Microservicii

Student: Lăpuște Raul

1.Concept

Microserviciile reprezintă o arhitectură de dezvoltare software în care o aplicație este structurată ca un ansamblu de servicii mici, independente, care comunică între ele prin intermediul unor interfețe bine definite, de obicei prin cereri HTTP sau mesagerie asincronă. Fiecare microserviciu este centrat pe o funcționalitate specifică de executat. Fiecare fiind dezvoltat, implementat și scalat independent, poate fi scris în tehnologii sau limbaje diferite.

Această mod de abordare abordare ajută la livrarea rapidă și continuă de proprietaților, și permite echipelor autonome să lucreze eficient la componente distincte ale aplicației. Arhitectura pe microservicii este frecvent utilizată în aplicații cloud-native și în medii DevOps, unde flexibilitatea și viteza de livrare sunt esențiale.

2. Prezentarea proiectului

Bazându-mă pe conceptul arhitecturii microservicilor și experiențe anterioare, am reușit dezvolatarea de la o aplicație monolitica spre fragmentarea și ordonarea eficient a codului. Proiectul prevede o bază Back-End, folosit pentru aplicația web dezvoltată personal, ”HRHelper”. Segmentarea masei inițiale de cod robust a dus la restructurarea codului in 6 controllere diferite, logic independente, posibile pentru comunicări predefinite între elemente pentru reutilizarea segvențelor de cod dar și independența servicilor.

Aplicația web a fost dezvoltată la nevoia unei interfețe utilizator simplă de comunicare ce permite logarea, transmiterea mesajelor între colegi, trimiterea de emailuri prin interent pentru comunicarea cu clienți și vizualizarea mesajelor. Aplicația ”HRHelper” va fi folosită ca metodă de retestare, împreună cu Swagger framework utilizat prin Visual Studio, în paralel. Testarea prin utilizarea paginilor Html și a fișierelor JavaScript prevede testarea accesului, a compatibilității și utilizarea la maxim a resurselor oferite de fiecare serviciu. Interfața Swagger permite testarea la nivel crud al fiecărui controller, la nivelul fiecărei funcții, pentru vizualizarea răspunsurilor ce altfel ar fi invizibile, pierdute, odată la legarea în lanț a funcților.

3. Definirea serviciilor

Proiectul presupune dezvoltarea și utilizarea a 6 servicii. Fiecare având la bază un controller ASP.NET :

1. Microservice1-Controller
2. Logging-Controller
3. Message-Controller
4. Person-Controller
5. Outside_Email-Controller
6. Admin-Controller

În cele 6 servicii prezentate mai sus, doar serviciul ”Microservice1” nu are acces la metode ce pot comunica cu baza de date folosită pentru stocarea informațiilor. Fiecare alt serviciu prevede acces la una singură tabelă de informații, urmărind astfel conceptul secționării și independenței, din punct de vedere logic. Pentru baza de date a fost folosit ”SQL Server Manager Studio”:

1. dbo.Logers
2. dbo.Messages
3. dbo.Persons

3.1 Microservice1

Prin scurta explicatie la punctul 3, serviciul ”Microservice1” nu utilizează metode pentru accesarea, preluarea, adăugarea sau modificarea unor elemente în oarecare tabelă a bazei noastre de date. Scopul principal îl are de-a urmări activitatea utilizatorilor înregistrați în baza de date, a memora logarea acestora și permite accesul la restul serviciilor. Acesta utilizează interfata IMemoryCache pentru stocarea informației pentru perioade limitate de timp.

Favorizarea utilizării memoriei Cache este datorată persistetei acesteia dincolo de ciclul de viață al metodelor controllerului. Instanțierea memoriei în serviciu inseamnă doar preluarea unei referințe deja creeată la pornirea programului, dat fapt aceasta fiind unică la nivelul întregii aplicații. Programarea serviciului doar ca acesta să dispuna de access la memoria Cache urmărește prezervarea conceptului de individualitate dintre servicii. Utilizarea memoriei Cache se poate prin introducerea următoarei comenzi în fisierul ”Starup.cs” sau ”Program.cs”

```
builder.Services.AddMemoryCache()
```

Totodată, controllerul API, împreună cu majoritatea celorlaltor controllere, folosește funcții ale unor clase statice pentru execuția și ordonarea unor secvențe de cod. Serviciul ”Microservice1” utilizează clasa statică ”Journal”.

1. ADD_GUY([FromBody] String nm)- metodă Controller
    1) Utilizează funcția Journal.ADD_GUY, ce returneaza un StatusCodeResult: 200 pentru adaugare îm memorie, 403 când criterile pentru memorare nu sunt respectate, sau 500 pentru ”Internal Server Errors”.
    2) Clasa statică Journal dispune de două variabile private, readonly, ” MemoryCacheEntryOptions” pentru stabilirea ciclului de viață: _xpx= new MemoryCacheEntryOptions() { SlidingExpiration = TimeSpan.FromMinutes(1), AbsoluteExpiration = DateTimeOffset.MaxValue, Priority = CacheItemPriority.NeverRemove };
    și _xpx2= new MemoryCacheEntryOptions() { SlidingExpiration = TimeSpan.FromMinutes(180), AbsoluteExpiration = DateTimeOffset.MaxValue, Priority = CacheItemPriority.NeverRemove };
2. CHECK_GUY([FromBody] String nm)-metodă Controller
    1) Utilizează funcția Journal.CHECK_GUY pentru verificarea existenței unei entități
    2) Returnează un tuplet format din StatusCode și informatie: 200 pentru obiect gasit, 204 pentru obiect inexistent
3. DeLOGS([FromBody] String nm)- metoda Controller
    1) Utilizează funcția Journal.DeLOGS pentru ștergerea obiectului din memorie
    2) Returnează StatusCodeResult: 200 pentru obiectul a fost îndepărtat, 500 pentru ”Internal Server Error”, 204 pentru obiect negasit

Funcția statica ”ADD_GUY” poate analiza mesajul text primit în două feluri. Primul, în care variabila ”nm” nu prezintă caracterul ”_”, și obiectul adaugat e de cheie ”nm” si informație DateTime. În al doilea, sunt două șiruri separate prin ”_”, în care primul va servi ca și cheie, iar al doilea ca și informație.

Accesul la controller este destinat pentru serviciile enumerate anterior la punctul 3, pentru validarea existenței utilizatorului și execuția completă a metodelor.

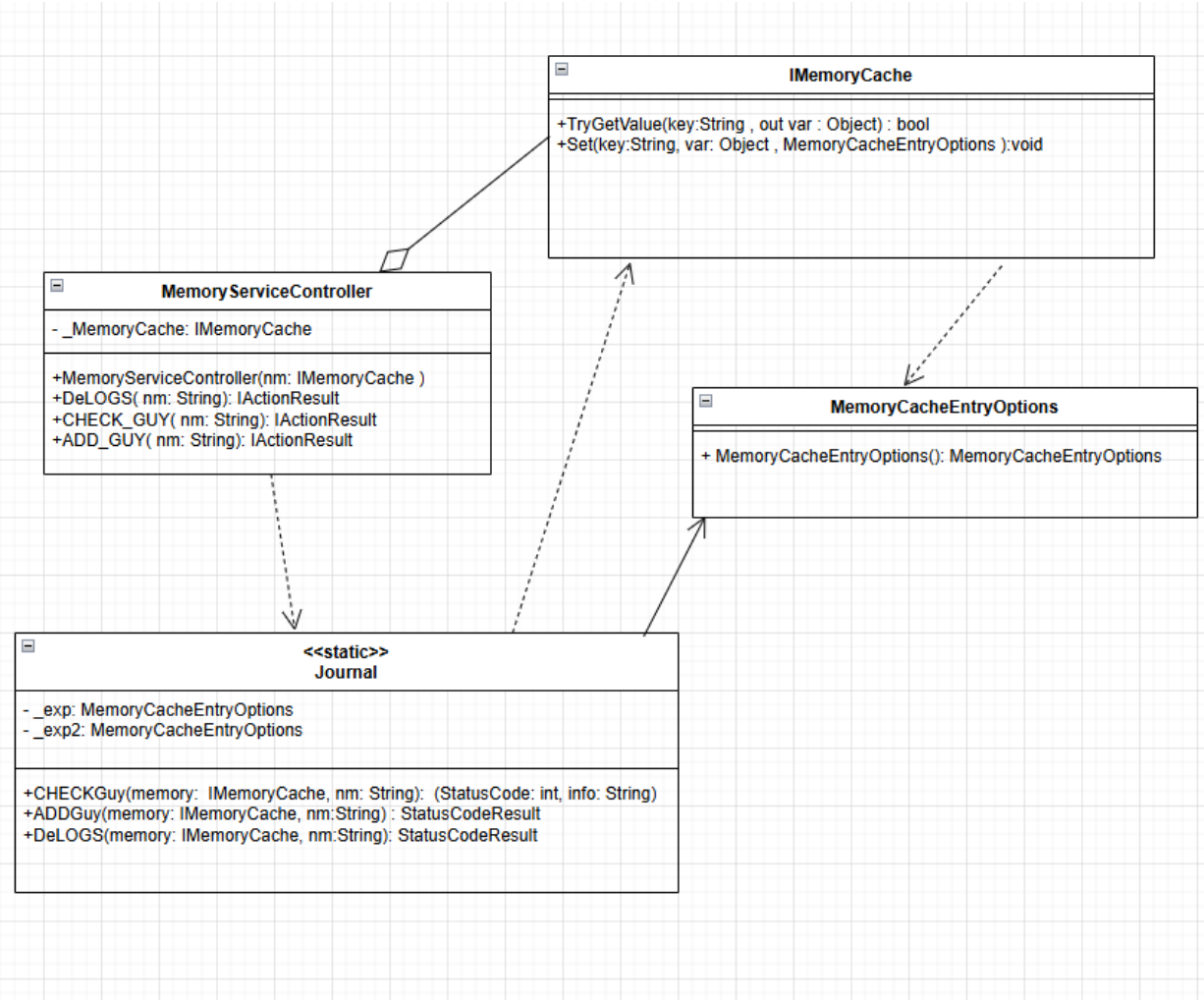


Figura 1. Diagrama clasei a serviciului Microservice1

Relațiile între clasele folosite este în principal ca ”Dependență”, unde instanțieri de clase sunt creeate în cadrul metodei, fie sunt transmise ca parametrii pentru alta. Controllerul este în dependență de clasa statică Journal, la care la rândul ei, dependentă de interfața ImemoruCache și în asociere cu MemoryCacheEntryOptions. Interfața este în agregație cu controllerul, deoarece acesta primește o referință spre un obiect existent înafara programului, care ciclu de viață de pășeste serviciul.

Diagrama secvențială a serviciului prezintă un actor, care în următoarele pagini va fi reprezentat de clasele statice ”Writterr” și ”CALL”. Aceste clase sunt folosite de restul servicilor pentru numărul minim de necesități

Clasa ”Writterr” aplică 3 funcții pentru aplicarea metodelor serviciului ”Microservice1”. Clasa ”CALL” dispune de doar o singură funcție, pentru verificarea statusului utilizatorului, online sau offline.

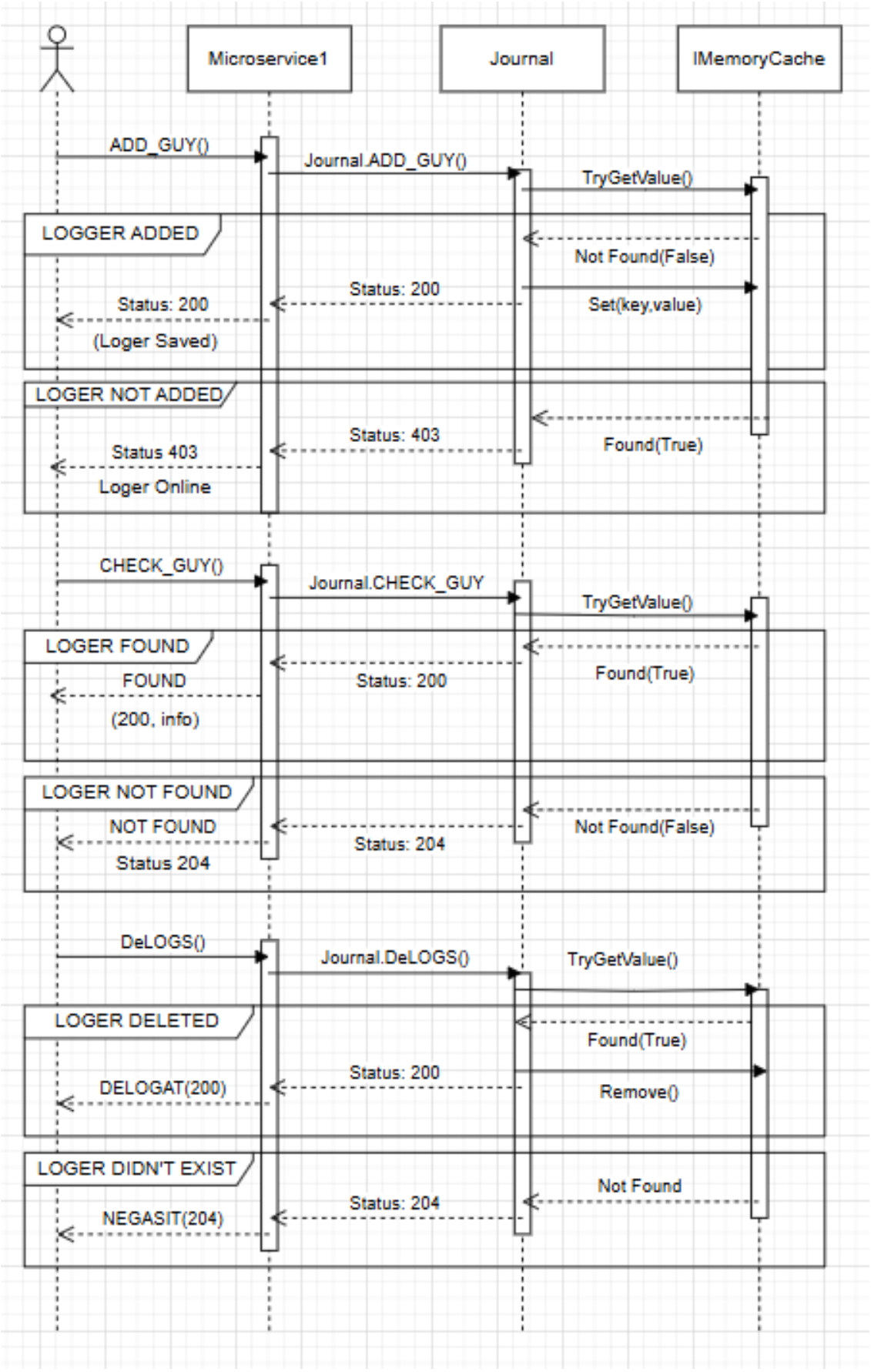


Figura 2. Diagrama segvențială a serviciului

3.2 Logging

Serviciul ”Logging” este o funție cu acces public, utilizat direct de aplicația web ”HRHelper”, cu scopul de a verifica autenticitatea utilizatorului, și oferirea accesului la restul funcționalităților aplicației.

