquery 17](#_Toc10819620)

[Prezentarea aplicatiei 18](#_Toc10819621)

[Structura 18](#_Toc10819622)

[Securitate 19](#_Toc10819623)

[Client-Side 23](#_Toc10819624)

[Server-Side 33](#_Toc10819625)

[Librarii / Plugin-uri folosite 44](#_Toc10819626)

[Concluzii 47](#_Toc10819627)

[Lista figuri 48](#_Toc10819628)

[Bibliografie 50](#_Toc10819629)

Introducere

Ideea aplicației pleacă de la tehnologia Apache Cassandra, o bază de date NoSQL (not only SQL) destinată gestionării cantităților mari de date. Din păcate interactionarea cu această bază de date de către un programator nu este foarte complexă.

Folosind limbajul CQL (Cassandra Query Language), un programator poate prelua date sau executa comenzi asupra bazei de date. Din păcate, din punct de vedere vizual și din punct de vedere al eficienței, această procedură nu este una foarte benefică. Interfața constă într-o consolă care primește comenzile CQL și returnează rezultatul. Astfel nu există nicio modalitate de a avea o privire de ansamblu a bazei de date fără a trece prin mai mulți pași inevitabili.

Am ales să dezvolt această aplicație în special pentru acest motiv, de a eficientiza gestionarea acestei baze de date printr-o interfață prietenoasă, eficientă și accesibilă oricui (de aici și numele aplicației).

Am ales ca acesta aplicație să fie web în primul rând pentru ușurința și accesibilitatea pe care o oferă interacțiunii și comunicării între membri. În al doilea rând, pentru a introduce aplicația mai ușor în mediul online, care după părerea mea este cel mai eficient mod de a promova o creație, mai ales aplicații în genul acesteia.

Având ca exemplu negativ interfața curentă a bazei de date, a fost ușor de imaginat o aplicație eficientă care să-i ocupe locul. Astfel, comparația dintre cele două interfețe poate fi destul de semnificativă.

\*De acum înainte voi folosi termenul de “interfata veche” și “interfata noua” pentru a face referire la interfața curentă a bazei de date Cassandra, reprezentată de consola de tip terminal, respectiv aplicația(interfața) prezentată, Cassandra Interface.

Spre deosebire de interfața veche, care putea fi accesată numai prin query-uri, interfața nouă oferă atât facilitatea de a executa query-uri de tip text, cât și accesarea/executarea datelor prin doar câteva click-uri. Acesta conține câte o pagină specială pentru acestea: “View/Edit”, “Console/Script”, prin care utilizatorul poate face (într-un procent destul de mare) cam tot ceea ce poate face și interfața veche, însă mult mai ușor.

Altă diferență seminificativa, este aceea că interfața veche nu oferă siguranță și comunicarea eficientă între doi sau mai mulți utilizatori care au acces la aceleași date. Cassandra Interface oferă această facilitate prin simplul fapt că orice schimbare asupra bazei de date este salvată în Log și se poate accesa de către membrii cu rol superior în orice moment. De asemenea orice schimbare de update sau delete asupra bazei de date creează automat un backup. Pe lângă aceste lucruri, utilizatorii cu acces la aceleași date, pot avea roluri și acces diferit asupra lor, făcând totul să se desfășoare într-un mediu sigur. Astfel, siguranța datelor în participarea colectivă este asigurată.

Principalul scop în dezvoltarea acestei aplicații fiind eficienizarea din punct de vedere vizual și temporal, Cassandra Interface oferă acest lucru prin fiecare componentă a aplicației. Fiecare pagină în parte oferă ceva aparte utilizatorului, care poate beneficia ce acel lucru într-un timp cât mai scurt.

Pentru a putea beneficia de calitate în dezvoltarea acestei aplicații atât din punct de vedere al programatorului cât și din perspectiva unui utilizator, am folosit tehnologii complexe în acest sens.

Din motive evidente, baza de date folosită pentru aplicație este Apache Cassandra, împreună cu limbajul CQL.

Ca limbaj principal de programare pentru backend am folosit Java 8, fiind una dintre cele mai bune tehnologii în dezvoltarea aplicațiilor web. De asemenea, este și o preferință subiectivă. Împreună cu aceasta, am folosit framework-ul Spring Boot integrat pentru Java, oferind o ușurință și o calitate mai mare în dezvoltarea aplicației. Pentru o securitate eficace am folosit o tehnologie derviata din Spring și anume Spring Security.

Comunicarea între backend și frontend este realizată prin tehnologia Thymeleaf, o librărie din Java, care aplică transformări și procesează template-urile afișând datele produse de aplicație. De asemenea, a fost posibilă integrarea cu Spring Security și crearea template-urilor de tip layout prin adăugarea unor dialecte la această librărie.

Pentru frontend, fiind o aplicație web, am folosit HTML5, împreună cu elementele de design alte CSS-ului și Bootstrap-ului și crearea de scripturi în Javascript și Jquery.

În continuare urmează descrierea în amănunt a tehnologiilor folosite și a aplicației.

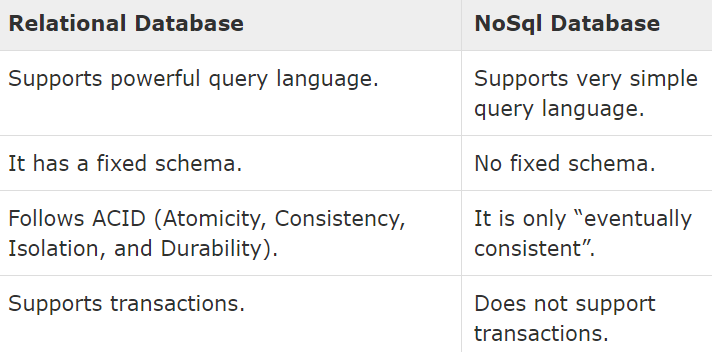
Tehnologiile si arhitecturile folosite

Apache Cassandra 3.11.3

Apache Cassandra este o bază de date “open-source”, distribuită (bază de date în care nu toate dispozitivele de stocare sunt atașate la același procesor, fiind posibilă stocarea pe mai multe calculatoare fie locale, fie dispersate într-o rețea de calculatoare interconectate [15]), non-relaționară (NoSQL) de tip “wide column store”, putând fi interpretată ca o depozitare cheie-valoare în două dimensiuni [16], creată pentru a putea face față unui număr foarte mare de date [3].

Avinash Lakshman și Prashant Malik au dezolvtat inițial Cassandra la Facebook pentru a îmbunătăți motorul de căutare, apoi Facebook a lansat oficial Cassandra ca proiect “open source” la codul Google în iulie 2008 [3].

O bază de date NoSQL prezintă un mod de a depozita și recupera date diferit față de mecanismul tradițional al bazelor de date relaționare, având avantajul procesării unui număr foarte mare de date mult mai eficient [17].



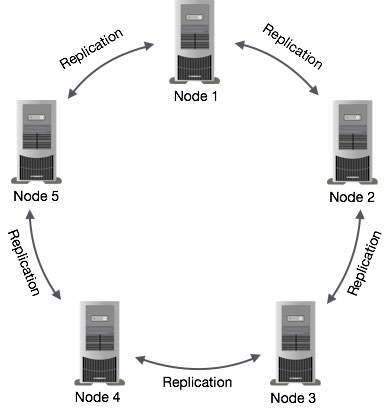
[17]

Cassandra a devenit populară datorită numeroaselor caracteristici de care dispune, cum ar fi capacitatea foarte mare de scalabilitate, având posibilitatea de adăugare de echipament pentru mai multe date. Pe lângă aceasta, Apache Cassandra este făcută în așa fel încât nu poate avea niciun punct de eroare, fiind disponibilă în orice moment. De asemenea, cu cât baza de date este mai complexă în arhitectură cu atât viteza este mai bună.

Spre deosebire de alte baze de date NoSQL, Cassandra a fost dezvoltată pentru “fast writes”, fiind capabilă să stocheze sute de tera de date fără a îngreuna citirea acestora. Un alt atribut este și flexibilitatea tipruilor de date, putând depozita toate formatele de tipuri de date, incluzând cele structurate și semi-structurate.

Arhitectura constă într-un sistem “peer-to-peer” distribuit de-a lungul mai multor noduri (în care sunt distribuite la rândul lor datele), aceste noduri aparținând de un cluster (grup). Un cluster conține unul sau mai multe centre de date (un centru de date este format din mai multe noduri). Fiecare nod dintr-un cluster are același rol și este independent și în același timp interconectat cu celelalte noduri. Atributul principal este că fiecare nod poate accepta cereri de scriere/citire, iar dacă un nod pățește ceva, aceste cereri pot fi satisfăcute de celealte noduri.

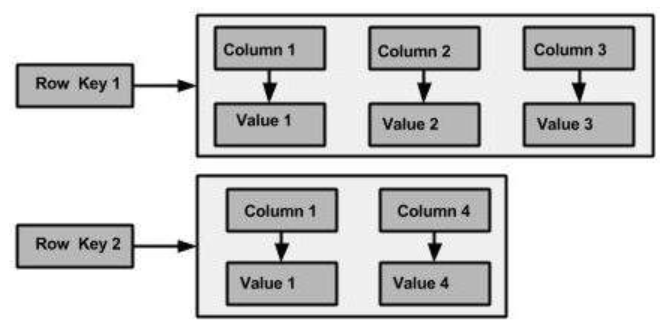
Cassandra lucrează prin replicarea datelor, astfel unul sau mai multe noduri dintr-un cluster conține replicări pentru o anumită bucată din datele din acel cluster. Fiecare cerere de scriere este captată de un log din fiecare nod, apoi detele sunt trimise la “mem-table”, un tabel de memorie unde sunt trimise temporar datele, înainte de a fi partiționate și replicate în cluster. Dacă “mem-table” este plin, datele se vor stoca de asemenea temporar în “SStable”, un fișier pe disk, pe care Cassandra îl va interoga periodic pentru a elimina datele inutile. Imaginea de mai jos arată o schemă de noduri într-un cluster și cum funcționează replicarea astfel încât să nu existe niciun fel de eroare [17].



[17]

Termenul de keyspace reprezintă un container pentru o listă de una mai mai multe “column families” (structură în care sunt depozitate datele – tabel). Ca și atribute, un keyspace poate fi determinat atât de un factor de replicare, care reprezintă numărul de copieri a datelor în clusterul din care face parte, cât și de strategia pentru a replica aceste date.

O familie de coloane la rândul ei este un container de colecții de linii, fiecare linie conținând coloane ordonate, fiecare linie având cel puțin o coloană ca făcând parte din “partion key” (cheia primară). Cheia primară poate fi simplă sau compusă (din mai multe coloane) și este responsabilă pentru distribuirea datelor în partiția curentă, astfel căutarea se va face implicit după această cheie. Pentru o căutare în care nu este menționată cheia primară, performanța va scădea senimificativ; în acest caz, pentru unele coloane care nu fac parte din cheia primară, se introduce termenul de index pe coloana respectivă, adaugandu-i anumite atribuite pentru a facilita căutarea. Pe lângă cheia primară mai există și coloane “clustering key”, care sunt reponsabile pentru sortarea datelor în partiție. Figura de mai jos reprezintă schema unei familii de coloane [17].



[17]

Apache Cassandra oferă un limbaj ca suport sub forma unui terminal pentru comunicarea între ușer și baza de date, și anume CQL (Cassandra Query Language), având posibilitatea de a crea keyspace-uri, insera date sau executa diferite query-uri [17]. Atât sintaxa limbajului cât și tipurile de date seamănă destul de mult cu cea de la bazele de date relaționare cum ar fi SQL, astfel există comenzi asemănătoare pentru: selectare, creare, alterare, ștergere, inserare, actualizare, etc. Pe lângă tipurile de date cunoscute cum ar fi: text, int, boolean, date, etc, Cassandra introduce termeunul de “frozen”, un tip de dată care acceptă atât tipuri create de utilizator (UDT – user data type), cât și colecții cunoscute ca liste, seturi și hărți. Ca și în alte baze de date, aceasta oferă posibilitatea creării de funcții și triggere asupra tabelelor. Mai mult decât atât, căutarea fiind restricționată de cheia primară, se poate crea un “materialized view”, adică un alt tabel conectat cu tabelul de bază, în care pe lângă cheia primară inititala, se adaugă și alte coloane la aceasta, făcând posibilă căutarea după acestea (fără a altera tabelul inițial, iar modificările sunt sincronizate). Mai jos sunt prezentate câteva exemple de sintaxa pentru acest limbaj.

CREATE KEYSPACE IF NOT EXISTS admin WITH REPLICATION = { ‘class’ : ‘SimpleStrategy’, ‘replication\_factor’ : 3 } AND DURABLE\_WRITES = true;

CREATE TABLE IF NOT EXISTS admin.users (user\_id text PRIMARY KEY, age int, name frozen<fullname>, data list<text>);

CREATE TYPE admin.fullname (first\_name text, last\_name text);

INSERT INTO admin.users (id,age) values (‘1’,21);

CREATE MATERIALIZED VIEW IF NOT EXISTS admin.users\_by\_age

AS SELECT \* FROM admin.users

WHERE user\_id IS NOT NULL AND age IS NOT NULL

PRIMARY KEY (age, user\_id);

SELECT (age,user\_id) FROM admin.users\_by\_age;

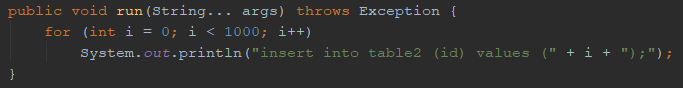
Motivul pentru care am ales această bază de date este prezentat în Introducere.

Java 8

Java este un limbaj de programare orientat pe obiecte, creat de către James Gosling la începutul anilor ’90, fiind lansat în 1995. Este un limbaj cu o întindere largă, fiind folosit în numeroase aplicații distribuite, incluzând cele mobile.

În sintaxa limbajului se regăsesc o mare parte din elemente de C sau C++, dar modelul obiectelor este mai simplu și are mai puține facilități de nivel jos. Ca de exemplu, Java nu are supraîncărcarea operatorului, iar moștenirea multiplă este valabilă doar pentru interfețe nu și pentru clase. Avantajul major este că un program Java corect și compilat, poate fi rulat fără modificări pe orice platformă care are instalată o mașină virtuală Java. Acest lucru este posibil deoarece sursele sunt compilate într-un format numit cod de octeți, acesta fiind independent de tipul platformei.

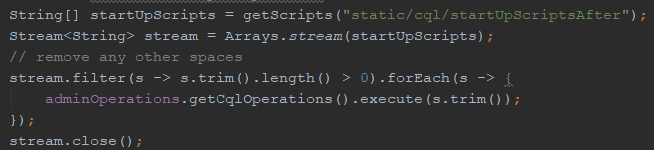
Pentru a demonstra aceste referințe, urmează o imagine sustrasă din proiect, care conține un bloc de cod care se apelează la pornirea serverului și afișează în consola acestuia (System.out.println) 1000 de linii (for de la 0 la 999), fiecare conținând un query de inserare pentru baza de date. Acest bloc a fost folosit pentru testare și nu are un rol funcțional în aplicație.



Dintre cele 4 platforme furnizate de Oracle (Java Card, Java ME – micro edition, SE – standard edition și EE – Enterprise Edition, aplicația Cassandra Interface folosește versiunea 8 a platformei standard, care este destinantă sistemelor de tip workstation [4].

Java SE 8 a apărut pe 18 martie 2014 și este considerată cea mai mare actualizare, modificările asigurând faptul că Java rămâne o prioritate la nivelul aplicatilor de întreprindere. Spre deosebire de celelalte versiuni, acesta aduce îmbunătățiri seminificative.

Printre acestea se numără expresiile lamda, care sunt considerate cele mai mari caracteristici în Java 8. Aceste expresii acționează ca o funcție anonimă, într-o bucată de cod foarte scurtă, ajutând astfel și la coding style. De asemenea, tot în acesta versiune au fost introduse stream-urile, o altă actualizare majoră, care vine cu diferite tipuri de operațiuni de flux (filtrare, sortare, etc), folosite în mod eficient împreună cu expresiile lamda [5]. Urmează o imagine din aplicație care conține un astfel de exemplu, care pentru un vector de stringuri, îl filtrează, eliminând stringurile vide, iar pentru fiecare dintre acestea, execută o anumită comandă.



Am ales acest limbaj în primul rând pentru performanța pe care o oferă atât ca viteză și ușurință de scriere, dar și pentru calitatea dovedită de-a lungul anilor, fiind platformă pentru milioane de dispozitive. În al doilea rând pentru diversitatea de care dispune acest limbaj, având posibilitatea de integrare cu aproape orice domeniu.

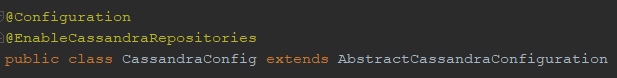
Spring, Spring Boot 2.0.5 și Spring Security

Spring este un framework de aplicații pentru platforma Java, iar prima versiune a sa a apărut pe 1 octombrie 2002 și a fost scrisă de Rob Johnson. Framework-ul poate fi folosit de orice aplicație Java, însă principalul său atribut facilitează dezvoltarea aplicațiilor web.

