UNIVERSITATEA "TITU MAIORESCU" DIN BUCUREȘTI

FACULTATEA DE INFORMATICĂ

LUCRARE DE LICENȚĂ

IMPLEMENTAREA UNEI APLICAȚII MULTIPLATFORMĂ UTILIZÂND O SINGURĂ SOLUȚIE

COORDONATOR ȘTIINȚIFIC:

Conf. univ. dr. ing. Pîrnău Mironela

ABSOLVENT:

Chanchian M. Armin Andrei

Chanchian Armin Andrei Digitally signed by Chanchian Armin Andrei Date: 2024.06.14 15:45:51 +03'00'

SESIUNEA IUNIE

2024

Cuprins

1 Introducere 3

2 Fundamentele teoretice și tehnologice 5

2.1 Noțiuni teoretice elementare în mediul petiționării 5

2.2 Tehnologiile utilizate în sistem 8

2.2.1 Prezentarea arhitecturii Qt Framework 8

2.2.2 Utilizare Qt Quick si QML 14

2.2.3 Sistemul de gestiune al bazelor de date relaționale PostgreSQL 18

3 Analiza și proiectarea sistemului 21

3.1 Configurarea mediului de dezvoltare Qt 21

3.2 Arhitectura aplicației 25

3.2.1 Implementarea modulului front-end cu ajutorul Qt Quick și QML 25

3.2.2 Implementarea modulului back-end prin intermediul Qt Core și C++ 37

3.2.3 Nivelul de acces la date folosind SGBD-ul relațional PostgreSQL 44

3.3 Analiza sistemului – model analist 50

4 Evaluarea și testarea 52

4.1 Testarea și depanarea implementării 52

4.2 Performanța și scalabilitatea 54

4.3 Analiza securității 54

5 Concluzii 56

6 Bibliografie 57

1 Introducere

În epoca post-decembristă, societatea a cunoscut o creștere semnificativă a conștiinței de sine, observându-se o dezvoltare exponențială. Cetățenii au îmbrățișat principii cu adevărat democratice, făcând din concepte și valori precum libertatea, justiția, egalitatea, diversitatea și solidaritatea un obiectiv comun. Lipsa unei soluții civice concrete în ceea ce privește accesibilitatea facilă a spiritul dreptului constituțional de petiționare, a condus la conceperea unei aplicații tip portal modular integrat de sesizări. Astfel, acest proiect, denumit în continuare, "Raport Civic" are ca scop crearea unei platforme prietenoase cu cetățeanul care dorește să își simplifice procesul de depunere petiții către autorități.

Aplicația este un sistem de tip portal, ce generează sesizări si petiții. În dezvoltarea ei, s-a folosit framework-ul Qt, alături de mediul de dezvoltare încorporat, anume Qt Creator. În partea de interfață a utilizatorului, este folosită tehnologia Qt Quick cu limbajul de marcare QML, fiind totodată complementat de funcții dezvoltate în limbajul dinamic JavaScript. În fundal, funcționalitățile sunt executate de limbajul de programare C++, alături de clasele Qt Core ce îl îmbunătățesc. Pentru conectarea sistemului de gestiune al bazei de date, PostgreSQL, s-a avut în vedere adaptarea cu sistemul cloud Aiven, ce are la bază infrastructura DigitalOcean. Comunicarea în cadrul sistemului, se stabilește cu managere și unelte de tip singleton.

Infrastructura aplicației are următoarele componente:

- Sistem de generare a petițiilor în funcție de șablonul selectat, fie în mod text sau extern
- Funcționalitate de afișare ghiduri civice
- Management de conturi, având funcționalitatea de înregistrare, autentificare și meniu cu profil utilizator
- Securizarea datelor, prin hashing

„RaportCivic” își propune să faciliteze generarea de modele sesizări într-un mod cât mai profesional și concis, urmând cu strictețe toate rigorile legii. Lucrarea de licență prezintă o imagine

de ansamblu asupra filosofiei acestei aplicații dar și modul de funcționare, conceptele sale precum și o privire tehnică a codului.

Tema „Implementarea unei aplicații multiplatformă utilizând o singură soluție” cu funcționalitatea de portal petiții a fost aleasă întrucât, în piața actuală, există prea puține soluții inteligente ce își propun să ajute cetățeanul să își expună părerea sau necesitățile. Din cercetările personale, absolut toate soluțiile disponibile, se concentrează asupra dispozitivelor portabile, de tip „smartphone”, neglijând piața desktop. Drept urmare, acest proiect ce folosește tehnologia Qt își propune să meargă cu un pas mai în față, și să creeze cu o implementare unică a soluției, cu un proiect capabil și compatibil pe toate sistemele de operare, inclusiv cu mici adaptări, a celor mobile.

O alta problematică o constituie faptul că unele instituții au o lipsă clară de soluții digitale concrete, cetățeanul trebuind să învețe pașii pe cont propriu. În contextul în care o autoritate pune la dispoziție o platformă de raportare a problemelor, de multe ori, acestea sunt neglijate întrucât nu sunt acoperite de legile în vigoare, ci totul stă în decizia instituției de a rezolva cererea, fără a avea neapărat repercusiuni legale.

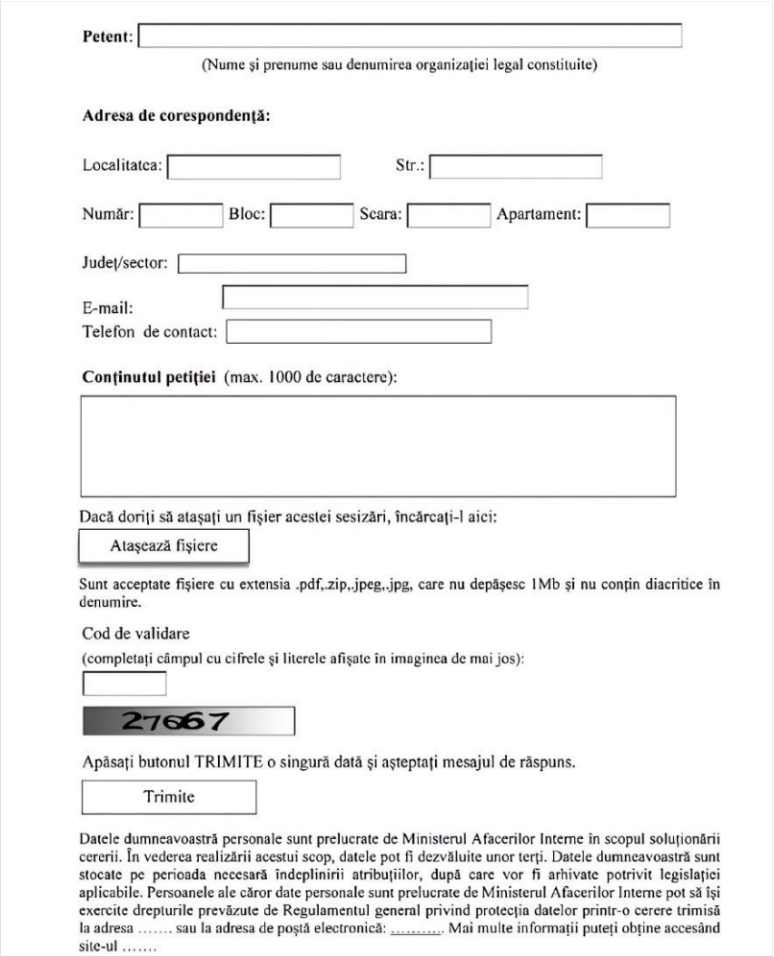
Sistemul are în vedere generarea de text, fie redirecționare automată a email-urilor, despre care se va discuta în capitolele ulterioare, care pot fi utilizate pentru a intra în contact cu autoritățile și instituțiile publice, ori private, combinând și adaptând eficiența unui portal integrat cu siguranța legii ce este posibilă prin transmiterea de mesaje prin contact de tip poștă electronică. Pe lângă aspectele anterior menționate, pentru promovarea unei astfel de inițiative, s-a avut în considerare și elemente necesare ce țin de eficiența și aspectul plăcut al interfeței.

Structura lucrării este formată inițial din partea teoretică, ce are în vedere introducerea teoriei legate de domeniul lucrării alese, aspecte civice și petiționare, continuând cu examinarea de suprafață a tehnologiilor alese în implementarea aplicației iar mai apoi făcându-se o imagine de ansamblu asupra părții tehnice de funcționalitate a soluției, cu dezvoltarea componentei de design, dar și cea ce ține de logica de lucru a aplicației. Totodată se prezintă elemente ce țin de analiza sistemului, prin diagrame ale schemei bazei de date dar și a ciclului de viață al aplicației. Se vor aduce în vedere și ideile ce țin de depanarea proiectului. În încheiere, vor fi subliniate ideile principale, rezultatele și oportunitățile de viitor ale proiectului

2 Fundamentele teoretice și tehnologice

2.1 Noțiuni teoretice elementare în mediul petiționării

Petițiile, adesea cunoscute și folosite în concept de „sesizări”, chiar notificări, sunt, instrumente legislative specifice ce permit cetățenilor să își comunice perspectivele personale, informații, problemele, întrebările, solicitările către autorități, instituții publice, organizațiilor sau altor entități reglementate de legile in vigoare, precum partidele politice.