În cadrul serviciului vom folosi modului ”Loger” pentru segmentarea automată a string-urilor de format JSON pentru o accesibilitate mai ușoară. Modulul ”Loger” predispune de următoarele proprietăți:

1. Id : int – identificator incrementat automat de baza de date
2. loger: string – email unic de adresă ”@ttp.com”
3. loger_pass: string – parola utilizatorului
4. Nume: string – nume persoană fizică
5. allow: bool – parametru pentru stabilirea accesului, setat, schimbat de administrator

Serviciul prezintă două funcționalități la îndemâna utilizatorului:

1. LogHERE([FromBody] Loger lgnm)- metodă Controller
    1) Returnează un ”IActionResult” : Ok(”MATCH FOUND”) dacă există o compatibilitate între input si baza de date, Unauthorized("YOU ARE BLOCKED!") daca parametrul allow este ”false”, Unauthorized("Email OR Password WRONG!") dacă emailul sau parola transmisă nu duc la o compatibilitate, BadRequest("!SOMETHING WENT WRONG IN THE SERVER!") dacă a aparut o eroare internă, Unauthorized("ALREADY ONLINE! GO AWAY!") dacă utilizatorul este deja online.
2. DeLOGS([FromBody]String nm)- metodă Controller
    1) Returnează un ”IActionResult” : Ok() dacă deconectarea, respectiv ștergerea utilizatorului din memoria Cache a fost un succes, NoContent() dacă utilizatorul deja nu existase.

Relația între aproape toate elementele serviciului este de dependență, cu excepția interfeței IConfiguration, ce este injectată la început de serviciu, folosită pentru preluarea parametrilor folosiți în accesarea bazei de date.

Serviciul ”Logging” depinde de fiecare altă clasă, fie module sau clasa statică ”Writterr”, doar executând metodele acesteia pentru modificarea memoriei. Clasele ”HttpClient” și ”HttpResponseMessage” sunt în simplă dependență de ”Writterr”, clasa statică instanțiend tipul claselor în interiorul funțiilor, nu la nivel de clasă.

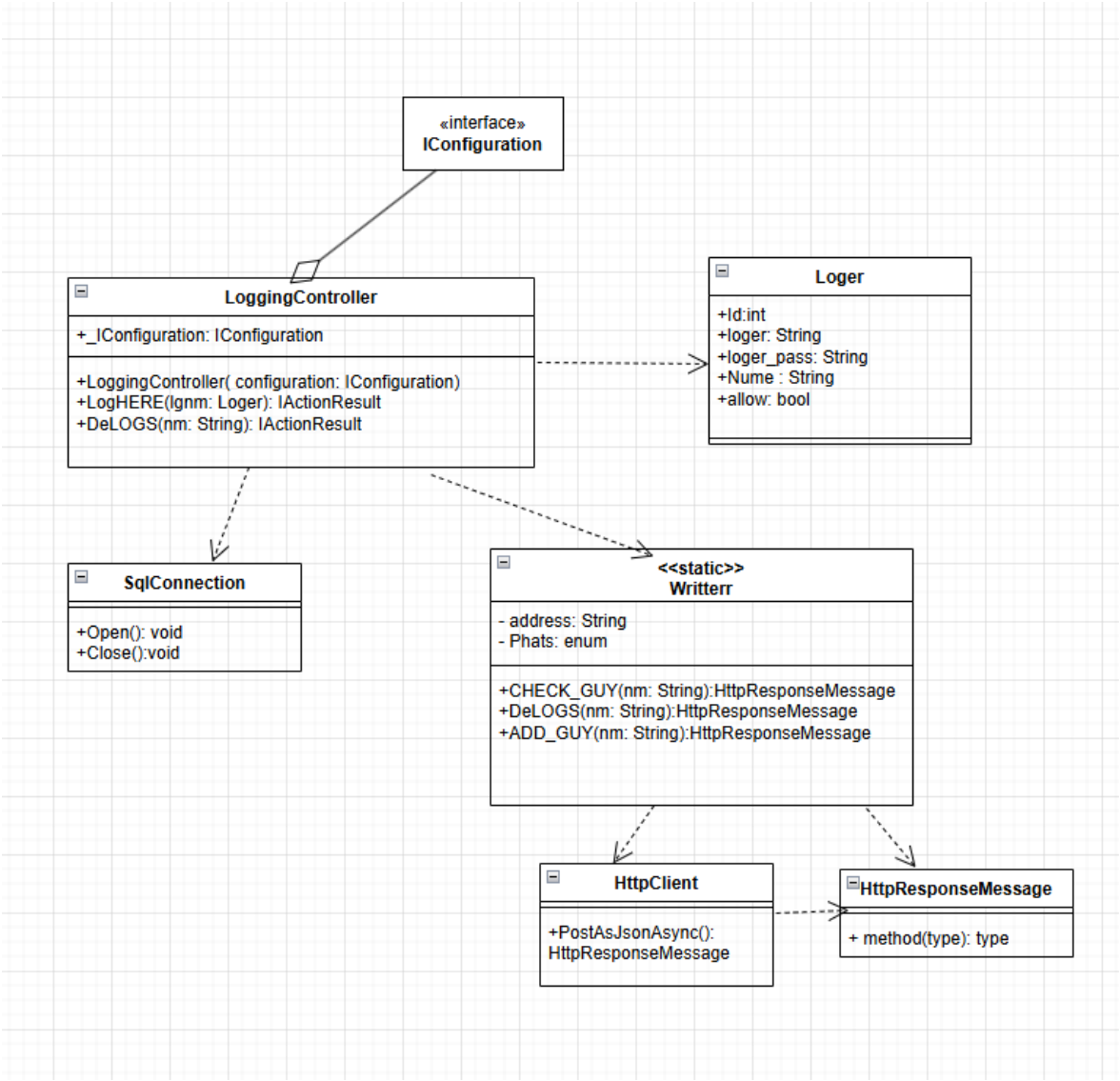


Figura 3. Diagrama claselor serviciului ”Logging”

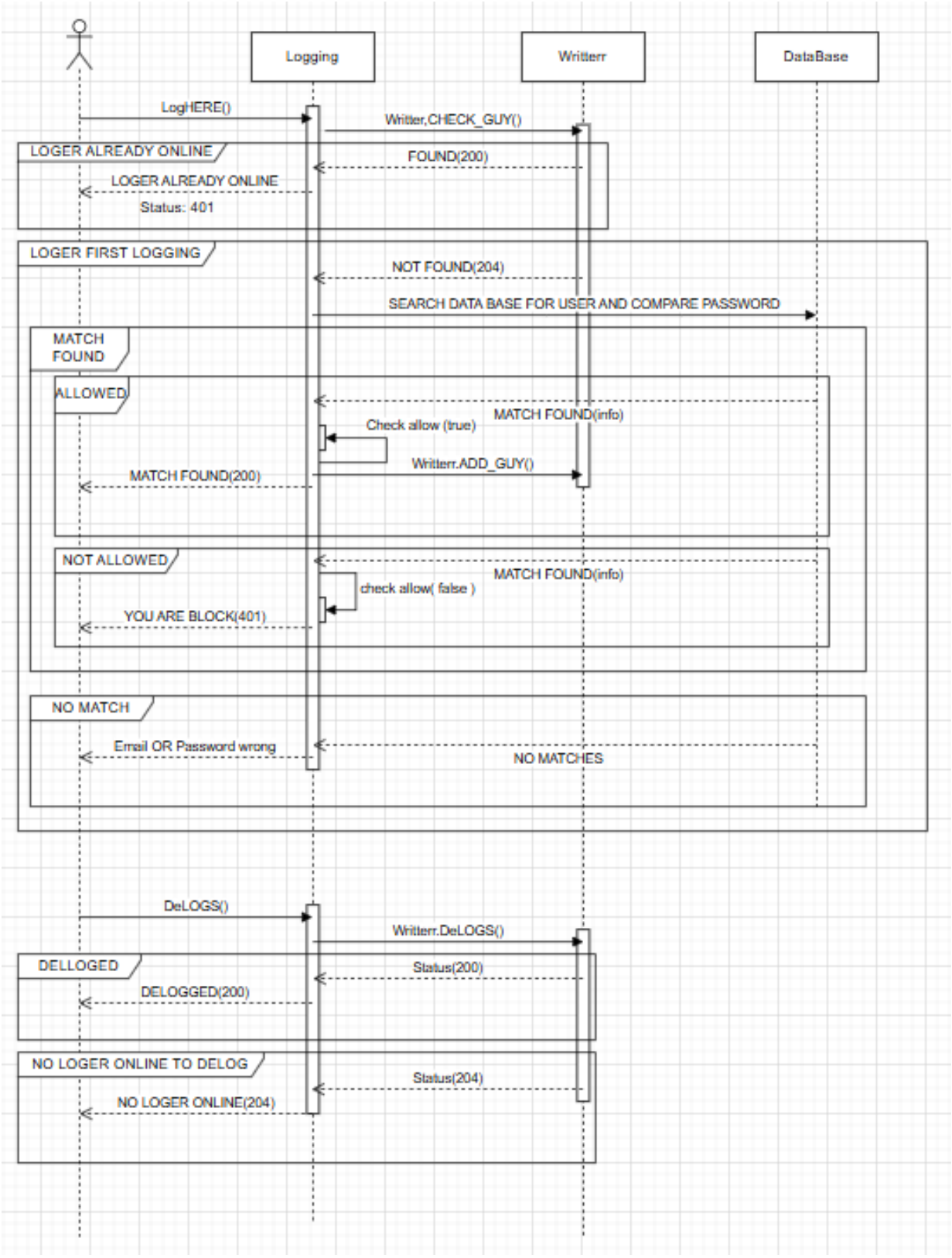


Figura 4. Diagrama segvențială a serviciului ”Logging”

3.2 Messages

Serviciul ”Messages” este folosit în aplicație pentru preluarea salvarea și returnarea mesajelor transmise de utilizator în baza de date. Totodată trebuie precizat, doar mesajele destinate între utilizatori, ci nu clienți înafara aplicației, sunt salvate în baza de date. În scurt, serviciul este utilizat pentru mesaje unde destinatatul are emailul terminat in ”@ttp.com”. Selecția direcției de transmisie, către baza de date sau către un client printr-un domeniu email pe internet se face în funcțiile logice Javascript ale aplicației web.

Acest serviciu a fost primul conceput ce folosește clasa statica ”CALL”, capabilă sa caute prezenta unui obiect în memoria Cache, folosind un text string ca și parametru cheie de căutare.

Serviciul folosește totodata modulul ”Message” pentru segmentarea mesajului JSON transmis prin Body de cererea Http, fiind format din:

1. ForName: string- destinatarul
2. FromName: string – expeditor
3. iMessage: string – mesajul în sine

Metodele disponibile prin utilizarea serviciului sunt:

1. SendMessage([FromBody] Message nm)- metodă Controller
    1) Metoda preia mesajul și salvează în baza de date
    2) Returneaza IActionResult: Ok(”MESAJ TRIMIS”) daca mesajul este salvat, BadRequest("MESAJUL NU A FOST TRIMIS") contrar, Unauthorized("LOGER OFFLINE! GET LOST") dacă funcția CALL.CHECK_GUY nu găsește utilizatorul în memoria Cache, BadRequest("!INTERNAL ERROR!") managementul excepțiilor.
2. GetMessages([FromBody] String email)- metoda Controller
    1) Returneaza o List<Message> sub format JSON string cu mesajele destinate utilizatorului, cautand după email.
    2) Returnează: Ok(JsonConvert.SerializeObject(list)) dacă fetchul se execută corect, Unauthorized("LOGER OFFLINE! GET LOST") dacă funcția CALL.CHECK_GUY nu găsește utilizatorul în memoria Cache și BadRequest("!INTERNAL ERROR!") la apariția unor erori interne de server.

Relația între clase este practic identică cu relațiile claselor serviciului ”Logging”, singurele diferențe fiind controllerul ”MessageController” și modulul ”Message”.