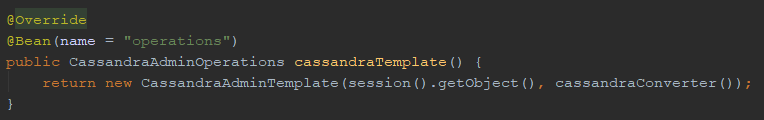
Specific pentru Spring este container-ul Ioc (inversion of control), care produce configurații și se ocupă de obiectele din Java cu ajutorul reflexiei (abilitatea unui proces de a-și examina și modifica propria structură și comportament [8]). Container-ul se ocupă de organizarea duratei de viață (creare, apelare, configurare, etc) a obiectelor care se mai numesc “beans” și poate fi configurat cu ajutorul unui fișier XML (Extensible Markup Language) sau prin detectarea unor adnotări specifice asupra claselor de configurare. Aceste obiecte la rândul lor pot fi obținute printr-o cerere către container pentru un obiect cu un anumit nume sau de un anumit tip (dependency lookup) sau prin-o metodă care trimite obiectul după nume într-un alt obiect (dependency injection) [6]. Imaginea de mai jos prezintă a două metodă, în care obiectul (bean-ul) UserService creat și stocat în contextul Spring, este “injectat” într-un obiect într-o clasă cu ajutorul adnotarii @Autowired.



Spring are propriul AOP (aspect-oriented programming), care modularizeaza relațiile de tip cross-cutting în aspecte. Cu alte cuvinte, spre deosebire de modulele tradiționale care încapsulează obiectele și conțin codul necesar pentru ca acestea să implementeze serciviile sitemului, modulele AOP conțin codul necesar implementării unui singur serviciu a sistemului, considerând toate obiectele posibile [9]. În imaginea următoare, se poate observa o clasă pe care framework-ul o va recunoaște ca fiind una de configurare și o va manipula în consecință (o va instanția ca un bean) datorită adnotației corespunzătoare.



Altă caracteristică importantă este accesarea datelor în Spring. Acesta oferă suport pentru cele mai populare framework-uri de acces de date din Java cum ar fi Cassandra (în cazul de față), JDBC, Hibernate, JDO, JPA, etc. De asemenea, tranzacțiile au o diversitate destul de mare, de la tranzacții locale și globale până la tranzacții în aproape toate mediile de platforme din Java [6]. Imaginea de mai jos arată crearea unui Bean pentru un obiect dintr-o clasă de configurare a bazei de date Cassandra și anume CassandraAdminTemplate, prin care se va permite executarea comenzilor și preluarea datelor asupra bazei de date.



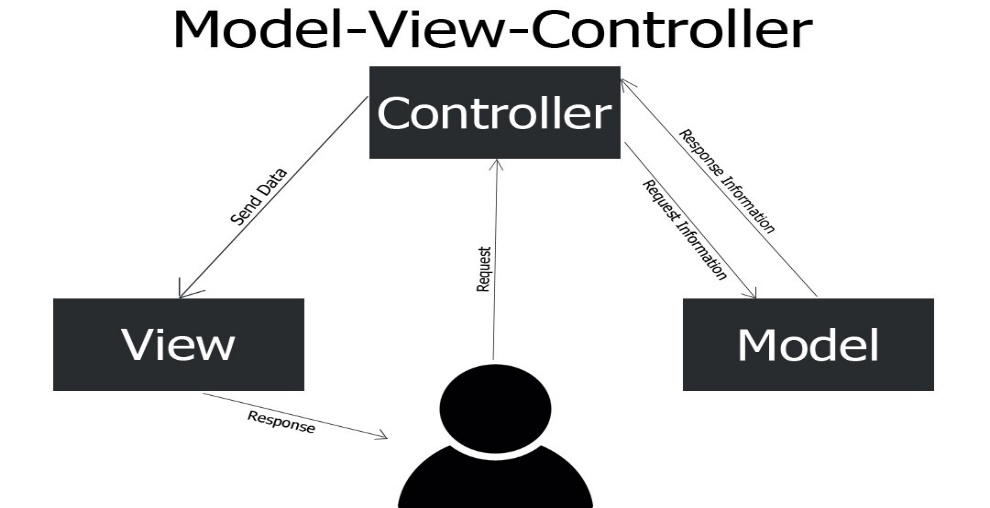
Framework-ul Spring folosește ca arhitectură propriul MVC (Model-View-Controller) pentru aplicații web. Această arhitectură este folosită în mare parte pentru dezvoltarea interfețelor pentru utilizatori și împarte aplicația în trei părți interconectate:

Model, componentă centrală, independentă de interfață și care se ocupă de logica și regulile cu privire la datele aplicației (bază de date);

View, reprezentând informația vizuală pe care o vede utilizatorul, partea de frontend;

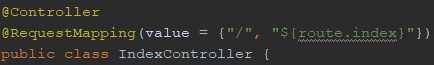
Controller, care primește intrări și trimite mai departe către model sau view.

Cu alte cuvinte, modelul este responsabil pentru organizarea datelor aplicației și primește intrări din view cu ajutorul controllerului, view-ul reprezentând astfel prezentarea modelului într-un format aparte, iar controllerul răspunde intrărilor din view, opțional le validează și acționează în consecință asupra modelului [10].

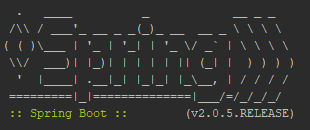


[11]

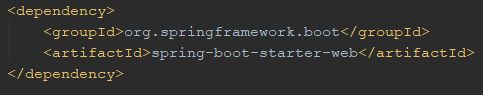
Figura următoare arată cum framework-ul va recunoaște, printr-o adnotare, o clasă ca fiind o componentă a arhitecturii MVC și anume controller și va trata întreaga clasă cu componentele ei ca atare.



Spring Boot reprezintă o soluție bazată pe o convenție de configurații considerate ca fiind unele dintre cele mai bune din platforma Spring pentru crearea de aplicații “stand alone”, aplicații care pornesc de la o configurare minimă și eficientă pentru a putea rula cât mai rapid (“just run”). Acesta oferă simplitate, configurând automat când este cazul și folosind Project Object Models (POMs), simplificând astfel configurațiile Maven [6].



Un POM este un fișier XML care conține informații despre proiect și despre configurațiile ce contribuie la construcția acestuia. În acest fișier pot fi incluse toate dependințele proiectului, plugin-urile, versiunile acestora și multe altele, oferind astfel atât simplitate cât și complexitate în dezvoltarea aplicației [13]. Imaginea următoare reprezintă o dependință Maven din POM-ul proiectului care permite integrarea web cu framework-ul Spring Boot.

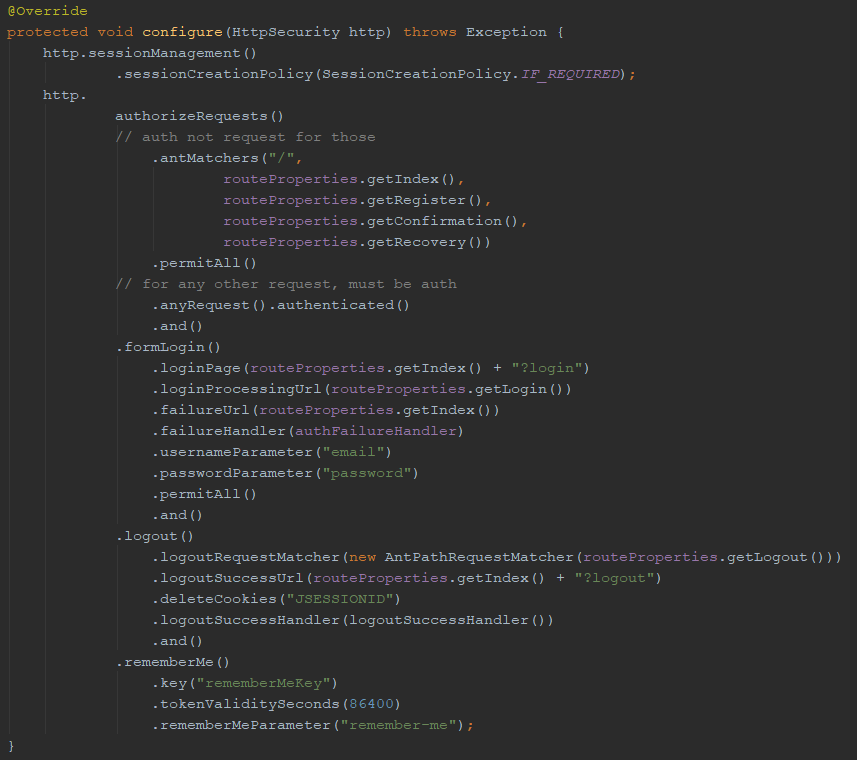


Spring Boot cuprinde implicit și server-ul pe care va rula aplicația, și anume Tomcat. Apache Tomcat este un mediu de server web în care un program în Java poate rula, implementând implicit câteva specificații din Java EE incluzând Java Servlet, JSP (JavaServer Page), Java EL și WebSocket [12], fiecare făcând parte din componentele unei aplicații web. Figura următoare arată inițializarea server-ului la pornirea Spring Boot-ului.

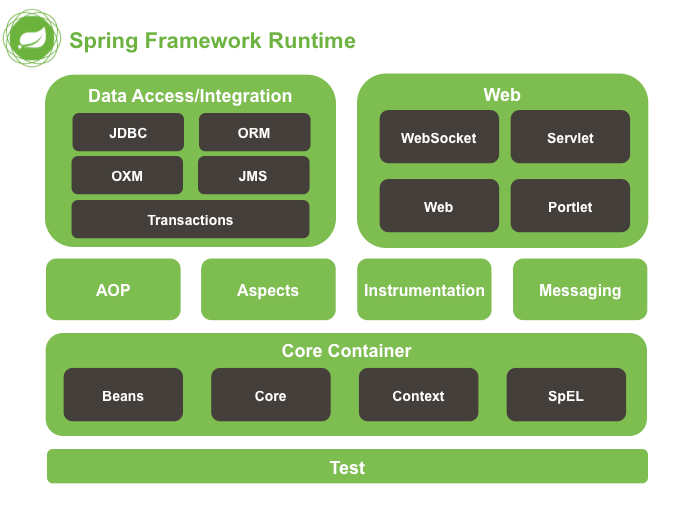


Spring Security este un framework care aduce conceptul de autentificare și autorizare pentru securizarea aplicațiilor Java. Autentficiarea reprezintă procesul de identificare a utilizatorului ce vrea să acceseze aplicația, iar autorizarea se ocupă de accesul în executarea acțiunilor asupra ei (cereri web, metode, acces pe un anumit domeniu, etc). Alte avantaje constau în suportul oferit pentru configurările în Java, posibilitatea de integrare cu Spring MVC, portabilitate, CSRF (Cross-site request forgery) protection (diferite atacuri asupra aplicației prin comenzi neautorizate provenite de la un utilizator în care aplicația are încredere [20]), etc [19].

Imaginea de mai jos reprezintă funcția de configurare a framework-ului pentru Cassandra Interface. Acesta configurare constă în restricționarea accesului asupra paginilor web ale aplicației utilizatorilor care nu sunt autentificati; în acest caz, accesul lor va fi limitat la pagina de pornire și la cele care țin de autentificare. De asemenea apar și setările subiective pentru pagina de login și logout (link-uri, handlers, parametrii); în plus, se configurează și facilitatea de “remember-me”, care memorează user-ul la autentificare pentru un anumit interval de timp.



Am ales acest framework pentru performanța și complexitatea de care dispune, fiind unul dintre cele mai folosite framework-uri pentru aplicații web în Java.



[14]

Thymeleaf

Thymeleaf este o librărie din Java pentru template-uri XML/XHTML/HTML5, care poate rula atât pe un mediu web cât și pe un mediu non-web, aplicând transformări asupra acestor template-uri în urma cărora se pot afișa date produse de aplicație [21], permițând astfel “comunicarea” între backend și frontend.

Arhitectura Thymeleaf permite procesarea rapidă prin parsarea fișierelor astfel încât să se execute cât mai puține operații.

O puternică unealtă a acestei librării o constituie extensibilitatea acestuia, permițând definirea anumitor noduri (DOM – Document Object Model), care împreună cu alte “artefacte”, formează un dialect [2]. Thymeleaf oferă default dialectul Standard, însă mai pot fi adăugate și altele.

În imaginea de mai jos se întâmplă următoarele: prin aplicarea precompilarii Thymeleaf cu dialectul Standard, acest element de HTML (<div>) va fi atașat paginii, numai în cazul în care în request va exista obiectul param.login (th:if). În caz afirmativ, conținutul div-ului va fi înlocuit cu mesajul corespunzător cheii “login.redirect.message” din fișierul de mesaje al aplicației (th:text).



Alt dialect foarte folosit este cel pentru Layout, care permite “decorarea” paginilor după un anumit layout, fiind foarte eficient pentru a nu duplica același cod pentru fiecare pagină în parte.

Prin această imagine este reprezentată sintaxa dialectului Layout care spune librăriei că acesta pagina HTML este layout pentru alte pagini.



Iar prin acesta sintaxă, Thymeleaf va știi că acest bloc HTML este suport (fragment) pentru pagina care va “decora” acest layout. Astfel, dacă într-o pagină care folosește acest layout există același fragment dar cu conținut, acel conținut va fi înlocuit în pagina de layout în fragmentul respectiv.



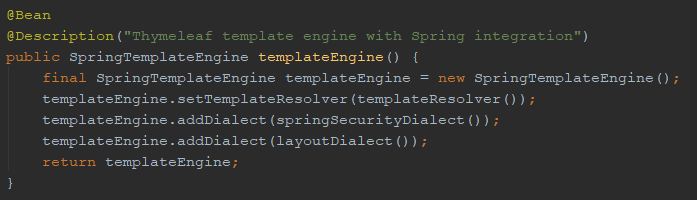
Dialectul pentru Spring Security este de asemenea folosit. Prin integrarea cu acest framework, Thymeleaf va putea avea acces la statusul legat de autentificare și autorizare a utilizatorului care accesează pagina. Astfel se pot impune restricții direct asupra blocurilor de html din frontend, în funcție de accesul pe care îl are la momentul respectiv.

Imaginile de mai jos fac referire la sintaxa dialectului care restricționează blocul respectiv de html (<li>) utilizatorilor care sunt autentificati, respectiv utilizatorului (autentificat și el) cu rol de ADMIN.





Ca ultim exemplu, în figura de mai jos este un fragment din configurarea din Cassandra Interface a librăriei Thymeleaf, în care se creează bean-ul (integrare cu Spring) pentru motorul care va fi folosit în compilarea template-urilor, unde se adaugă cele două dialecte (cel standard este inclus automat).

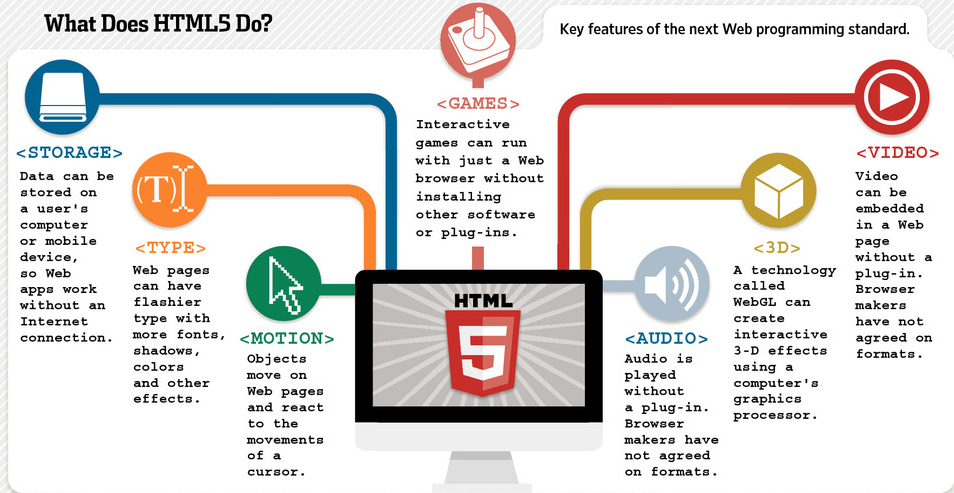


HTML și HTML5

HTML (Hype Text Markup Language) este un limbaj de marcare (metodă de formatarea unui text într-o pagină web) folosit pentru crearea site-urilor web. Browserele web primesc documente HTML fie din spațiul local fie de la un server web și le parsează în pagini web.

HTML creează documente structurate folosind blocuri pentru texte cum ar fi paragrafe, liste, link-uri, imagini și alte obiecte de tip bloc. Acestea sunt delimitate prin tag-uri ca <p> </p> pentru paragrafe sau <img> </img> pentru imagini. O pagină web este delimitată de tag-ul <html> </html>, iar conținutul vizibil pentru user este delimitat de tag-ul <body> </body> [22]. Astfel, întreg documentul este o structură arborescentă, fiecare tag putând include alte tag-uri. De asemenea fiecare tag în parte poate avea informații suplimentare (atribute) despre obiectul respectiv, informații pe care HTML le va folosi în parsarea documentului.

HTML5 este a cinea și cea mai bună versiune a limbajului HTML, apărut pe 22 ianuarie 2008 și are ca scop îmbunătățirea limbajului astfel încât să poată avea suport pentru cele mai noi fișiere multimedia și alte caracteristici, fiind în același timp citibil atât pentru user dar și pentru calculatoare. Facilitează și încorporarea cu diferite API-uri pentru aplicații web mai complexe [23]. Pentru a declara documentului că este folosit HTML se folosește intaintea tag-ului <html>, tag-ul <!DOCTYPE html>.