Petent:

(Nume şi prenume sau denumirea organizaţiei legal constituite)

Adresa de corespondenţă:

Localitatea: Str.:

Număr: Bloc: Scara: Apartament:

Judeţ/sector:

E-mail:

Telefon de contact:

Conţinutul petiţiei (max. 1000 de caractere):

Dacă doriţi să ataşaţi un fişier acestei sesizări, încărcaţi-l aici:

Ataşează fişiere

Sunt acceptate fişiere cu extensia .pdf,.zip,.jpeg,.jpg, care nu depăşesc 1Mb şi nu conţin diacritice în denumire.

Cod de validare

(completaţi câmpul cu cifrele şi literele afişate în imaginea de mai jos):

27667

Apăsaţi butonul TRIMITE o singură dată şi aşteptaţi mesajul de răspuns.

Trimite

Datele dumneavoastră personale sunt prelucrate de Ministerul Afacerilor Interne în scopul soluţionării cererii. În vederea realizării acestui scop, datele pot fi dezvăluite unor terţi. Datele dumneavoastră sunt stocate pe perioada necesară îndeplinirii atribuţiilor, după care vor fi arhivate potrivit legislaţiei aplicabile. Persoanele ale căror date personale sunt prelucrate de Ministerul Afacerilor Interne pot să îşi exercite drepturile prevăzute de Regulamentul general privind protecţia datelor printr-o cerere trimisă la adresa ……. sau la adresa de poştă electronică: ……. Mai multe informaţii puteţi obţine accesând site-ul …….

Figura 1. Formular exemplu pentru petiții on-line[1]

[1] https://lege5.ro/Gratuit/gm3denzqhezq/model-formular-pentru-petitii-on-line-ordin-33-2020?dp=gmytcnbsgmytamq

Procesul de petiționare se poate defini, ca aspect general, prin conceptul de cerere scrisă, fie electronică, o plângere, o sesizare sau o propunere care poate să fie depusă ori de către un cetățean sau de pildă, ONG-uri la diverse instituții, autorități răspunzătoare. Lista este, și nu se limitează la diverse departamente ce țin de autoritatea guvernamentală, chiar companii cu participații sau în deplină subordonare a statului, fie autorități la nivel local, fie cele de tip central.

În ceea ce privește petițiile, acestea funcționează după un set de reguli ce trebuie urmate, adică, în general, dacă este în format scris, pe foaie, să aibă în subsol o semnătură și să includă datele de contact și de stabilire a identității trimițătorului. Ca o notă, serviciile ce coordonează departamentele ce țin de comunicare, pot să respingă acele cereri depuse fără datele anterior menționate. Instituțiile conform cu legea actuală ar trebui să ofere un răspuns la petiție în termen de cel mult 30 de zile trecute, întrucât dacă intervin aspecte tehnice fie ce țin de complexitate, se poate lărgi timpul permis de la cele 15 zile implicite.

Dacă cetățeanul nu cunoaște sau se află într-o dilemă cu privire la autoritatea ce ar trebui să se ocupe de soluționare celor transmise, autoritatea va transmite cu promptitudine petiția instituției competente responsabile și va oferi și o notificare de redirecționare cetățeanului. În acest caz, se va avea în vedere restabilirea timpului inițial de așteptare încât autoritatea notificată să poată soluționa.

O petiție, în ansamblul ei, este în fapt un document, fie în format digital, de pildă o cerere prin email sau un fișier deseori PDF, fie un act fizic, pe suport de foaie scris de mână sau în format tipărit, care are să fie semnat în numele unei persoane sau chiar a mai multora, ori chiar din partea unui grup ori organizație.

Deși anterior a fost definit conceptul de petiție, aceasta, în limbaj generic, este și o sesizare. O petiție are în spectru larg, utilitatea de punere în vedere a autorităților a unor situații generice, ce pot fi rezolvate și în termen mediu-lung. În contrast, sesizarea este o comunicare ce are în vedere o situație, fără îndoială, ce necesită rezolvare în termen scurt. Pentru o comparație succintă, se poate pune în imagine diferența între o cerere de amenajare peisagistică a unei grădini și o cerere legată de un drum care indică semne de surpare majore; primul exemplu se asociază, mai degrabă, în tematica unei petiționări iar cel de-al doilea exemplu, are forma unei sesizări.

Se are în vedere OG. 27/2002, care este printre altele, o reglementare în aspect legislativ a raportului de interacțiune dintre cetățeni și autoritate. De drept, aceasta este legea care stă la baza conceptului de petiție. Forma consolidată (actualizată) a ordonanței, definește în art. 1 de la alin. 1 cum, de altfel, scopul ordonanței este acela de a conferi o reglementare modului în care persoanele fizice își arogă dreptul de a înainta petiții în nume propriu autorităților și organizațiilor publice, precum și procesul de soluționare a acestora. Este important de menționat că autoritățile iau în considerare și petițiile formulate de anumite grupuri, care vor fi incluse ca solicitări în atributul adunării participante.

Complementar, mai există și Legea 544/2001[2] ce aduce în vedere faptul că cetățenilor statului li se oferă, în raport cu drepturile universale dar și cele garantate de către Constituția României, permisiunea deplină și neîngrădită la orice informație cu caracter public. Astfel, potrivit art. 2, lit. b, este luat în considerare cum că, în fapt, orice element ce ține de domeniul unei instituții publice se definește ca informație cu un caracter de interes public.

Totodată, se garantează dreptul lipsit de discriminare, oricărei persoane în uzul legii. Legea mai prevede și că pentru documente de tip clasificat, cele secrete sau care au o importanță majoră ori strategică pentru stat, nu se poate oferi accesul cetățenilor obișnuiți.

Într-un scenariu realistic, atunci când petițiile și sesizările sunt depuse, nu vor fi luate în considerare absolut toate prevederile legale, astfel încât unele instituții, de la caz la caz, vor accepta și date incomplete ori în mod complet anonime. Spre exemplu, la trimiterea unui e-mail înspre autoritatea Poliției Locale, cu o sesizare referitoare la un caz exemplificativ, de pildă o mașină abandonată, ce are o descriere minimală, precum localizarea și în plus, câteva imagini atașate, lipsa datelor de identificare nu va invalida în mod automat sesizarea transmisă. Alte instituții pot să aibă un regulament intern ce prevede invalidarea din oficiu în lipsa tuturor datelor necesare.

[2] https://legislatie.just.ro/Public/DetaliiDocument/31413

2.2 Tehnologiile utilizate în sistem

2.2.1 Prezentarea arhitecturii Qt Framework

Definire concept framework

Framework-urile sunt în fapt un ansamblu bine definit de unelte pregătite a fi puse la dispoziția dezvoltatorului pentru a-i fi de ajutor în dezvoltarea propriului sistem software și hardware. Prin această metodă, utilizatorul nu trebuie să lucreze și la fundamentarea propriilor unelte, reducând semnificativ din timpul alocat dezvoltării propriei aplicații.

Trecere prin istoricul Qt

Mediul Qt este unul foarte diversificat. Acesta își are începuturile încă din anul 1990 în Norvegia. Totul a luat naștere atunci când creatorii acestuia, anume Haavard Nord cât și Eirik Chambe-eng au dorit să contureze un sistem conceput și dedicat în mod special aplicării imagisticii cu ultrasunete. Acest sistem necesita o compatibilitate vastă, astfel că pentru a acoperi o gamă variată de sisteme de operare, aveau nevoie de o interfață grafică puternică care putea să le confere un mediu dedicat de instrumente avansate în programarea sistemului. În timp ce aceștia își petreceau timpul liber, chiar într-un parc, au avut o revelație, astfel că unul dintre fondatorii acestui framework, Haavard și-a exprimat ideea a pune bazele unui sistem de afișare grafic, cu specializarea orientării pe obiecte. Această arhitectura permitea afișarea interfețelor grafice utilizator, cu capabilitatea de a avea compatibilitate cu o largă plajă de dispozitive

Figura 2. Logo Qt 1996

Prima iterație, anterior menționată, avea să fie eliberată sub egida licenței FreeQt 1.0. Această tipologie de licență prezenta primul pas în popularitatea Qt Framework. Pe de o parte, permitea utilizatorilor să folosească Qt în mod lipsit de obligații financiare, iar pe de altă parte, stipula necesitatea transmiterii codului sursă integral a produsului final, în cazul în care s-a optat

pentru versiunea gratuită. În anul 1997, printre altele, Qt este integrat cu compatibilitate cu recunoscuta bibliotecă cu caracteristici grafice, OpenGL.

