



**UNIVERSITATEA DIN
BUCUREȘTI**

**FACULTATEA DE
MATEMATICĂ ȘI
INFORMATICĂ**



SPECIALIZAREA INFORMATICĂ

Lucrare de disertație

Aplicație Cloud Distribuită pentru Managementul și Dezvoltarea de Software

Absolvent

Alexiu Adrian-Ștefan

Coordonator științific

Prof. Kevorchian Cristian

București, iunie 2023

Rezumat

HotSpotWeb, soluția dezvoltată de mine ce se vrea a fi un proof-of-concept al ideii de management al software-ului dintr-o singură platformă, încearcă să demonstreze prin mai multe aspecte ale funcționalităților sale și prin procesul de dezvoltare al acesteia în sine avantajele dezvoltării de software folosind automatizări.

Arhitectura aplicației HotSpotWeb constă în mai multe module cheie: HotSpotWeb.Application, HotSpotWeb.Core, HotSpotWeb.Migrator, HotSpotWeb.Web.Core și HotSpotWeb.Host, care sunt implementate în .NET Core și utilizează MSSQL drept bază de date.

Interfața cu utilizatorul a aplicației HotSpotWeb permite generarea diverselor tipuri de aplicații, precum Ruby on Rails, .NET, Angular, React și altele, pe baza configurațiilor și dependențelor specificate. Utilizatorii pot descărca aplicația generată, crea un repository pe Github și încărca aplicația generată pe acest repository sau pe unul creat anterior. De asemenea, utilizatorului îi este permisă configurarea multiplelor profile de Github pe care acesta poate încărca aplicațiile.

Tehnologiile cheie utilizate în cadrul aplicației HotSpotWeb includ Angular pentru partea de frontend, Ruby on Rails, Redis și Postgresql pentru HotSpotWebCommandsService, Sidekiq pentru gestionarea sarcinilor în fundal, Docker pentru containerizare și Azure Service Fabric pentru implementarea și distribuția aplicației în cloud.

Această lucrare examinează în detaliu arhitectura și funcționalitățile aplicației HotSpotWeb, precum și tehnologiile utilizate pentru dezvoltarea și gestionarea acesteia. De asemenea, sunt prezentate aspecte legate de implementarea, alături de discuții și concluzii relevante.

În concluzie, aplicația HotSpotWeb reprezintă o soluție generală pentru dezvoltarea și gestionarea aplicațiilor software distribuite în cloud. Această lucrare oferă o prezentare cuprinzătoare a aplicației, tehnologiilor utilizate și rezultatelor obținute, deschizând calea către dezvoltări viitoare și îmbunătățiri ale aplicației HotSpotWeb.

Abstract

În lumea modernă, unde a apărut o nevoie nestăvilită de viteză, fie că vorbim de deplasare sau alte aspecte ale vieții, am ajuns într-un punct în care orice secundă contează. Lumea dezvoltării de software este, în opinia mea, unul dintre factorii principali care determină această manieră rapidă în care ne trăim viețile.

Companiile, organizațiile și indivizii se bazează pe aplicații software pentru a eficientiza proceduri și operațiuni, îmbunătăți productivitatea și livra produse de calitate în cel mai scurt timp. Totodată, cu cât se dezvoltă mai mult software, cu atât acesta devine mai complex, durează mai mult timp începerea dezvoltării efective și este cu atât mai greu de administrat.

Utilizatorii au ajuns să se aștepte de la aplicații la experiențe fără întreruperi și un mod intuitiv și ușor de folosit. De asemenea, datorită dezvoltării masive a unor domenii precum inteligența artificială, realitatea augmentată și virtuală și altele, o interfață cu trei butoane nu mai este de ajuns pentru utilizatori. Software-ul se dezvoltă la viteze uluitoare și pentru asta avem nevoie ca el să fie stabil, scalabil și sigur, pentru ca alte procese să se poată desfășura fără întreruperi.

Cuprins

Introducere.....	4
1.1 Contextul și motivația lucrării	4
1.2 Obiectivele lucrării	5
Concepte Teoretice	6
2.1 Tehnologii utilizate.....	6
2.2 Structura bazelor de date.....	11
2.3 Concepte și metodologii folosite.....	14
2.4 Distribuirea aplicațiilor în Azure Service Fabric	15
Arhitectura aplicației HotSpotWeb.....	17
3.1 ASP.NET – Backend	17
3.2 Angular – Frontend	19
3.3 Ruby on Rails – Serviciul de rulat comenzi	23
Discuții și Concluzii.....	25
4.1 Limitări și probleme întâmpinate	25
4.2 Direcții viitoare de dezvoltare pentru HotSpotWeb	25
4.3 Concluzie	26

Capitolul 1

Introducere

1.1 Contextul și motivația lucrării

Ideea aplicației a venit în urma unei experiențe personale ce a constat în nevoia de a dezvolta o serie de aplicații de Ruby on Rails cu diferite specificații, dar toate având, în principiu, același schelet. Cu siguranță soluția găsită nu a fost cea mai eficientă deoarece am sfârșit prin a copia și modifica manual toate aplicațiile. O soluție de genul HotSpotWeb ar fi reprezentat un avantaj imens în dezvoltarea acestor aplicații.

Aplicația HotSpotWeb facilitează crearea de aplicații personalizate pentru diferite tehnologii, cum ar fi Ruby on Rails, .NET, Angular, React și multe altele, bazate pe configurații și dependențe specifice. Orientarea proiectului către mediul cloud și utilizarea tehnologiilor moderne precum .NET Core, Angular, Ruby on Rails, Docker și Azure Service Fabric vine în întâmpinarea nevoilor actuale ale dezvoltatorilor de a obține o infrastructură scalabilă și ușor de administrat pentru aplicațiile lor distribuite.

Scopul final al aplicației este acela de a elimina pe cât posibil pașii mici și redundanți de la începutul fiecărei dezvoltări de software. Desigur, există soluții ce împlinesc acest obiectiv însă foarte puține dintre ele pot fi configurate în detaliu. Una dintre aceste soluții se numește ASP.NET Boilerplate [1] și a fost folosită în dezvoltarea aplicației din această lucrare.

1.2 Obiectivele lucrării

Precum am menționat anterior, aplicația se vrea a fi un proof-of-concept al ideii de management al software-ului dintr-o singură platformă, constând în configurare, generare, descărcare și încărcare (pe Github). Scopul aplicației este să demonstreze că nu este nevoie de ore pentru a începe dezvoltarea efectivă a logicii de business ci doar de câteva minute pentru a crea o configurare cu dependențele necesare.

De asemenea, această soluție de alcătuire a unui „schelet” poate însemna introducerea de bloatware (dependințe inutile) care necesită timp pentru a le înlătura. Din acest motiv, HotSpotWeb are capacitatea de a defini multiple configurări cu diferite dependențe ce pot fi alese în funcție de nevoi.

Un alt obiectiv specific al lucrării este acela de a analiza conceptele și tehnologiile utilizate în dezvoltarea produsului, precum microservicii, containere, rulare de job-uri în fundal dar și .NET Core, Angular, Ruby on Rails, Docker și Azure Service Fabric. În cadrul aceluiași obiectiv se va prezenta arhitectura și structura aplicației HotSpotWeb. De asemenea, se va analiza modul în care aceste componente interacționează și comunică între ele pentru a oferi funcționalități complexe.

Unul dintre obiectivele vitale este modul în care putem să definim cât se poate de general un proces pentru a îl putea automatiza, în speță generarea de aplicații personalizate pentru nevoi specifice. Pentru acest obiectiv voi descrie modul în care am reușit să definesc metoda prin care, folosind configurări predefinite, se rulează comenzi specifice generării de aplicații.

Obiectivul final este de a reflecta asupra procesului de dezvoltare și a problemelor întâmpinate, de a oferi concluzii și de a identifica posibile direcții viitoare de dezvoltare ale aplicației. Acest obiectiv va fi satisfăcut prin discuția punctelor specifice în care aplicația are probleme și funcționalități noi care pot face aplicația mai utilă.

Capitolul 2

Concepte Teoretice

2.1 Tehnologii utilizate

Deoarece natura și utilitatea microserviciilor este diferită, am folosit tehnologii diferite pentru acestea astfel încât să acopere toate nevoile și funcționalitățile. Pe scurt, interfața cu utilizatorul folosește framework-ul Angular, alături de Bootstrap, AdminLTE, css, ABP Framework [2], backend-ul ce face atât legătura cu baza de date și cu frontend-ul cât și cu microserviciul ce se ocupă cu rularea de comenzi este scris în .NET Core 7.0 iar microserviciul pentru rularea comenzilor este un monolit web dezvoltat folosind framework-ul Ruby on Rails cu versiunea de Ruby 3.1.2. În continuarea lucrării, urmează să descriu în detaliu de ce am ales tehnologiile respective și cum m-au ajutat acestea în procesul de dezvoltare al aplicației.