[24]

CSS și Bootstrap

CSS (Cascading Style Sheet) este un limbaj de stilizare pentru prezentarea unui document scris într-un limbaj de marcare (HTML). Numele de “cascading” vine de la puterea limbajului de a determina prioritatea regulilor (pe care să o aleagă) în caz că există mai multe.

CSS-ul are o sintaxa simplă, astfel ca un fișier de stilizare (style sheet), conține o listă de reguli, fiecare regulă având unul sau mai mulți selectori și un bloc de declarare. Selectorul determină cărei părți din document îi este aplicată stilizarea.

Mai jos este un exemplu de sintaxa a CSS-ului, care setează culoarea roșie și textul bold pentru conținutul tuturor paragrafelor care au clasa “paragraf” dintr-un document, “p.paragraf” fiind selectorul, iar ceea ce se află între acolade este blocul de reguli.

p.paragraf {

color: red;

font-weight: bold;

}

Boostrap este un framework “open source” pentru CSS. A apărut inițial sub numele de Twitter Blueprint, fiind creat de către Mark Otto și Jacon Thornton pentru platforma Twitter. Au apărut apoi mai multe versiuni, iar Boostrap 3 (cel folosit în Cassandra Interface) s-a stabilit pe 19 august 2013, urmat apoi de Boostrap 4 pe 29 octombrie 2014 în versiunea alpha și finalizat pe 18 ianuarie 2018 [26].

Boostrap este format în mare parte din o colecție de reguli CSS și opțional sripturi de JavaScript, care folosite și apelate sub forma unor atribute ale elementelor dintr-un document, transformă elementele respective și le stilizează pe baza regulilor definite de framework.

Astfel, având posibilitatea de a modifica numai acele atribute după bunul plac al utilizatorului, stilizând prin acestea documentul, Boostrap devine un framework în același timp complex, dar și maleabil și ușor de folosit, economisind astfel timp și cod.

Javascript și Jquery

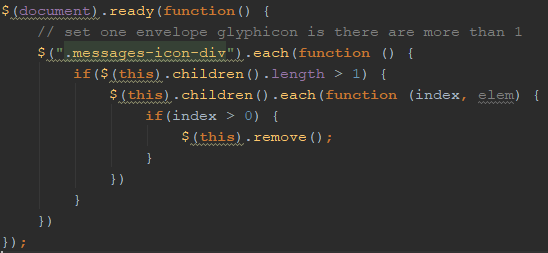
Javascript este un limbaj de programare interpretat (intrucțiunile sunt executate direct, fără compilare), orientat pe obiecte bazate pe prototipe (obiectele se pot refolosi). Alături de CSS și HTML, Javascript este unul dintre tehnologiile nucleu ale World Wide Web-ului și este o parte esențială pentru aplicațiile web.

Chiar dacă sunt unele asemănări între Javascript și Java din punct de vedere al sintaxei și al librăriilor, acesta a fost influențat în mare parte de limbaje de programare ca Self și Scheme. De asemenea serializarea JSON este bazată pe JavaScript [27].

O mare caracteristică a limbajului o consituie facilitatea de a putea comunica atât în partea de client (client-side), cât și în partea serverului (server-side), creând astfel o comunicare eficientă între frontend și backend. O pagină web poate avea sau nu, unul sau mai multe fișiere (scripturi) javascript, fiecare rulându-se atunci când pagina web este încărcată. Unele dintre aceste scripturi creează triggere, părți de script care se vor apela atunci când un anumit eveniment este întâlnit în pagină.

Jquery este o librărie din JavaScript creată pentru a simplifica manipularea elementelor DOM din cadrul paginii HTML, a evenimentelor, animațiilor, cererilor ajax, etc, făcând navigarea într-un document mult mai ușoară [28]. Astfel, scriptul scris în Jquery este mult mai scurt și mai lizibil decât unul scris în Javascript, încurajând în acest sens coduri mai complexe.

Imaginea de mai jos demonstrează complexitatea sintaxei Jquery. Odată cu încărcarea paginii, scriptul se rulează (apelându-se funcția de încărcare completă a paginii ($(document).ready()), apoi prin Jquery, preiau toate elementele găsite care conțin clasa “messages-icon-div” și pentru fiecare element găsit, dacă are mai mult de un element-copil, șterg acele elemente până rămâne unul singur. Astfel scriptul elimină duplicatele din interioriul unui element prin doar câteva comenzi simple.



Prezentarea aplicației

Structura

Aplicația pe partea de frontend, este structurată în aproximativ 4 părți importante:

Prima este pagina de Home, la care au acces toți utilizatorii, fie ei autentificati sau nu. În cadrul acestei pagini este prezentată o descriere a aplicației, mai mult decât atât, de pe acesta pagină utilizatorii se pot autentifica sau înregistra în aplicație.

A doua parte constă în pagina de Profil a membrului autentificat cu succes, care este disponibilă spre a vizualiza și a aduce modificări în cadrul contului acestuia.

A treia parte și cea mai reprezentativă o reprezintă paginile cu interacțiune în baza de date. De asemenea aceste pagini pot fi accesate numai după ce membrul este autenificat și pe lângă acest lucru, baza de date în sine are autentificare și roluri de acces proprii. Aceste pagini permit utilizatorului cu acces să vizualizeze / creeze / editeze / exporte / caute în baza de date, în funcție de nevoile acestuia și de setările făcute.

A patra parte constă într-o consolă universală cu acces la baza de date, care este disponibilă numai membrilor cu acces de ADMIN global, neavând legătură cu baza de date personalizată pentru fiecare utilizator.

Toate aceste pagini sunt expuse membrilor sub forma unui meniu mereu prezent și ușor de utilizat. De asemenea, folosind Thymeleaf (descris mai sus), aplicația folosește un layout cu mai multe fragmente, astfel paginile propriu-zise sunt construite dinamic, fără replicare de cod.

Pe partea de backend, aplicația este structurată în 2 părți esențiale:

Prima este partea de configurare, care conține toate clasele care configurează dependințele proiectului, cât și clasele de proprietăți, validări, și tot ce ține de partea “fizica” a aplicației.

A doua parte este cea de web, în care se află controllerele, modelele, serviciile, event-urile, și tot ceea ce ține de o aplicație REST, partea de comunicare cu frontend-ul și cu baza de date.

Toate acestea vor fi explicate la capitolul de Server-Side.

Securitate

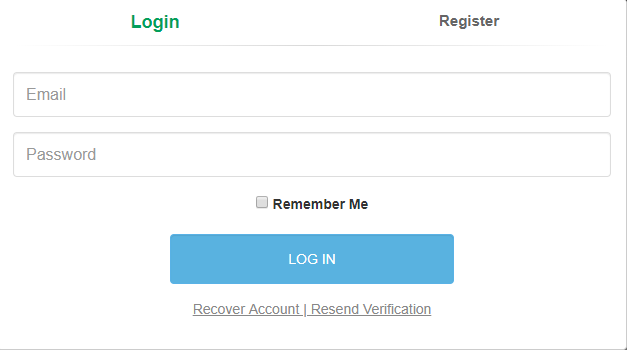
Cassandra Interface a fost dezvoltată cu scopul de a simplica anumite procese și de a prezenta o nouă metodă de organizare a datelor într-o bază de date NoSQL. Astfel, inevitabil, platforma este dependentă de baza de date pentru care rulează, devenind nefuncțională în cazul în care Apache Cassandra nu este activă.

Fiind o aplicație web, aplicația se bazează pe conceptul de REST, cereri făcute de către utilizatori către server (controller), la care se trimite un răspuns adecvat acțiunii lor. De asemenea, din cauza faptului că mai mulți utilziatori pot accesa în același timp aplicația, respectiv baza de date, atât eficiența cât și securitatea trebuie să fie pe măsură. Astfel, o conexiune a utilizatorului cu aplicația este legată de sesiunea acestuia în browser-ul web, devenind inactivă odată cu lipsa activității din cadrul aplicației în sesiunea curentă.

În mare parte, Spring Security se ocupă de securitatea la nivelul aplicației, însă cea la nivelul accesării bazei de date a trebuit construită manual. Mai departe este descris modul în care funcționează securitatea în Cassandra Interface.

Figura de la pagina 13, care descrie configurația framework-ului Spring Security în cadrul aplicației, expune cât de poate de simplu puterea acestuia. Utilizatorii sunt restrictionati din a accesa anumite pagini în funcție de starea lor de autentificare și autorizație. Pe lângă aceste setări, Spring Security se ocupă și cu “retinerea” unui utilizator autentificat în contexul aplicației, pentru a cunoaște în orice moment datele acestuia. Astfel, urmează o prezentare a modului de autentificare a unui user și cum este reținut acesta în context.

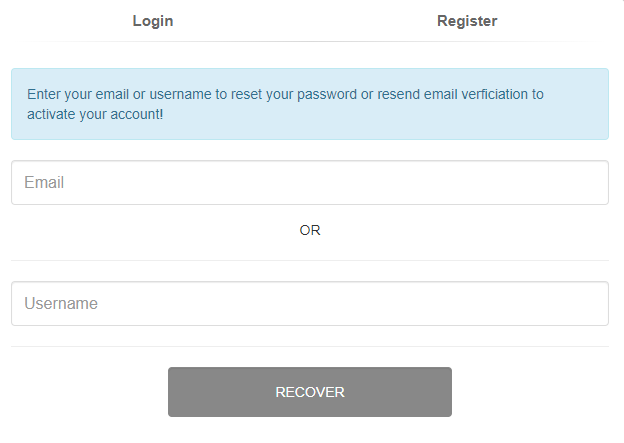
În imaginea următoare este prezentat formularul de login în aplicație. Acesta apare sub forma unui “modal” în pagină, la apăsarea butonului de Login din meniul alicatiei.



După cum se poate observa, autentificarea unui utilizator deja înregistrat se face după email-ul și parola acestuia. De asemenea există posibilitatea de a bifa “remember-me”, facilitate garantată de către Spring Security, menținând utilizatorul logat chiar dacă acesta părăsește aplicația. În caz de eroare la autentificare, aplicația afișează mesaje corespunzătoare, ca de exemplu:



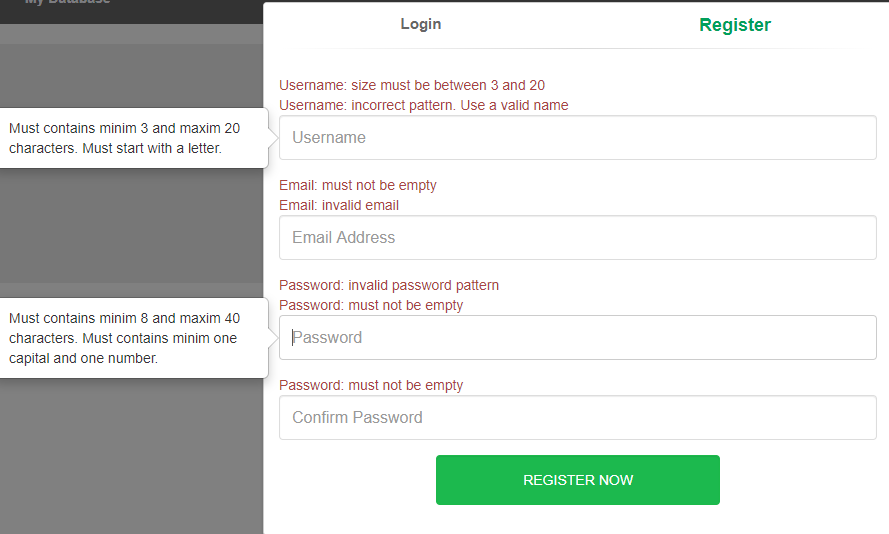
În cazul pierderii contului sau a neactivarii acestuia, user-ul are posibilitatea de a recupera parola prin email sau a retrimiterii email-ului de activare (vom discuta despre asta la înregistrare) printr-un click în subsolul formularului de logare (Recover Account…), care va deschide alt modal cu următorul formular:



Intern, aplicația își va da seama dacă utilizatorul dorește să-și reseteze parola sau să rerimita email-ul de activare prin testarea stării contului respectiv.

Revenind la autentificare, dacă aceasta a avut succes, datele din formular sunt transmise către Spring Security, care oferă posibilitatea de customizare al acestui proces, lăsând programatorul să decidă care anume din datele utilizatorului să se păstreze în context și sub ce formă. Astfel, aplicația Cassandra Interface, preia datele (în CassandraUserDetailsService), verifică dacă user-ul există și îl preia din baza de date, urmând apoi să se stabilească rolurile acestuia în cadrul aplicației și să se returneze aceste date mai departe către framework (CassandraUserDetails, care implementează clasa default UserDetails). În orice alt caz, se retruneaza o eroare sub forma unui mesaj reprezentativ pentru utilizator.

Pe lângă acestea, securitatea nu trebuie să se oprească aici, urmează o imagine cu formularul de înregistrare în aplicație (disponibil în pagina de Home, în meniul aplicației), iar mai departe descrierea sistemului.

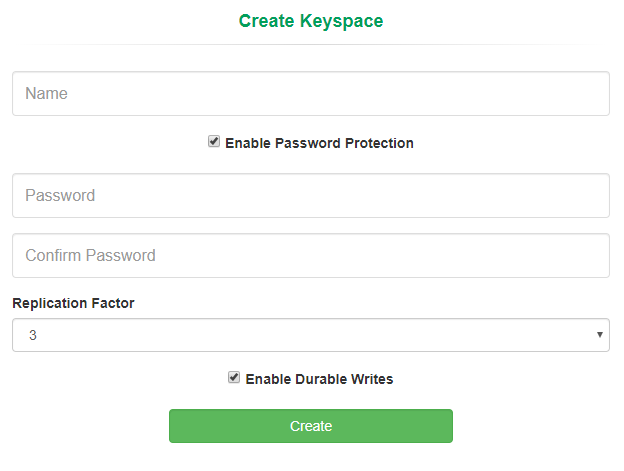


Pentru a se putea observa sistemul de erori, imaginea de mai sus este făcută după trimiterea formularului gol către server, acesta încercând să-l valideze, însă fără succes, returnând astfel erorile specifice. De asemenea, aplicația oferă câteva popover-uri cu informații legate de validarea datelor care trebuie introduse. Înregistrarea constă într-un email și username unic global și o parola cu un minim de dificultate.

Informațiile sunt transmise la controllerul de register, unde sunt validate. Dacă nu există erori, se creează un user cu aceste date, se generează un ID, se criptează parolă, se setează rolul de USER și se adaugă în baza de date ca utilizator inactiv. Totodată se generează un link care se trimite pe email-ul cererii de înregistrare, care conține un token de active a contului. Dacă utilizatorul va încerca să se logheze fără accesarea acelui link din email, va primi un mesaj corespunzător de eroare. Astfel, confirmarea prin email a contului reprezintă o metodă eficientă de securitate împotriva creării de conturi invalide și validarea integrității utilizatorului.

Mai departe vom vorbi despre securitatea în cadrul interacțiunii cu baza de date.

Un utilizator înregistrat și logat cu succes, are acces la pagina personală în baza de date indiferent de rolul pe care îl deține (prin apăsarea butonului “My Database” din meniul aplicației). Platforma îi oferă utilizatorului posibilitatea să-și creeze propriul keyspace (structură descrisă în capitolul anterior), protejată cu parolă sau nu. Figura de mai jos prezintă formularul pentru acest proces (tot sub forma unui modal-view).



Se poate observa că un keyspace, poate fi sau nu securizat prin parolă. Creatorul astfel are alegerea liberă dacă dorește să-și protejeze sau nu keyspace-ul. Odată creat, acesta este stocat atât în baza de date a userilor ca făcând parte din colecția creatorului, cât și în baza de date a keyspace-urilor, cu setările alese în formular. Astfel, în acest moment singurul cu acces la structura creată este însuși creatorul, însă acesta poate acorda acces cu diferite roluri (membru, editor sau admin) și altor utilizatori, adaungandu-se astfel keyspace-ul și în colecția acestora. Prin aceste facilități, un membru poate să-și creeze propria “comunitate” care să interacționeze cu aceleași date, chiar și în același timp.

Cu toate că mai mulți useri pot accesa în același timp aceleași date, Cassandra Interface oferă securitate și în acest sens. Pentru fiecare interacțiune cu baza de date, se verifică în primul rând dacă utilizatorul este conectat la keyspace-ul pentru care se face acțiunea, apoi dacă are dreptul de a o executa (în funcție de rolul pe care îl deține în keyspace-ul respectiv). Conexiunea este stabilită după cum am precizat și mai sus, sub forma unei sesiuni, care va expira în lipsa inactivității, astfel un utilizator nu poate fi conectat în același timp la mai multe keyspace-uri.

Altă caracteristică majoră a securității bazei de date o reprezintă faptul că toate interacțiunile de editare sunt salvate în log-ul keyspace-ului. De asemenea, unele interacțiuni creează un link de backup pentru modificarea făcută, din care se pot downloada, sub format JSON, datele de dinainte de modificare, obținând astfel o siguranță continuă. Pe lângă aceste backup-uri dinamice, se pot exporta și manual datele și logurile printr-un singur click.

Prin toate aceste caracteristici, cât și cele descrise la capitolul anterior, aplicația Cassandra Interface își câștigă securitatea și controlul asupra acțiunilor factorilor externi.