Trolltech devine parte a grupului Nokia din anul 2009, KDE constituie cel de-al treilea acord cu Qt, în urma transferului către compania Nokia și cu dorința de a însemna relicențierea Qt în licență LGPL. În acord cu figura 3, LGPL este o extensie a licenței GPL, aceasta având în plus o permisivitate mai mare a dezvoltatorului. Nu obligă în publicarea codului sursă decât dacă este cerut în mod expres. În același scop, se constituia necesitatea ca noi module produse să fie lansate sub licență tip GPL iar cei doritori puteau achiziționa o licență extra ca să le permită crearea de proiecte încapsulate în cod sursă închis. Un principiu de bază îl constituia trecerea și de la versiuni mai vechi, de pildă GPLv2 la GPLv3, ce evită conceptul de tivizare. Ca o ilustrare, în probabilitatea în care o companie ce folosește cod cu sursă deschisă GPLv2, poate să speculeze detaliile acesteia, chiar dacă codul folosit este publicat înapoi, putând să impună restricții utilizatorului final, prin care acesta din urmă nu are dreptul de a altera software-ul sistemului pe motiv că este nevoie de pachetul autorizat și semnat digital.

	Open Source												Enterprise		
	GPL						LGPL								
	Desktop		Embedded		Mobile		Desktop		Embedded		Mobile		Desktop	Embedded	Mobile
Subscription Cost (monthly or annually)	✗		✗		✗		✗		✗		✗		✓	✓	✓
License per seat (Developer)	✗		✗		✗		✗		✗		✗		✓	✓	✓
Per Device/App Royalty fee	✗		✗		✗		✗		✗		✗		✗	✓	✓
Dedicated Support from Qt	✗		✗		✗		✗		✗		✗		✓	✓	✓
Must open-source code	✓		✓		✓		✗		✗		✗		✗	✗	✗
Must provide source code	B2B	B2C	B2B	B2C	B2B	B2C	B2B	B2C	B2B	B2C	B2B	B2C	✗	✗	✗
	✓	✓	✓	✓	✓	✓	✗✓	✓	✗✓	✓	✗✓	✓			
Must provide installation instructions	B2B	B2C	B2B	B2C	B2B	B2C	B2B	B2C	B2B	B2C	B2B	B2C	✗	✗	✗
	✗	✓	✗	✓	✗	✓	✗✓	✓	✗✓	✓	✗✓	✓			

* Given the complexity of licensing and its legal implications, it is highly advisable to consult with legal professionals or Qt's licensing experts. They can provide precise advice based on your specific project, distribution model, and business goals.

Figura 3. Comparație licențiere Qt[3]

Versiunea 4.7 prezintă o noutate, Qt Quick acompaniat de QML. Tehnologii grafice, multi-modulare și „markup language" grafic.

[3] https://extenly.com/2024/02/07/qt-licensing-gpl-vs-lgpl-vs-commercial-choosing-the-right-fit-for-you/

Versiunea Qt 5 aducea cu aceasta suport pentru grafici accelerate bazate pe parte hardware, punându-se un accent extrem de important pe suportul JavaScript și QML. Se introduce suport Wayland (continuator spiritual al X11), suport limitat pentru sistemele de operare mobile precum cel iOS dar și Android, aspect nativ dat de controalele Qt Quick, noi extensii pentru rețelistică, module 3D avansate, printre care și cele de navigație (hărți), suport NFC dar și unirea mult module sub umbrela Qt Quick. În plus, anul 2020, aducea noua versiune Qt 6. Începând cu această iterație a framework-ului, se permite numai limbaj C++ de standard 2017, sistemul de construcție propriu Qbs este abandonat și trecut în domeniu public, fiind în continuare dezvoltat de comunitate, sistemul QMake, tot o producție proprie însă sintactic mult mai apropiată de CMake este pus în plan secund în favoarea sistemului ce l-a influențat, CMake care este adoptat ca sistem principal de testare și construcție a proiectului în prezent.

Figura 4. Brand (2023) actual al grupului Qt

Prezentarea arhitecturii Qt Framework

Pentru început, Qt este, așa cum a fost adus in vedere în subcapitolul anterior, un ansamblu de dezvoltare cu capacități multiplatformă. Poate fi proiectat pentru sisteme de tip mobile sau desktop însă, cu toate acestea, partea mobile fiind mai puțin avansată. Dacă se are în vedere configurarea sistemului build al proiectului, Qt poate compila aplicații, fără a modifica părți esențiale ale codului, pentru Linux (mediul inițial), Windows, chiar și Mac. De asemenea, are capacități foarte avansate în partea de embedded.

Observând ilustrarea ulterioară a figurii 5, se poate înțelege imagina de ansamblu a Qt Framework. Este un sistem foarte modular, având o gamă variată de API-uri, librării și clase C++ modularizate, inclusiv șabloane de cod care pot fi utilizate de utilizatori fără a fi neapărat modificate.

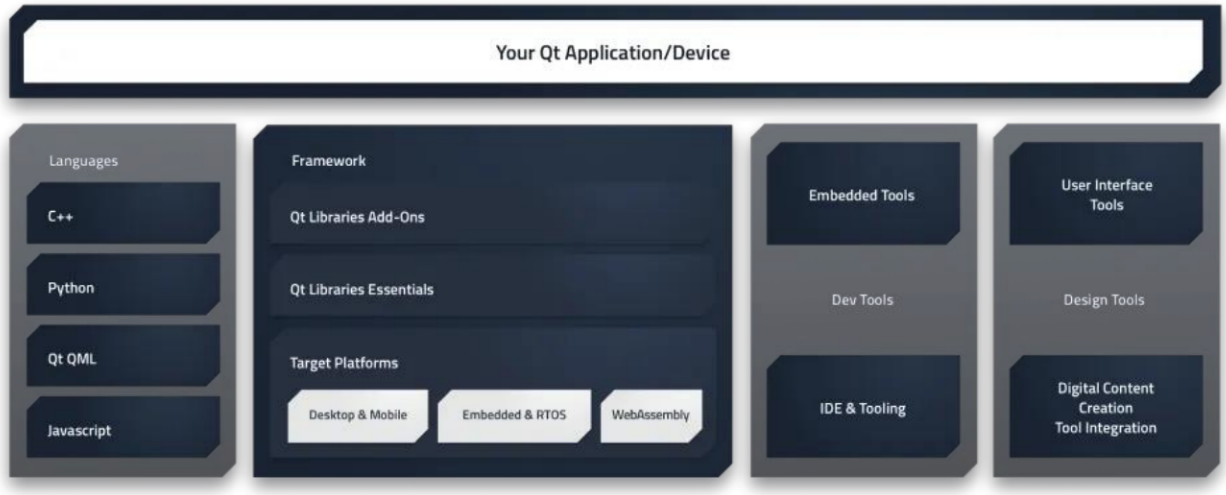


Figura 5. Componente Framework Qt

Mediul de dezvoltare Qt poate construi sisteme în mai multe domenii de programare. În această manieră, printre limbajele suportate se regăsesc: limbajul de programare C++, ce este orientat pe obiecte, Python, care fără îndoială este un contrastat de cele două atribute, cel al ineficienței în raport cu C++ dar și atributul pozitiv, cel al eficienței în programare, în moment ce prezintă o tipologie multiparadigmă (pe scurt, îmbină elemente ale limbajelor de programare procedurale dar și ale celor cu orientare pe obiecte).

Pe lângă partea de back-end, se distinge și prezența limbajului de scripting la nivel declarativ, denumit Qt Modeling Language, pe scurt, QML care face parte din Qt Quick, ce este un modul de unelte ce face parte din întregul ansamblu Qt Framework. Este influențat în mod direct de JavaScript, prescurtat JS, un limbaj larg răspândit, cu funcționare pe parte de navigatoare web, conform figurii 6, ce expune indexul de popularitate Tiobe. Se caracterizează ca un limbaj declarativ, la nivel abstract, având o abordare mixtă, sistem hibrid de interpretare și compilare, ce poate fi folosit și regăsit pe o multitudine de sisteme.

Jun 2024	Jun 2023	Change	Programming Language	Ratings	Change
1	1		Python	15.39%	+2.93%
2	3	^	C++	10.03%	-1.33%
3	2	v	C	9.23%	-3.14%
4	4		Java	8.40%	-2.88%
5	5		C#	6.65%	-0.06%
6	7	^	JavaScript	3.32%	+0.51%
7	14	^^	Go	1.93%	+0.93%
8	9	^	SQL	1.75%	+0.28%
9	6	v	Visual Basic	1.66%	-1.67%
10	15	^^	Fortran	1.53%	+0.53%

Figura 6. Indicele Tiobe[4]

Lucrarea teoretică prezentă cât și aplicația practică se concentrează pe iterația a 6-a a framework-ului. În categoria modulelor Qt (fig. 7) se regăsesc:

- Qt Essentials; modulele care constituie fundamentul Qt în totalitatea platformelor sale. Se regăsesc pe toate platformele compatibile.
    - Qt Core: modulul care stă la baza Qt, cu funcții și clase fără capabilitate grafică. Este folosit de restul modulelor. Aduce un strat de îmbunătățiri și unelte limbajului C++, printre care se poate enumera terminologia de semnale și slot-uri (în mod original definit ca „signals and slots”).
    - Qt D-BUS: clase care facilitează utilizarea protocoluluui D-Bus, ce reprezintă un mod de comunicare între procesele care rulează simultan în același sistem, în mod uzual pe Linux.
    - Qt GUI: clase ce permit elementele de utilizator grafice, inclusiv pe OpenGL și Vulkan, imagini, text și chiar font.
    - Qt Network: clase cu suport predefinit pentru rețelistică.
    - Qt QML: suport de dezvoltare aplicații dar și librării pentru QML și JS. Conține rețeaua de limbaj programare QML.
    - Qt Quick: set de librării standard cu scopul dezvoltării aplicațiilor având elemente predefinite QML.