Deoarece prima interacțiune a utilizatorului cu aplicația HotSpotWeb se realizează prin intermediul interfeței din browser, voi începe să descriu ce conține și cum a fost realizată această componentă. La fel cum am menționat anterior, frontend-ul folosește framework-ul Angular, mai exact versiunea 15.0.3. În cadrul acestei aplicații am folosit diverse biblioteci ajutătoare sub forma de pachete npm precum Bootstrap pentru stilizarea componentelor aplicației, adminLTE ce se folosește de Bootstrap pentru a crea o interfață prestilizată de tip “Dashboard” și altele. Framework-ul ABP predefinește anumite elemente stilistice precum ferestre modale, avertizări, mesaje toast și elemente de loading pentru a eficientiza procesul de definire a astfel de entități.

Angular este un framework de frontend dezvoltat de Google și se remarcă prin multiple particularități de restul, însă am ales să vorbesc în principal de locul său în piață. Angular este un framework matur, ce și-a format deja o comunitate mare de utilizatori care stau la dispoziția și iau parte la dezvoltarea de elemente pentru Angular și plecând de la acesta. Fiind inițial lansat în 2009 sub numele de AngularJS și apoi fiind rescris

Capitolul 2

complet în 2016 (Angular-ul pe care îl folosim astăzi) [3]. Având o experiență vastă în anii săi de dezvoltare, a ajuns să fie un framework ușor scalabil și pregătit pentru Enterprise și aplicații mari. Din aceleași considerente documentația s-a dezvoltat foarte mult și acum este destul de complexă pentru a ajuta atât începători prin tutoriale cât și avansați prin aspecte complexe.

Un alt mare avantaj pe care îl are Angular este faptul că folosește Typescript. Sigur, multe alte framework-uri pot fi configurate să folosească acest superset a Javascript-ului precum React sau Vue, însă a doua iterație a Angular a fost construită pentru a folosi Typescript. Acest limbaj de programare vine în sine cu o multitudine de avantaje precum posibilitatea de a defini tipurile de date a variabilelor ceea ce va elimina erori legate de incompatibilitatea variabilelor.

În continuare voi prezenta framework-ul .NET Core 7.0 cu limbajul C# asociat. Pot spune că acest serviciu este cel mai important din toată configurația aleasă. Proiectul .NET este un REST API ce leagă aplicația de baza de date, frontend și microserviciul de rulat comenzi.

Cu riscul de fi nu fi imparțial și a nu face alegeri doar pe bază de logică, principalul motiv pentru care am ales acest framework este datorită faptului că sunt foarte familiar cu el și îmi place procesul de dezvoltare a aplicațiilor web în acesta. Din fericire, există multe alte motive pentru care .NET Core și ASP.NET sunt îndragite de comunitatea de developeri.

Înainte de apariția .NET Core, Microsoft a atras foarte multe critici în urma blocării acestui ecosistem în cadrul Windows și a refuzat ani la rând să îl generalizeze pentru mai multe platforme. În 2015 în schimb, gigantul tehnologic a anunțat faptul că .NET Core va fi open source [5] și astfel putea fi migrat pe Linux și alte platforme (Mac OSX). Acest lucru l-a făcut din ce în ce mai popular și în final face parte în continuare dintre cele mai puternice soluții pentru dezvoltarea de aplicații web.

Chiar și înainte de a fi făcut cross platform și portabil, framework-ul .NET era recunoscut pentru performanța și scalabilitatea sa superioară. Platforma beneficiază de un runtime optimizat și eficient ce permite rularea rapidă a informațiilor și gestionarea volumului mare de cereri. Arhitectura modulară și orientată pe microservicii a .NET Core facilitează dezvoltarea aplicațiilor scalabile, permițând împărțirea logicii în microservicii independente și gestionarea lor într-un mediu distribuit. De asemenea, .NET oferă suport nativ pentru sarcini asincrone, ceea ce permite API-ului să răspundă eficient unui număr mare de cereri.

Cu toate că Microsoft și-a înglobat toate aceste unelte puternice într-o “cușcă” proprietară

Capitolul 2

și nu le-a dat voie să iasă până nu demult, ecosistemul Microsoft are o multitudine de avantaje precum suport și documentație foarte exactă și cuprinzătoare, o bibliotecă cuprinzătoare de pachete și framework-uri (ASP.NET Core, Entity Framework Core sunt două exemple pe care le-am integrat în proiect) și, în final, o gamă extinsă de instrumente de dezvoltare alături de care formează o sinergie care îi ușurează munca developer-ului. Două dintre aceste instrumente pe care le-am integrat în proiectul meu sunt: mediul de dezvoltare Visual Studio și platforma pentru cloud Microsoft Azure.

Un alt aspect ce merită menționat cu privire la .NET Core este faptul că acesta pune un accent deosebit pe securitate, stabilitate și performanță. Limbajul C# se aseamănă foarte mult cu Java însă, spre deosebire de Java, acesta folosește un compilator just-in-time ce compilează codul în timpul execuției spre deosebire de înainte de aceasta [6]. Ca și măsuri de securitate, .NET Core furnizează diverse biblioteci pentru gestionarea autentificării, autorizării, gestionarea erorilor și auditarea sub formă de log-uri.

Alături de .NET Core și Entity Framework, pentru stocarea datelor am folosit serviciul de baze de date Microsoft SQL Server. Acesta a fost rulat local, apoi pe un container de Docker și accesat folosind SSMS (SQL Server Management Studio). Abordarea creării bazei de date și a tabelor a fost de tip code-first folosind migrări.

Deoarece planul de dezvoltare a aplicației avea să conțină un microserviciu ce se ocupă cu rularea de comenzi pe un sistem ce rulează Linux, am decis să aleg un limbaj de programare ce se integrează foarte bine într-un astfel de mediu, și anume Ruby. Ruby, asemenea Python, este un limbaj de nivel înalt, interpretat ce rulează pe diferite platforme și se folosește în principal pentru server-side scripting [7].

Una dintre particularitățile care m-au convins că Ruby on Rails este o alegere potrivită pentru această aplicație este datorită simplității și rapidității în dezvoltare, lucru care subliniază nevoia de unelte care trec peste pașii repetitivi și inutili în dezvoltarea unei aplicații. Ruby on Rails adoptă principiul “convention over configuration” ceea ce înseamnă că prezintă o structură și convenții bine stabilite, eliminând astfel necesitatea repetiției și configurării manuale a aspectelor comune ale unei aplicații [8]. Acest framework înglobează foarte bine ideea de reducere a “overhead-ului” dezvoltării de software întrucât, în final, developer-ul se poate concentra asupra implementării logicii de business decât asupra structurii codului.

Deoarece conceptul pe care îl introduce Rails este atât de important și relevant pentru această lucrare, voi prezenta în continuare modalitatea prin care se dezvoltă o aplicație de

Capitolul 2

Ruby on Rails de tip MVC conținând migrări către baza de date, frontend stilizat cu tailwind css, un controller, un model și trei pagini separate.