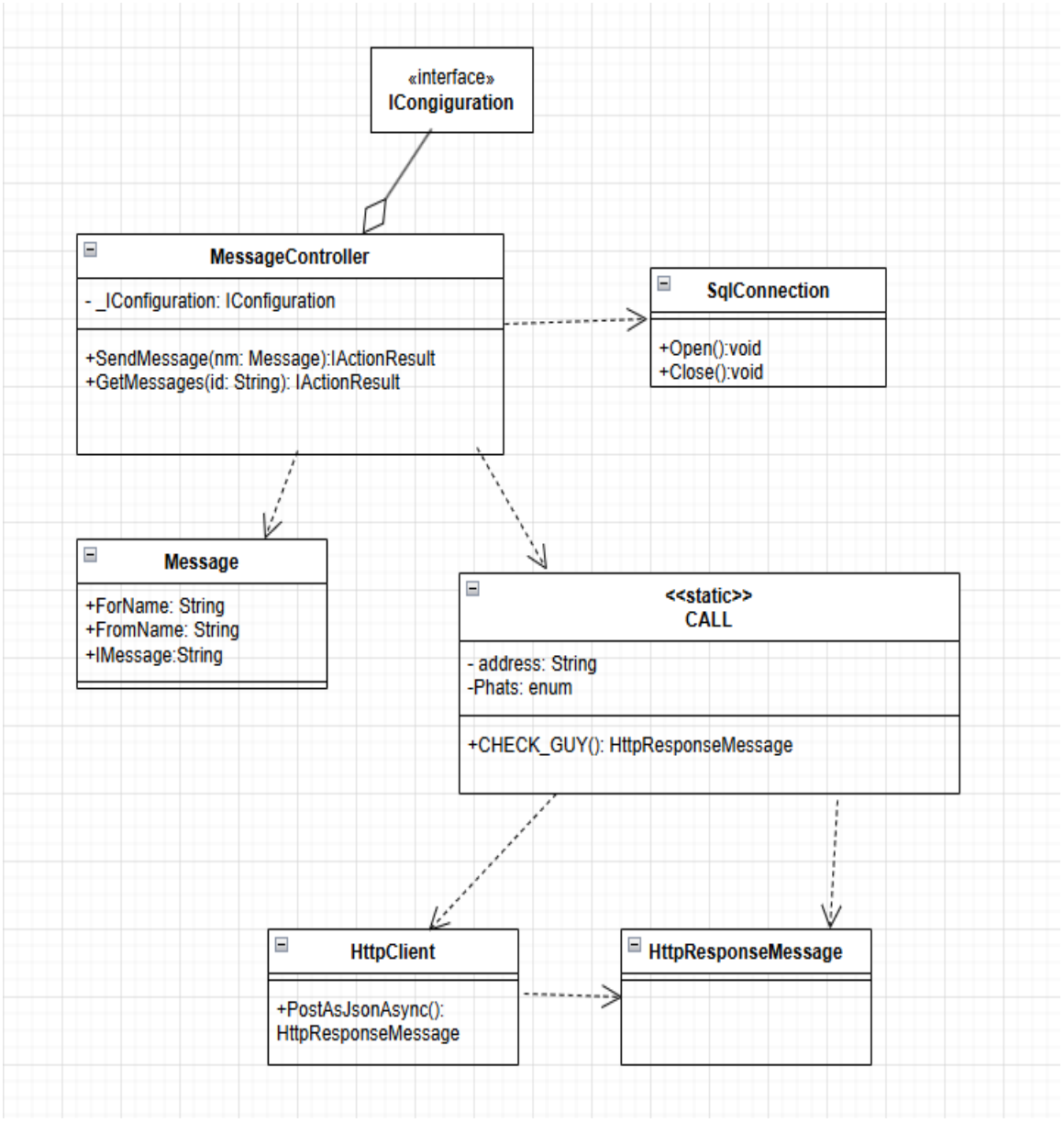


Figura 5. Diagrama claselor serviciului ”Messages”

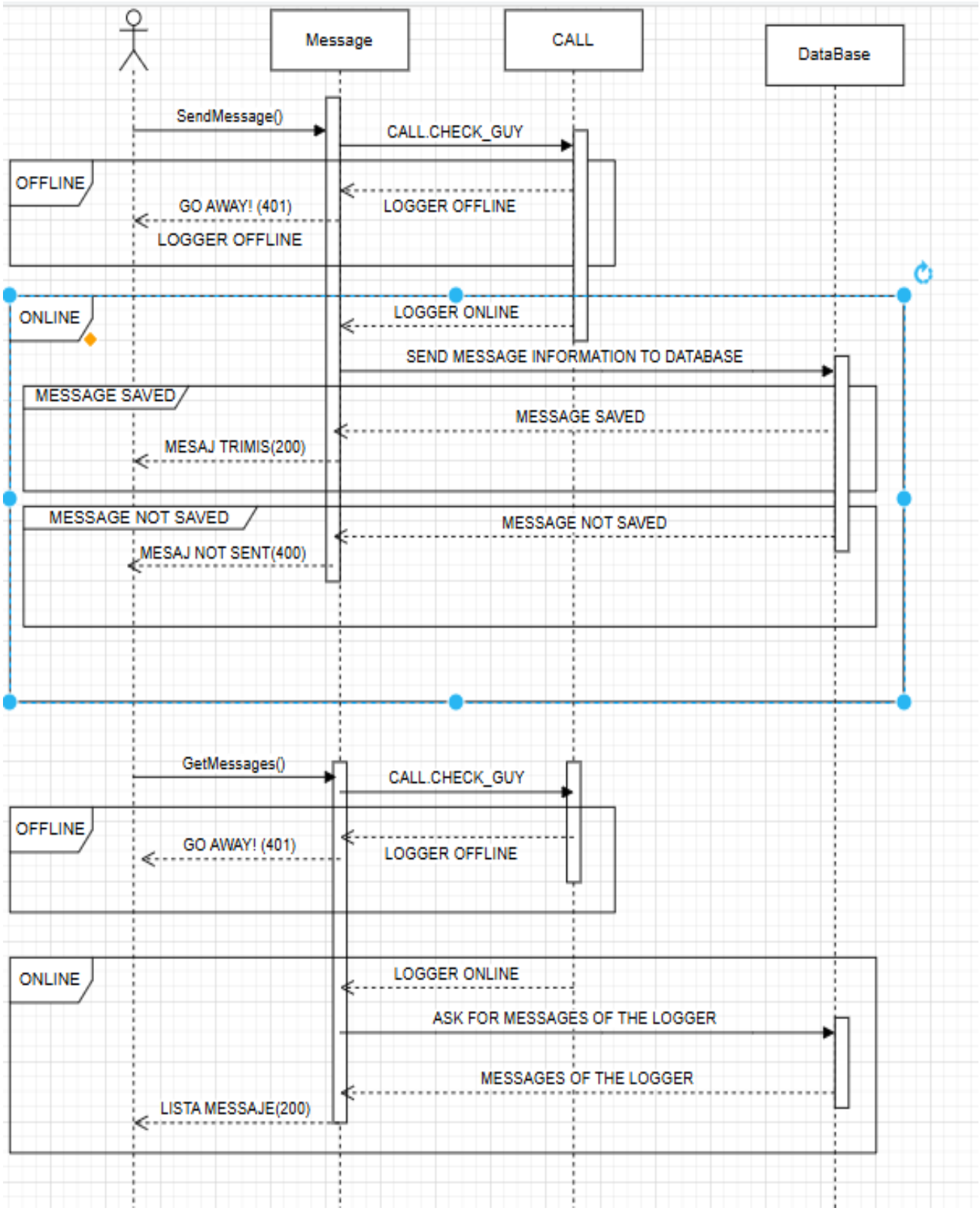


Figura 6. Diagrama segvențială a serviciului ”Messages”

3.4 Persons

Serviciul ”Persons” este folosit de utilizator pentru adaugarea, stergerea și vizualizarea clienților din tabela ce serveste ca agenda pentru utilizator. În aplicația web, acesta este necesară pentru preluarea informațiilor mandatorii pentru trimiterea mesajelor.

Serviciul utilizează clasa statica ”CALL” pentru verificarea statusului utilizatorului ce trimite mesajul către serviciu, mandatoriu în majoritatea serviciilor ca mod adițional de securitate. Modulul ”Person” este folosit pentru segmentarea datelor transmise prin format JSON prin corpul cererii Http, având următoarele proprietăți:

1. Id: int – identificator unic incrementat de baza de date la adaugare
2. Name: string – numele contactului
3. Company: string – numele companiei pentru care contactul lucrează
4. Phone: string – număr de telefon
5. Info: string – scurtă detaliere a contactului
6. Email : string – email pentru contactare
7. Idf: string – email utilizator pentru care a fost salvat

Metodele procurate de serviciul ”Persons” sut următoarele:

1. GetThis([FromBody] String x)- metodă Controller
    1) Returnează un IActionResult :Ok(JsonConvert. SerializeObject(list)) ce cară lista contactelor specifice utilizatorului, Unauthorized("UTILIZATOR OFFLINE! GET LOST") dacă funcția CALL.CHECK_GUY nu găsește utilizatorul în memoria Cache, BadRequest(ex.Message) dacă există excepții.
2. AddPerson([FromBody] Person x) – metodă Controller
    1) Folosit pentru adaugarea unui nou contact
    2) Returneaza un IActionResult: Ok("PERSON ADDED") dacă persoana a fost adăugată cu succes la baza de date, BadRequest("Couldn't ADD to DataBase") dacă are loc o eroare la nivelul bazei de date, Unauthorized ("UTILIZATOR OFFLINE! GET LOST") dacă funcția CALL.CHECK_GUY nu găsește utilizatorul în memoria Cache și BadRequest(ex.Message) pentru erori la nivel de server.

3. DeletePerson([FromBody]String id)- metoda Controller
    1) Șirul preluat din body este un șir JSON, format din numele contactului ce trebuie șters și emailul utilizatorului
    2) Folosit pentru ștergerea unui nou contact
    3) Returneaza un IActionResult: Ok("PERSOANA ȘTEARSĂ") dacă persoana a fost ștearsă cu succes din baza de date, Ok("PERSOANA NU A FOST GASITA ") dacă nu există în baza de date, Unauthorized ("UTILIZATOR OFFLINE! GET LOST") dacă funcția CALL.CHECK_GUY nu găsește utilizatorul în memoria Cache și BadRequest (ex.Message) pentru erori la nivel de server.

Relația între clase este practic identică cu relațiile claselor serviciului ”Logging”, singurele diferențe fiind controllerul ”PersonsController” și modulul ”Person”.

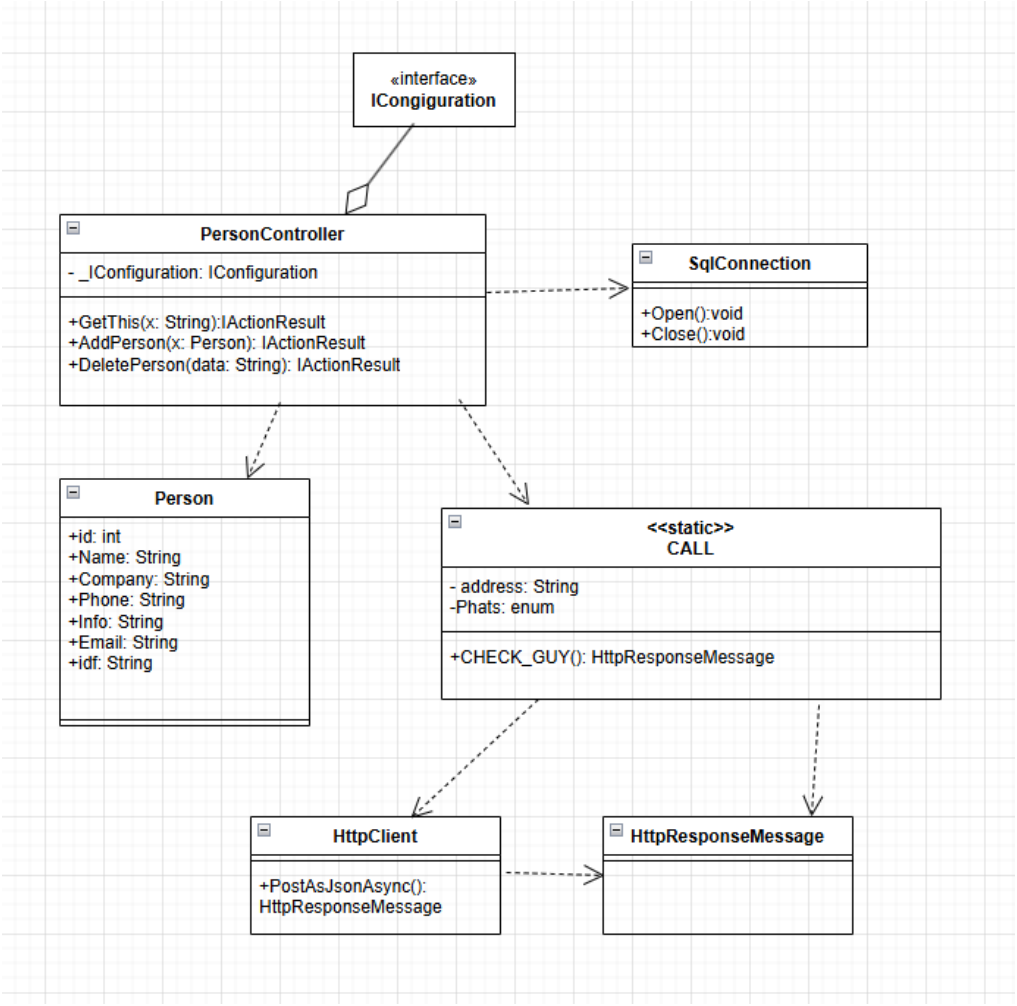


Figura 7. Diagrama claselor serviciului ”Persons”