[4] https://www.tiobe.com/tiobe-index/

- Qt Quick Controls: elemente de stilizare, precum butoane, popup-uri, meniuri, input-uri, indicatoare printre altele, a aplicației utilizând tipuri QML.
- Qt Quick Dialogs: modul ce permite dezvoltarea și interacțiunea cu dialoguri de sistem constituite de QML.
- Qt Quick Layouts: tipuri QML ce definesc pozitia în pagina a obiectelor. Se remarcă așezarea pe coloană (column), rând (row), în tip grilă (grid), stack.
- Qt Quick Test: un pachet de teste pentru QML sub formă de funcții JS.
- Qt Test: clase destinate testării aplicațiilor dar și a librăriilor Qt.
- Qt Widgets: o serie de componente pentru interfața utilizator. Este antagonistul Qt Quick. Destinat aplicațiilor ce nu doresc să folosească elemente grafice animate și sunt destinate dezvoltării în formă C++ pură.

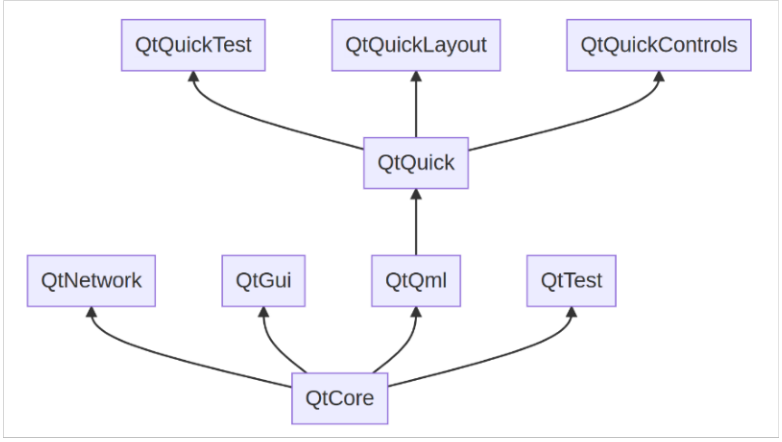


Figura 7. Ierarhie funcționare module Qt Essentials

- Qt Addons; module suplimentare cu funcții mai degrabă specific care se regăsesc pe anumite platforme compatibile. Așadar, conform Jürgen Bocklage-Ryannel et al.(2021), se disting următoarele:
  - „Rețelistică: Qt Bluetooth, Qt Network Authorization
  - Componente UI: Qt Quick 3D, Qt Quick Timeline, Qt Charts, Qt Data Visualization, Qt Lottie Animation, Qt Virtual Keyboard
  - Grafică: Qt 3D, Qt Image Formats, Qt OpenGL, Qt Shader Tools, Qt SVG, Qt Wayland Compositor

- Unelte: Qt 5 Core Compatibility APIs, Qt Concurrent, Qt Help, Qt Print Support, Qt Quick Widgets, Qt SCXML, Qt SQL, Qt State Machine, Qt UI Tools, Qt XML."

2.2.2 Utilizare Qt Quick si QML

Qt Quick este un termen generic pentru tehnologia folosită în interfața Qt. În mare parte, în conformitate și cu figura 8, aceasta acoperă următoarele limbaje:

- QML – ce este limbajul de tip marcaj al Qt
- JavaScript – limbajul de programare dinamic
- C++ – limbajul de bază și funcționalitățile îmbunătățite în Qt

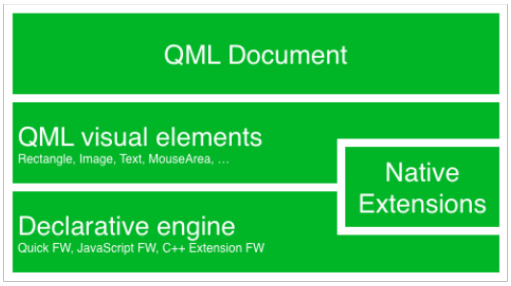


Figura 8. Diagramă Qt Quick

QML este un limbaj specializat, utilizat în crearea de elemente ce vor fi afișate pe ecran. Elementele pot fi, mai degrabă, descrise din punctul de vedere al așezării, relaționării și al aspectului. QML poate fi comparat în modul lucrului, cu HTML. Utilizează etichete numite tipuri (tr. „types"), care sunt cuprinse între acolade. Permite dezvoltatorilor crearea în cel mai eficient mod a aplicațiilor. De asemenea, o caracteristică importantă pentru dinamizarea interfeței grafice, este posibilitatea adaptării cu cod de tip JavaScript. QML-ul suportă și îmbunătățirea prin comunicarea cu codul C++ specializat al Qt.

În mod obișnuit, se pot folosi QML și JavaScript pentru partea front-end iar pentru back-end se va folosi C++, care comunică în mod direct cu interfața sistemului și întreprinde toate funcțiile complexe. Astfel, se va constitui o departajare directă în ceea ce reprezintă partea cu rol de conținut al design-ului și cea de dezvoltare funcțională.

Partea declarativă a interfeței utilizator, este denumită în mod succint, front-end iar părțile ce compun funcțiile din spatele acestor interfețe, sunt cunoscute adesea drept back-end. Prin acest aspect, se permite discrepanța între partea interfeței și cea funcțională, de calcul a aplicației.

Pentru exemplificarea situației, Jürgen Bocklage-Ryannel et al. (2021), în ghidul Qt prezintă o abordare simplificată. Pentru conturarea unei interfețe făcând uz de Qt Quick, se va defini un fișier de tip .qml. Dat fiind tipul limbajului, cel de marcare (tr. „markup language”), un fișier QML este necesar să aibă numai un anume tip rădăcină. În această situație, se va impune un exemplu în figura 9, ce folosește tipul „Item”, ce moștenește lungimea (tr. „height”) și lățimea (tr. „width”) imaginii de fundal.

```
import QtQuick

Image {
    id: root
    source: "images/background.png"
}
```

Figura 9. Exemplu cod 1

Din moment ce QML nu impune un anumit tip inițial, s-a ales tipul „Item” la care s-a stabilit proprietatea imaginii de fundal „background.png” ca rădăcină.

Figura 10. Imagine background.png

De notat: fiecare tip este asociat cu proprietăți specifice. De pildă, un tip „Image” are proprietăți precum cele menționate anterior, printre care sursa fișierului dar și mărimea pe lățime și lungime care reprezintă numărul de pixeli. În general, mărimea tipului „Image” este dedus din mărimea imaginii. Totodată, tipurile de bază se găsesc în modulul „QtQuick”.

„id” are rolul de identificator unic al unui element din fișierul qml. Nu poate fi alterat dacă a fost stabilit. În același timp, fișierul nu poate fi modificat dacă este în uz. Următoarele figuri reprezintă elemente de fundal în imagine:

Figura 11. Băț

Figura 12. Roată

Scopul este de a poziționa tija orientat pe parte orizontală, în centrul fundalului, în partea inferioară. Roata va fi poziționată exact în partea centrală, așa cum este expus în figura 13.

```
Image {
    id: root
    ...
    Image {
        id: bat
        anchors.horizontalCenter: parent.horizontalCenter
        anchors.bottom: parent.bottom
        source: "img/bat.png"
    }

    Image {
        id: roata
        anchors.centerIn: parent
        source: "img/roata.png"
    }
    ...
}
```

Figura 13. Exemplu cod 2

O proprietate complexă cunoscută sub numele de ancoră (tr. „anchor”) este utilizată pentru poziționarea roții în centru. Ancorarea facilitează definirea relațiilor geometrice între obiectele de tip cap și moștenitor. Factual, se are în vedere amplasarea în centrul altui tip „anchors.centerIn: parent”. Este normal ca mai multe ancore să fie în reciprocitate atunci când există uz comun. Este greșit să se ancoreze partea dreaptă a unui tip în partea inferioară a unui alt tip. În acest caz, nu este nevoie decât de o simplă ancoră.

Ocazional, poate fi necesar să existe diverse ajustări minore, cum ar fi decalarea ușoară a unui tip (ex. „anchors.horizontalCenterOffset”, „anchors.verticalCenterOffset”).

Un concept esențial al unui limbaj declarativ este ilustrat prin includerea unei imagini ca descendent al tipului rădăcină „Image”.