Presupunem că sistemul are instalat Ruby și gem-urile Ruby on Rails și bundler [9]. Vrem să dezvoltăm o aplicație ce administrează cărțile unei biblioteci. Pentru asta vom rula:

```
rails new BookManagementApp --db=postgres --css=tailwind
rails g scaffold Book ISBN:string pages:integer
rails db:create db:migrate
./bin/dev
```

Figură 1 Comenzi pentru generarea unei aplicații Ruby on Rails

Și rezultatul final este următorul:



Books [New book](#)

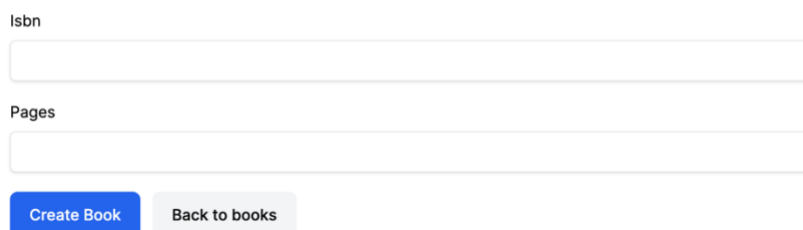
Isbn:
978-3-16-148410-0

Pages:
346

[Show this book](#) [Edit this book](#)

Figură 2 <http://localhost:3000/books>

New book



Isbn

Pages

[Create Book](#) [Back to books](#)

Figură 3 <http://localhost:3000/books/new>

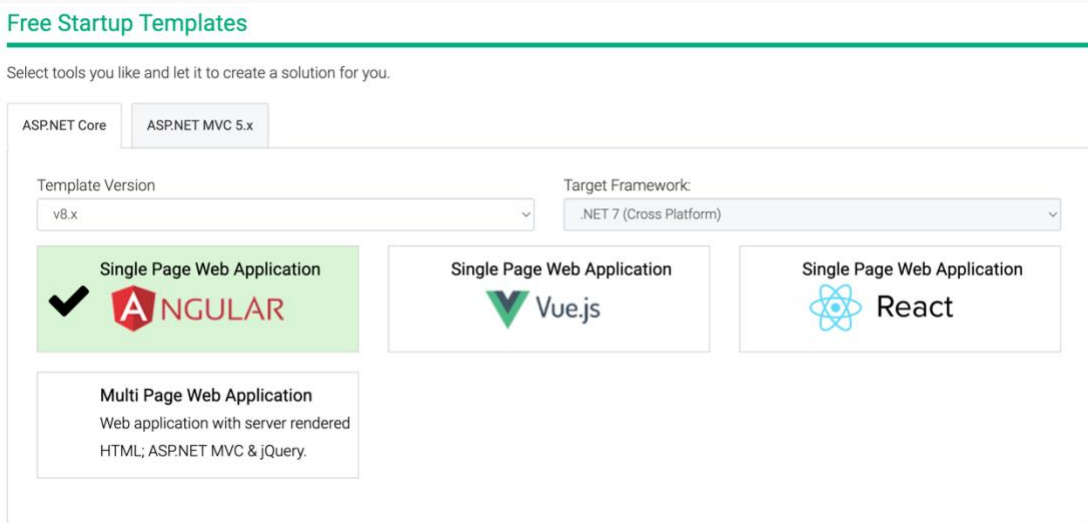
Capitolul 2

Rulând aceste comenzi simple au fost implementate toate operațiile CRUD (Create, Read, Update, Delete/Destroy) cu trei view-uri în care poți accesa toate aceste informații. Sigur că acesta este un exemplu simplu și lipsit de însemnătate privind utilitatea aplicației, însă, la fel ca și HotSpotWeb, acesta reprezintă dovada că dezvoltarea rapidă și bună este posibilă.

HotSpotWeb folosește Postgresql pentru baza de date întrucât aceasta este o combinație des întâlnită în lumea developer-ilor de Ruby on Rails, deoarece este open source și este a patra cea mai folosită baze de date din lume [10].

Deși nu este un limbaj de programare sau un framework propriu zis dar a jucat un rol mare în dezvoltarea aplicației HotSpotWeb, voi vorbi în continuare despre Aspnetboilerplate [1]. Această unealtă a reprezentat în primul rând o inspirație pentru mine și apoi un punct de plecare pentru aplicația dezvoltată, evidențiând încă odată faptul că reducerea timpului de dezvoltare și eventualele probleme apărute în fazele incipiente ale dezvoltării sunt importante.

ASP.NET Boilerplate generează aplicații web cu o structură bine definită, sigure și destul de complexe dar cu destul de puține dependențe cât să nu dăuneze performanței. Spre deosebire de HotSpotWeb, această platformă este specializată pe aplicații ASP.NET Core și un framework de frontend (Angular, Vue, React sau MVC). Prin completarea formularului, de mai jos se va genera o aplicație și va fi descărcată local, fără posibilitatea de încărcare pe platforme de versionare și descărcare ulterioară.



The image shows a web interface titled "Free Startup Templates". Below the title is a subtitle: "Select tools you like and let it to create a solution for you." There are two tabs: "ASP.NET Core" (selected) and "ASP.NET MVC 5.x". Below the tabs are two dropdown menus: "Template Version" set to "v8.x" and "Target Framework" set to ".NET 7 (Cross Platform)". Below these are three main options for "Single Page Web Application": "ANGULAR" (highlighted with a green background and a checkmark), "Vue.js", and "React". Below these is a fourth option for "Multi Page Web Application" with the description "Web application with server rendered HTML; ASP.NET MVC & jQuery."

Figură 4 Formular generare aplicație ASP.NET Boilerplate

Capitolul 2

Aplicația generată este furnizată sub formă de arhivă ce se descarcă în momentul generării. Pe lângă aceste funcționalități de generare, aplicațiile generate oferă o structură bine definită ce conține implementări ale mai multor design pattern-uri precum Dependency Injection și Singleton și dispune de alte beneficii ca și autorizare, validare, audit, auto-mapping [11]. Structura este bine explicată în documentația de pe site-ul oficial, care conține și tutoriale pentru o mai bună înțelegere a framework-ului.

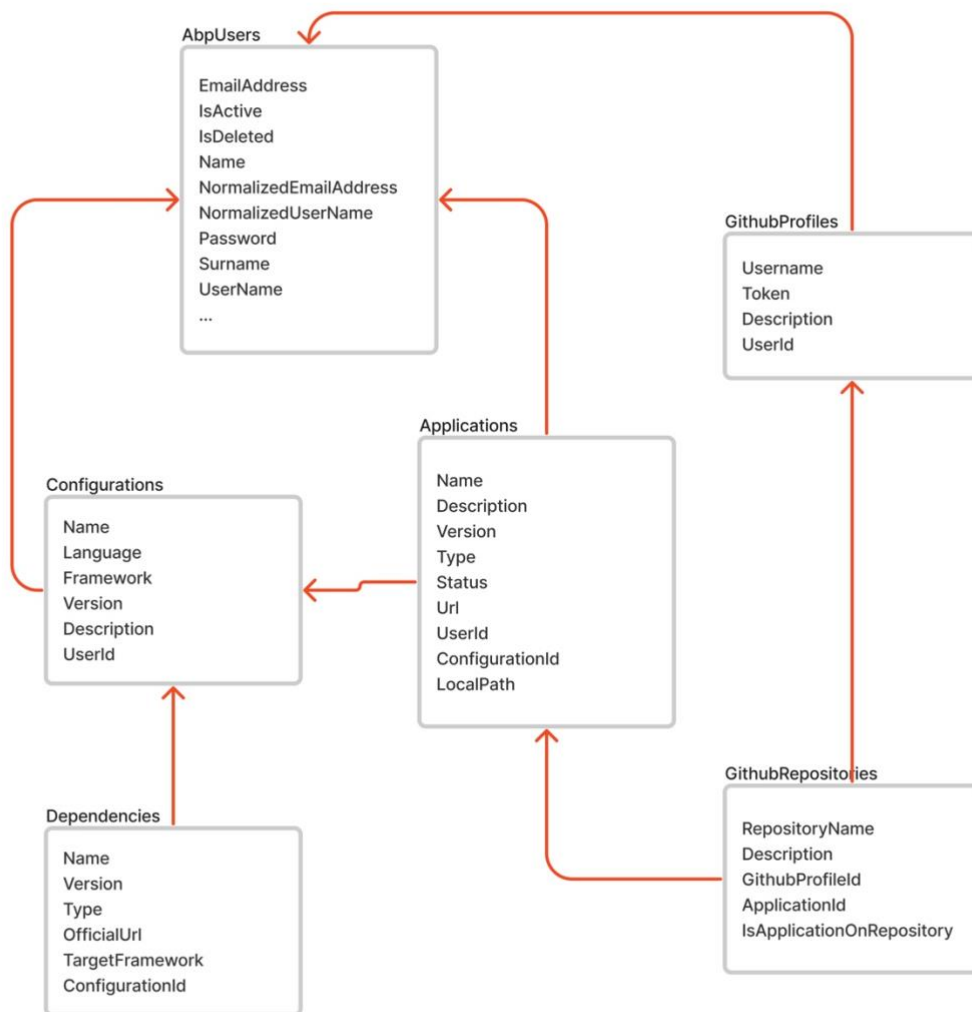
2.2 Structura bazelor de date

Framework-ul ASP.NET Boilerplate prezentat anterior vine, după cum am menționat, cu o multitudine de alte funcționalități și servicii care fac parte din baza de date a aplicației. Din acest considerent am ales să includ doar tabelele relevante în dezvoltarea logicii din spatele funcționalităților de bază a acestei aplicații.