Client-Side

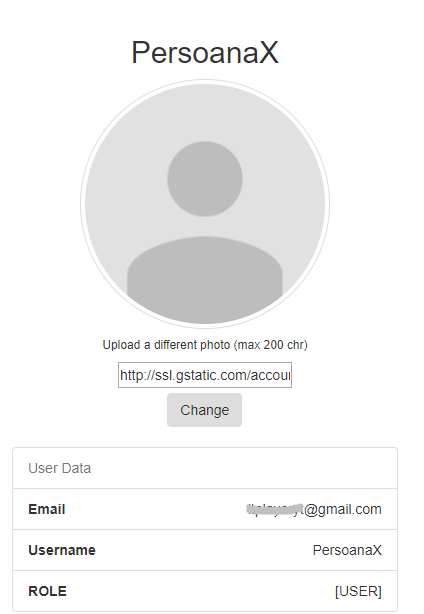
În această parte vom vorbi despre facilitățile de care dispune aplicația din punct de vedere al unui utilizator, fie el cu rol de membru, fie admin, utilizator neautentificat neavând acces decât la pagina de Home de unde se poate loga sau înregistra, și bineînțeles, de a citi descrierea aplicației.

Am prezentat mai sus sistemul de autentificare și de securitate, care sunt valabile atât pentru utlizatorul normal cât și pentru admin. Mai departe vom elabora tot ceea ce aplicația poate oferi.

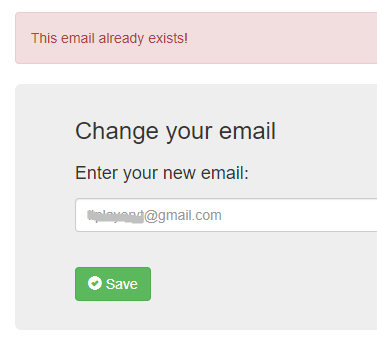
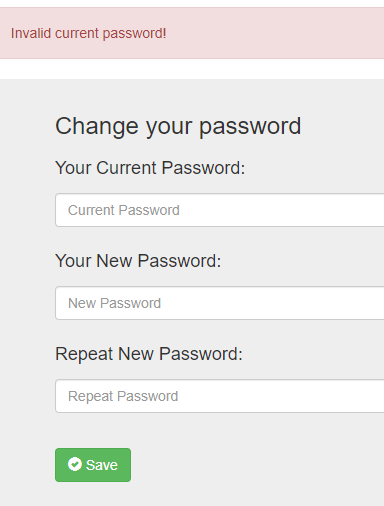
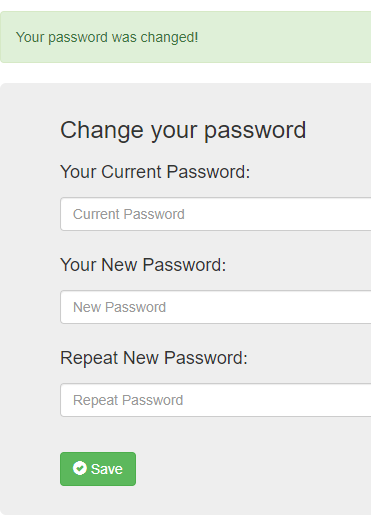
Să presupunem că o persoană (pot fi și mai multe), pe care o să o notăm arbitrar cu PersoanaX, are nevoie de o bâză de date pentru a stoca datele afacerii sale. Citind despre diferite sisteme de gestiune, ajunge la concluzia că o bază de date NoSQL i-ar satisface nevoile, astfel că ajunge față în față cu aplicația Cassandra Interface. După ce se întregistrează, confirmă adresa de email și se loghează, PersoanaX are la dispoziție mai multe opțiuni, însă alege să-și consulte pagina de profil accesând butonul cu Username-ul ei din meniul aplicației (în partea dreaptă, lângă Logout).



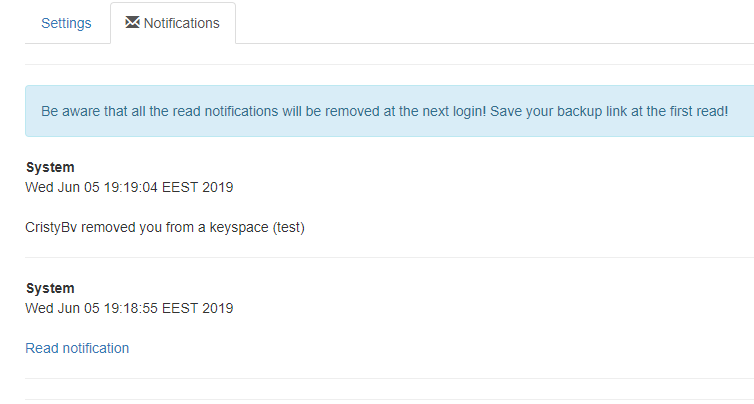
Ajuns pe pagina de Profil, PersoanaX observă că sunt afișate câteva din datele personale și are la dispoziție mai multe setări asupra contului sau. Pentru început decide să-și schimbe avatar-ul, înserând un URL al unei fotografii (URL ce nu trebuie să depășească o anumită lungime), și apasă pe butonul Change, care, folosind o cerere Ajax, transmite către server (ProfileController) URL-ul și actualizează atât utilizatorul autentificat în context-ul Spring, cât și în baza de date.



Apoi, poate decide de exemplu să-și schimbe email-ul sau parola, primind mesaje reprezentative dacă operațiunea a avut succes sau nu.

De asemenea, tot în pagina de profil sunt afișate notificările PersoaneiX (dacă are). Acestea sunt transmise de către Sistem cu ocazia anumitor evenimente cum ar fi: adăugarea utilizatorului într-un keyspace, ștergerea utilizatorului din keyspace sau ștergerea keyspace-ului din care face parte. Pentru eficientă, aceste notificări, după cititre, vor fi șterse la urmatoroarea logare. Dacă PersoanaX are notificări necitite, în dreptul Username-ului său din meniu, va apărea un icon cu un plic pentru a-l anunța despre acest lucru.

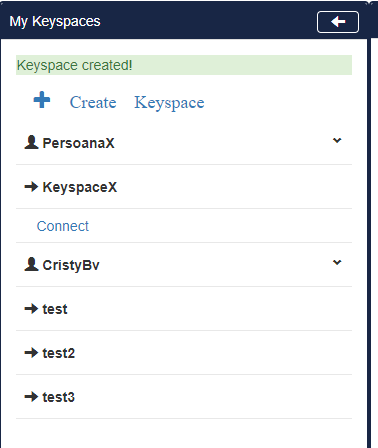
 

Acum că PersoanaX și-a actualizat profilul, dorește să înceapă să folosească baza de date și să-și invite colegii (dacă este cazul) să aibă acces la aceasta, așa că dă click pe butonul “My Database” din meniul aplicației.



Să presupunem că PersoanaX mai are cunoștințe care folosesc aplicația și a fost adăugat în trei keyspace-uri de către aceștia: primul are numele de “test” și aici X are rol de membru, apoi “test2” cu rol de editor și “test3” cu rol de admin. Cu toate acestea, încă nu dorește să interacționeze cu acestea și decide să-și creeze propriul keyspace, acesta fiind motivul pentru care folosește aplicația.

În partea stânga îi este prezentat tab-ul cu keyspace-uri (“My keyspaces”), care momentan conține butonul de creare și cele 3 keyspace-uri la care are acces. Astfel, dă click pe “Create Keyspace” și îi apare modal-ul prezentat mai sus (la Securitate), unde își setează datele pentru keyspace și îl creează.



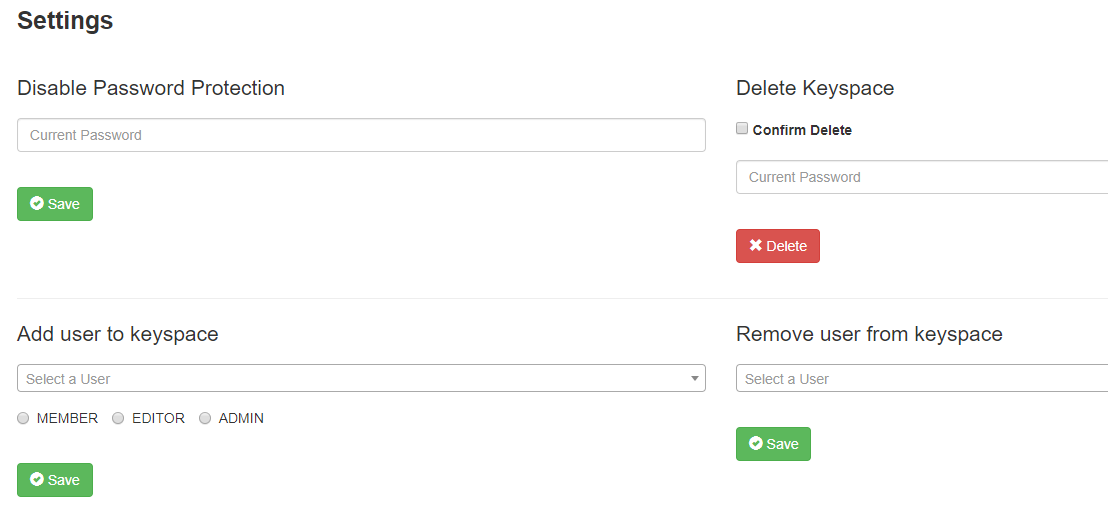
În poza de mai sus se poate observa mesajul de succes al creării keyspace-ului cu numele “KeyspaceX”, creat de “PersoanaX”. De asemenea cele trei keyspace-uri de mai jos la care are acces, au ca și creator pe utilizatorul cu username-ul “CristyBv”.

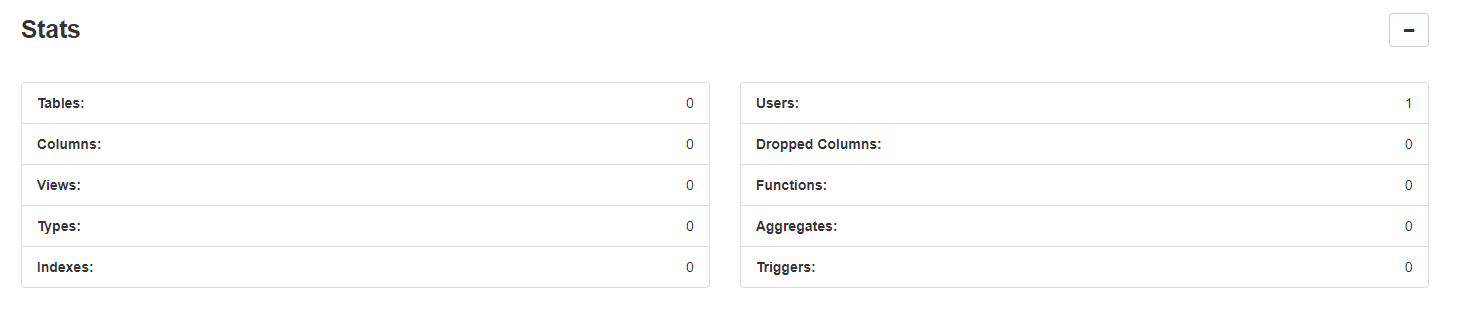
Mai departe, PersoanaX se conectează la keyspace prin butonul de Connect (folosind parola dacă este cazul) și astfel, intern, se adaugă la sesiunea curentă ca fiind conectat la KeyspaceX. Odată conectat, în partea liberă a paginii apare un meniu reprezentativ pentru user-ul cu rol de creator și este redirecționat pe prima pagină a meniului, aceea de Manage.

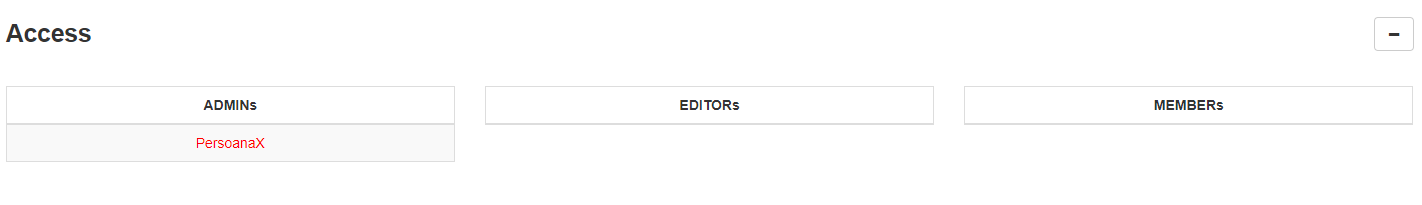


În această pagină, PersoanaX poate vizualiza și modifica setările keyspace-ului sau, îl poate șterge, poate activa/dezactiva protecția prin parolă, poate adăuga/șterge membrii (folosind căutare Live prin cereri Ajax prin plugin-ul Select2 – va fi descris mai încolo). De asemenea, tot aici apar informații utile despre content-ul keyspace-ului (numărul de tabele, coloane, etc), cât și tabele cu userii adăugați și rolurile lor.





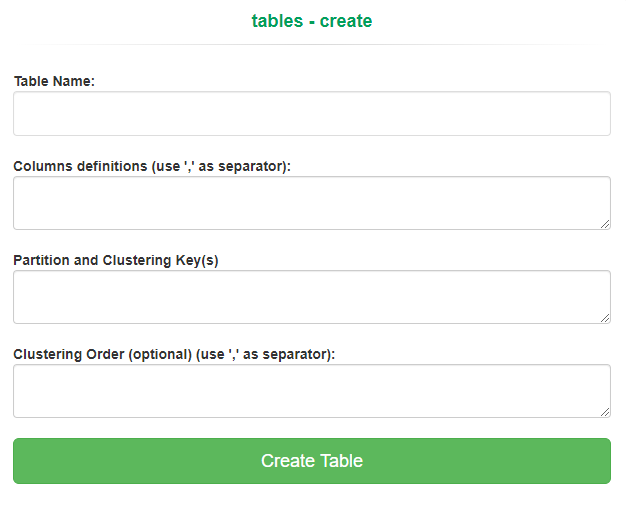


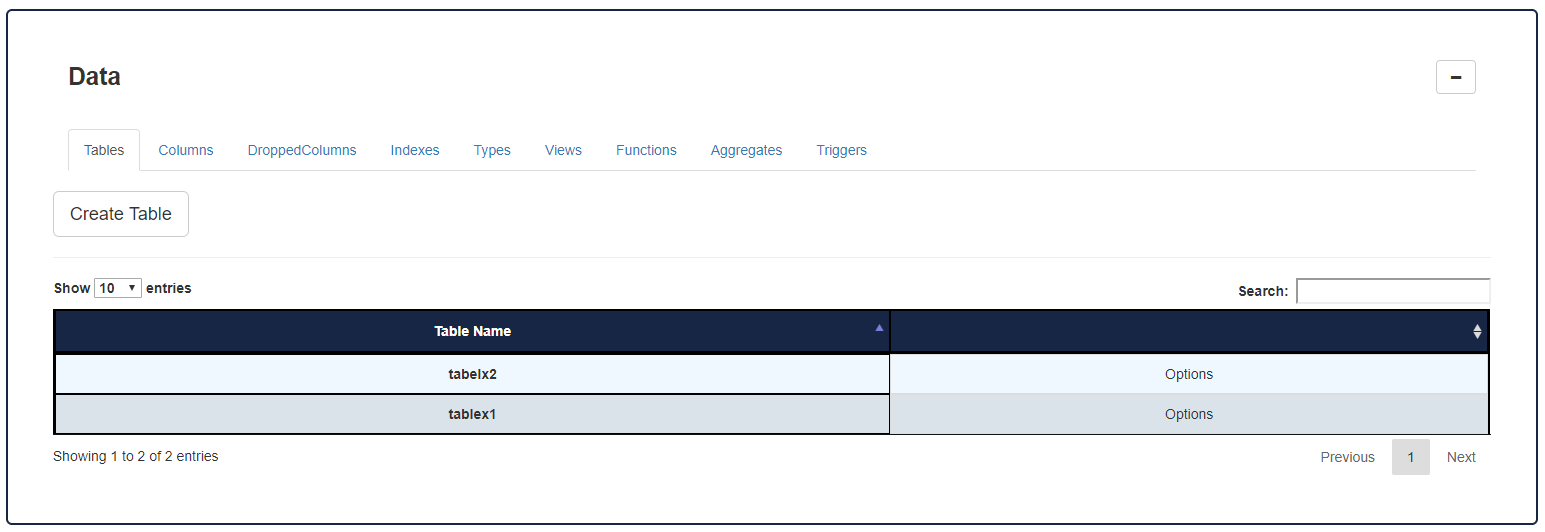


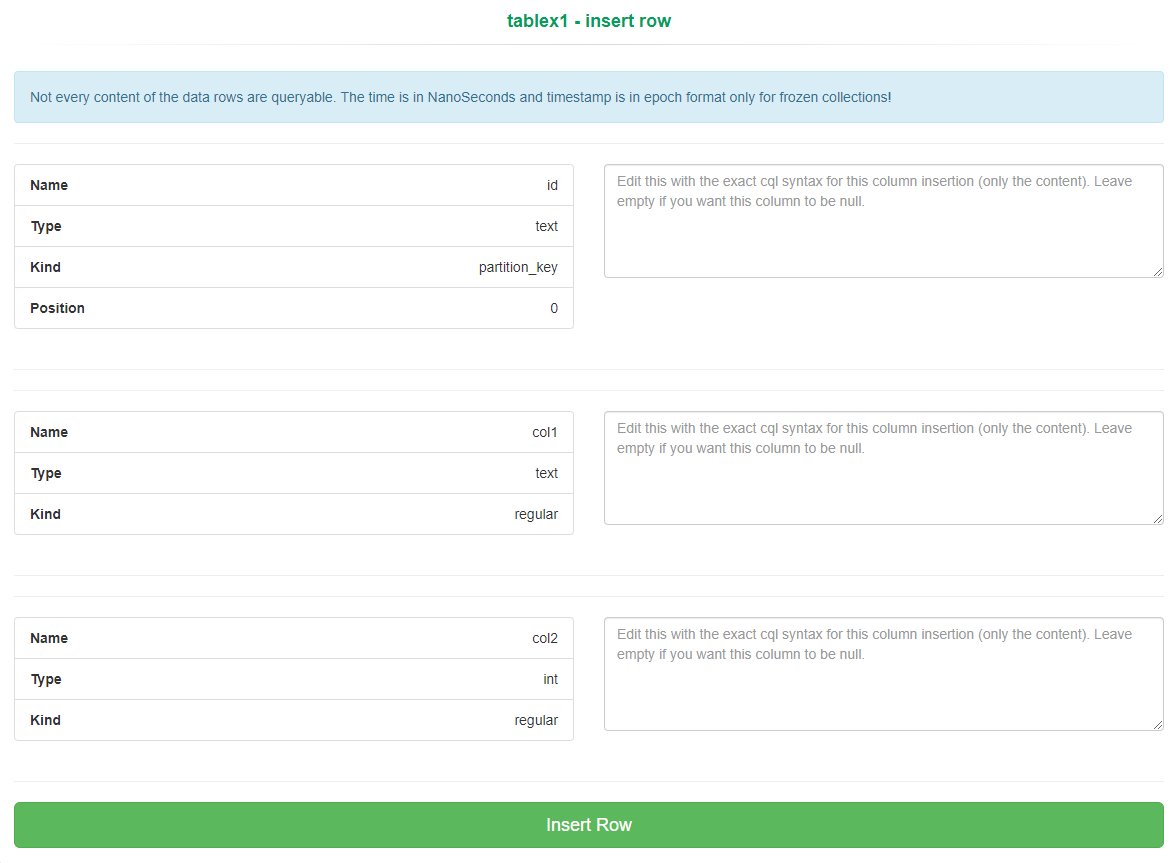
După ce verifică sau editează aceste informații, PersoanaX cercetează mai departe aplicația și dă click pe a doua pagină din meniu și anume View/Edit. Dând click pe View/Edit, intern, i se adaugă în sesiunea curentă faptul că pagina aleasă este aceasta, astfel, chiar dacă părăsește baza de date, când se va întoarce, aplicația va știi să afișeze această pagină.