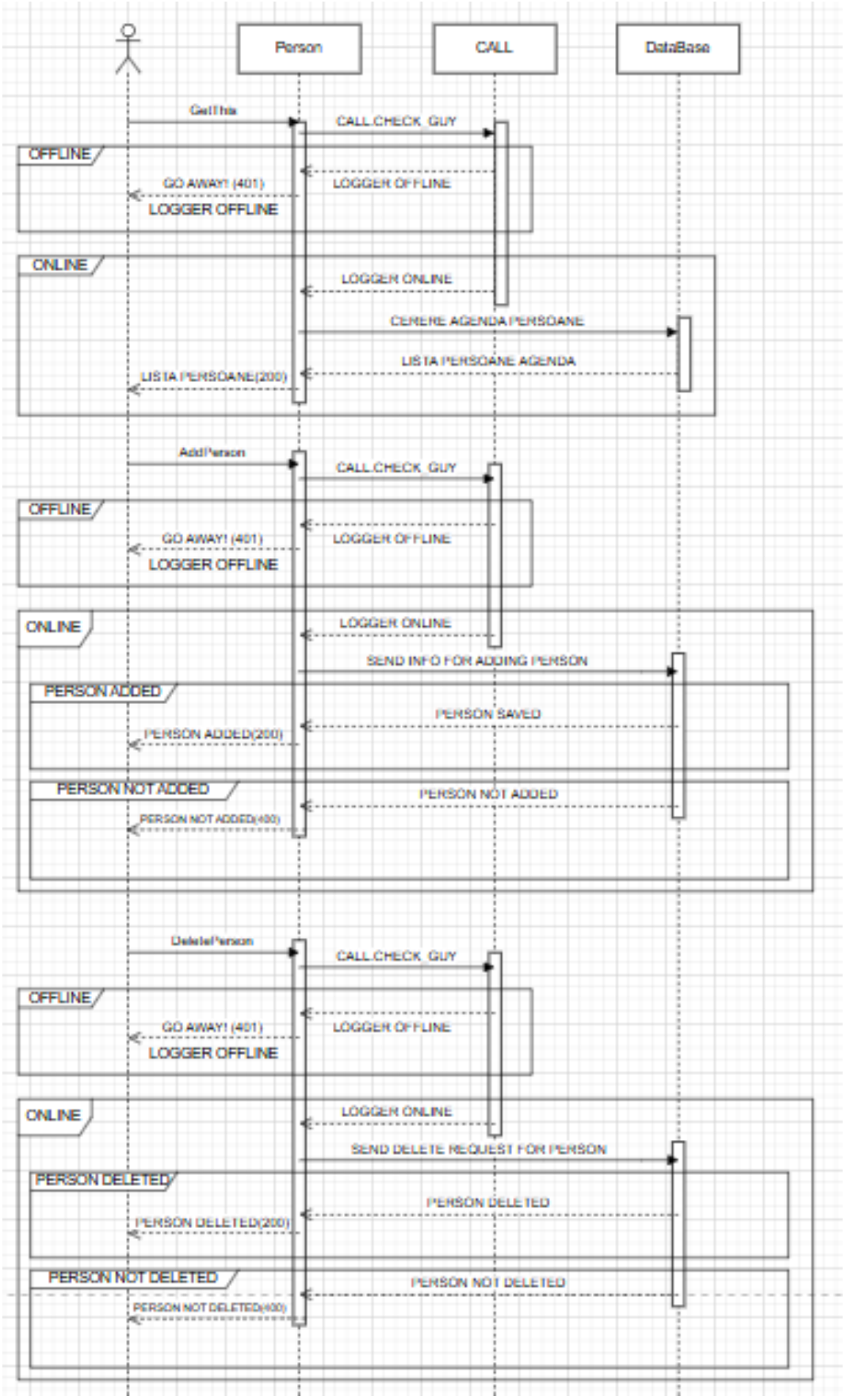


Figura 8. Diagrama segvențială a serviciului ”Persons”

3.5 Outside_Email

Serviciul a fost creat pentru genstionarea mesajelor ce trebuie trimise pe domenii diferite față de ”ttp.com”, în esență care este folosit pentru comunicarea intre utilizatori la nivelul firmei. Menționat mai devreme, atât serviciul ”Messages” cât și ”Outside_Email” sunt alese a fi utilizate la nivelul logicii javascript din cadrul aplicației web ”HRHelper”.

Pe lângă utilizarea clasei statice ”CALL” pentru determinarea starii online sau offline a utilizatorului, controllerul utilizează o clasă statică secundară, ”EmailSender” ce conține funcțiile necesare.

3.5.1 Clasa statica EmailSender

O clasă cu arhitectură relativ simplă, formată din 3 funcții și 3 câmpuri statice readonly ce prezintă ” base_mail”, ” App_Password” și ” subject”. Datorită resurselor limitate, a fost utilizat un domeniu ”3th party” pentru transmiterea emailurilor pe internet, Yahoo.com. Utilizarea unui cont deja existent presupune creearea unei parole aplicație pentru conectarea la contrul respectiv. Câmpul ” base_mail” presupune adresa contrului creat, ” App_Password” este parola generata de opțiuniile domeniului Yahoo pentru a putea fi accesat contul folosind un program ”3th party”. Subject este o variabilă a clasei cu valoare ”no_reply_email”.

Funcțiile dezolvate pentru asigurarea transmiterii emailului sunt:

1. SendEmailAsync(string email, string message)- funcție asincronă
    1) Returnează :un Task<string> ce este un raspuns string de format Guid, creea de clasa cu același nume, un identificator pe 128 de biti, unic, o eroare dacă emailul nu a putut fi trimis din cauza nedetectării unui domeniu valid.
    2) Funcția instanțiază obiecte pentru clasele MimeMessage, utilizată în formatarea mesajului și SmtpClient, folosită pentru conectarea la domeniu și trimiterea emailului.
    3) A fost incercată introducerea unui now ”Header” pentru urmărirea emailului, conținând identificatorul Guid, însă in urma testelor, a fost determinat că domeniul Yahoo.com duce la modificarea acestuia, nereușind a urmări sau gasi de aplicație în cazul procesului de verificare ”bounce”. Acesta a fost introdus după în corpul mesajului.

2. CheckBouncesAsync(Message msg)- funcție asincronă
    1) In cazul acestei funcții, msg.iMessage, proprietatea modulului Message, conține un șir JSON format din mesajul string vechi și identificatorul Guid.
    2) Nu returnează nimic. Dacă emailul anterior ce a fost trimis se va dovedi a fi eșuat, va fi apelată cea a treia funcție a clasei ”EmailSender”, care transmite un mesaj de ”fail” ce va fi salvat în baza de date, destinat utilizatorului ce a trimis emailul.
    3) Prezintă conectarea la contul din domeniu folosind ”base_email” și ”App_Password” folosindu-ne de clasa ”ImapClient”
    4) Odată conectați, putem cere emailurile existente în ”inbox”, o priprietate de tip IMailFolder.
    5) Emailurile sunt filtrate după două condiții: dacă sunt eșuate, funcția ” SearchAsync (SearchQuery.SubjectContains ("Failure Notice"))” , dacă trimiterea emailului a fost cerută în ultimele 10 minute și dacă a fost trimis în aceeași zi.
3. SendBACK(Message msg)- funcție clasa statică
    1) Nu returnează nimic.
    2) Apelată doar la gasirea unui email eșuat, în interiorul funcției CheckBouncesAsync().
    3) Folosită pentru a apela serviciul ”Messages” și a salva mesajul formatat de funcția CheckBouncesAsync().

3.5.2 Controllerul

Controllerul serviciului dispune de o singură metodă ce utilizează optim toate resursele clasei statice ”EmailSender”:

- SendEmail([FromBody] Message msg)- metoda asincronă
    1) Execută procesul de trimitere a mesajului prin email
    2) Returnează Task<IActionResult>: Unauthorized("LOGER OFFLINE! GET LOST") dacă utilizatorul ce a cerut transmiterea nu este online, BadRequest(resp) dacă în urma executarii funcției SendEmailAsync() a fost returnată o eroare.

3) Dacă funcția SendEmailAsync() returnează pozitiv, respectiv șirul identificator Guid emailului trimis, se poate începe cautarea emailului eșuat, dacă acesta există, după identificator.
4) Un emai poate fi învalid din două motive însă rezulta în moduri diferite: email invalid însă domeniu corect, pentru care expecție este folosită funcția CheckBouncesAsync, sau când domeniul este invalid, astfel funcția SmtpClient.Send() va arunca o eroare ce va fi gestionată în SendEmail().
5) Funcția CheckBouncesAsync() este aplicată folosind Task.Run(), astfel aceasta poate rula pe un thread separat, micșorând timpul de așteptare pentru utilizator.

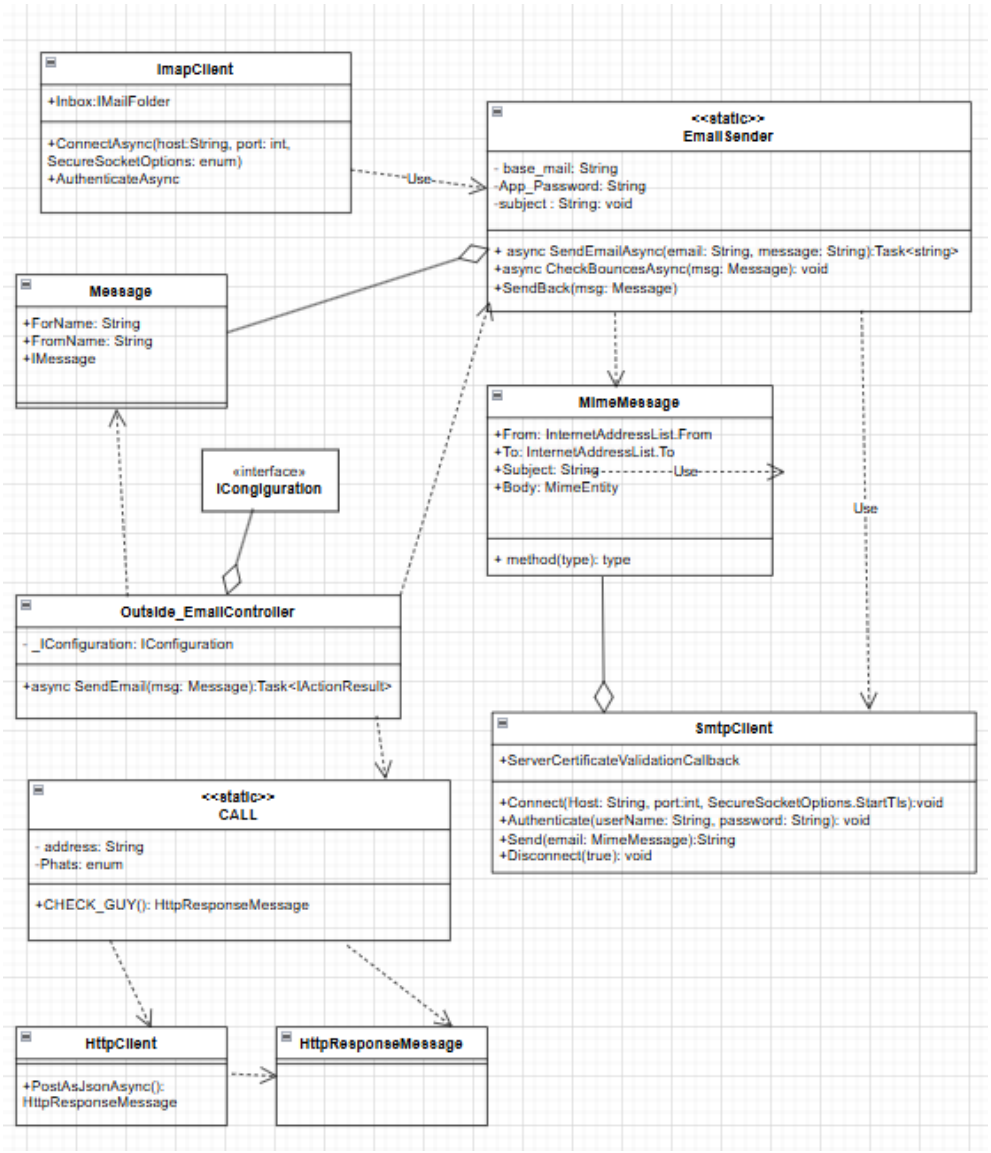


Figura 9. Diagrama claselor serviciului ”Outside_Email”