În principiu, baza de date are 6 entități relevante: Applications, Configurations, Dependencies, GithubProfiles, GithubRepositories și Users. Relațiile dintre tabele, descrise în diagrama de mai jos, sunt următoarele: o dependență este de sine stătătoare și poate aparține oricărei configurații; o aplicație poate avea o singură configurație și un utilizator; un utilizator poate avea mai multe profile (GithubProfile) și mai multe configurații; un repository (GithubRepository) aparține unei aplicații și are un profil.

Pe lângă câmpurile prezente în imagine, sunt prezente, pentru fiecare tabel în parte câmpurile CreationTime, LastModificationTime și Id.

Capitolul 2

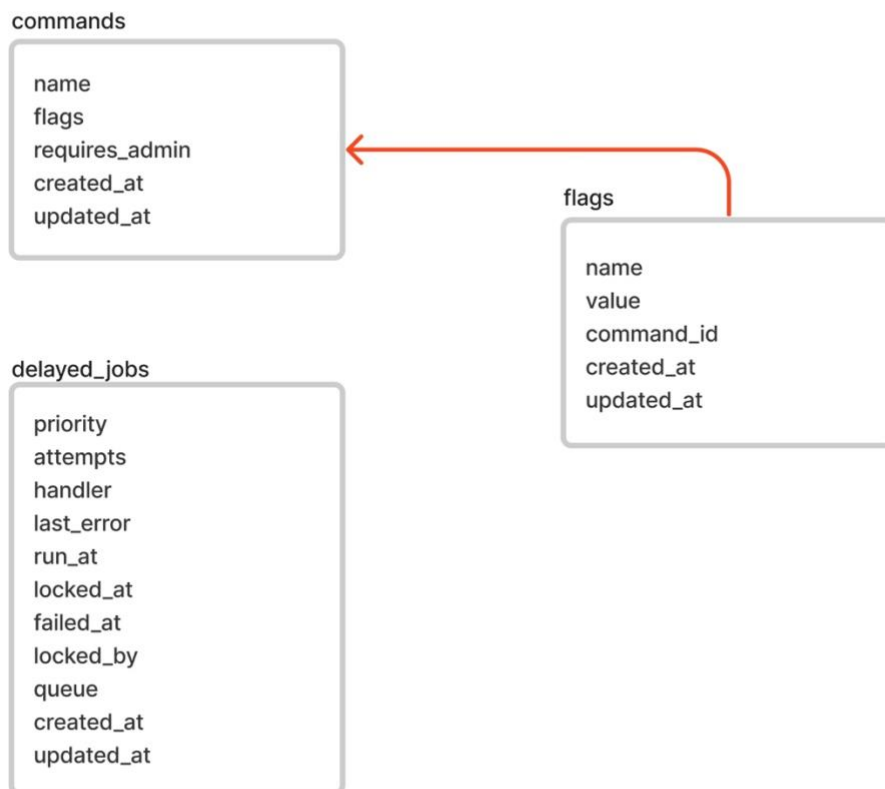


Figură 5 Schemă bază de date HotSpotWeb

Am menționat că și microserviciul HotSpotWebCommandsService are o bază de date de tip Postgres iar aceasta este folosită exclusiv pentru audit. Fiecare comandă ce vine de la server va conține comanda în sine alături de câmpurile `requires_admin` și `flags`, câmp care conține restul comenzii. În această bază de date se mai află tabelul `flags` ce conține flag-urile din comandă

Capitolul 2

după parsarea acestora (ele sunt inițial într-un string și apoi sunt despărțite) și tabelul `delayed_jobs` asociat cozii de execuție sidekiq.



Figură 6 Schemă bază de date `HotSpotWebCommandsService`

Capitolul 2

2.3 Concepte și metodologii folosite

În cadrul dezvoltării aplicației HotSpotWeb, o serie de concepte și metodologii sunt aplicate pentru a asigura o dezvoltare eficientă și o arhitectură scalabilă. Acest capitol conține câteva dintre aceste elemente precum microservicii, dependency injection și Sidekiq (drept coadă de execuție).

Arhitectura bazată pe microservicii a fost adoptată în cadrul aplicației HotSpotWeb pentru a permite dezvoltarea și gestionarea independentă a componentelor. Această abordare implică împărțirea aplicației în servicii mici, autonome și specializate, care pot fi dezvoltate, testate și implementate independent, unul față de celălalt. Folosirea microserviciilor facilitează scalabilitatea, flexibilitatea și reutilizarea codului permițând aplicației să fie dezvoltată ulterior pe mai multe planuri în paralel.

Dependency Injection (DI) este un design pattern care facilitează gestionarea dependențelor între componente. În cadrul aplicației HotSpotWeb, DI este utilizată pentru a inversa controlul asupra dependențelor, acestea din urmă fiind injectate într-o componentă din exterior, permițând înlocuirea ușoară a implementărilor [12]. În dezvoltarea aplicației acestea au fost folosite în principal pentru injectarea managerilor unui model în serviciul respectiv. Spre exemplu, GithubProfileManager are implementarea pentru crearea unui GithubProfile prin metoda Create, însă apelul pe care îl realizează frontend-ul se oprește în GithubProfileAppService. Pentru aceasta a fost injectat managerul în serviciu și chemată metoda CreateAsync după cum urmează:

```
private readonly IGithubProfileManager _githubProfileManager;

public GithubProfileAppService(IGithubProfileManager githubProfileManager)
{
    _githubProfileManager = githubProfileManager;
}

public async Task CreateAsync(CreateGithubProfileDto input)
{
    var githubProfile = GithubProfile.Create(input.Username, input.Token,
        input.Description, AbpSession.GetUserId());
    await _githubProfileManager.CreateAsync(githubProfile);
}
```

Figură 7 Exemplu de Dependency Injection

Capitolul 2

Sidekiq este un sistem de gestionare a sarcinilor în fundal, utilizat în cadrul aplicației HotSpotWebCommandsService. Sidekiq facilitează procesarea asincronă a sarcinilor, ceea ce permite aplicației să răspundă rapid la cereri, acesta programând sarcinile în fundal folosind o coadă de execuție. Acest program folosește un worker ce rulează permanent și este chemat în momentul în care se dorește a se executa o comandă nouă. Comanda intră în coadă, este procesată și abia în momentul în care se termină comanda respectivă o poate executa pe următoarea, astfel evitând erori precum descărcarea unei aplicații ce nu a fost încă generată.

2.4 Distribuirea aplicațiilor în Azure Service Fabric

În acest capitol, voi explora procesul de distribuire a aplicațiilor în Azure Service Fabric utilizând containere Docker. Voi discuta despre beneficiile utilizării containerelor și cum Azure Service Fabric facilitează gestionarea și scalarea aplicațiilor distribuite într-un mediu cloud.

Pentru a distribui serviciile HotSpotWeb, am ales să le containerizez cu ajutorul tehnologiei Docker. Docker oferă un mediu de izolare portabil și ușor de gestionat pentru aplicații, permițând rularea acestora într-un mod consistent și reproductibil pe diferite platforme și infrastructuri. Containerele Docker, datorită portabilității sale, poate fi rulat în același mod în diverse medii, inclusiv pe mașini locale de dezvoltare, în centre de date și în mediul cloud. Rulând în medii izolate, containerele nu interferează cu alte aplicații care rulează atât pe mașină cât și pe cluster. Un alt avantaj al folosirii Docker este acela că un container conține toate dependențele necesare pentru rularea aplicației, ceea ce asigură că aceasta poate fi reprodusă cu ușurință în același mod în orice mediu.

Pentru a gestiona imaginile Docker, am ales să utilizez Azure Container Registry. Azure Container Registry este un serviciu de gestionare a registrelor de containere Docker în Azure, care permite stocarea, administrarea și distribuirea imaginilor Docker. Principalul motiv pentru care am ales să folosesc acest serviciu a fost faptul că se integrează perfect cu alte servicii Azure, precum Azure Service Fabric, permițând o distribuire și gestionare simplă a aplicațiilor containerizate.

Azure Service Fabric este un serviciu de orchestrare a aplicațiilor care facilitează gestionarea și

Capitolul 2

scalarea aplicațiilor distribuite în cloud. Prin utilizarea Azure Service Fabric, putem distribui și monitoriza aplicațiile noastre containerizate într-un mod eficient și automatizat. Azure Service Fabric este doar una din soluțiile ce se ocupă cu distribuirea containerelor sub formă de cluster, alte opțiuni fiind Docker Swarm și Kubernetes. Totodată, Azure Service Fabric se remarcă față de celelalte prin faptul că permite scalarea automată a aplicațiilor în funcție de cerințele de încărcare, asigurând performanță și disponibilitate ridicate. De asemenea, Azure Service Fabric gestionează automat detecția și recuperarea de erori în aplicații, asigurând funcționarea continuă în cazul eșecurilor hardware sau software.