Se are în vedere dinamizarea elementelor. Principiul ar fi mișcarea la apăsarea cu mouse-ul în cadru. „MouseArea” va fi folosit pentru interactivitate. Totodată, trebuie să înglobeze tot tipul „root”, ca în figura 14.

```
Image {
    id: root
    ...
    MouseArea {
        anchors.fill: parent
        onClicked: roata.rotation += 90
    }
    ...
}
```

Figura 14. Exemplu cod 3

Semnalele sunt generate de zona mouse-ului atunci când utilizatorul selectează în limita cadrului ce îl conține. Prin intermediul suprascrierii funcției „onClicked”, este posibil să se stabilească o conexiune la acest semnal. La momentul de conectare al semnalului, funcția căreia îi corespunde are să fie invocată ori de câte ori semnalul este emis. În acest caz particular, se declară faptul că tipul cu „id” de „roata”, se va roti la 90 de grade în cazul acțiunii cu mouse-ul. Această metodă este aplicabilă tuturor semnalelor, având convenția „on” și „NumeSemnal”. Toate proprietățile transmit un semnal de fiecare dată când există o schimbare în cadrul valorilor. În acest caz, schema este „on${proprietate}Changed”. De pildă, se poate monitoriza modificare unei proprietăți precum „width” folosind metoda „onWidthChanged: print(width)”.

Acum, roata se va putea roti prin acțiunea utilizatorului, însă rotația este una singulară, nu perpetuă. Se poate obține o mișcare lină prin intermediul animațiilor. O animație trasează procesul prin care o proprietate are parte de schimbare într-un interval de timp specificat. În aceeași idee, se poate utiliza proprietatea „Behavior” a tipului „Animation”. Proprietatea „Behavior” stabilește o animație pentru o proprietate specifică drept răspuns la schimbările survenite proprietății, conform figura 15. Așadar, la fiecare modificare, animația se va încărca.

```
Image {
    id: root
    Image {
        id: roata
        Behavior on rotation {
            NumberAnimation {
                duration: 250
            }
        }
    }
}
```

Figura 15. Exemplu cod 4

În prezent, proprietatea de rotație a roții are să fie animată prin intermediul „NumberAnimation” cu un timp de 250 milisecunde, la fiecare modificare. Astfel, va exista o tranziție fină în rotire, precum în figura 16.

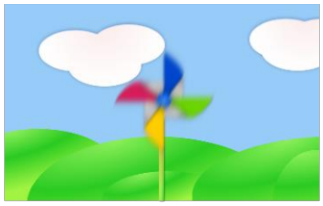


Figura 16. Cadru final exemplificare cod scenă

2.2.3 Sistemul de gestiune al bazelor de date relaționale PostgreSQL

În acord cu Luca Ferrari & Enrico Pirozzi (2023), „PostgreSQL este o bine-cunoscută bază de date relațională cu cod sursă deschis, iar mottoul său afirmă ceea ce proiectul intenționează să fie, cea mai avansată bază de date open-source. Principalele calități care atrag mase de utilizatori noi în fiecare an și mențin entuziasmul utilizatorilor actuali sunt stabilitatea, scalabilitatea și siguranța sa, precum și toate caracteristicile pe care trebuie să le ofere un sistem de gestionare a bazelor de date la nivel de întreprindere”.

Figura 17. Logo PostgreSQL

Baza de date relațională este un set bine structurat de informații care aranjează datele în relații prestabilite. Datele sunt păstrate în mod normal, în tabele, denumite generic și sub denumirea de relații, care sunt formate din coloane și rânduri. Acest mod de structurare facilitează vizualizarea și înțelegerea interconexiunilor dintre diferitele structuri de date. Relațiile reprezintă o asociere rațională între tabele distincte, pe baza interacțiunii dintre aceste tabele.

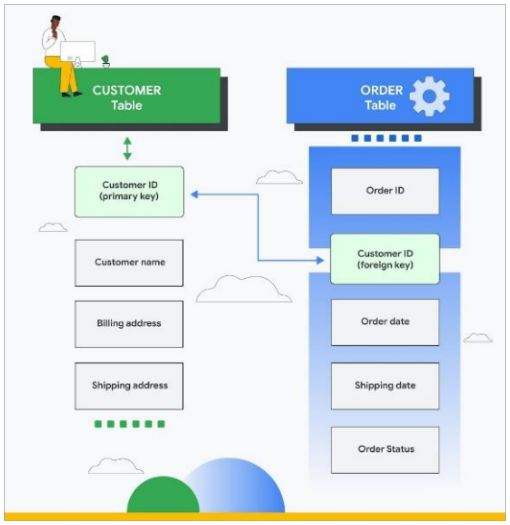
În același context, Google (f.a.) detaliază: „Dezvoltat de E.F. Codd de la IBM în anii 1970, modelul bazei de date relaționale permite ca orice tabel să fie legat de un alt tabel folosind un

atribuit comun. În loc să folosească structuri ierarhice pentru a organiza datele, Codd a propus trecerea la utilizarea unui model de date în care datele sunt stocate, accesate și legate în tabele fără a reorganiza tabelele care le conțin”.

Baza de date de acest tip, relațional, este un fel de catalog cu foi de calcul bine organizate, ce sprijină gestiunea, administrarea și evidența datelor. Datele sunt stocate în diverse tabele proiectate și definite sub forma unor de coloane (atribute) și rânduri (înregistrări).

Într-o bază de date, fiecare informație, cunoscută drept înregistrare conține un o valoare a unui anumit tip de date și în cele din urmă, acestea sunt identificate, pe tiologia datei, de către atribute.

Figura 18. Model bază de date relațional



Tot Luca Ferrari & Enrico Pirozzi (2023) explică: „În timp ce PostgreSQL este o bază de date relațională, ecosistemul său a crescut de-a lungul timpului, oferind o platformă bogată în extensii, instrumente și limbaje interconectate de comunități răspândite în întreaga lume. De asemenea, acesta este deplin compatibil ACID și are o bază solidă în integritatea datelor și concurență.” Dat fiind natura acestui proiect, cu cod sursă la vedere, baza de date nu necesită eforturi financiare, fiind în mod complet gratuit și la disponibilitatea tuturor dezvoltatorilor. Licența este permisivă și orice persoană care dorește, poate contribui.

Parte importantă a capacității ce ține de partea relațională este componenta ACID, ce definește sistemul ca unul de tip tranzacțional. Parte din operațiunile care cuprind straturile tranzacționale, au proprietățile precum, atomicitatea, unde toate operațiunile, precum cele de citire, scriere, sunt încadrate ca un întreg, consecvența, unde se aduce o stabilitate în ceea înseamnă datele tabelare, care nu pot fi alterate fără acțiuni complete ce pot corupe înregistrările, izolarea, ce individualizează în grupuri tabelele, astfel orice activități ce accesează aceeași instanță, prin proces de concurență, ce evită producerea de conflicte dar și durabilitatea, ce asigură că datele ce sunt alterate sunt conservate în cea mai mare parte, chiar și în cazul unor incidente ce țin de partea tehnice.

De asemenea, PostgreSQL are un tip de limbaj procedural denumit PL/PgSQL, ce poate fi folosit ca o metodă de scripting, permițând bucăți de cod reutilizabile, printre care se enumeră funcții și proceduri. De asemenea, tehnologia procedurală acceptă declanșatorii „before” și „after”, vizualizări, vizualizări materializate, tabele partiționate, „warappere” de date străine, scheme multiple, coloane generate și așa mai departe.

De asemenea, Amazon AWS aduce în prim-plan câteva caracteristici: “Acesta reprezintă un SGBD obiectual-relațional (ORDMBS), ceea ce înseamnă că are capacități relaționale și un design orientat pe obiecte.

Utilizându-l, dezvoltatorii pot printre altele, să comunice cu serverele bazei de date folosind obiecte în codul lor, să definească tipuri de date personalizate complexe, să definească funcții care lucrează cu propriile tipuri de date și să stabilească moștenirea sau relațiile părinte-copil între tabele.”[5]

Prin urmare, PostgreSQL este un sistem relațional extrem de flexibil, ce permite adaptarea a numeroase conexiuni și tipologii intern, fără a fi nevoie să se folosească intervenienți externi, precum aplicația ce accesează conținutul. De asemenea, fiabilitatea este un aspect important iar performanța poate fi una foarte ridicată prin implementarea acțiunilor dorite.

[5] https://aws.amazon.com/rds/postgresql/what-is-postgresql/

3 Analiza și proiectarea sistemului

3.1 Configurarea mediului de dezvoltare Qt

Pentru instalarea kitului de dezvoltare software (tr. SDK) al Qt, trebuie descărcat instalatorul online de pe site-ul www.qt.io/download-qt-installer-oss. Acesta este pachetul destinat uzului gratuit, cu licență L/GPLv3, în funcție de caz. Utilizatorul va alege sistemul de operare compatibil. De asemenea, este disponibilă și o versiune cu funcționalitate offline. În urma descărcării pachetului, este necesar crearea unui cont de utilizator pe platforma Qt, pe care utilizatorul va trebui să îl autentifice în unealta Qt de mentenanță, în conformitate cu figura 19.