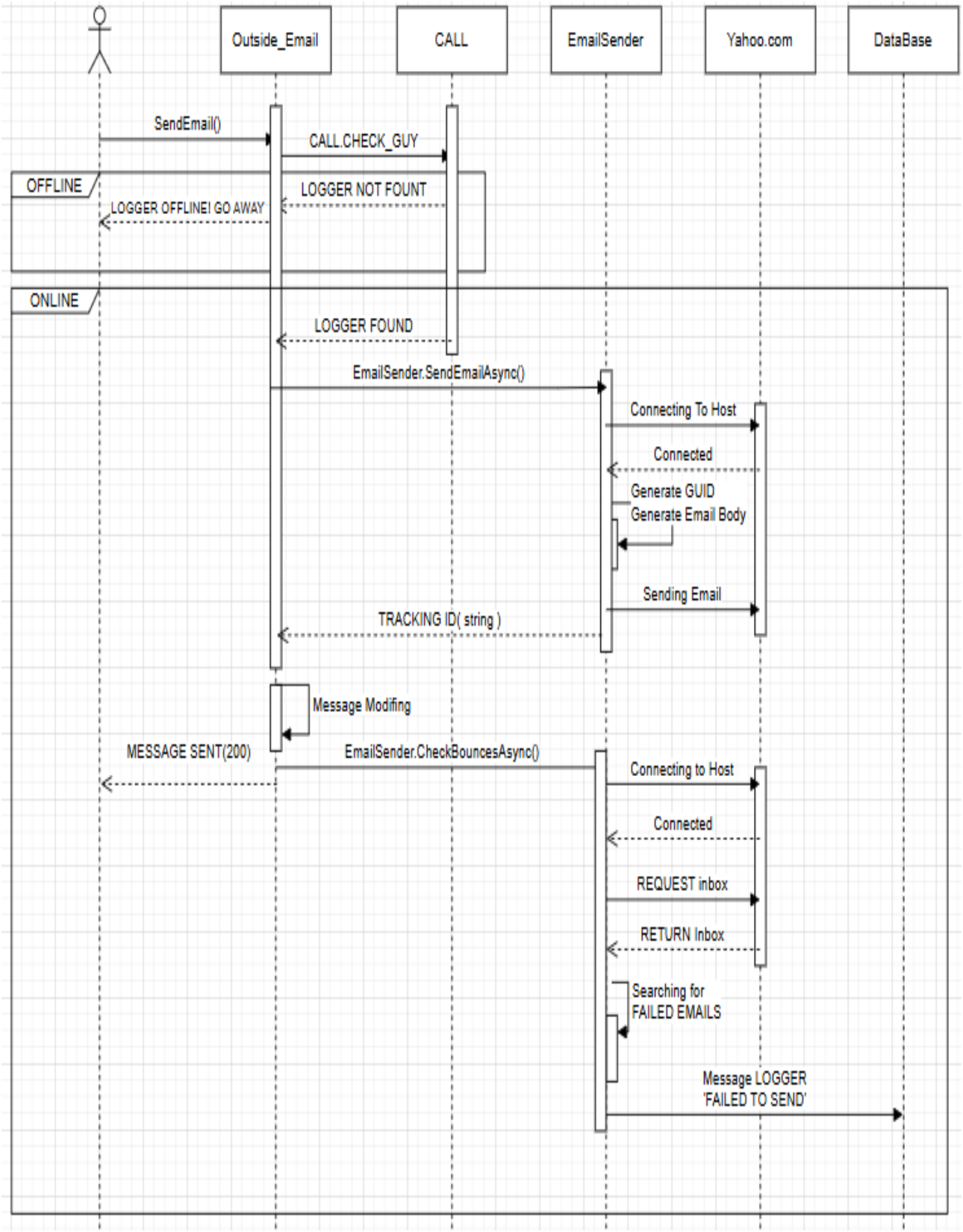


Figura 10. Diagrama segvențială a serviciului ”Outside_Email”

3.6 Admin

Serviciul ”Admin” este special rezervat presupusei echipe tehnice al domeniului ”@ttp.com”, nefiind valabilă utilizatorilor. Acesta pune la dispoziție 5 metode ce pot fi utilizate pentru a urmări și ajuta utilizatorii în diferite activități. Totodată, în cadrul acestui serviciu vom introduce o nouă funcție statica numită ”Brainiac”, din nou, pentru ordotarea și segmentarea codului.

Serviciul dispune de elemente de elemente Front-end pentru funcționalitatea schimbării parolei utilizatorului:

1. PASS_FORM.html- schița standard formular parolă.
2. Cl.js-fișier cu funcționalități Javascript.
3. Folder ”ONLINES”- conține fișiere Html personalizate.

Funcția statica are 4 funcții folosite în metodele controllerului:

1. GET_LOGERS(IConfiguration _Configuration)
    1) Returnează o List<Loger>: o listă a utilizatoriilor din baza de date dacă a fost primit ur raspuns, chiar dacă nu există utilizatori, o listă cu un singur element ce transmite mesajul de eroare.
    2) Variabilele preluate din tabela bazei de date sunt doar variabilele ”loger”, adresa de email a utilizatorului și parametrul ”allow”
2. Access_CHANGER(IConfiguration _Configuration, string nm)
    1) Folosită pentru schimbarea parametrului de access între ”true” și ”false”
    2) Returnează StatusCodeResult: 200 dacă parametrul a fost schimbat cu succes, 400 dacă a avut loc o problemă la updatarea parametrului utilizatorului, 500 pentru erori interne de server
3. Send_Email_Pass_CHANGER(string nm)
    1) Folosit pentru a trimite un email pentru schimbarea parolei utilizatorului. Trimiterea se face prin clasa statica ”EmailSender”.
    2) Parametrul ”nm” are nevoie să fie doua șire separate prin ”_”, unde primul șir este adresa spre care va fi trimis emailul și al doilea, emailul utilizatorului din baza de date.

3) Fiindcă domenile email nu permit executarea de fișiere Javascript în cadrul interfeței utilizator, a fost nevoie de transmiterea unui mesaj conținând link către pagina Html hostată în domeniul propriu.
4) Returnează Task<StatusCodeResult>: 200 dacă nu au aparut probleme la trimiterea emailului, 400 dacă au aparut erori, 500 pentru erori interne.
5) Activarea funcții duce la crearea fișiere individuale pentru fiecare cerere de la utilizator, pe baza fișierului html ”PASS_FORM”. Fiecare copie este după modificată să conțină un identificator Guid unic.
6) După formatarea mesajului, acesta este trimis către adresa folosită de utilizator pentru a-și schimba parola.
7) Dacă emailul de resetare al parolei a fost trimis cu succes, funcția continuă cu salvarea adresei utilizatorului în memoria Cache folosind clasa statică ”Writterr”, folosind ca și cheie identificatorul generat fișierului pentru a fi folosit mai târziu, după ce utilizatorul a ales o nouă parolă.

4. PASS_CHANGER(IConfiguration _Configuration, PSSWRD user)
    1) Folosit pentru modificarea parolei utilizatorului
    2) Returnează Task<string>: "INFORMATION WRONG" dacă identificatorul Guid nu este valid, "SESSION EXPIRED" datorită proprietății SlidingExpiration, care precizează ce obiectul stocat va fi șters dacă nu este accesat în perioada de timp precizată. Totodată, în acest caz, fișierul personalizat este șters. "PASSWORD WAS UPDATED" dacă parola a fost schimbată cu succes, "COULDN'T UPDATE THE PASSWORD" dacă am eșuat, "File was not found" dacă fișierul nu a fost creat sau posibil șters, mesaj de eroare.
    3) După schimbarea reușită a parolei, obiectul memorat în Cache și fișierul sunt șterse.

Metodele serviciului ce utilizează funcțiile clasei statice ”Brainiac” sunt:

1) GET_LOGERS()- metodă Controller
    1) Rulează funcția GET_LOGERS(IConfiguration _Configuration)
    2) Returnează IActionResult: Ok(JsonConvert.SerializeObject (Brainiac.GET_LOGERS(_Configuration))), BadRequest (ex.Message);
2) Check_LOGER([FromBody] string nm) -metodă Controller
    1) Rulează Writterr.CHECK_GUY(string nm)
    2) Returnează IActionResult: Ok(" Loger IS ONLINE"), NoContent() dacă logerul nu este online, BadRequest(ex.Message) la apariția unei erori interne.
3) Access_CHANGER([FromBody] string nm)- metodă Controller
    1) Rulează funcția Access_CHANGER(IConfiguration _Configuration, string nm)
    2) Returnează IActionResult: Ok("Allow parameter CHANGED") dacă parametrul de acces a fost schimbat, BadRequest ("SOMETHING WENT WRONG") la aparitia unei erori la nivelul bazei de date, BadRequest(ex.Message) pentru erori interne de server.
    3) Dacă schimbarea parametrului a fost un succes, si utilizatorul e online, acesta fa fi deconectat prin Writterr.DeLOGS(string nm)
4) Send_Link_Password_Changer([FromBody] string info)
    1) Rulează funcția Send_Email_Pass_CHANGER(string nm)
    2) Returnează Task<IActionResult>: Ok() dacă emailul e trimis cu succes, BadRequest(resp) dacă a apărut o eroare la nivelul domeniului folosit, BadRequest(ex.Message) pentru erori interne.
5) PASS_CHANGER([FromBody] PSSWRD user)- metodă Controller
    1) Rulează funcția PASS_CHANGER(IConfiguration _Configuration, PSSWRD user), unde PSSWRD este un modul pentru segmentarea mesajului JSON transmis de la pagina Html.
    2) Accesată de fișierul ”Cl.js” partajat fișierului personalizat Html
    3) Returnează Task<IActionResult>: Ok(resp), unde ”resp” este raspunsul către utilizator, poziti sau negatic, BadRequest (ex.Message) mesaj în cazul unei erori interne.

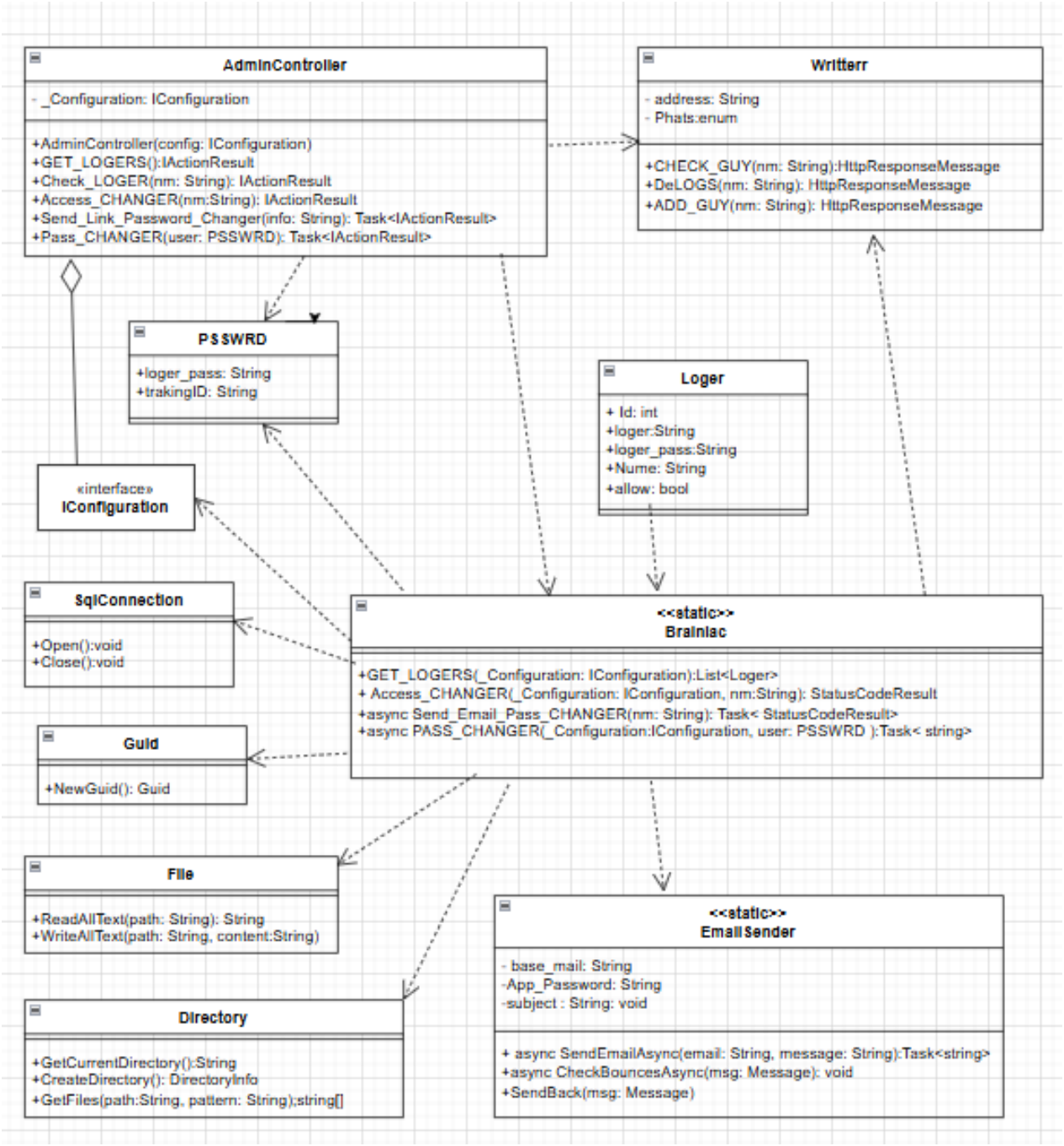


Figura 11. Diagrama claselor serviciului ”Admin”