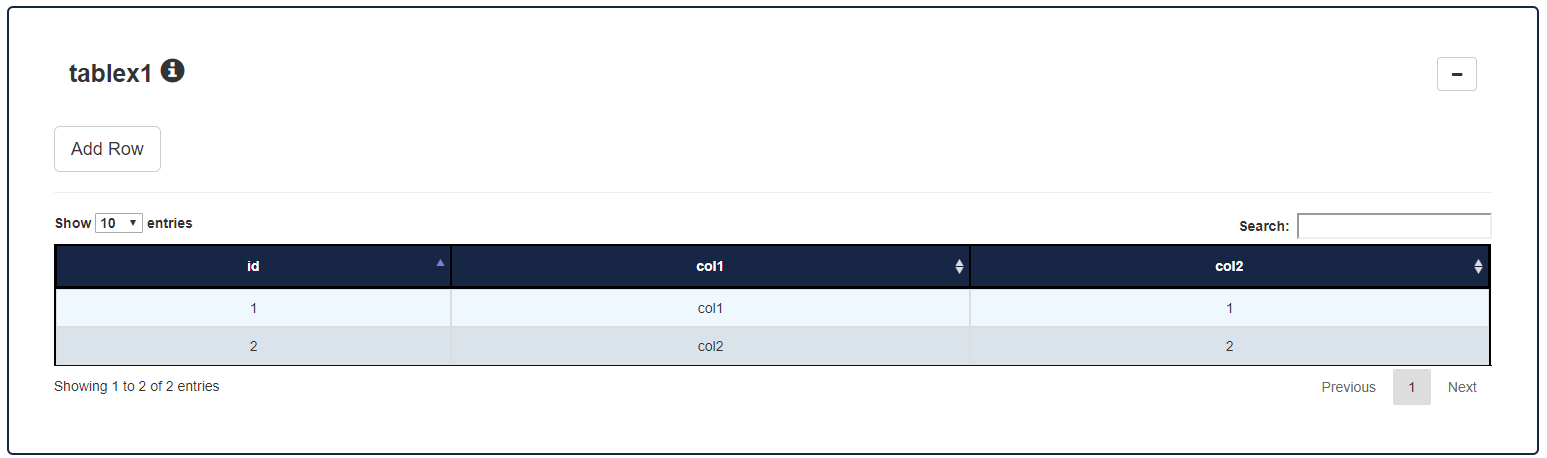
Pagina de View/Edit este împărțită în 2 corpuri. Cel de sus (Data) unde sunt afișate toate tabelele / coloanele / indecșii / type-urile / funcțiile / etc, fiecare cu proprietățile lor, putând fi adăugate / editate sau șterse unde este cazul. În corpul de jos (Content) vor fi afișate datele dintr-un tabel / view, atunci când se dă click pe unul (de asemenea, alegerile sunt memorate în sesiune), și la fel ca mai sus, se pot adăuga, edita sau șterge linii în tabel.

PersoanaX începe prin a crea câteva tabele și însera câteva date pentru a testa aplicația.





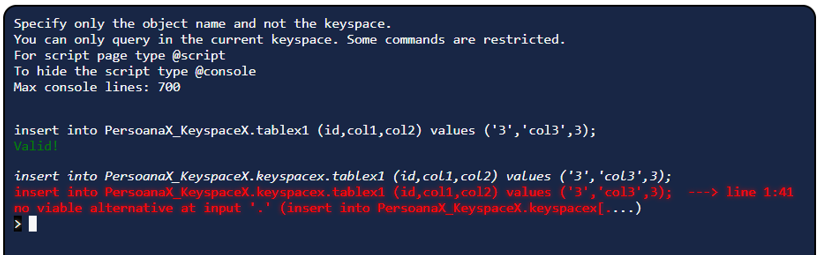




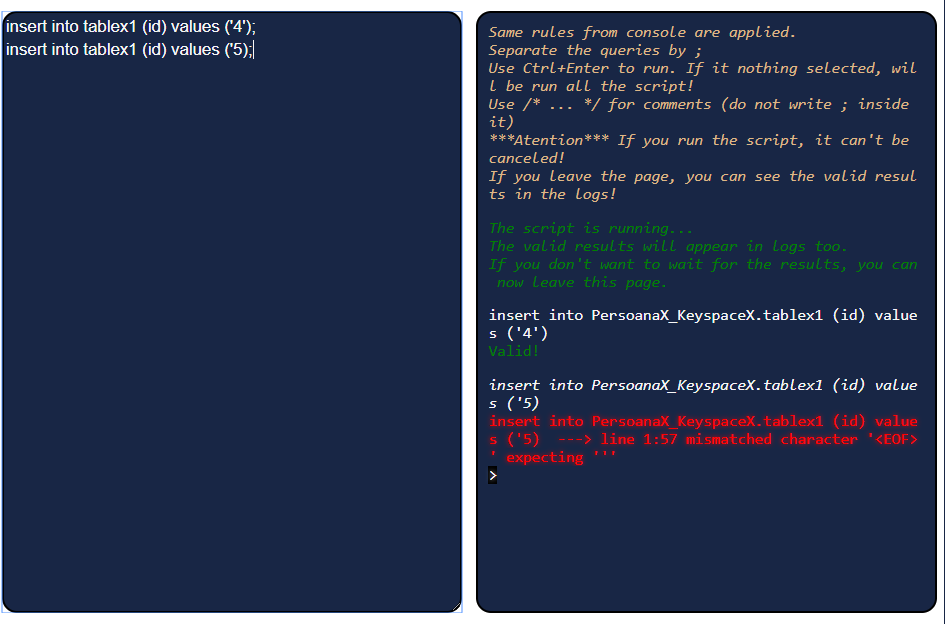
În imaginile de mai sus se poate observa cum PersoanaX, dă click pe Create Table, introduce datele corespunzătoare, creând cele 2 tabele, care apar în tabelul de mai jos. Apoi, dând click pe unul dintre acestea, respectivul va apărea în corpul de jos, de unde X va însera cele două linii. Este de precizat că cele două tabele (create cu plugin-ul DataTable – va fi descris mai încolo) primesc datele diferit. Cel de sus, având date relativ puține, vor fi încărcate toate în partea de client-side, în timp ce tabelul de jos, putând avea sute de mii de date, își preia datele (filtrate sau nu) din server-side, îmbunătățind astfel considerabil eficiența aplicației.

Mai departe, platforma oferă mai multe setări. În afară de paginare, sortare, căutare, facilități oferite de plugin, PersoanaX poate redimensiunea coloanele tabelului prin scroll up/down, poate ascunde coloane prin dreapta click pe acestea și îl poate reseta prin click pe titlul acestuia. Pentru a edita o linie, X trebuie doar să dea dublu-click pe aceasta și îi va apărea un modal cu datele respective pe care le poate upadata sau șterge.

Toate aceste operațiuni se repetă mai mult sau mai puțin pentru toate structurile din Cassandra (tabele / coloane / funcții / etc) în pagina de View/Edit. Mai departe, PersoanaX ajunge la pagina de Console/Script, unde este activat plugin-ul de terminal (va fi descris mai încolo), afișându-l sub forma unei console în care X poate introduce aproape toate tipurile de query existente în Cassandra (create / alter / drop / select / insert / update / delete / …). Aplicația fiind securizată, nu va valida niciun query destinat altui keyspace în afară de cel curent, astfel, query-urile nu trebuie să conțină și numele keyspace-ului, fiind adăugat automat intern.



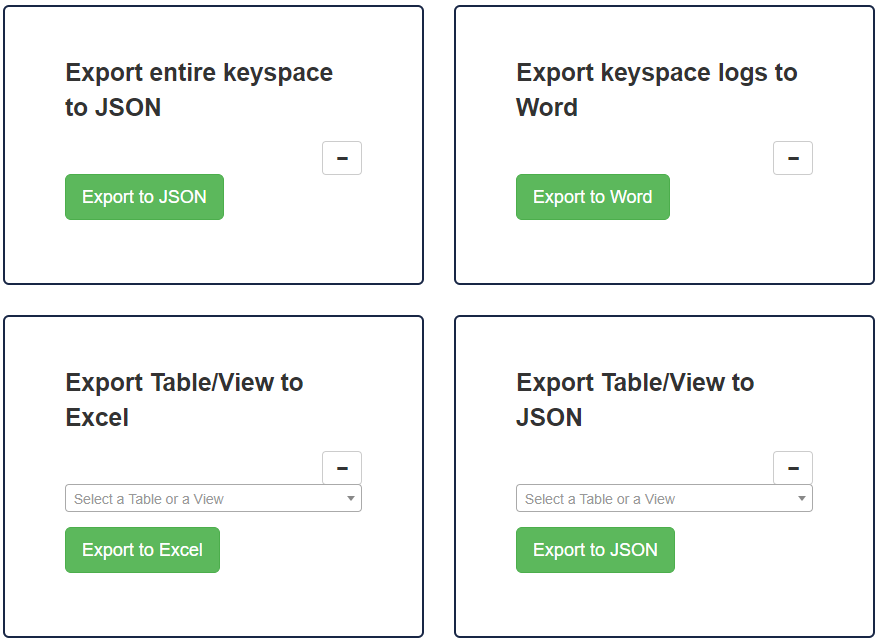
Pe lângă facilitatea de consolă, prin tastarea comenzii @script, va apărea încă un corp în care se pot introduce mai multe query-uri odată și rulate (prin Ctrl+Enter, dacă nu este nimic selectat, va fi rulat tot scriptul, altfel doar ceea ce este selectat).



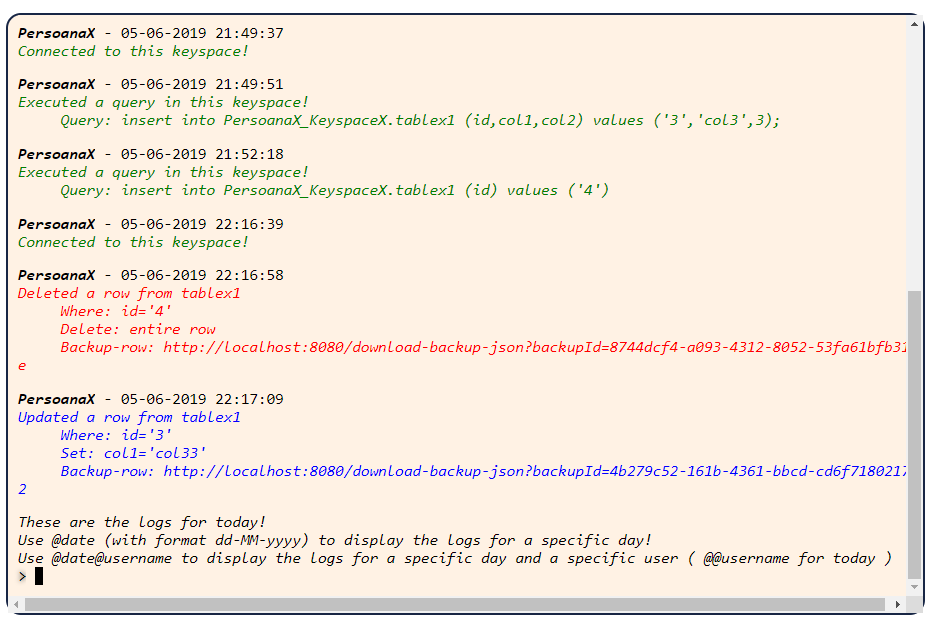
Acest mecanism este posibil prin cereri Ajax către server cu respectivele query-uri, care se detectează, validează și execută, returnând rezultatul în consecință. Astfel, odată ce scriptul a fost activat, cererea Ajax este trimisă, iar PersoanaX dacă nu dorește să aștepte rezultatele (în caz că sunt foarte multe query-uri), poate părăsi pagina, fiindcă oricum editările valide vor apărea de asemenea și în log-ul keyspace-ului.

Ca și aproape celelalte facilități, atât alegerile cât și conținutul din console/script, este salvat în sesiune pentru ca PersoanaX să nu piardă conținut din greșeală. În cazul unei comenzi de select, în care datele pot fi nenumărate, consola afișează numai câteva, lăsând posibiltatea tastării comenzii @more pentru a încărca mai multe dintre acestea, evitând astfel blocarea paginii.

Mai departe, X ajunge la pagina de Export, în care are posibilitatea atât de a-și exporta tabelele sau view-urile ca Excel (folosind librăria ApachePOI) dar și ca JSON (folosind librăria Jackson), cât și posibilitatea de exportare a întregului keyspace ca JSON sau a log-ului ca fișier de tip Word (folosind de asemenea ApachePOI).

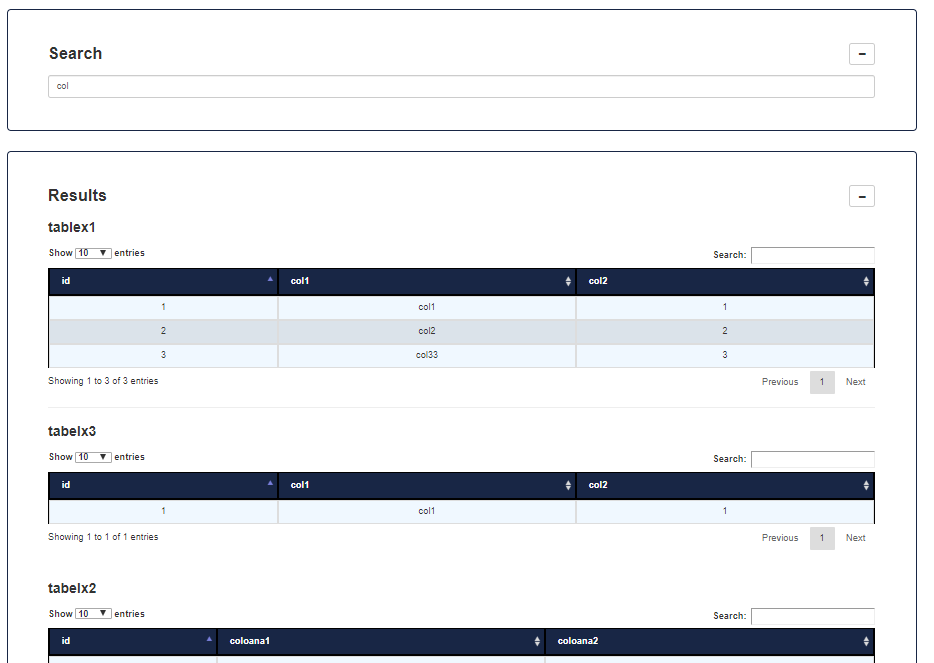


Urmează pagina de Log, în care tot sub forma unei console sunt afișate log-urile keyspace-ului pentru ziua curentă. Pentru a vedea alte log-uri, PersoanaX trebuie să urmeze comenzile afișate în consolă.