Capitolul 3

Arhitectura aplicației HotSpotWeb

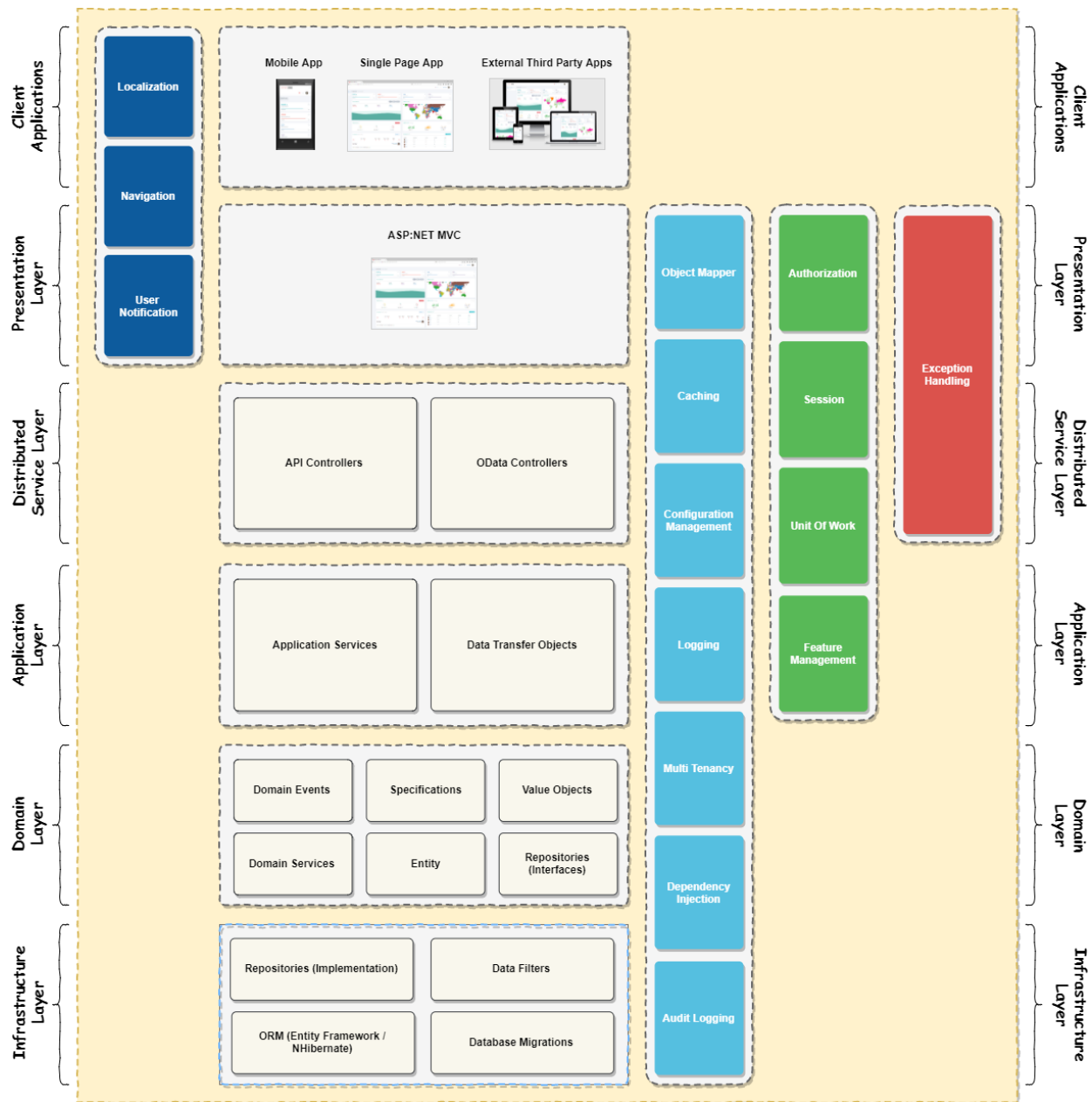
3.1 ASP.NET – Backend

În capitolul anterior am prezentat motivele pentru care am considerat că ASP.NET Core este framework-ul potrivit pentru backend-ul aplicației HotSpotWeb. În acest capitol voi prezenta structura internă a întregii soluții HotSpotWeb.

Soluția în cauză este alcătuită din 6 proiecte fiecare reprezentând un nivel din arhitectura Nlayer. Această arhitectură împarte aplicația într-un număr n de layere distincte. Spre exemplu, layerul de prezentare poate conține un proiect de frontend deoarece aici se află interacțiunea cu utilizatorul, în timp ce un layer de business poate reprezenta un REST API iar un layer de date (layer de infrastructură) poate fi o bază de date sau Entity Framework [13]. Această arhitectură, deși complexă la prima vedere, organizează codul într-o manieră precisă ce sprijină scalabilitatea codului. Modularitatea și reutilizarea sunt de asemenea două caracteristici ce reies din faptul că fiecare strat este independent și poate fi dezvoltat, testat și întreținut separat.

În contextul HotSpotWeb, sunt folosite layerele din paradigma DDD (Domain Driven Design), și anume Presentation Layer, Application Layer, Domain Layer și Infrastructure Layer. Aceste layere sunt prezentate succint în schema de mai jos cu exemple concrete. Aceste layere sunt implementate în template-urile generate de Aspnet Boilerplate.

Capitolul 3



Figură 8 Schema arhitecturii DDD pentru aplicațiile Aspnet Boilerplate [14]

Aplicațiile Client (Client Applications) pot fi considerate toate aplicațiile în cadrul cărora ajunge informația prin HTTP, spre exemplu aplicația Angular din HotSpotWeb. În cadrul HotSpotWeb, layerele corespund proiectelor în felul următor:

- HotSpotWeb.Application reprezintă layerul de aplicație (Application layer) și include servicii specifice aplicației care utilizează logica de domeniu și obiectele din cadrul domeniului

Capitolul 3

- HotSpotWeb.Core reprezintă layerul de domeniu (Domain layer) și conține logica principală a aplicației. Aici se găsesc entitățile și managerii care implementează logica de business
- HotSpotWeb.Migrator este responsabil cu migrarea și gestionarea schemei de bază de date pentru aplicație. Acest proiect conține de fapt un executabil care rulează toate migrările noi asupra bazei de date
- HotSpotWeb.EntityFrameworkCore implementează interfețele de repository definite în HotSpotWeb.Core utilizând framework-ul Entity Framework Core pentru a comunica cu baza de date
- HotSpotWeb.Web.Core reprezintă stratul de prezentare și include funcționalități legate de prezentarea datelor utilizatorului, cum ar fi rutarea, maparea obiectelor și gestionarea localizării
- HotSpotWeb.Web.Host asigură hosting-ul aplicației și gestionează aspecte precum configurarea, interpretarea cererilor HTTP și integrarea cu alte componente și servicii externe