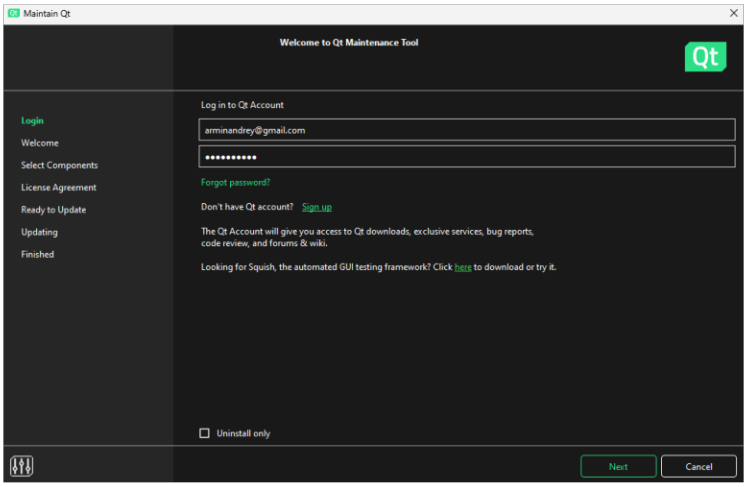


Figura 19. Autentificare și înregistrare cont

Proiectul „RaportCivic”, folosește următoarele componente (fig. 20):

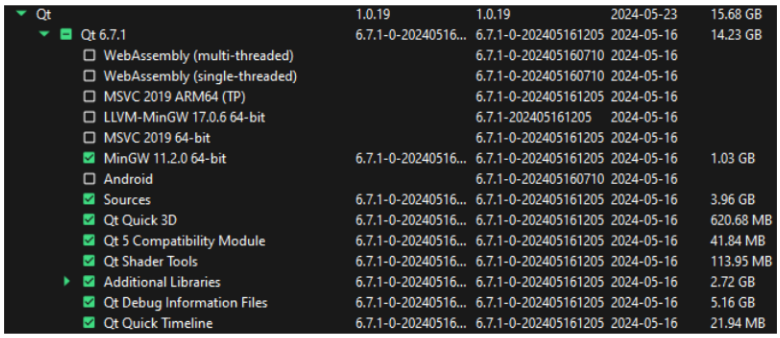


Figura 20. Componente Qt

În primul rând, pentru sistemul de operare în uz, a fost aleasă combinația dintre MSYS2 (instalat extern), MinGW-w64 și CMake. Berik I. Tuleuov & Ademi B. Ospanova (2024), constată că „MSYS este acronimul ce provine de la «Minimal SYStem» și este conceput pentru a oferind proces de construire în Windows a programelor și bibliotecilor care au fost inițial orientate pentru construirea în sisteme de tip Unix”.

MSYS are în conținut, o multitudine de pachete GNU, precum bash, grep, make, gzip, tar, gawk și wget. Un aspect important este că MSYS nu are în competență biblioteci sau compilatoare C/C++, el în fapt fiind o componentă a MinGW, adică Minimalist GNU for Windows. MinGW pune la dispoziție headere WinAPI, utilitare, biblioteci și arhivatoare, compilatoare, linkere pentru dezvoltarea de soluții și biblioteci native pentru sistemul de operare Windows.

CMake este o suită de instrumente ce pot să genereze sisteme de construire a proiectelor software. Este cu sursă deschisă și implicit, gratuit. De asemenea, poate să funcționeze pe majoritatea sistemelor de operare. CMake depinde de alte instrumente de sistem pentru a efectua compilarea propriu-zisă, crearea de legături și chiar alte sarcini. Acesta nu poate crea nimic în mod individual. Pe scurt, rolul este de a fi un manager de construire a proiectului ce leagă elementele necesare pentru rezultatul final.

Rafal Swidzinski (2022) prezintă în detaliu modul de funcționare și etapele (a se observa figura 22) ce sunt parcurse.

Astfel, CMake se distinge prin cele trei etape: configurarea, generarea cât și construirea.

1. Etapa de configurare:

În această parte, este necesar să se obțină detaliile proiectului, detalii care au fost înmagazinate într-un director, ce poartă numele arbore sursă. În continuare, este necesar să se pună la dispoziție un director de ieșire, dar și pentru etapa de generare, arborele de construcție:

„Pentru început, CMake creează un build tree gol și colectează toate detaliile despre mediul de lucru, cum ar fi, arhitectura, compilatoarele disponibile, linkerele și arhivatoarele. Suplimentar, verifică dacă un program simplu de testare poate fi compilat corect. În continuare, fișierul de configurare a proiectului, denumit „CMakeLists.txt” este analizat și executat. Proiectele de tip

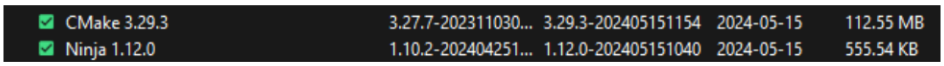
CMake utilizează limbajul de codare CMake. Acest fișier este minimul pentru un proiect CMake, alte fișiere sursă putând fi adăugate ulterior. CMakeLists informează CMake despre structura proiectului, ținte (tr. «targets») și dependențe (biblioteci și alte pachete). Pe parcursul acestui proces, CMake stochează detalii de sistem, configurații de proiect, jurnale și fișiere temporare în build tree pentru a fi utilizate ulterior”.

2. Etapa de generare:

Se va crea un sistem de compilare pentru mediul de lucru. Sistemul de lucru este implicit ca fiind un se de fișiere de configurare ce întâmpină alte instrumente, precum Makefiles și Ninja să construiască în mod corect proiectul:

„Ninja și Make sunt ambele „buildsystems”. Se pot furniza liste de fișiere și instrucțiuni de compilare sub formă de comenzi (ex. bash) și pot fi utilizate pentru compilarea software-ului. În mod implicit, acestea se adaptează la scenariu dat, putând să altereze prin recompilare doar fișierele care au fost modificate în primă instanță.”

În mod asemănător, CMake este o unealtă de generare însă aceasta se află în sensul ierarhiei la nivel mai înalt, fiind într-o relație de necesitate, CMake creând obiectele necesare pentru Make și Ninja.



CMake 3.29.3	3.27.7-202311030...	3.29.3-202405151154	2024-05-15	112.55 MB
Ninja 1.12.0	1.10.2-202404251...	1.12.0-202405151040	2024-05-15	555.54 KB

Figura 21. Componente de dezvoltare Qt

3. Ultima etapă din procesul CMake este reprezentată de etapa de construcție, astfel,

„pentru a produce elementele necesare în proiect, este nevoie să se ruleze instrumentul de build corespunzător. Acesta poate fi invocat în mod direct, prin intermediul unui IDE sau folosind comanda CMake. Instrumentele pentru build, la rândul lor, vor executa etape pentru a produce targets cu compilatoare, linkere, instrumente de analiză statică și dinamică cadre de testare, instrumente de raportare și așa mai departe”.

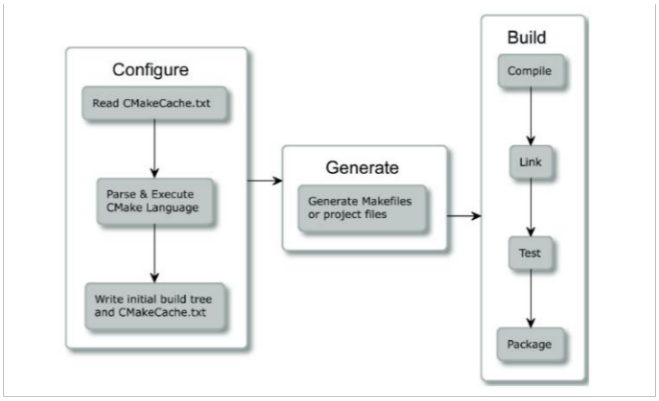


Figura 22. Etape CMake

Punctul forte al acestei tehnici constă în capacitatea de a genera „buildsystems” pentru diverse platforme, la nevoie, utilizând doar o configurare, ajutând conceptul de multiplatformă al proiectului.

Pe lângă acestea, în figura 23, sunt selectate și componente precum „Sources”, sursa de sine a Qt, ce permite legarea și compilarea la nevoie a altor componente din proiect sau aplicație, și chiar biblioteci suplimentare, cum se poate vedea în figura 36, ce pot avea diferite forme de licența.