După cum se poate vedea, log-urile pentru conectare / creare / inserare sunt reprezentate de culoarea verde, cele de alter / update de albastru, iar cele pentru deconectare / drop / delete de roșu. Mai mult decât atât, după cum am precizat și mai sus, editările / ștergerile făcute în pagina de View/Edit, generează automat și un fișier JSON de backup, iar link-ul acestuia apare în loguri.

Ultima, este pagina de Search, unde PersoanaX poate căuta în toate tabelele din keyspace-ul lui. Rezultatele sunt grupate după aceste tabele, iar dacă sunt prea multe rezultate, se afișează numai câteva, și se sugerează să se restrângă criteriul de căutare.



Acum că PersoanaX a parcurs și a testat funcționalitățile aplicației pentru keyspace-ul său, urmează să se conecteze și la celelalte keyspace-uri în care a fost adăugat. Făcând asta, el observă următoarele:

Rolul de membru nu-i oferă decât informații despre keyspace și acces de vizualizare, adică la pagina de View, însă tot ceea ce este legat de editare / ștergere nu mai apare, la pagina de Export, unde poate exporta numai tabele sau view-uri și în cele din urmă, are acces la Search.

Rolul de editor, față de membru, îi oferă și posibilitatea de a vizualiza, pe lângă informațiile keyspace-ului, și statisticile acestuia cât și utilizatorii adăugați. Fiind editor, are acces de editare / ștergere în cadrul paginii de View/Edit, poate folosi de asemenea pagina de Console/Script. La Export și Search este la fel ca la member.

Rolul de admin îi oferă toate drepturile creatorului, în afară de: posibilitatea de a modifica setările keyspace-ului, de a activa / dezativa protecția cu parolă, și evident, de a șterge keyspace-ul.

Astfel, PersoanaX cunoaște aplicația și poate începe să-și invite echipa de afaceri și să-și populeze baza de date personală după bunul plac.

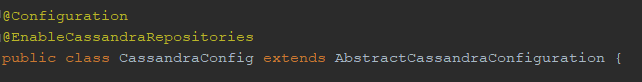
Diferența dintre un utilizator normal al aplicației și unul cu rol de ADMIN, este aceea că adminul are la dispoziție încă o pagina (disponibilă în meniul aplicației), numită Console. Aici apare același terminal ca în pagina din baza de date Console/Script (dar fără partea de script), însă nicio restricție nu mai este aplicată, query-urile putând conține orice keyspace din toată baza de date. Astfel, un admin poate face absolut orice modificare asupra aplicației prin intermediul bazei de date, inclusiv adăugarea altui admin, modificând în acest sens tabelul de useri din keyspace-ul admin.

Server-Side

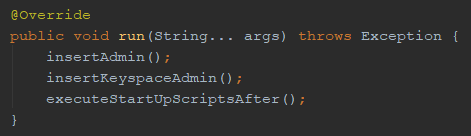
După cum a fost prezentat și la sub-capitolul Structură, backend-ul cuprinde două părți importante: cea de configurare și cea de web. Vor fi discutate pe rând.

Pentru început, tot mai sus am precizat că aplicația este dependentă de baza de date, dat fiind scopul aceștia. Astfel, o parte importantă din configurațiile aduse aplicației sunt pentru Apache Cassandra. Datorită suportului adus de către aceasta în aplicațiile Java, a fost folosită o clasă de configurare automată a bazei de date, în care au fost suprascrise anumite metode pentru adaptare. Acesta clasă se numește AbstractCassandraConfiguration și a fost importată prin dependințe Maven.

Odată ce baza de date Apache Cassandra este pornită pe server, clasa CassandraConfig, care extinde AbstractCassandraConfiguration (cea precizată mai sus), auto-configurează aplicația, conectând-o cu instanța bazei de date pe care o detectează pe server cu ajutorul metodelor suprascrise cu datele de conectare preluate din fisirul de proprietăți al aplicației. Mai mult decât atât, unele metode vor prelua din anumite fișiere de pe server (startUpScripts, shutDownScripts), scripturi predefinite care să se execute la pornirea, respectiv oprirea server-ului. Aceste scripturi constau în crearea / ștergerea unor materialized-view-uri pentru tabelele definite în aplicație, spre a putea prelua date din acestea după un alt criteriu în afară de cheia primară (acest mecanism este descris la capitolul Tehnologii folosite).



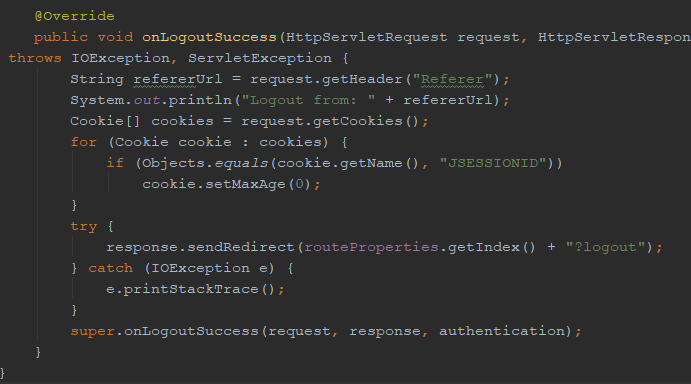
Trebuie precizat însă că, în Cassandra, un tabel nu poate fi șters sau recreat dacă materialized-view-ul care depinde de el încă există; acest lucru este valabil și invers, un materialized-view neputând fii creat dacă tabelul de bază nu există. Astfel, din cauza acestui lucru, atât StartUpScrips cât și ShutDownScripts fac același lucru și anume să șteargă toate materliazed-view-urile care depind de tabelele ce vor fi recreate la pornirea server-ului Spring, astfel eliminând orice șansa de a apărea vreo eroare la pornire. Apoi, după ce AbstractCassandraConfiguration își face treaba, și creează tot ceea ce trebuie creat, într-o clasă numită CommandLineAppStartRunner care implementează CommandRunner, permite prin metoda “run” a se rula cod după pornirea cu succes a server-ului Spring Boot, putând astfel aici rula scripturile de creare dintr-un alt fișier numit “startUpScripsAfter”. Tot în acesta metodă sunt inserate conturile default și keyspace-ul de admin, în care se află toate datele de construcție a aplicației (useri / keyspace-uri / etc).



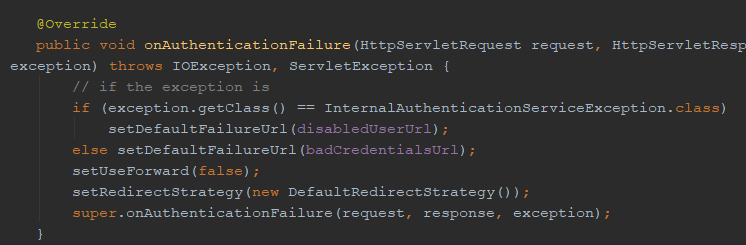
Mare parte din configurarea framework-ului Spring Security a fost descrisă anterior, însă mai sunt de precizat câteva amănunte.

Ca și Apache Cassandra, și SpringSecurity a trebuit importat prin Maven în fișierul de configurare pom.xml al aplicației. Cu acest import a fost posibilă crearea clasei SecurityConfig care extinde clasa de configurare WebSecurityConfigurationAdapter, permițând astfel suprascrierea anumitor proprietăți. În aceste setări descrise și anterior, este de spus că au fost intoduse clase de tip “handle” pentru anumite evenimente.

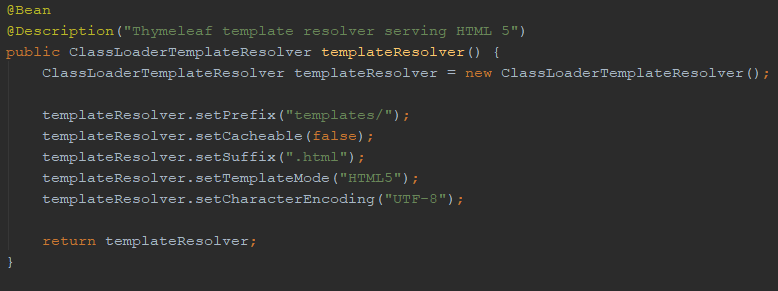
Unul din aceste handlere este “CustomLogoutHandler”, care se apelează atunci când utilizatorul dorește să se delogheze. În mare parte, aici sunt afișate mesaje în consola server-ului, se șterg cookie-uri și se redirecționează către pagina de Home, urmând ca Spring Security să-și continue protocolul.



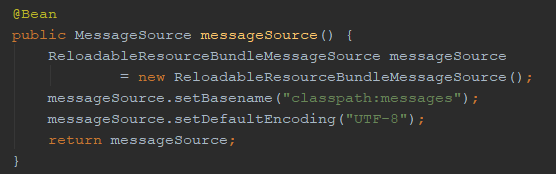
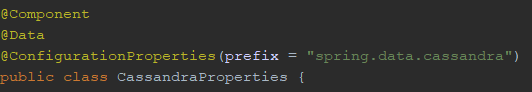
Un alt handler important este “CustomAuthenticationFailureHandler”, care se apelează atunci când este anunțată o eroare la autentificarea user-ului în aplicație. Aici se verifică ce tip de eroare este și se stabilește mesajul de eroare potrivit.



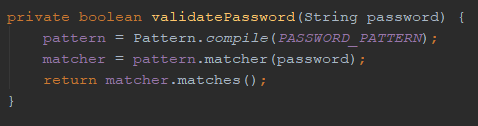
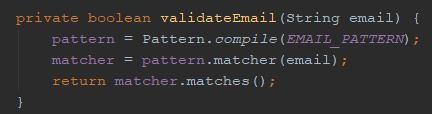
O configurare importantă o constituie cea de setare a comunicării cu frontend-ul. Clasa WebMvcConfig implementează clasa predefinită pentru aplicații web (importată tot prin Maven), WebMvcConfigurer, care permite setarea “engine”-ului de compilare a template-urilor. Astfel aici este configurat Thymeleaf-ul, setând locația (prefixul), extensia (sufixul) și formatul documentelor de html pe care să le proceseze, cât și dialectele folosite (descrise mai sus).

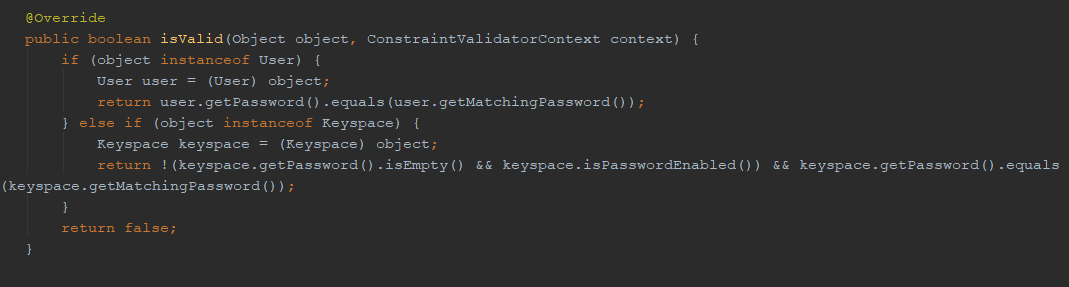


Tot la partea de configurare sunt adăugate clasele de proprietăți: “CassandraProperties”, “KeyspaceProperties”, “QueryProperties”, “RouteProperties”, prin care se preiau setări din fișierul “application.properties” (fișier de proprietăți specific pentru Spring). De asemenea, clasa “MessagesGetter”, care face același lucru pentru fișierul de message “messages.properties” și clasa “ResourceGetter”, care preia conținutul altor fișiere (cum ar fi cele descrise mai sus pentru scripturi).



În final, pentru partea de configurare mai intră și clasele de validări, cum ar fi “EmaiValidator”, “PasswordMachesValidator” și “PasswordPatternValidator”, care, setate pe un anumit câmp (sau chiar clasa în cazul PasswordMachesValidator), ajută la validarea conținutului pentru acel câmp / clasă. Astfel, validarea pentru email se face după un anumit pattern specific email-urilor, validarea pentru parole identice compară cele două parole, iar cea pentru pattern-ul parolei, de asemenea folosește un regex pentru verificarea parolei.



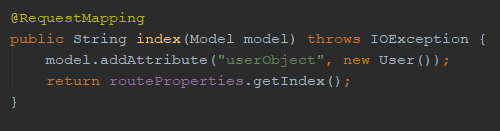


Urmează partea de web, care constă în mai multe controllere, modele, servicii, evenimente și alte caracteristici acestei comunicări între frontend și backend (REST).

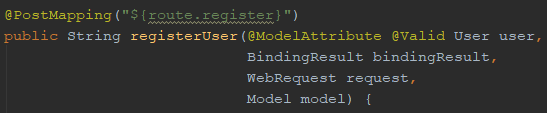
Aplicația Cassandra Interface, are 9 clase ce îndeplinesc rolul de controller. Folosind Thymeleaf, controllerul poate returna un string cu numele view-ului care trebuie afișat. Pentru stilizare și eficientă, numele respectiv este preluat din fișierul de proprietăți.

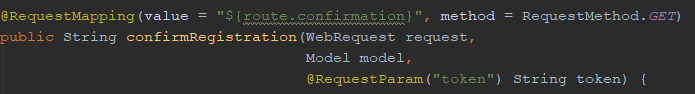
De asemenea este important de precizat că, fiind o aplicație în care cererile sunt aproape întotdeauna dinamice (tabele, tipuri, funcții, fișiere, etc create dinamic de către ușeri), iar modelele sunt denormalizate (datele cu interes comun în principiu sunt toate într-un singur tabel, dat fiind că este o bază de date NoSQL, adică nerelationara) , aplicația primește aceste cereri numai sub formă de GET și POST (în general cerearea GET este când se preiau date din baza de date, iar POST când se modifică), nefolosind astfel cereri de tip PUT sau DELETE. În consecință, controllere acceptă numai astfel de cereri (GetMapping și PostMapping – adnotări folosite pentru framework-ul Spring pentru a știi ce fel de cerere să intercepteze controllerul).

Primul este HomeController și are ca scop prelucrarea request-urilor pentru pagina de Home. În esență, acesta trebuie doar să creeze obietul User pe care să-l transmită în view, ca Thymeleaf să-l poată folosi pentru formularul de register.



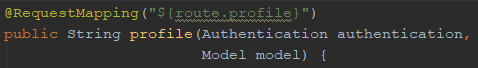
Următoarele trei sunt legate de înregistrarea în aplicație a unui utilizator. Primul din cele trei este RegisterController care primește datele din formularul de înregistrare, le validează și returnează rezultatul. Apoi urmează ConfirmationController care este apelat atunci când utilizatorul dă click pe link-ul trimis pe email, verificând astfel token-ul de confirmare și validându-l sau respingându-l dacă este cazul. Ultimul este RecoveryController, care primește datele din formularul de recovery, testează ce tip de cerere este (pentru resestare parolă sau pentru retrimitere email de activare) și acționează în consecință.







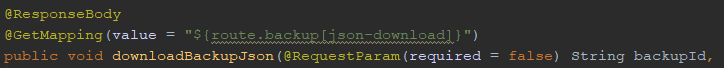
Următorul este ProfileController și are ca scop prlucrarea request-urilor pentru pagina de Profil. Ruta principală, care returnează pagină de profil are sarcina de a pregăti notificările utilizatorului (dacă există), sortându-le descrescător după dată și anunțarea dacă sunt sau nu notificări necitite. Apoi mai sunt rutele care sunt apelate la anumite evenimente: schimbarea avatarului (care primește o cerere Ajax și returnează un obiect în format JSON), citirea notificatilor, schimbarea parolei și a email-ului, email care și de data aceasta trebuie confirmat.



Mai departe vorbim despre SearchController, care se ocupă de cererile Ajax provenite de la căutările Live din cadrul aplicației. Se poate observa aici adnotarea diferită a controllerului, fiind @RestController în loc de @Controller ca la celalalte, diferența fiind că un RestController va întoarce un “body” (de obicei string JSON) în loc de un view, rezultat acceptat de către cererile Ajax. Căutarea Live este eficientă datorită faptului că rezultatele apar în timp real pe măsură ce parametrul de căutare este scris, elementele astfel (în număr mare și nefiltrate) nefiind afișate în view de la început, economisindu-se astfel memorie și spațiu. Astfel, acest controller se ocupă de căutările Live pentru useri, tabele, view-uri, coloane și funcții, pregătind datele pentru că plugin-ul Select2 să le poată primi și afișa în frontend (vom discuta mai încolo despre acest sistem).



BackupController se ocupă de descărcarea unui anumit backup (după id) sub forma unui fișier JSON (backup-ul are deja stocate datele sub forma unui JSON, acest controller doar trebuind să le copieze în răspuns-ul către frontend, care este transformat într-un răspuns de tip fișier). Controller-ul caută backup-ul după id în baza de date, îl găsește dacă există și acționează în consecință. Răspunsul returnat fiind un fișier și nu un view, în loc de adnotarea @RestController asupra clasei, s-a ales folosirea adnotarii @ResponseBody, care semnifică același lucru însă nu pe toată clasa ci doar pe funcția respectivă.



Următorul este AdminConsoleController care are sarcina de a gestiona cererile legate de pagina Console, disponibilă numai adminilor. În principiu acest Controller verifică autenticitatea persoanei care îl accesează (să aibă drept de acces), iar la trimiterea unei comenzi (query) printr-o cerere Ajax, o procesează și returnează rezultatul care va fi apoi afișat în consolă.