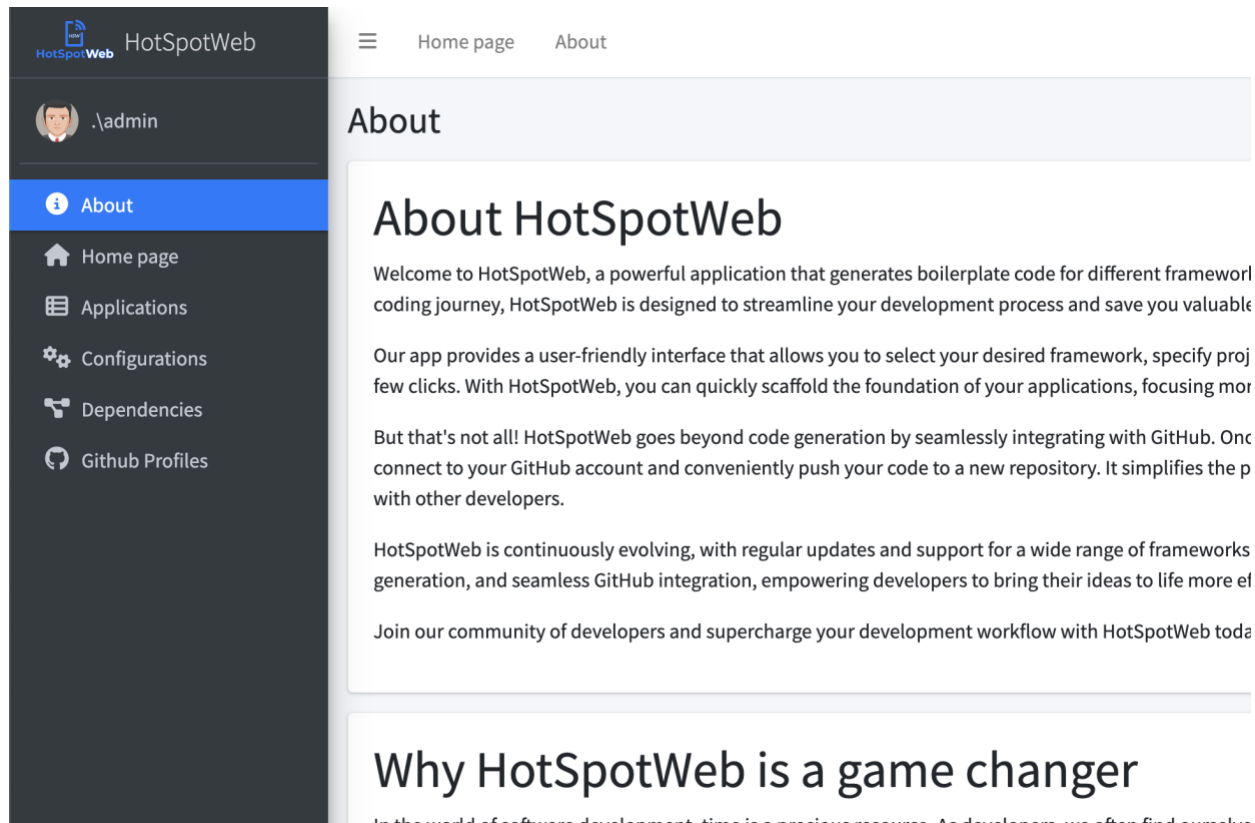
3.2 Angular – Frontend

Interfața cu utilizatorul este scrisă în Angular și stilizată folosind AdminLTE și Bootstrap. Am ales Angular datorită arhitecturii structurate pe componente, rapiditatea în dezvoltare și suportului. Aplicația pornește cu pagina de start ce conține elemente de statistică precum numărul de aplicații create, numărul de aplicații active, numărul de configurații și cea mai folosită configurație. De asemenea, aici se află shortcut-uri pentru restul paginilor de interacționat cu aceste elemente.

În momentul accesării platformei, fiecare utilizator este obligat să își introducă credențialele pentru a se loga și a accesa resursele aferente contului său. În acest moment, configurațiile sunt private și nu pot fi accesate de alți utilizatori însă, pe viitor, se află în plan o funcționalitate de „hub” pentru configurări.

Pe lângă aceste pagini și funcționalități de bază, aplicația conține paginile pentru generarea de aplicații. Există componente cu operațiunile CRUD pentru Aplicații, Configurații, Dependințe și Profile de Github. Desigur, pentru a crea o aplicație fără bătăi de cap se recomandă folosirea tutorialului din pagina de About. Acesta explică pas cu pas cum se poate genera o aplicație folosind o configurare de bază și cum se poate adăuga aceasta pe Github.

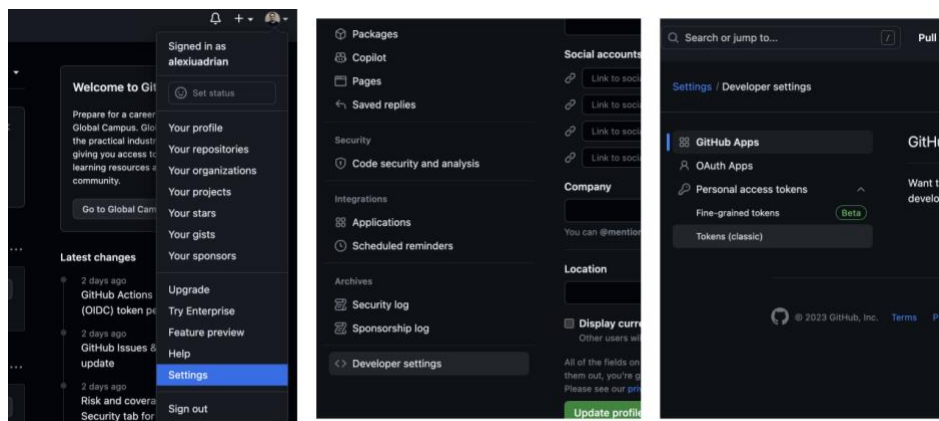
Capitolul 3



Figură 9 Interfața HotSpotWeb

Flow-ul pentru crearea cu succes a unei aplicații și încărcarea acesteia pe Github este următorul:

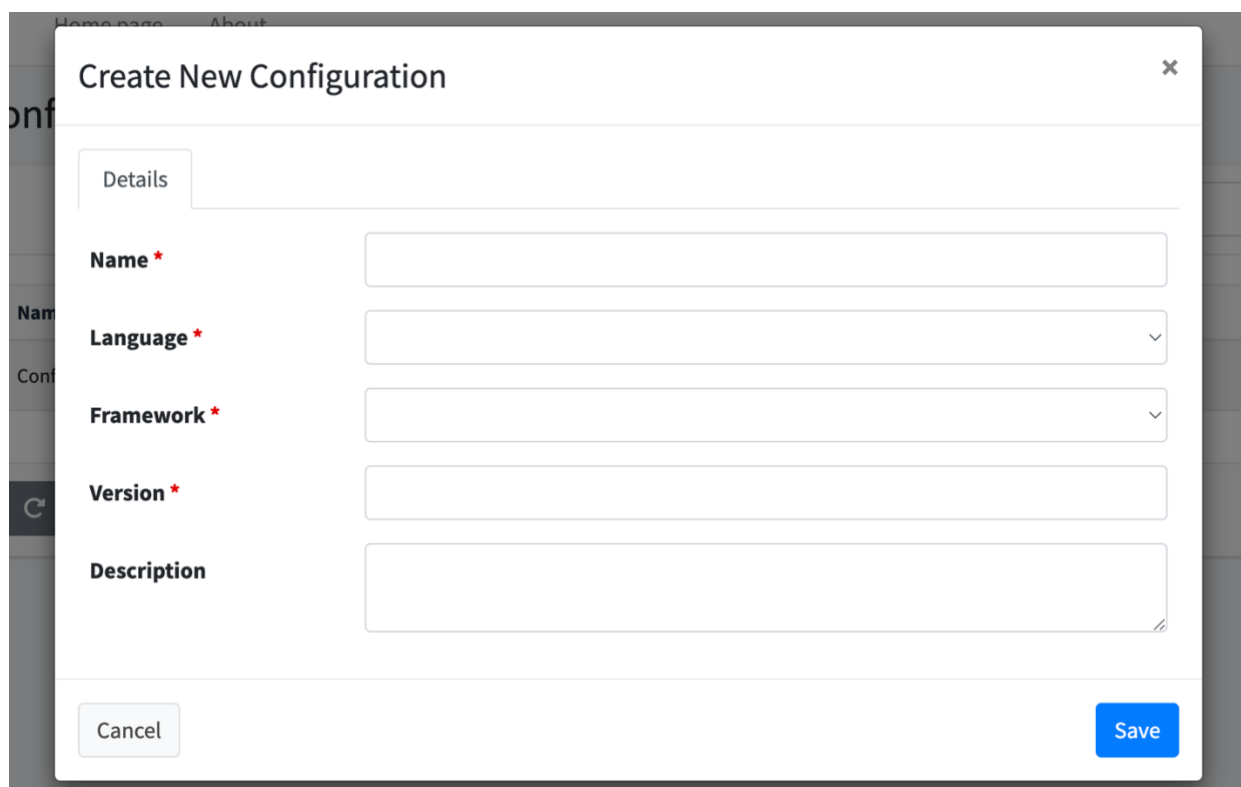
1. Se adaugă un profil nou de Github folosind un username și un token. Token-ul poate fi generat conform schemei de mai jos. (doar în cazul în care se vrea ca aplicația să fie urcată pe Github)