Additional Libraries	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	2.72 GB
Active Qt	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	94.26 MB
Qt 3D	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	859.65 MB
Qt Charts	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	139.55 MB
Qt Connectivity	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	30.00 MB
Qt Data Visualization	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	71.72 MB
Qt Graphs (TP)	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	100.80 MB
Qt HTTP Server (TP)	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	8.68 MB
Qt Image Formats	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	11.28 MB
Qt Insight Tracker (TP)	6.7.1-0-20240516...	6.7.1-0-202405161403	2024-05-16	872.32 KB
Qt Language Server	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	50.54 MB
Qt Location (TP)	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	126.66 MB
Qt Lottie Animation	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	29.03 MB
Qt Multimedia	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	287.77 MB
Qt Network Authorization	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	9.61 MB
Qt PDF	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	10.86 MB
Qt Positioning	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	43.89 MB
Qt Protobuf and Qt GRPC (TP)	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	185.42 MB
Qt Quick 3D Physics	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	376.02 MB
Qt Quick Effect Maker	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	35.40 MB
Qt Remote Objects	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	40.32 MB
Qt Sensors	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	35.45 MB
Qt Serial Bus	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	33.00 MB
Qt Serial Port	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	4.32 MB
Qt Speech	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	13.79 MB
Qt State Machines	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	62.86 MB
Qt Virtual Keyboard	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	87.64 MB
Qt WebChannel	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	9.17 MB
Qt WebEngine	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	0.00 bytes
Qt WebSockets	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	12.95 MB
Qt WebView	6.7.1-0-20240516...	6.7.1-0-202405161205	2024-05-16	8.85 MB

Figura 23. Biblioteci suplimentare Qt

Instrumentul de dezvoltare integrat (tr. IDE) se numește Qt Creator. Qt Creator este opțional. Se poate opta pentru alte variante inclusiv VS Code cu extensiile necesare. Prin folosirea extensiilor, unele caracteristici nu vor fie disponibile în mod direct, cum ar fi completarea sintactică. În plus, acest tip de configurare implică o documentare extensivă în prealabil. Pentru

folosirea IDE-ului Qt Creator pus la dispoziție, trebuie selectate opțiune ce referă Qt Creator (fig. 24):

Qt Creator 13.0.2	13.0.2-0-2024060...	13.0.2-0-202406051...	2024-06-05	529.71 MB
Qt Creator 13.0.2 CDB Debugger Support	13.0.2-0-2024060...	13.0.2-0-202406051...	2024-06-05	106.75 MB
Debugging Tools for Windows	2024-02-26-0-20...	2024-02-26-0-2024...	2024-06-05	68.47 MB

Figura 24. Instalare IDE Qt Creator

Acesta este un instrument specializat ce servește dezvoltatorii să își construiască proiectele Qt, așa cum apare în figura 25. Acest mediu integrat este accesibil pe toate platformele sistemelor de operare. Adesea, poate fi folosit și pe dispozitive încorporate. O caracteristică importantă este constituită de eficiența în productivitate. Poate fi capabil de lucru cu cod C++, inclusiv compilarea sa dar totodată, poate rula interfețe de tip Qt Quick.

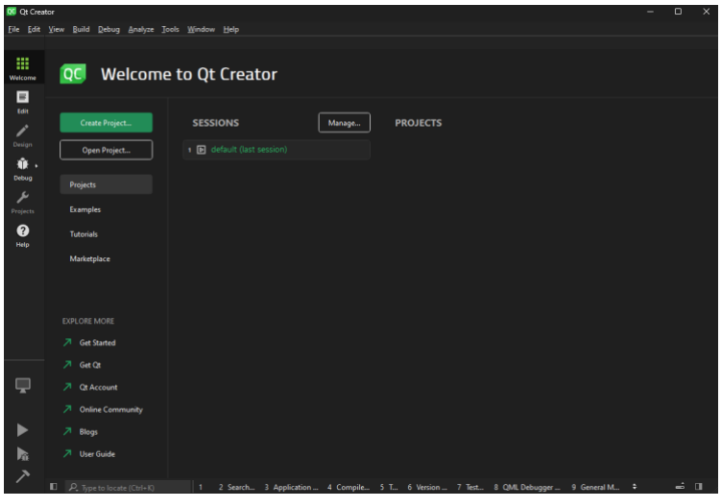


Figura 25. Pagină principală IDE Qt Creator

3.2 Arhitectura aplicației

3.2.1 Implementarea modulului front-end cu ajutorul Qt Quick și QML

Sursa proiectului în ceea ce privește componenta front-end este inițializată în fișierul CMakeLists.txt, conform cu figura 26, alături de restul asset-urilor ce definesc structura vizuală a aplicației.

```
set(QML_SOURCES
    qml/App.qml
    qml/board/ItemsBase.qml
    qml/board/ItemsMain.qml
    qml/board/SharedState.qml
    qml/pane/About.qml
    qml/pane/AccountContent.qml
    qml/pane/AccountLogin.qml
    qml/pane/AccountRegister.qml
    qml/pane/Crash.qml
    qml/pane/Licenses.qml
    qml/pane/PageLayer.qml
    qml/pane/PaneBackend.qml
    qml/screen/544Guide.qml
    qml/screen/CaseGuide.qml
    qml/screen/CivicalGuide.qml
    qml/screen/Home.qml
    qml/screen/ReportData.qml
    qml/screen/ReportDetails.qml
    qml/screen/ReportDisplay.qml
    qml/screen/ReportInput.qml
    qml/screen/ReportSelection.qml
    qml/screen/Settings.qml
    qml/screen/UserManagement.qml
)
```

Figura 26. Definire QML în CMakeLists

Asset-urile folosite se regăsesc in directorul „assets”. Acest director este amplasat in rădăcina definită ca „src”. Asset-urile vizuale sunt împărțite in diferite categorii, printre care „svg”, „pdf”, „img” și „icons”.

Un exemplu sugestiv poate fi directorul „img” ce înmagazinează imagini de tip fundal, iconițe și alte tipuri de butoane.

Figura 27. Asset-uri proiect

În aceeași manieră, este de notat faptul că toate resursele vizuale au fost comprimate în unelte precum pinga și TinyJPG. Unealta pinga permite o compresare foarte ridicată cu zero pierderi.[6]

[6] https://css-ig.net/benchmark/png-lossless

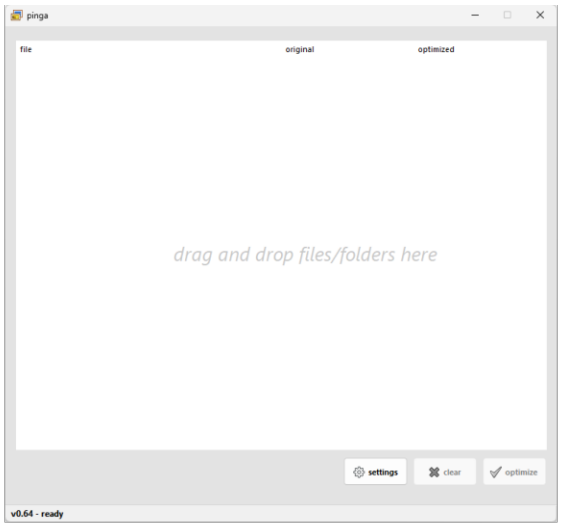


Figura 28. Unealtă comprimare pinga

Conținut aplicație practică

Aplicația „RaportCivic”, construită pe un ansamblul Qt Quick, și suplimentată cu un pachet de componente grafice Qt predefinite, influențate de principiile de design FluentUI, conturate de Microsoft, denotă un design elegant și primitor, care pune în evidență modernitatea și ușurința în utilizare și navigare. Interfața aduce în prim-plan un mediu profesional, prin utilizarea unei palete de culori strălucitoare, vibrante și prismatice, așa cum reiese chiar și din primul moment al contactului cu aplicația (figura 29). Designul minimalist este, de asemenea, îmbogățit prin utilizarea umbrelor și zonelor cu gradient subtil, astfel, adăugându-se o profunzime vizuală.

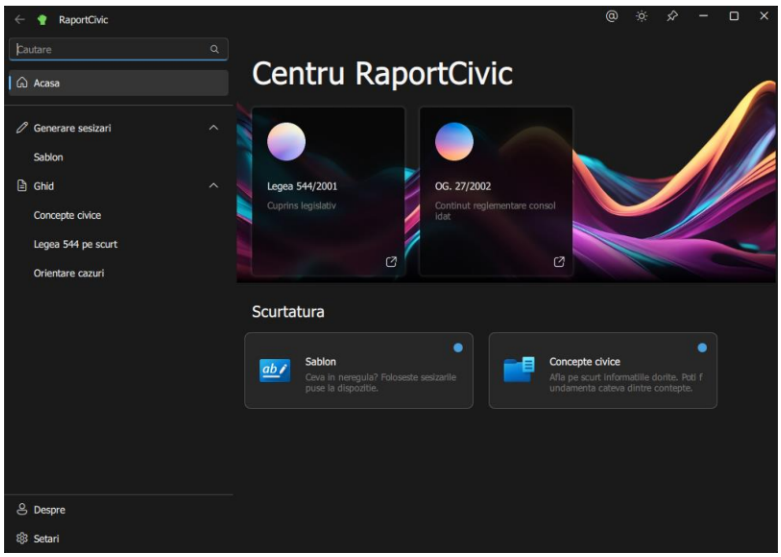


Figura 29. Pagină principală

Utilizatorului îi este prezentată o ierarhie vizuală distinctă, cu o organizare ce denotă meticulozitate în ceea ce privește design-ul ales.