Ultimul și cel mai complex este MyDatabaseController care se ocupă de toate cererile care țin de paginile din My Database descrise la Client-Side. Pentru început putem spune că aproape toate metodele sunt pe baza sesiunii utilizatorului. Odată ce acesta se conectează la un keyspace, mereu se va testa dacă acel keyspace este încă activ și bineînțeles dacă are acces. Astfel, acest controller poate primii cereri pentru:

* Conectare, deconectare, editare, ștergere keyspace.
* Adăugare / ștergere membru keyspace
* Crearea, ștergerea, alterarea unei structuri.
* Inserarea, editarea, ștergerea datelor dintr-o structură.
* Selectarea datelor dintr-o structură și pregătirea lor pentru afișare (în funcție de tipul de date).
* Executarea query-urilor dinamice din consolă / script, detectând tipul acestora și acționând în consecință.
* Exportarea unei structuri.
* Crearea de log-uri pentru fiecare modificare a bazei de date.
* Schimbare panel (pagina) din MyDatabase (actualizare sesiune).
* Schimbare conținut în consolă sau script (cerere ajax).
* Căutare în baza de date.

Atunci când utilizatorul preia date din baza de date sau le vizualizează în pagina de View, acestea trebuie procesate înainte, deoarece Cassandra, având numeroase tipuri de date și colecții, integrarea cu Java îngreunează acest proces mai ales că datele sunt dinamice. Astfel, tot în acest controller a fost simulată o interpretare a datelor în funcție de tipul lor, iar în cazul colecțiilor s-a folosit o funcție recursivă pentru a altera primitivele din acestea. Pentru cazurile tipulor frozen, care pot fi de orice tip (creat sau nu de utilizator), aplicația încă nu oferă suport vizual la fel de bun, însă în viitor se va implementa și acest lucru. Din acest motiv nici importul datelor nu este posibil momentan.

De asemenea, fiind o aplicație la care mai mulți utilizatori pot executa comenzi în același timp, datele sunt re-actualizate înainte de fiecare acțiune pentru a nu exista conflicte. De asemenea, sunt afișate mesaje informative (preluate din fișierul de mesaje) pentru fiecare acțiune, astfel utilizatorul va știi mereu în ce situație se află. Astfel, acest controller se ocupă de toate aceste caracteristici, facilitând astfel comunicarea dintre utilizator și baza de date.

Acum urmează descrierea modelelor aplicației. Totul fiind mai mult dinamic, nu există decât 3 modele și anume User, Keyspace și Backup. Aceste modele la rândul lor au anumite câmpuri de tip “frozen”, fiind create 4 tipuri (UDT) pentru ele. Sunt făcute în așa fel încât să conțină destul de multe informații într-un singur “tabel”, fiind o bază de date NoSQL, relațiile nu există, iar tabelel sunt eficiente dacă sunt denormalizate. În continuare urmează prezentarea lor:

Modelul User are 13 câmpuri:

* Id – string, cheiea primară în care se va depozita un id de forma UUID (universally unique identifier).
* Email – string, validat cu validatorul de email descris mai sus.
* Username – string, validat cu pattern și indexat în baza de date, fiind un câmp foarte folosit.
* Password și MatchingPassword – string-uri, trebuie să fie identice (se validează întreaga clasă), însă în baza de date este trecut numai câmpul Password.
* Roles – listă de stringuri, conține rolurile de care dispune un user în aplicație (user sau admin).
* RegisterDate – tip date, data de înregistrare.
* Enabled – boolean, true dacă user-ul a confirmat email-ul, false în caz contrar.
* Token – string, reține token-ul care va fi comparat cu cel trimis pe email pentru confirmarea acestuia.
* ExpiryDate – tip date, reține data la care token-ul de confirmare va expira.
* EmailToken – string, reține token-ul care fi comparat cu cel trimis pe email pentru schimbarea acestuia.
* Avatar – string, reține url-ul avatarului utilizatorului.
* Notifications – listă de UserNotification (UDT), reține notificările utilizatorului. UserNotification conține următoarele:
  + Author – string, reține autorul notificării.
  + Date – tip date, conține data la care este trimisă notificarea.
  + Read – boolean, true dacă notificarea a fost citită, false în caz contrar.
  + Content – string, conține conținutul notificării.
* Keyspaces – listă de UserKeyspace (UDT), reține keyspace-urile din care face parte userul. UserKeyspace conține următoarele:
  + Keyspace – acest câmp nu apare în baza de date, dar există pentru că acest UDT este folosit pentru a reține în sesiunea utilizatorului atunci când este conectat la un keyspace, iar în acest câmp se va reține keyspace-ul (modelul) propriu-zis.
  + Name – string, numele keyspace-ului.
  + Access – string, accesul pe care îl are userul în keyspace-ul respectiv.
  + CreatorName – string, numele creatorului keyspace-ului.

Modelul Keyspace are 9 câmpuri și este de precizat că la crearea unui keyspace, pe lângă că este trecut în acest tabel, este și creat fizic în Cassandra.

* Id – string, cheiea primară în care se va depozita un id de forma UUID.
* Name – string, validat cu pattern, reține numele keyspace-ului care este format astfel: NumeCreator\_NumeKeyspace, astfel evitând conflictele de nume în baza de date.
* PasswordEnabled – boolean, true dacă este protejat cu parolă, false în caz contrar.
* Password și MatchingPassword – string-uri, trebuie să fie identice (se validează întreaga clasă), însă în baza de date este trecut numai câmpul Password.
* CreationDate – tip date, reține data la care este creat keyspace-ul.
* ReplicationFactor – integer, minim 1 – maixm 8, reține factorul de replicare al keyspace-ului.
* DurableWrites – boolean, true dacă keyspace-ul permite durable writes, false în caz contrar.
* Log – listă de KeyspaceLog (UDT), reține log-urile pentru keyspace-ul respectiv. KeyspaceLog conține următoarele:
  + Content – string, reține conținutul logului.
  + Date – tip date, reține data la care a fost creat logul.
  + Username – string, reține userul care prin acțiunile lui a creat logul respectiv.
  + Type – string, reține tipul logului, care poate fi de: creare, editare, ștergere.
* Users – listă de KeyspaceUser (UDT), reține userii care aparțin de keyspace-ul respectiv. KeyspaceUser conține următoarele:
  + Username – string, reține username-ul utilizatorului care are acces la acest keyspace.
  + Acces – string, reține accesul pe care îl are utilizatorul cu acces la acest keyspace.

Modelul Backup are 4 câmpuri:

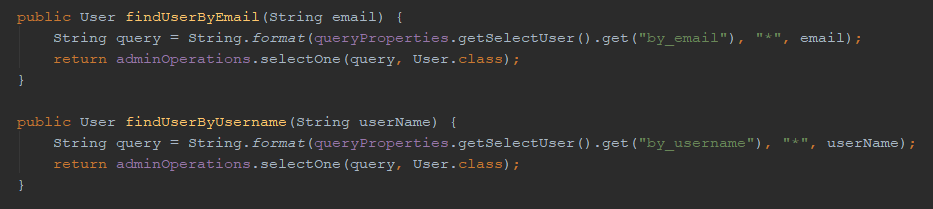
* Id-string, cheiea primară în care se va depozita un id de formă UUID.
* Content – string, reține conținutul backup-ului, în general sub format JSON.
* Date – tip date, reține data la care a fost creat backup-ul.
* KeyspaceName – string, reține numele keyspace-ului în contextul căruia a fost creat backup-ul.

Aceste modele sunt adnotate în primul rând cu @Table, notație care spune framework-ului să creeze aceste tabele fizic în baza de date (fiind integrat cu Cassandra, acest lucru este posibil). Apoi, mai sunt adnotații din cadrul librăriei Lombok, care o să fie discutată mai încolo).

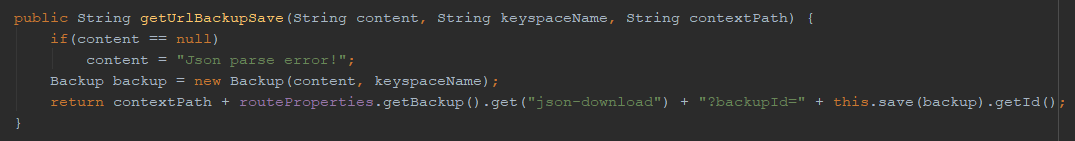
Mai departe vorbim despre clasele adontate cu @Service, notație care spune framework-ului să trateze aceste clase ca făcând parte din contextul aplictiei și care vor fi folosite spre a oferi diferite servicii. În acest caz, acestea se vor ocupa de diferite tranzacții asupra bazei de date, folosind un “bean” creat la configurarea Cassandrei (descrisa anterior) și anume CassandraAdminOperations, care permite conexiunea directă cu baza de date, permițând executarea comenzilor și preluarea datelor. Aplicația Cassandra Interface are 3 servicii.

Primul este UserService, care se ocupă de tranzacțiile pentru tabelul modelului User, descris mai devreme. Aceste tranzacții pot fii de: adăugare, ștergere, editare, activare și căutare după mai multe criterii.

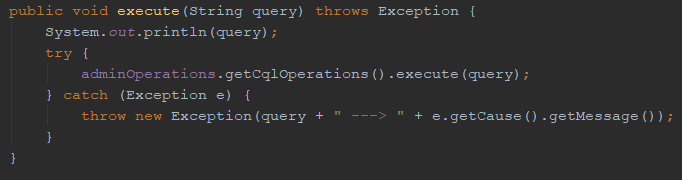




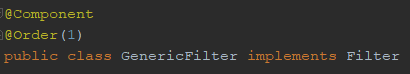
Al doilea este BackupService, care se ocupă de crearea, căutarea după id și formarea link-ului de download pentru un obiect de tip Backup.

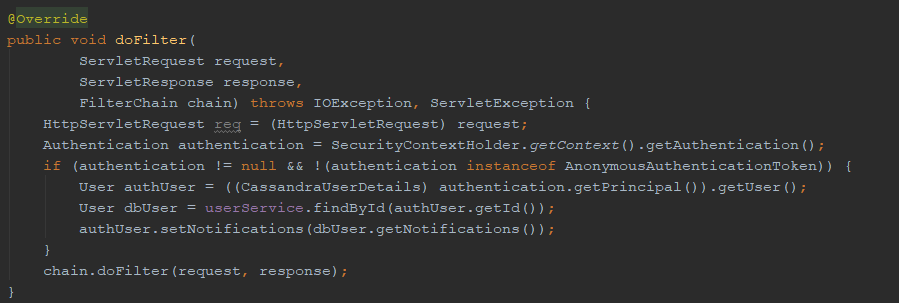


Ultimul este KeyspaceService, care servește la mai multe tranzacții, nu numai celor legate de modelul Keyspace. În afară de tranzacțiile pentru modelul Keyspace, tot de acest serviciu se folosește aproape întreg MyDatabaseController-ul, facilitând crearea, editarea, ștergerea și preluarea tuturor structurilor din baza de date create dinamic de către utilizatori. Acestea se află în acest serviciu datorită faptului că utilizatorii depind și pot interacționa cu baza de date numai prin intermediul keyspace-ului la care sunt conectați.

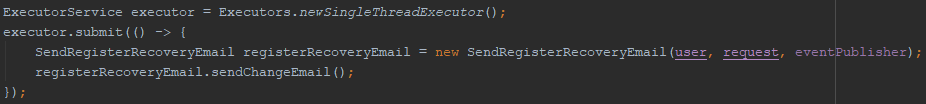


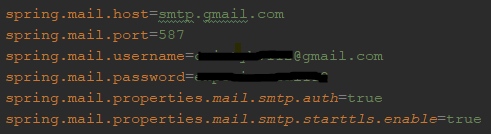
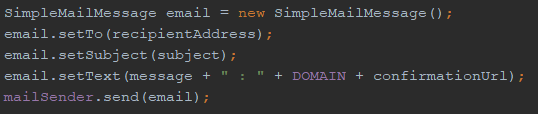
Tot pe partea de server în categoria web, aplicația mai conține GenericFilter, o clasă care implementează clasa Filter și este adnotată cu @Component, spunând astfel framework-ului să creeze un bean și să folosească clasa acesta în scopul pentru care a fost defnitia (dacă are un scop). În aplicațiile web, filterul are rolul de a se apela înaintea controller-ului, pentru a “filtra” informațiile din request. Pentru aplicația Cassandra Interface, filterul este apelat înaintea oricărui Controller, verificând dacă utilizatorul care a făcut cererea este autentificat, iar în caz pozitiv, actualizează notificările acestuia în contextul Spring. Acest sistem este important pentru a afișa în timp real utilizatorului notificările primite, iar utilizarea filter-ului este esențială fiind inutilă actualizarea în fiecare controller separat.





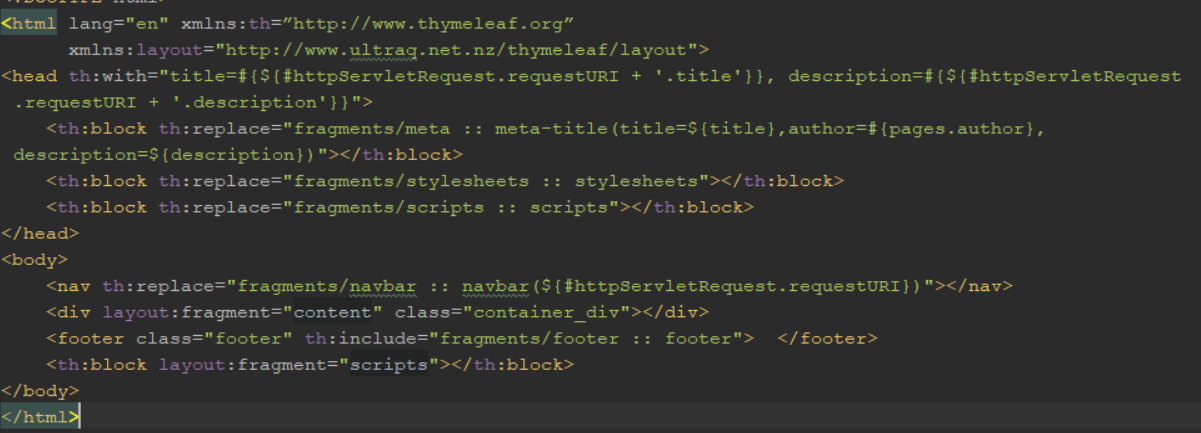
Acum vorbim despre sistemul de trimitere al email-ului de confirmare / recuperare. Pentru ca aplicația să nu aștepte până la efectuarea acestuia, s-a folosit un Thread. Thread-ul permite executarea în paralel a unui alt proces. Pe lângă acest lucru, a fost folosit sistemul de evenimente, care stabilește trimiterea email-ului ca un eveniment în cadrul apicatiei, nefiind eficient ca acest proces să se execute în cadrul controllerului. Astfel, atunci când se trimite un email, se creează un thread în interiorul căruia se “publica” evenimentul de trimitere a email-ului, acesta urmând să fie trimis chiar dacă aplicația și-a continuat cursul. Expedierea este posibilă importând anumite clase (prin Maven), printre care și SimpleMailMessage, care permite trimiterea de email-uri, setând înainte în proprietăți datele necesare conectării aplicației cu un cont de Gmail (stabilind astfel și expeditorul).





În final, vorbim puțin despre arhitectura template-urilor în Cassandra Interface.

Folosind dialectul Layout din Thymeleaf, avem un document html denumit layout.html, care este folosit în “decorarea” celorlalte pagini. În acest layout sunt incluse de asemenea fragmente de meta-uri, script-uri javascript, fișiere css, meniul aplicației cât și footer-ul acesteia. Important de precizat este ca, folosind comenzi din Thymeleaf, atât titlul cât și descrierea paginii sunt construite dinamic, detectând pagina care implementează layout-ul și preluând din fișierul de mesaje titlul sau descrierea respectivă.



Folosind sesiunea pentru monitorizarea acțiunilor utilizatorului, paginile ce desemnează interacțiunea cu baza de date nu sunt toate încărcare, ci doar acelea care aparțin de spațiul curent “ales” de utilizator și care se regăsește în sesiunea acestuia, înlăturând astfel încărcarea codului nefolosit. Astfel, cam toate documentele html sunt contruite la rândul lor dinamic din mai multe fragmente. Se poate

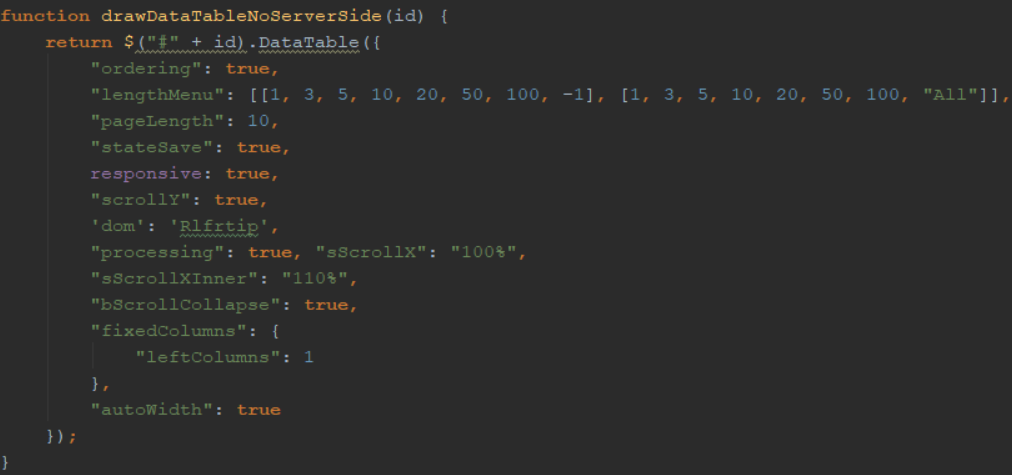
Mai este de precizat că datele de pe fiecare linie a unui tabel nu sunt adăugate în formularul de editare de la început. Atunci când utilizatorul accesează datele respective (prin click pe opțiuni sau prin dublu-click pe linia respectivă), datele sunt introduse în formularul unic, prelucrat la inițializarea paginii folosind numele și tipul coloanelor tabelului respectiv.