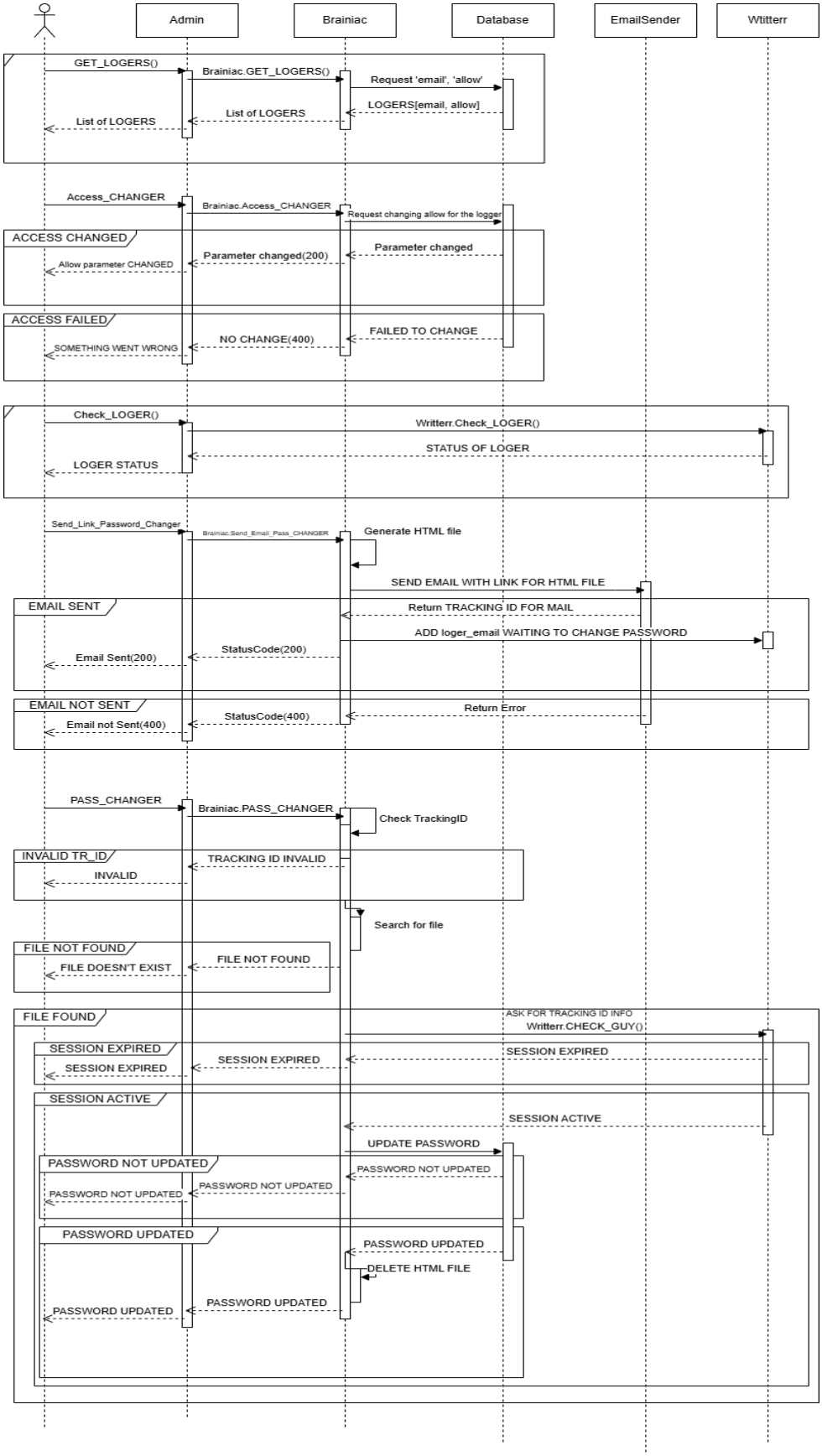


Figura 12. Diagrama segvențială a serviciului ”Admin”

4. Testare

Testarea se va executa atât prin aplicația web, cât și prin interfața Swagger pentru a urmării rezultatele din ambele perspective. Conturile de utilizator folosite vor fi arlapuste@ttp.com și tcarmen@ttp.com, cu parole ”rain”, respectiv ”dog”

4.1 Microservice1

4.1.1 ADD_GUY

Adaugarea unui utilizator în memoria Cache, fară ca el să fie prezent, e un success. Adaugarea repetată al aceluiaș utilizator va rezulta într-o eroare.

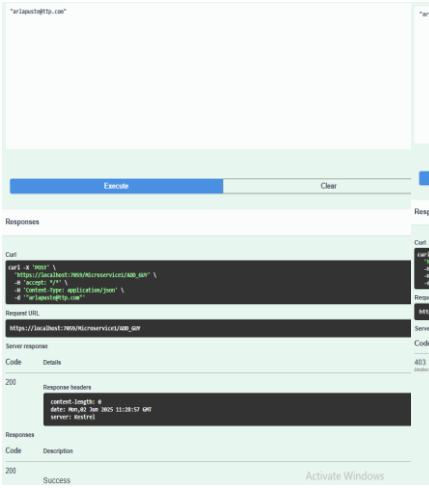


Figura 13

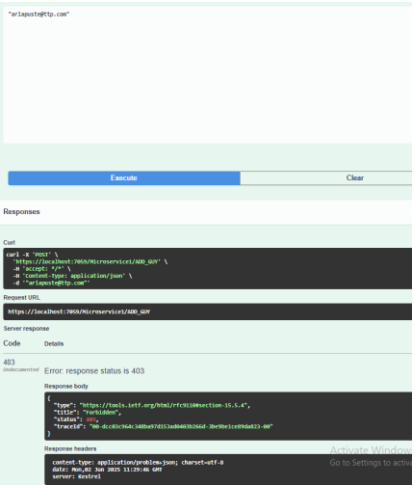


Figura 14

4.1.2 CHECK_GUY

Verificarea pentru un utilizator deja online produce un status code ”success”, lipsa acestuia produce ”NoContent”.

Figura 15.

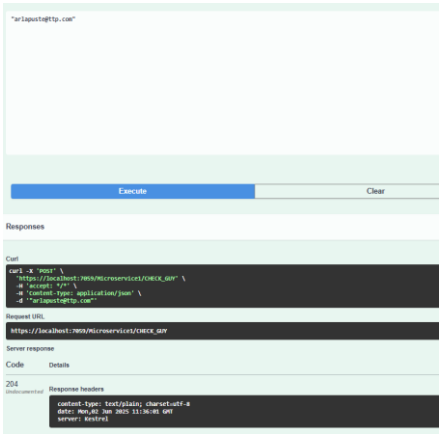


Figura 16.

4.1.3 DeLOGS

Apelarea metodei produce un succes la prezenta utilizatorului și un ”NoContent” la lipsa sa.

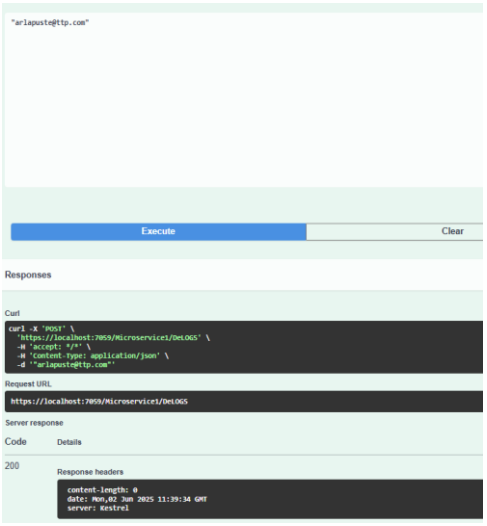


Figura 17.

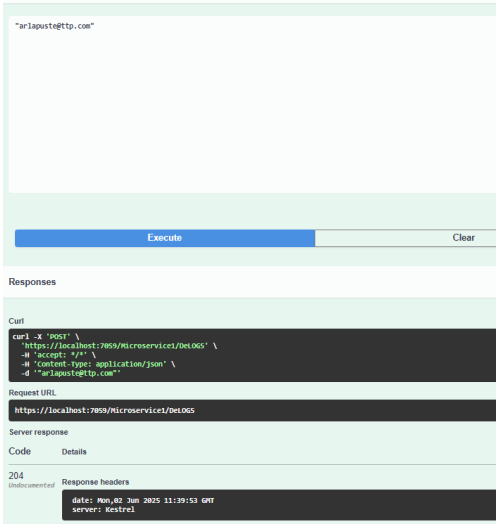


Figura 18.

4.2 Logging

4.2.1 LogHERE

Logarea folosind adresa de emai și parola validă. Logarea când utilizatorul deja este online.

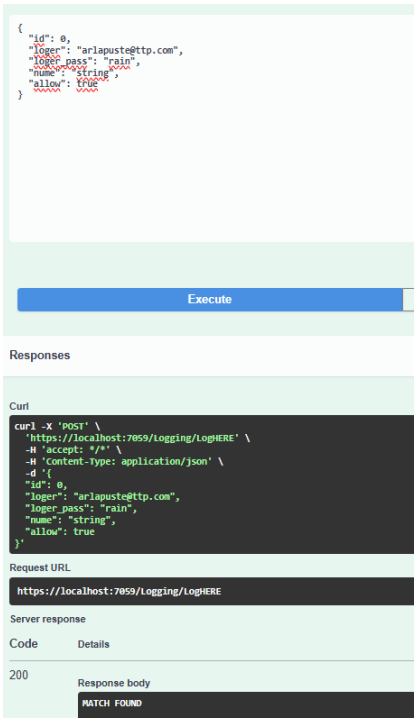


Figura 19.

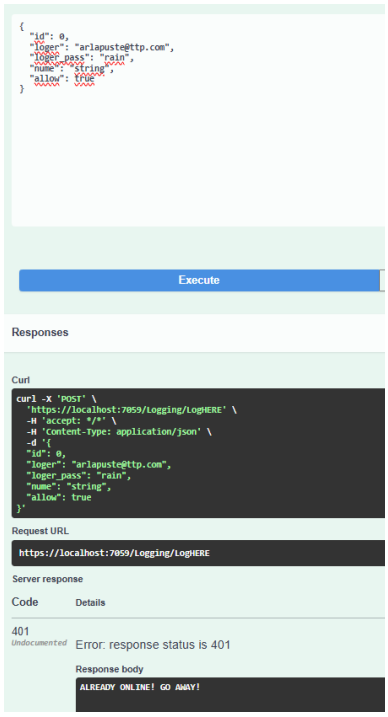


Figura 20.

Logarea când folosim o parolă sau adresă greșită

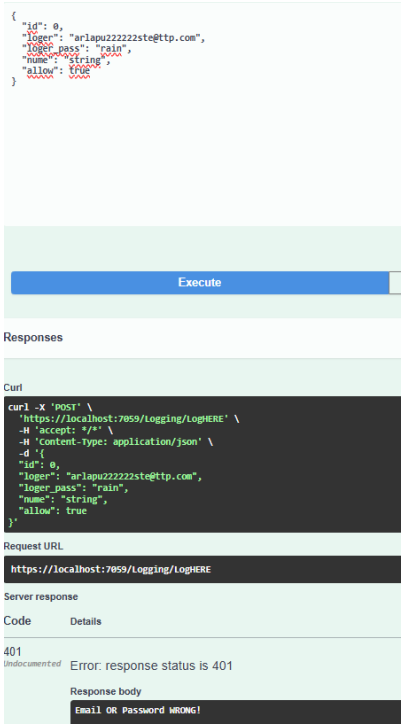


Figura 21.

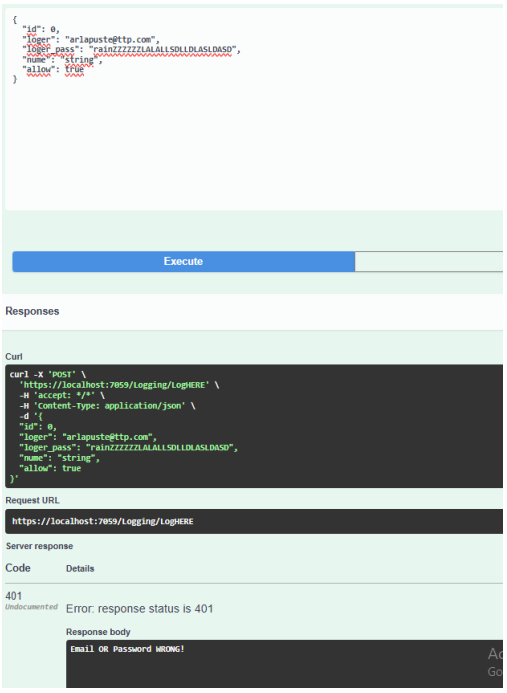


Figura 22.

Logarea cu email și parolă corecte, dar parametrul de acces e ”False”

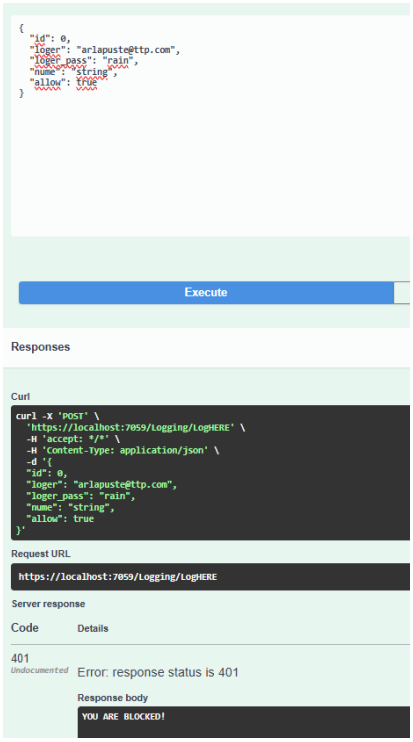


Figura 23.

4.2.2 DeLOGS

Delogarea când utilizatorul este online. Când utilizatorul este offline

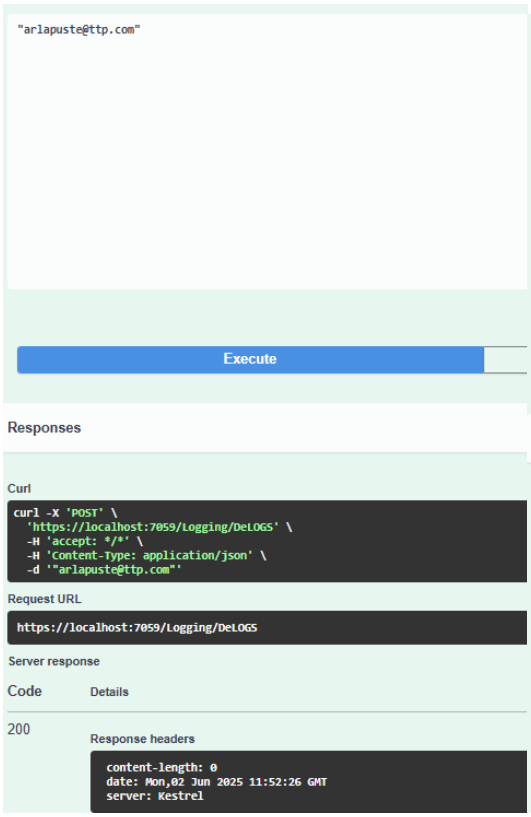


Figura 24.