Figură 10 Obținerea unui Personal Access Token

Capitolul 3

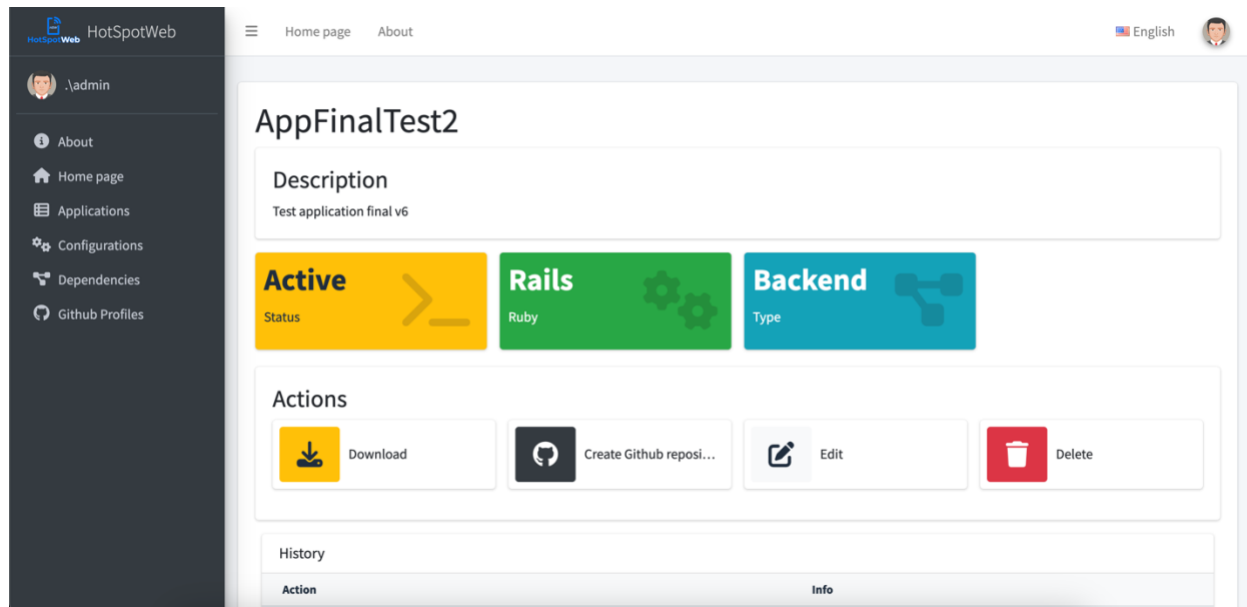
2. Se adaugă o configurație folosind modalul de creat configurații din pagina de Configurations. În acest pas se definesc numele, limbajul, framework-ul și versiunea configurației.

The image shows a 'Create New Configuration' modal window. It has a title bar with a close button (X). Below the title bar is a tab labeled 'Details'. The form contains five fields: 'Name *' (text input), 'Language *' (dropdown menu), 'Framework *' (dropdown menu), 'Version *' (text input), and 'Description' (text area). At the bottom of the modal are two buttons: 'Cancel' and 'Save'.

Figură 11 Adăugarea unei configurații

3. În final, se adaugă o aplicație completând formularul din pagina de Applications și apoi se accesează aplicația creată.

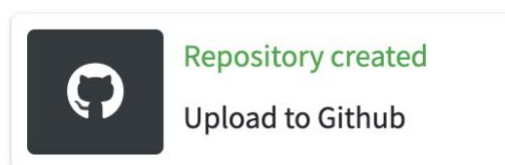
Capitolul 3



Figură 12 Pagina de vizualizare a aplicației

În cadrul paginii de vizualizare a aplicației se află majoritatea acțiunilor care pot fi făcute asupra acesteia. În primul rând, aici sunt implementate operațiile Edit și Delete ce au efect direct asupra acțiunii salvate. Editarea aplicației permite modificarea descrierii, versiunii și statusului, result câmpurilor fiind vitale pentru restul operațiunilor.

În momentul apăsării butonului Create Github Repository se va deschide o nouă fereastră modală în care se adaugă numele repository-ului ce va fi creat, o descriere și se va selecta profilul create anterior. Nu se poate genera un repository fără profil. În urma acestei operațiuni, butonul își va modifica aspectul semnalând că există un repository.



Figură 13 Butonul ce semnifică faptul că Repository-ul a fost creat

Făcând un nou click pe buton se va deschide o fereastră modală cu un buton ce are mesajul “Upload to Github” care, odată apăsă, va verifica dacă aplicația a fost generată ulterior și o va trimite către Github. Modalitatea prin care se fac aceste operațiuni va fi descrisă în capitolul următor.

Asemenea funcționalității de încărcare pe Github, descărcarea proiectului se face accesând

Capitolul 3

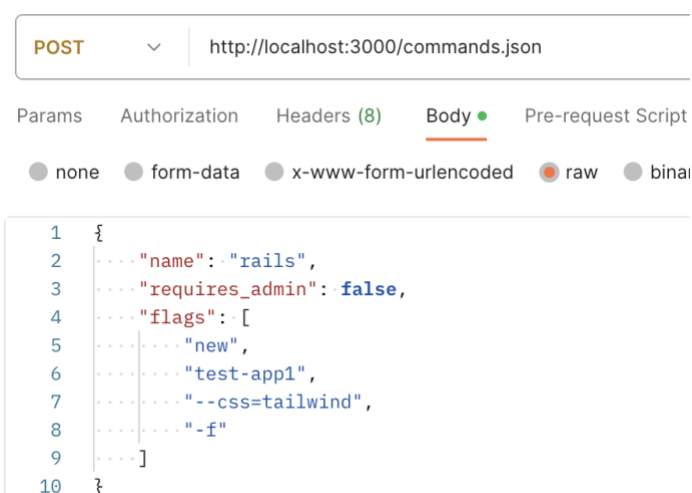
butonul “Download” care verifică dacă aplicația are deja un link de descărcare. În caz contrar, aplicația se va genera, se va arhiva și se va trimite către Azure Blob Storage pentru păstrare, urmând să se returneze un url pentru descărcare.

3.3 Ruby on Rails – Serviciul de rulat comenzi

Deoarece am avut în vedere de la început că toate comenzile vor fi rulate pe un sistem de tip Linux, am căutat o soluție familiară ce poate rula comenzi simplu și fără complicații. Structura aplicației este una foarte simplă, concepută pentru a evita eventuale probleme de configurare și pentru a generaliza pe cât se poate procesul de generare al aplicațiilor.

Prima idee de abordare a problemei în cauză a fost să definesc în această aplicație o multitudine de comenzi pre-existente care să fie rulate în momentul în care HotSpotWeb trimite un request pentru generarea aplicației respective. Apoi am realizat că funcționalitatea de rulare de comenzi pe un sistem poate fi de folos și în alte aspecte ale aplicației, precum arhivarea fișierelor și încărcarea acestora pe Github sau pe Azure Blob Storage. Din acest considerent am ales să nu fac acest serviciu unul de generat aplicații ci mai degrabă unul de rulat comenzi pe un sistem.

Baza de date asociată acestui proiect este folosită în mare parte pe post de audit, la fel ca și interfața grafică cu paginile asociate. Acestea păstrează și afișează toate comenzile rulate folosind API-ul. Request-ul pentru a rula o comandă necesită un nume ce reprezintă comanda în sine, urmat de un câmp “requires_admin” ce face diferența dintre comenzi rulate cu sau fără sudo (câmpul acesta nu este folosit în cadrul aplicației momentan), și, în final, flag-urile necesare pentru rularea aplicației. În imaginea de mai jos se află un exemplu de comandă ce generează o aplicație de Ruby on Rails folosind serviciul.



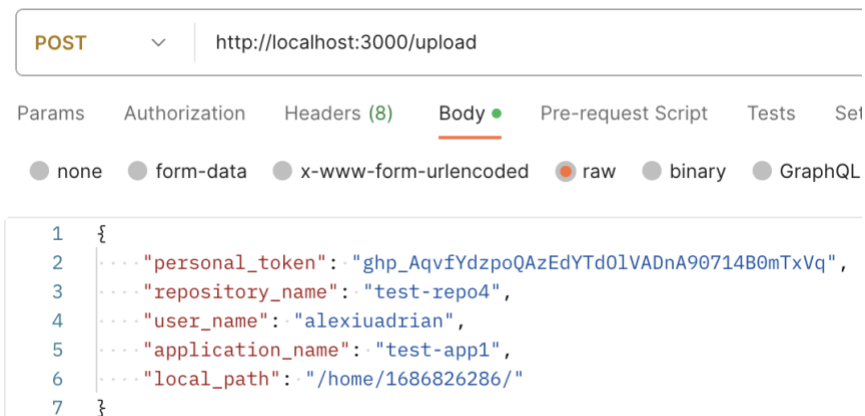
Figură 14 Exemplu de comandă pentru generarea unei aplicații

Capitolul 3

După ce serviciul primește request-ul cu comanda, acesta o salvează în baza de date și cheamă funcția “perform_async” din cadrul worker-ului numit run_command pentru serviciul sidekiq. Sidekiq are nevoie de un worker definit pentru a ști ce să ruleze asincron, worker-ul fiind un amalgam de funcții care să umple coada de execuție. În cadrul funcției numite “perform”, ce primește ca și parametri o listă de comenzi, fiecare comandă fiind un string ce conține și flag-urile, un final_path ce reprezintă path-ul de pe mașină unde se va executa comanda, un path ce reprezintă în aceeași măsură un path de pe mașină însă acesta este folosit pentru a stoca fișierul de tip “sh” și un timestamp pentru identificarea unică a directoarelor și script-urilor generate de pe mașină.