Librării / Plugin-uri folosite

Pentru eficientă, aplicația Cassandra Interface folosește anumite librării și plugin-uri pentru a facilita anumite procese.

Pentru început vom discuta despre plugin-ul DataTables, care este importat ca fișier javascript și stilizat prin css. Acest plugin oferă o automatizare a creării și filtrării tabelelor. Dat fiind niște headere și niște date (un tabel), DataTables le poate primi ca input sau le poate găsi direct în document, și aplicat asupra lor, va crea automat tabelul dorit. Pe lângă acest lucru, plugin-ul deține filtrări ca ordonare, căutare, paginare și numerotare a datelor din tabel, fiind extrem de util utilizatorului, având acces total și rapid asupra datelor respective.

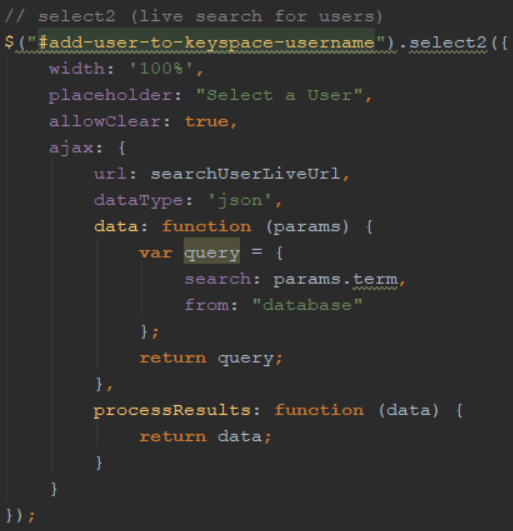
Un alt lucru important de precizat despre DataTables, lucru folosit și în aplicația Cassandra Interface, este că datele și filtrarile pot fi prelucrate atât în partea de client, cât și în partea de server. Astfel, în cadrul aplicației, datele tabelelor, acestea putând conține sute de mii de date, sunt prelucrate pe partea de server, plugin-ul folosind Ajax pentru a le cere (transmițând la server starea curentă a tabelului și cerințele de filtrare) și a le prelua. Legat de datele cu privire la structurile care aparin de keyspace-ul curent (partea de sus din View/Edit), nefiind vorba de atât de multe linii, acestea pot fi stocate și filtrate în partea de client.



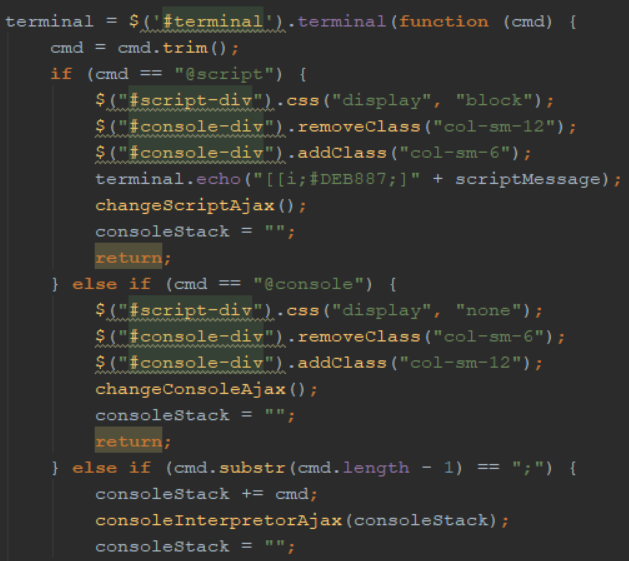
Am ales să folosesc acest plugin deoarece este foarte eficient și ușor de implementat, fiind setat numai javascript și stilizat în css.

Următorul plugin descris este Select2, care este folosit pentru a transforma un element de tip <select> din HTML, într-un input text folosit pentru comunicarea cu serverul, care la rândul lui va prelua textul respectiv, îl va prelucra și va returna date sub formă de listă de id-uri și texte, folosit apoi de plugin pentru a crea opțiuni pentru selectare de către utilizator.

Astfel, plugin-ul Select2 în aplicația Cassandra Interface este folosit pentru căutările Live în baza de date, permițând user-ului să introducă un parametru de catuare care este trimis la server prin Ajax, unde se caută rezultate pe baza acelui parametru și se returnează astfel încât să se creeze opțiuni dintre care utilizatorul să aleagă.



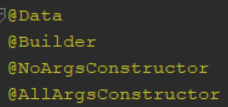
Un alt plugin important este Jquery Terminal, care permite simularea unei console și interacțiunea cu aceasta prin fișiere javascript. Aplicația Cassandra Interface controlează consola, primind comenzile efectuate de client, urmând apoi să fie trimise la server prin Ajax unde se detectează și validează (în mare parte query-uri), și se returnează rezultatul, care va fi apoi re-prelucrat și afișat în aceeași consolă. Figura de mai jos reprezintă inițializarea terminalului și prelucrarea a câtorva comenzi introduse în consola de la Console/Script.



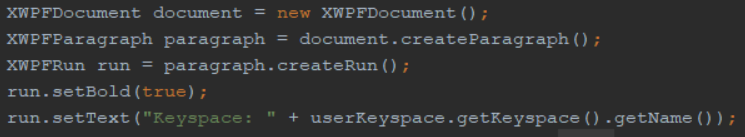
De asemenea, am folosit un plugin-ul ColResizable, pentru a permite utilizatorului să mute și să redimensioneze coloanele unui tabel.

Mai departe vorbim despre librăriile importante folosite în aplicație. Sunt 3 la număr:

* Lombok, o librărie care permite crearea anumitor metode prin adnotarea claselor. În aplicație, Lombok este folosit pentru a creat automat gettere, settere, contructori și buildere (metodă de a instanția o clasă fără a trebui incluși toți parametrii din constructor) pentru diferite clase și modele.



* ApachePOI, librărie care permite crearea fișierelor ca Word și Excel, folosite în aplicație pentru exportul datelor.



* ObjectMapper – Jackson, librărie care permite transformarea unui obiect într-un format JSON, folosit de asemenea pentru export și pentru crearea backup-urilor.



Astfel, tot ceea ce este descris mai sus face parte din partea de server a aplicației Cassandra Interface, totul încercând să fie cât mai eficient și mai coerent atât pentru programator cât și pentru utilizator.

Concluzii

În concluzie, aplicația Cassandra Interface aduce un mare avantaj în folosirea bazei de date Apache Cassandra.

Fiind o aplicație care definește o interfață, aceata oferă un suport vizual mult mai mare decât cea oferită de Cassandra, care constă în mare parte într-o consolă neatractivă în care datele nu se pot vizualiza concomitent, iar toate acțiunile trebuie să fie sub formă de comenzi query. Cassandra Interface, pe lângă că oferă aceleași beneficii, în plus permite prelucrarea și interacțiunea cu baza de date prin doar câteva click-uri, sub o formă plăcută, inteligibilă și complexă.

Vorbind de securitate, și interfața veche oferă autentificare, însă nu la nivelul pe care îl oferă aplicația prezentată, având mai multe roluri, mai mult acces și mai multă siguranță, existând log-uri și backup-uri care asigură acest lucru.

Din punct de vedere al exportului, Cassandra Interface oferă mult mai multe metode decât interfața veche, putând salva atât tabele și view-uri că Excel și JSON, dar și log-uri și chiar întreg keyspace-ul.

Am vorbit de trecut, de prezent și acum să vorbim despre viitor. Aplicația Cassandra Interface lasă loc de viitoare îmbunătățiri, cum ar fi:

* Posibilitatea de a importa datele din fișiere externe.
* Suport vizual pentru toate tipurile de date din Cassandra.
* Îmbunătățirea eficienței pentru un număr foarte mare de date.
* Adăugarea unui forum unde utilizatorii pot comunica.

Lista figuri

1. Funcționarea replicării datelor într-un cluster, pag6
2. Schema unei familii de coloane, pag7
3. Demonstrarea funcției de "run" din clasa CommandLineAppStartUpRunner, pag8
4. Demonstrare expresie lamda și stream-uri în Java, pag9
5. Obiect adnotat cu @Autowired (dependency injection), pag10
6. Clasa adnotată cu @Configuration (clasă de configurare), pag10
7. Metodă adnotată cu @Bean (creându-se astfel o instanță în contextul Spring), pag10
8. Schema Model-View-Controller, pag11
9. Clasă adnotată cu @Controller, pag11
10. Mesajul inițial la rularea serverului Spring, pag12
11. Exemplu importare dependință în pom.xml, pag12
12. Mesaj inițializare Tomcat la rularea serverului Spring, pag12
13. Configurare Spring Security, pag13
14. Schema pentru framework-ul Spring în timpul rulării, pag13
15. Import Layout Dialect pentry Thymeleaf în tag-ul <html>, pag14
16. Folosirea Layout Dialect pentru a înlocui cod cu un fragment, pag15
17. Exemplu sintaxa Spring Security pentru testarea de autentificare, pag15
18. Exemplu sintaxa Spring Security pentru testarea rolului de ADMIN, pag15
19. Configurarea dialectelor în Thymeleaf (server-side), pag15
20. Schema HTML5, pag16
21. Exemplu script Jquery de eliminare a duplicatelor dintr-un element, pag18
22. Formularul de autentificare în aplicație, pag20
23. Mesajul de eroare la autentificare, pag20
24. Formularul de recuperare a contului, pag21
25. Formularul de înregistrare în aplicație, pag21
26. Formularul de creare a unui keyspace, pag22
27. Buton de profil și logout din meniul aplicației, pag24
28. Datele utilizatorului din pagina de profil, pag24
29. Eroare la schimbarea email-ului, pag24
30. Eroare la schimbarea parolei, pag24
31. Schimbarea parolei cu succes, pag24
32. Panoul de notificări din pagina de profil, pag25
33. Buton de profil cu icon-ul de mesaje prezent, pag25
34. Butonul de Home și de MyDatabase din meniul aplicației, pag25
35. Lista keyspace-urilor utilizatorului, pag26
36. Informații despre keyspace, pag26
37. Setări pentru keyspace, pag27
38. Statistici pentru keyspace, pag27
39. Accesul în keyspace, pag27
40. Formular de creare tabel, pag28
41. Blocul structurilor și tabelul cu tabelele din keyspace (pagina View/Edit), pag28
42. Formular de inserare linie într-un tabel, pag28
43. Blocul cu datele unui tabel / view, pag28
44. Exemplu utilizare consolă, pag29
45. Exemplu utilizare script, pag30
46. Cele 4 tipuri de export, pag31
47. Exemplu afișare log-uri, pag31
48. Exemplu căutare în baza de date, pag32
49. Clasa de configurare a bazei de date care extinde AbstractCassandraConfiguration, pag33
50. Inserarea conturilor și keyspace-urilor default și executarea scripturilor inițiale, pag34
51. Handler-ul de logout, pag34
52. Handler-ul de autentificare eșuată, pag35
53. Configurare locație template-uri pentru Thymeleaf, pag35
54. Exemplu clasă de proprietăți, pag35
55. Exemplu clasă de preluare a mesajelor din messages.properties, pag35
56. Exemplu funcție de validare a email-ului, pag36
57. Exemplu validare a parolei, pag36
58. Exemplu validare a unui obiect cu parole identice, pag36
59. Antetul funcției index din controllerul de Home, pag37
60. Antetul funcției din controllerul Register ce se ocupă de înregistrarea în aplicație, pag37
61. Antetul funcției din controllerul Confirmation ce se ocupă de confirmarea email-ului, pag37
62. Antetul funcției din controllerul de Recovery ce se ocupă de recuperarea contului, pag37
63. Antetul funcției din controllerul Profile, pag37
64. Exemplu adnotație @RestController, pag38
65. Exemplu adnotație @ResponseBody, pag38
66. Exemplu adnotație @Service, pag42
67. Exemplu funcție de căutare user după email sau username, pag42
68. Exemplu funcție care creează un backup și returnează link-ul de download pentru acesta, pag42.
69. Exemplu funcție care execută un query în baza de date, pag42
70. Exemplu adnotație @Component, pag43
71. Funcția din GenericFilter care se ocupă cu actualizarea notificarilor utilizatorului, pag43
72. Exemplu thread care publică un eveniment de trimitere a unui email, pag43
73. Exemplu trimitere email, pag43
74. Setările din application.properties pentru trimiterea de email-uri, pag43
75. Prezentare a layout-ului folosit în aplicație, pag44
76. Exemplu creare tabel (client-side) în jquery folosind plugin-ul DataTables, pag45
77. Exemplu inițializare plugin Select2 pentru un element select, pag46
78. Exemplu inițializare plugin Jquery Terminal și interpretarea unor comenzi, pag46
79. Exemplu adnotații Lombok, pag47
80. Exemplu creare fișier Word folosind ApachePOI, pag47
81. Exemplu transformare obiect în format JSON folosind ObjectMapper-Jackson, pag47

Bibliografie

[1] Apache Cassandra Documentation 4.0, <http://cassandra.apache.org/doc/latest/>

[2] Thymeleaf – Tutorial: Using Thymeleaf, <https://www.thymeleaf.org/doc/tutorials/2.1/usingthymeleaf.html>

[3] Apache Cassandra – Wikipedia, <https://en.wikipedia.org/wiki/Apache_Cassandra>

[4] Java (limbaj de programare) – Wikipedia, <https://ro.wikipedia.org/wiki/Java_(limbaj_de_programare)>

[5] Java 8 programming for beginners: Go from zero to hero, <https://www.infoworld.com/article/3130466/java-8-programming-for-beginners-go-from-zero-to-hero.html>

[6] Spring Framework – Wikipedia, <https://en.wikipedia.org/wiki/Spring_Framework>

[7] Spring Boot Refference, <https://docs.spring.io/spring-boot/docs/2.1.5.RELEASE/reference/pdf/spring-boot-reference.pdf>

[8] Reflexion (computer programming) – Wikipedia, <https://en.wikipedia.org/wiki/Reflection_(computer_programming)>

[9] Programarea orientata pe aspecte – Wikipedia, <https://ro.wikipedia.org/wiki/Programarea_orientat%C4%83_pe_aspecte>

[10] Model-view-controller – Wikipedia, <https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller>

[11] Girish Margan, Model-View-Controller (MVC), May 17 2018 - <https://medium.com/datadriveninvestor/model-view-controller-mvc-75bcb0103d66>

[12] Apache Tomcat – Wikipeida, <https://en.wikipedia.org/wiki/Apache_Tomcat>

[13] Maven, Apache Maven Project – Introduction to the POM, <https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>

[14] Spring Framework – Introduction to the Spring Framework, <https://docs.spring.io/spring/docs/4.3.x/spring-framework-reference/html/overview.html>

[15] Distributed database – Wikipedia, <https://en.wikipedia.org/wiki/Distributed_database>

[16] Wide column store – Wikipedia, <https://en.wikipedia.org/wiki/Distributed_database>

[17] Tutorialspoint – Cassandra Tutorial, <https://www.tutorialspoint.com/cassandra>

[18] Datastax Documentation – CQL Refference, <https://docs.datastax.com/en/cql/3.3/cql/cql_reference/cqlCommandsTOC.html>

[19] Javapoint – Spring Security Tutorial, <https://www.javatpoint.com/spring-security-tutorial>

[20] Cross-site request forgery – Wikipedia, <https://en.wikipedia.org/wiki/Cross-site_request_forgery>

[21] Thymeleaf – Wikipedia, <https://en.wikipedia.org/wiki/Thymeleaf>

[22] HTML – Wikipedia, <https://en.wikipedia.org/wiki/HTML>

[23] HTML5 – Wikipedia, <https://en.wikipedia.org/wiki/HTML5>

[24] Web Techology and Trend: HTML5, <http://www-scf.usc.edu/~chenemil/itp104/webtech.html>

[25] Cascading Style Sheet – Wikipedia, <https://en.wikipedia.org/wiki/Cascading_Style_Sheets>

[26] Boostrap – Wikipedia, <https://en.wikipedia.org/wiki/Bootstrap_(front-end_framework)>

[27] JavaScript – Wikipedia, <https://en.wikipedia.org/wiki/JavaScript>

[28] Jquery – Wikipedia, <https://en.wikipedia.org/wiki/JQuery>

[29] DataTables – Manual, <https://datatables.net/manual/>

[30] Select2, <https://select2.org/>

[31] Jquery Terminal, <https://terminal.jcubic.pl/api_reference.php>

[32] JqueryScript – Jquery plugin for draggable resizable table columns – colResizable, <https://www.jqueryscript.net/table/jQuery-Plugin-For-Draggable-Resizable-Table-Columns-colResizable.html>

[33] Project Lombok, <https://projectlombok.org/>

[34] Apache POI, <https://poi.apache.org/>

[35] ObjectMapper – Jackson, <https://fasterxml.github.io/jackson-core/javadoc/1.9/org/codehaus/jackson/map/ObjectMapper.html>