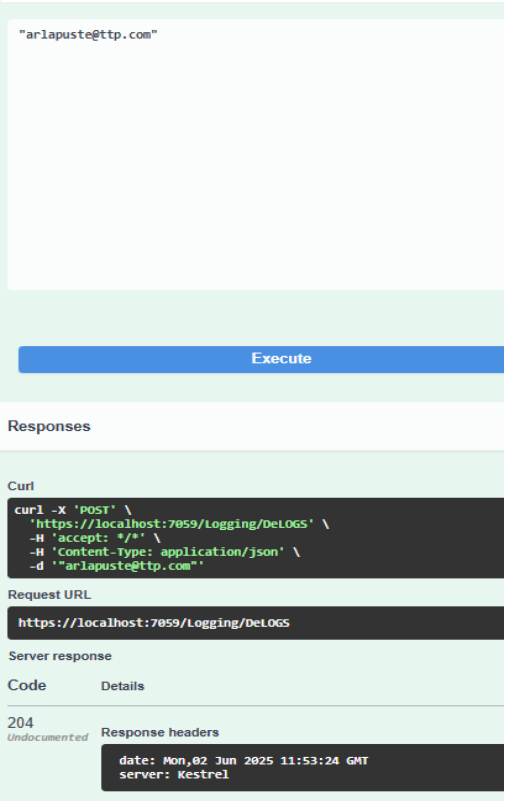


Figura 25.

4.3 Message

4.3.1 SendMessage

Pentru simplificarea testului, va fi utilizată aplicația wen unde unul arlapuste@ttp.com trimite mesaj către tcarmen@ttp.com

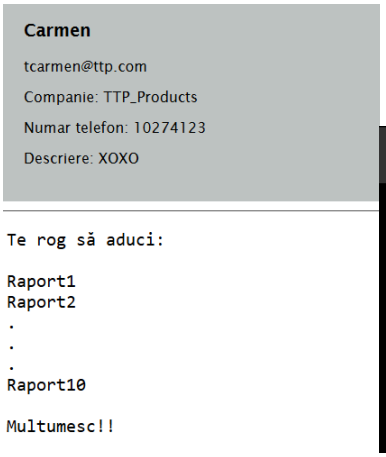


Figura 26.

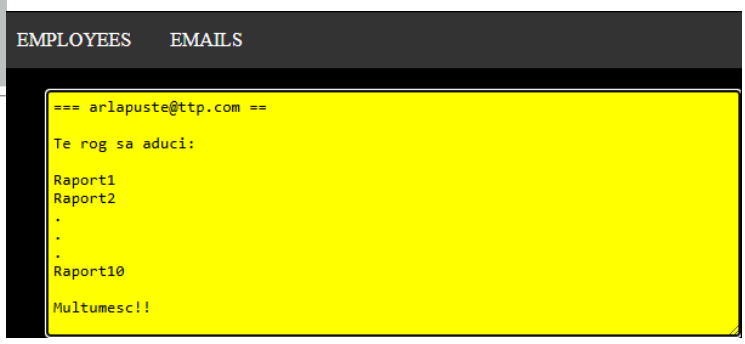


Figura 27.

Trimiterea unui mesaj utilizand Swagger când arlapuste@ttp.com nu e online. Utilizatorul tcarmen@ttp.com nu a primit mesajul.

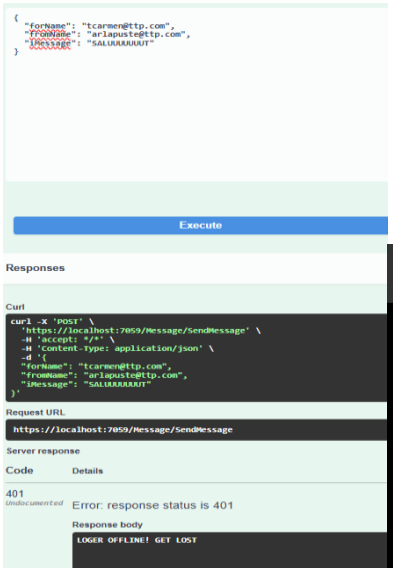


Figura 28.

Figura 29.

4.3.2 GetMessages

Pentru cazul în care utilizatorul nu este online :

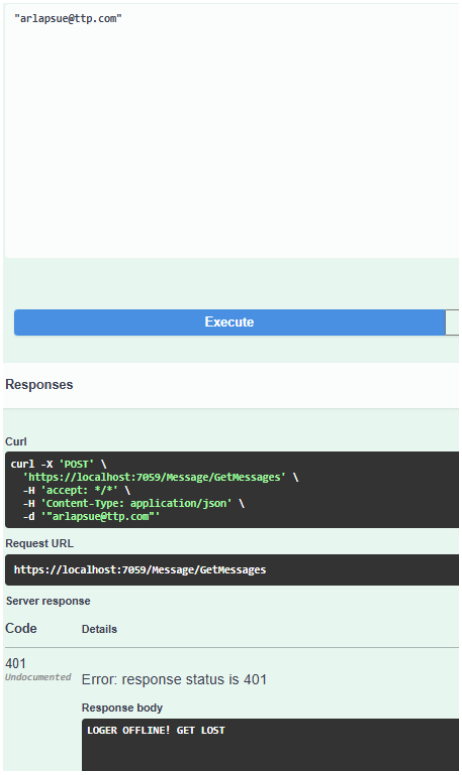


Figura 30.

4.4 Persons

4.4.1 GetThis

Apelarea metodei când utilizatorul nu este online. Când este online, se va returna lista comtactelor.

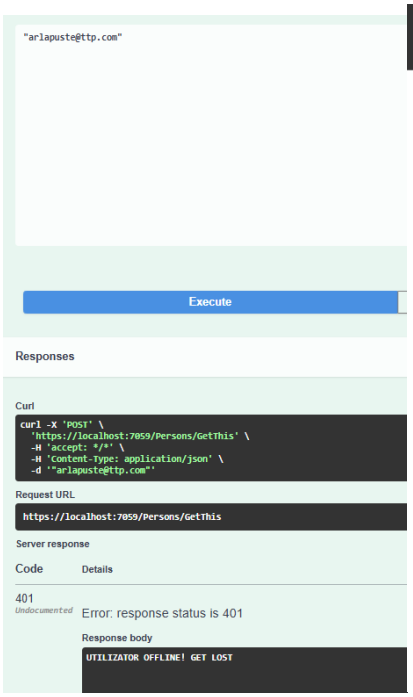


Figura 31.

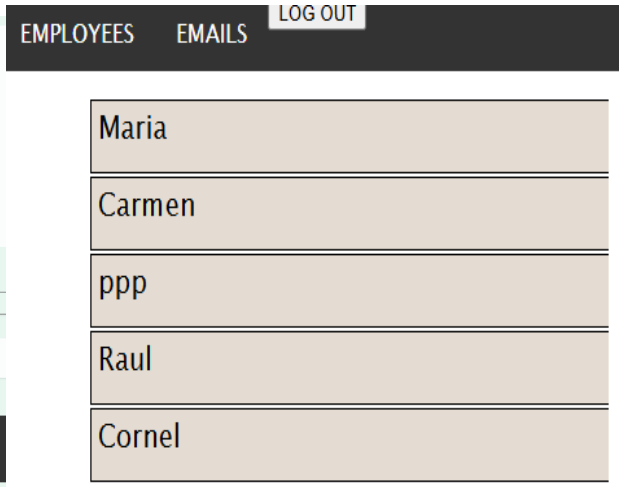


Figura 32.

4.4.2 AddPerson

Adaugarea unui contact nou.

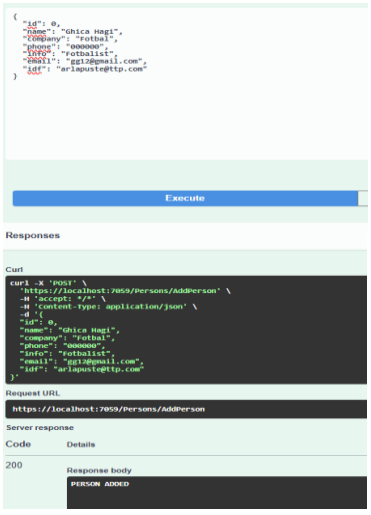


Figura 33.

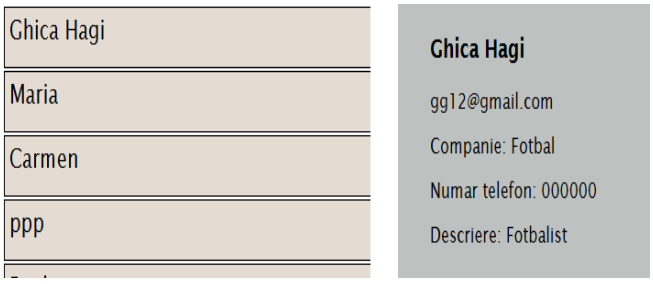
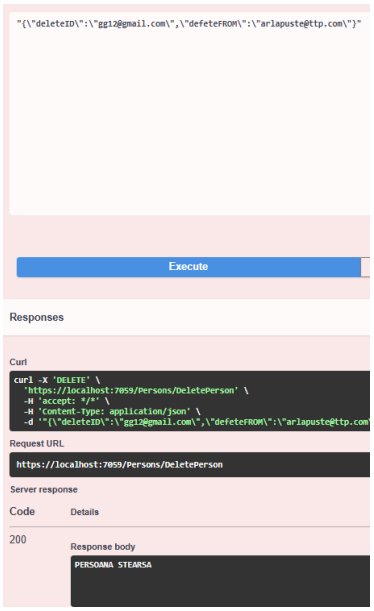


Figura 34.

4.4.3 DeletePerson

Ștergerea unui contact existent. Ștergerea unui contact inexistent. Parametrul este un șir JSON ce conține emailul contactului și al utilizatorului



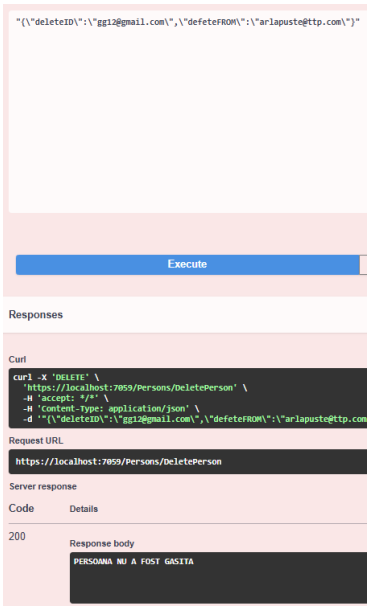


Figura 35. Figura 36.

4.5 Outside_Email

4.5.0 SendEmail

Utilizarea unui email valid pentru transmiterea mesajului și utilizator online.

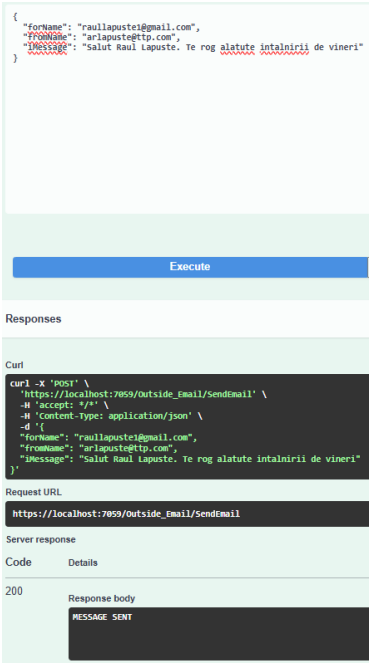


Figura 37.

Utilizarea unei adrese inexistente de mail. Mesajul serviciului către utilizator că mesajul nu a putut fi trimis la adresa introdusă.

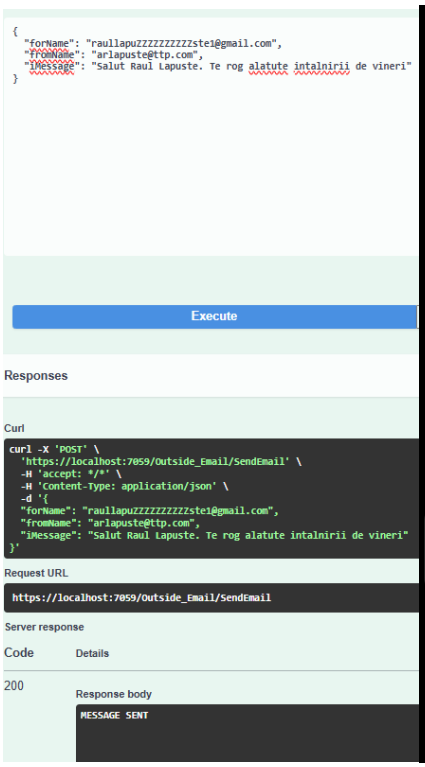


Figura 38.

Figura 39.

Utilizarea unei adrese cu domeniu invalid.

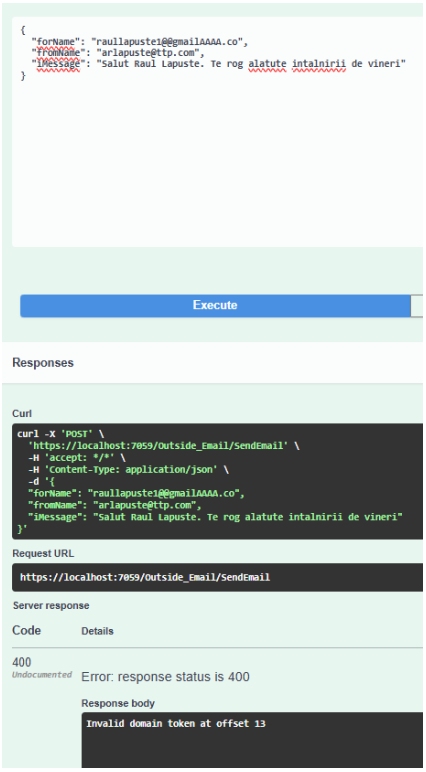


Figura 40.