Folosind acești parametri, funcția creează un script de tip “.sh” în care scrie comanda pentru a schimba directorul în directorul curent folosindu-se de local_path și scrie fiecare comandă din listă pe câte o linie nouă. Apoi, fișierului de tip script îi este dată permisiunea de rulare (0755) și apoi este chemată funcția „system” pentru rularea scriptului respectiv.

În cadrul aceleiași aplicații, însă fără a fi folosit endpoint-ul de „commands”, sunt făcute și încărcările către Github și Azure Blob Storage. Totuși, logica internă a endpoint-urilor pentru aceste două operațiuni se folosește de același worker de rulat comenzi, cu excepția celui care folosește în mod explicit API-ul Github pentru crearea repository-urilor goale. Un exemplu de request care încarcă o aplicație generată anterior pe Github arată astfel:



Figură 15 Exemplu de request pentru încărcarea unei aplicații într-un repository de pe Github

Unde „local_path” este obținut în urma rulării de generare a aplicației și reprezintă locația pe disk-ul mașinii unde este rulată comanda. De asemenea, serviciul mai conține și alte funcționalități precum cererea de date despre un anumit repository, crearea de repository-uri pentru organizații și ștergerea de repository-uri însă acestea nu au fost încă implementate în restul aplicației HotSpotWeb.

Capitolul 4

Discuții și Concluzii

4.1 Limitări și probleme întâmpinate

Pe parcursul dezvoltării aplicației HotSpotWeb am întâmpinat o multitudine de probleme, cele mai mult fiind de natură tehnică (bug-uri, comportament neașteptat, resurse insuficiente, etc.), însă au existat, mai ales la începutul dezvoltării, și probleme legate de structura finală a aplicației.

O mare parte a dezvoltării a fost stabilirea unei structuri nu foarte complicate dar care să aibă sens și de partea utilizatorului. Configurările erau combinate cu dependențele în formarea unui json care era trimis mai departe către generarea aplicațiilor însă am realizat că astfel pot apărea situații de command injection [15] ce pot compromite sistemul.

În același timp, simpla generare a unor aplicații de un singur tip ar fi reprezentat o limitare a funcționalității proiectului și, în final, neatingerea obiectivelor pe care le propunea aceasta. Am decis într-un final că utilizatorii vor alege din anumite configurații și dependențe predefinite.

În timpul dezvoltării limitările au fost reprezentate exclusiv de limitările rezultate din logică și de mediul de dezvoltare ales, însă adevăratele limitări au apărut în momentul configurării serviciului Azure Service Fabric. Începând cu obligația de a folosi sistemul de operare Windows pentru configurarea unui cluster local și până la probleme cu aplicația Azure Portal în care, datorită configurărilor eronate și lipsei de resurse educaționale cu privire la Azure Service Fabric, nu am reușit inițial să duc la bun sfârșit o configurare a acestui serviciu, am reușit să întâmpin o multitudine de probleme care să interfereze cu procesul de dezvoltare.

4.2 Direcții viitoare de dezvoltare pentru HotSpotWeb

Deși consider că am adus conceptul de management al software-ului dintr-o singură platformă într-o formă simplă și destul de configurabilă pentru o multitudine de nevoi, aplicația HotSpotWeb are o serie de idei ce, dacă vor fi implementate, vor îmbunătăți semnificativ utilitatea platformei în piață.

Una dintre aceste idei care a fost deja implementată parțial este posibilitatea de adăugare a profilurilor de organizații Github și, astfel, repository-uri de organizații. Această funcționalitate ar furniza companiilor și organizațiilor un mod simplu de creare a proiectelor noi.

Deși, momentan, configurările aplicațiilor nu reprezintă un aspect foarte personalizabil, putând alege dintr-o gamă predefinită de configurări și dependențe, pe viitor, utilizatorii își pot defini

Capitolul 4

propriile configurări după nevoile proprii și trimite spre aprobare pentru a fi adăugate în platformă și, astfel, a putea fi folosite. Odată cu această îmbunătățire, poate apărea un loc în care utilizatorii își împărtășesc între ei propriile configurări care pot fi copiate și folosite de alți utilizatori.

O altă idee este adăugarea posibilității de configurare a aplicațiilor pe servere. Acest lucru înseamnă că, prin procese automate sub formă de configurări, o aplicație generată este trimisă către un server și apoi modificate elemente precum nginx sau apache pentru a face aplicația publică. Desigur, este o idee ambițioasă, însă structura de bază pe care a construit-o HotSpotWeb permite o astfel de îmbunătățire.

4.3 Concluzie

Într-o lume în care viteza este un factor atât de important pe cât este în a noastră, o aplicație precum HotSpotWeb poate reprezenta un avantaj competitiv, o soluție sigură fără complicații sau doar un mod comod de a începe procesul de dezvoltare al unui produs software. Ideea a plecat dintr-o nevoie specifică într-un moment critic care poate ar fi făcut o diferență în lansarea produsului cu una sau două zile mai devreme decât a fost lansat în acest fel.

Lucrarea s-a axat atât pe descrierea în detaliu a soluției implementate cât și evidențierea importanței conceptului de generare a unui schelet de aplicație bine implementat. Din acest motiv aplicația se bazează pe simplitatea codului în schimbul punerii în valoare a logicii și a proceselor din spate.

Din aceste motive, consider că HotSpotWeb poate fi un punct de plecare (atât ca inspirație cât și ca structură) pentru alte soluții ce vor aduce în piață aplicații mai sigure, mai eficiente, mai bine organizate și într-un timp mai scurt decât până acum.

Bibliografie

- [1] *ASP.NET Boilerplate*, accesat la 24.04.2023, URL: <https://aspnetboilerplate.com/>
- [2] *Abp Framework*, accesat la 24.04.2023, URL: <https://abp.io/>
- [3] Swati Saini, Sakshi Arora, *8 Proven Reasons You Need Angular for Your Next Development Project*, accesat la 30.04.2023, URL: <https://www.grazitti.com/blog/8-proven-reasons-you-need-angular-for-your-next-development-project/>
- [4] Maitray Gadhavi, *Top 8 Reasons ASP.NET Core is the Best Framework for Web Application Development*, accesat la 18.05.2023, URL: <https://radixweb.com/blog/8-reasons-asp-dot-net-core-is-best-framework>
- [5] Alin Ciocan, *How many .NET developers are there?*, accesat la 22.05.2023, URL: <https://www.quora.com/How-many-NET-developers-are-there>
- [6] *Just-in-time compilation*, accesat la 22.05.2023, URL: https://en.wikipedia.org/wiki/Just-in-time_compilation
- [7] Shivi_Aggarwal, *Python vs Ruby*, accesat la 31.05.2023, URL: <https://www.geeksforgeeks.org/python-vs-ruby/>
- [8] *Convention over configuration*, accesat la 31.05.2023, URL: https://en.wikipedia.org/wiki/Convention_over_configuration
- [9] *Bundler.io*, accesat la 05.05.2023, URL: <https://bundler.io/>
- [10] Mahesh Chand, *Most Popular Databases In The World (2023)*, accesat la 07.05.2023, URL: <https://www.c-sharpcorner.com/article/what-is-the-most-popular-database-in-the-world/>
- [11] *What is the ASP.NET Boilerplate*, accesat la 08.05.2023, URL: <https://aspnetboilerplate.com/Pages/Documents/Introduction>
- [12] *Dependency injection*, accesat la 08.05.2023, URL: https://en.wikipedia.org/wiki/Dependency_injection
- [13] Mehmet Ozkaya, *Layered (N-Layer) Architecture*, accesat la 15.05.2023, URL: <https://medium.com/design-microservices-architecture-with-patterns/layered-n-layer-architecture-e15ffdb7fa42>
- [14] *ASP.NET Boilerplate Application Architecture Model*, accesat la 16.05.2023, URL: <https://aspnetboilerplate.com/Pages/Documents/NLayer-Architecture>
- [15] *Command Injection*, accesat la 26.05.2023, URL: <https://www.imperva.com/learn/application-security/command-injection/>