4.6 Admin

4.6.1 GET_LOGERS

Metoda nu are nevoie de parametri, aceasta returand fie rezultatul, fie mesajul în cazul apariției unei erori

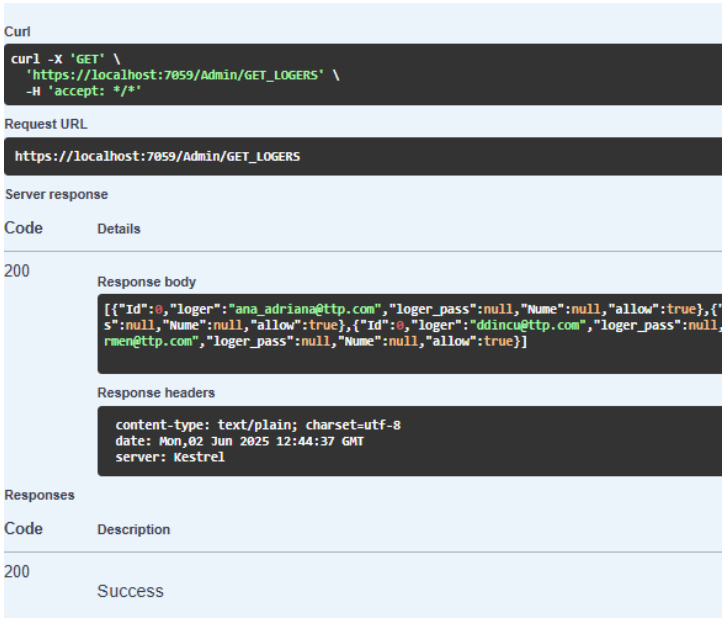


Figura 41.

4.6.2 Check_LOGERS

Verificarea utilizatorului online și offline prin serviciul ”Admin”

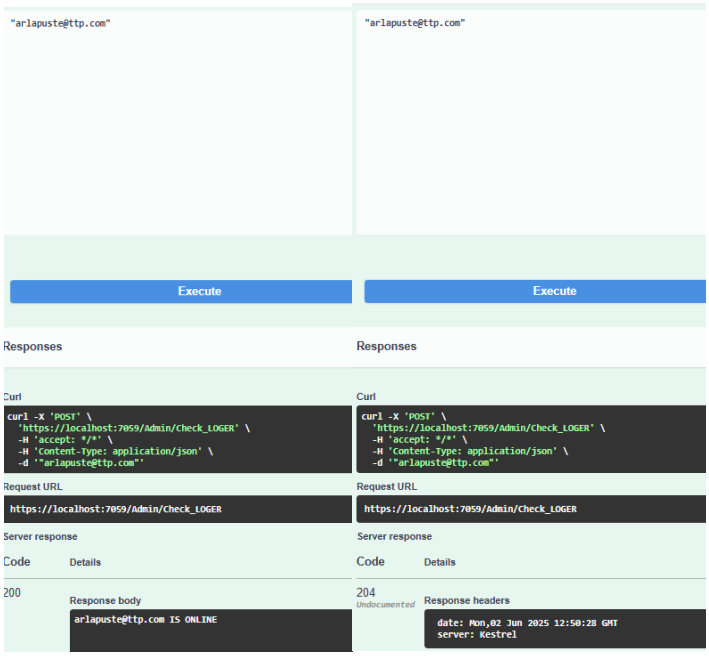


Figura 42. Figura 43.

4.6.3 Access_CHANGER

În figura următoare putem observa statusul fiecarui utilizator direct din baza de date. Utilizatorul arlapuste@ttp.com fiind trecut ”True”.

loger	loger_pass	Nume	allow
ana_adriana@ttp.com	qwert12	Ana Adriana	1
arlapuste@ttp.com	rain	Andrei Lapuste	1
ddincu@ttp.com	turcu	Daniel Dincu	1
tcarmen@ttp.com	dog	Tanase Carmen	1

Figura 44.

Apelarea metodei ”Access_CHANGER” inverseaza automat starea fie de adevarat sau false a parametrului ”allow”.

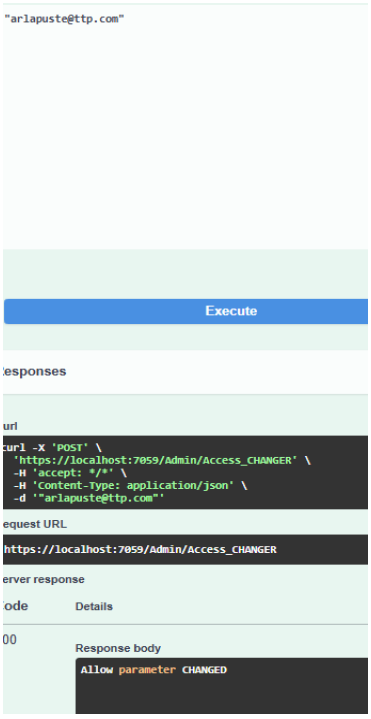


Figura 45.

idLoger	loger	loger_pass	Nume	allow
1	ana_adriana@ttp.com	qwert12	Ana Adriana	1
3	arlapuste@ttp.com	rain	Andrei Lapuste	0
4	ddincu@ttp.com	turcu	Daniel Dincu	1
5	tcarmen@ttp.com	dog	Tanase Carmen	1

Figura 46.

4.6.4 Send_Link_Password_Changer

Apelarea metodei are nevoie să primească pentru parametru două adrese de email valide raullapuste1@gmail.com_arlapuste@ttp.com. Rezultat va fi apariția emailului la prima adresă, crearea fișierului Html după identificatul unic și salvarea adresei utilizator în memoria Cache după cheia Guid.

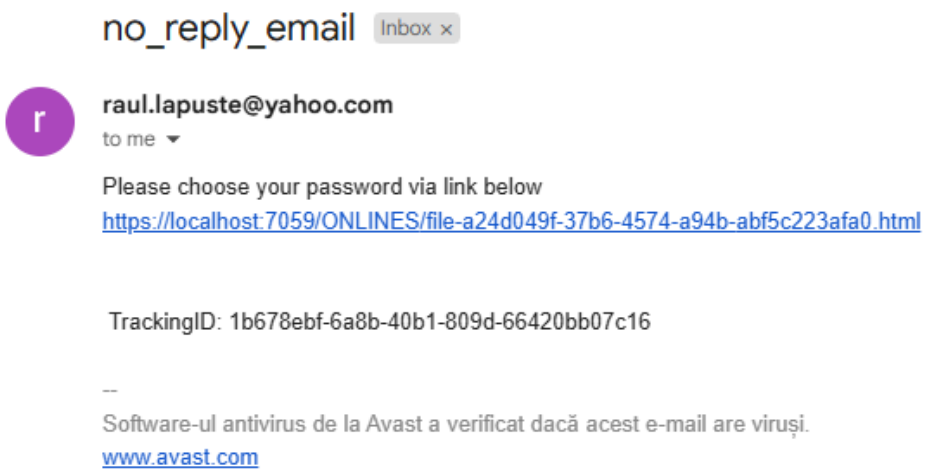


Figura 47.

Figura 48.

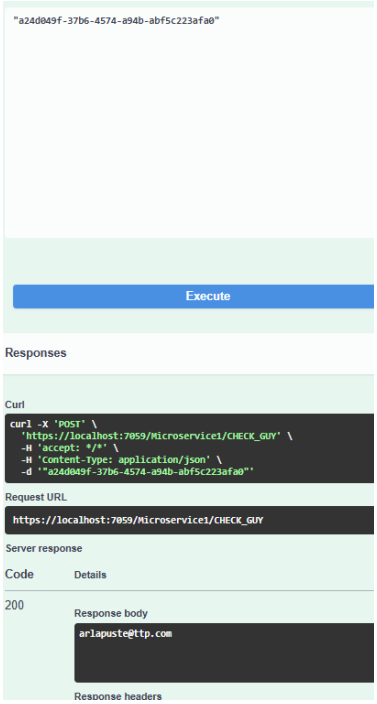


Figura 49.

Linkul oferă acces la formularul de schimbare al parolei.

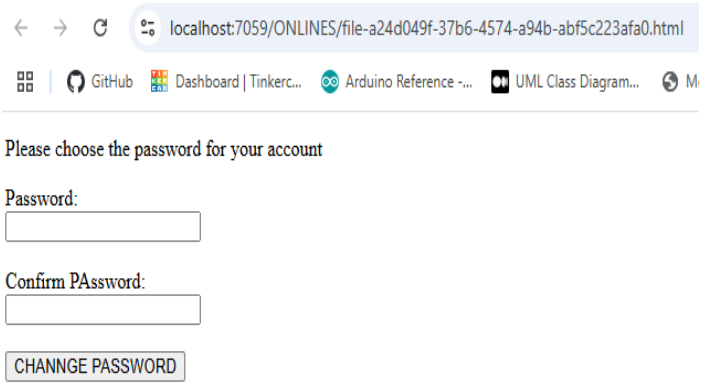


Figura 50.

Schimbarea cu succes a parolei va duce la ștergerea fisierului cât și la eliminarea obiectului din memoria Cache.

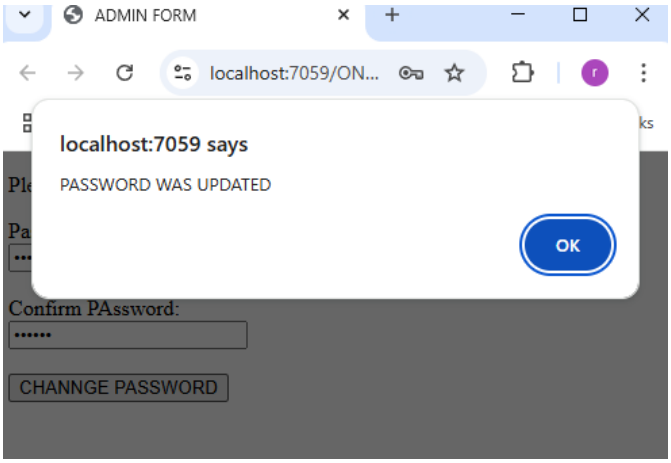


Figura 51.

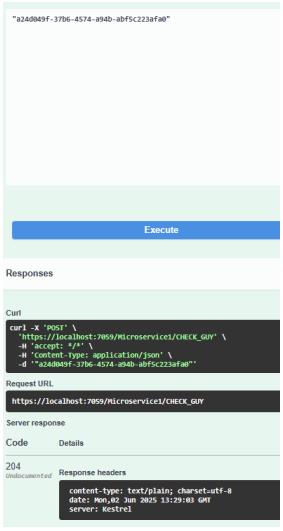


Figura 52.

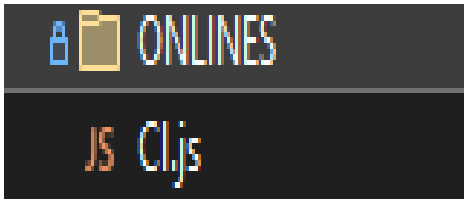


Figura 53.

Utilizatorul are la dispoziție 3 ore să aleagă o nouă parolă. În cazul când perioada depășește termenul limită, sesiunea va expira. Eliminarea se va face prin serviciul ”Microservice1”, motoda DeLOGS. Se va trimite un formular nou.

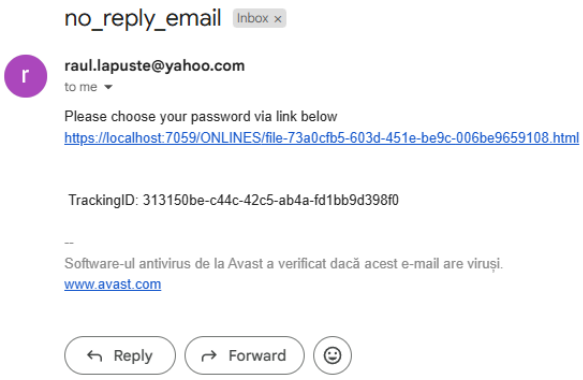


Figura 54.

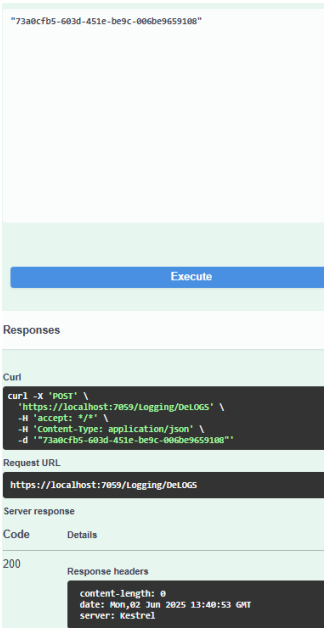


Figura 55.

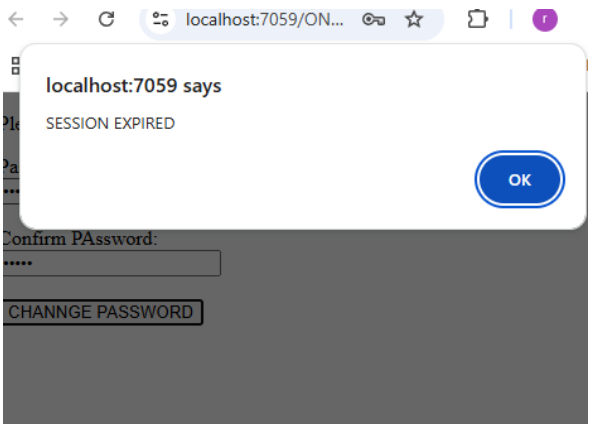


Figura 56.