Fundația aplicației începe din rădăcina fișierului „App.qml” (fig. 30). QML respectiv, specifică componenta principală a aplicației, de tip QLauncher care este o clasă derivată din QObject, clasa de bază a sistemului de obiecte al Qt. Succint, acesta se caracterizează ca un sistem meta-object, acționând cu „signals” și „slots” dar și având rolul proprietăților. Fișierul este un gestionar al inițializării, ocupându-se cu diferitele componente ale aplicației.

În acest inițializator, se leagă conexiuni cu clasele ce se ocupă cu funcționalitățile back-end dar și totodată, gestionează „routes” pentru accesarea specifică prin apel, folosindu-se de clasa QRouter. Rutele sunt definite la nivel de cheie și valoare, astfel se poate conferi un nume și o cale de urmat.

```
QLauncher {
    id: app

    Connections {
        target: QTheme
        function onDarkModeChanged() {
            SettingsHelper.saveDarkMode(QTheme.darkMode)
        }
    }

    Connections {
        target: QApp
        function onUseSystemAppBarChanged() {
            SettingsHelper.saveUseSystemAppBar(QApp.useSystemAppBar)
        }
    }

    Connections {
        target: TranslateHelper
        function onCurrentChanged() {
            SettingsHelper.saveLanguage(TranslateHelper.current)
        }
    }

    Component.onCompleted: {
        Network.openLog = false
        Network.setInterceptor(function(param) {
            param.addHeader("Token", "12082asdjf73pvh32hpzbk")
        })
        QApp.init(app, Qt.locale(TranslateHelper.current))
        QApp.windowIcon = "qrc:/qt/qml/src/assets/icons/logo.ico"
        QApp.useSystemAppBar = SettingsHelper.getUseSystemAppBar()
        QTheme.darkMode = SettingsHelper.getDarkMode()
        QTheme.animationEnabled = true

        QRouter.routes = {
            "/": "qrc:/qt/qml/src/qml/pane/PaneBackend.qml",
            "/about": "qrc:/qt/qml/src/qml/pane/About.qml",
            "/accountContent": "qrc:/qt/qml/src/qml/pane/AccountContent.qml",
            "/crash": "qrc:/qt/qml/src/qml/pane/Crash.qml",
            "/userManagement": "qrc:/qt/qml/src/qml/pane/UserManagement.qml",
            "/licenses": "qrc:/qt/qml/src/qml/pane/Licenses.qml"
        }
    }
}
```

Figura 30. App.qml – inițializator QML

De fel, sistemul de căi, la bază este setat în fișierul CMakeLists (fig. 31), sub tip „qml modules”, legându-se la executabilul proiectului și configurările predefinite.

```
# https://doc.qt.io/qt-6/cmake-build-qml-application.html
# qrc:/qt/qml/${URI}/...
qt_add_qml_module(${PROJECT_NAME}
    URI src
    VERSION 1.0
    SOURCES ${PROJECT_SOURCES} ${HEADER}
    QML_FILES ${QML_SOURCES}
    RESOURCES ${RESOURCES}
)
```

Figura 31. Definire resurse și cale sub formă de QML Module

Dacă URI-ul de bază în proiect este definit ca „src”, director implicit, atunci calea stabilită are să fie „qrc:/qt/qm/src/...”, unde punctele de suspensie reprezintă orice director/fișier conținut care se află sub directorul „src”.

În continuare, inițializatorul pe de o parte, trimite calea iconiței prin proprietatea „windowIcon” dar leagă și clasa ce conține preferința în cazul barei aplicației „getUseSystemAppBar”.

Se va folosi apelarea metodei „QRouter.navigate("/")” ce va duce pe calea fișierului, „PaneBackend.qml”, cu tipul clasei „QWindow,” componentă a Window. Fișierul QML se îngrijește de afișarea ferestrei principale (fig. 29). Pe scurt, componenta predefinită „QWindow” conține proprietăți precum alias-uri, argumente nule, definirea proprietăților clasei „QAppBar”, printre care „title”, „height” „width” și butoanele conținute.

Bara ferestrei aplicației (fig. 29), prezintă elementul de branding al „RaportCivic”, pumnul strâns, un simbol al solidarității, al rezistenței și al puterii. Culorile aplicației sunt stabilite, precum ghidul vizual, afișat în figura 32, în funcție de compatibilitatea paletei dar și a culorii de fundal a aplicației.

Figura 32. Ghid vizual branding – paletă culori[7]

[7] https://logo.com/

Tot în aceeași bară superioară, în partea dreaptă, se regăsesc trei butoane, primul este un mic meniu ce permite utilizatorului să se conecteze (fig. 33), cel de-al doilea permite schimbarea între modurile de afișare cu luminozitate sau întunecat iar cel de-al treilea fixează fereastra actuală în poziția cea mai din față a afișării sistemului de operare, pe scurt, neputând să fie suprapusă de o fereastră a unei alte aplicații.

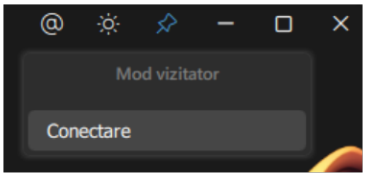


Figura 33. Meniu conectare cont

Se poate observa că prin lipsa contului, proprietatea de tip „QText” afișează doar string-ul „Mod vizitator”.

În componenta „PaneBackend.qml”, este definită integrarea componentei „appBar” (fig. 34) a clasei „QAppBar”, în principala fereastră.

```
appBar: QAppBar {
        width: window.width
        height: 30
        showAccount: true
        showDark: true
        darkClickListener: button => handleDarkChanged(button)
        closeClickListener: () => {
            dialog_close.open()
        }
        z: 7
    }
```

Figura 34. Definire „appBar”

Se stabilește ca în mod implicit, butoanele pentru Cont, prin proprietatea „showAccount” și Luminozitate, prin intermediul „showDark” să fie activate și vizibile.

Pentru meniul cu cont, componentei alterate „QAppBar” i s-au definit câteva aspecte, printre care se enumeră instanțe ale componentei „QMenuItem”, ce determină comportamentul în funcție de existența sau inexistența contului. Dacă clasa de tip singleton, „AccountManager” trimite semnal fals în ceea ce înseamnă existența unui cont, se va stabili doar butonul „Conectare”. În aceeași idee, se va defini un string (fig. 33) care va reprezenta, după caz, contul conectat.

Bara din partea stângă a ferestrei, prezintă un modul de căutare „autoSuggestBox”, ce primește inputuri de la utilizator, și se folosește de componenta „QAutoSuggestBox”, afisând sugestii. Funcția „filter” stabilește filtrarea elementelor iar „loadData” populează lista de sugestii.

Sub funcția de căutare, există un meniu, împărțit în două categorii. Prima categorie, face parte din clasa „ItemsMain”. Aceasta afișează itemii de tip „QPanelItem”, cum ar fi „Acasă” (fig. 35), „Șablon”, „Orientare cazuri” și așa mai departe.

```
QPaneItem {
    id: item_home
    title: qsTr("Acasa")
    icon: FluentIcons.Home
    url: "qrc:/qt/qml/src/qml/screen/Home.qml"
    function homeNav() {
        navigationView.push(url)
    }
    onTap: homeNav()
}
```

Figura 35. Item „Acasa” bara meniu lateral

Obiectele din meniu, primesc un „title”, calea navigabilă înspre fereastra qml dorită, dar și o iconiță, din pachetul predefinit „FluentIcons”. De notat, la apăsarea itemilor, nu se face tranziția în mod direct înspre pagina qml, ci se va folosi o funcție ajutătoare. Scopul acestui fapt se datorează codului dintre componente, ce în anumite cazuri, pentru legarea componentelor grafice cu cele de tip pagină, s-a folosit alterarea prin semnal (fig. 36).

```
Connections {
  target: QmlBridge
  function onLogoutSignal() {
      item_home.homeNav()
  }
...
```

Figura 36. Funcție semnal QML deconectare

Componenta predefinită „QNavigationView” conține funcțiile utilizate, totodată, de către clasele de „itemi”. Spre exemplu, toate categoriile din interior, se pot retracta total, prin intermediul funcției „collapseAll” (fig.37).

```
function collapseAll() {
    for(var i = 0; i < nav_list.model.length; i++) {
        var item = nav_list.model[i]
        if(item instanceof QPaneItemExpander) {
            item.isExpand = false
        }
    }
}
```

Figura 37. Funcție retractare expansiune

De altfel, meniul are caracteristica de a permite în mod dinamic apariția și dispariția unor categorii cu expansiune, în funcție de semnale. În acest context, la conectarea și deconectarea utilizatorilor.

Mai jos, se regăsește și meniul „ItemsBase”, ce conține „Setările” și fereastra „Despre”. Aplicația poate fi personalizată la nivel de animație (a se vedea tag-ul colorat al meniului selectat), tipul fontului prin intermediul proprietății „QTheme.nativeText”, declararea tipului de bară, fie integrat sau simplu (funcționalitate ce nu este adaptată la nivel Linux), schimbarea limbii, prin intermediul „Qt Linguist”. Se definește în aplicație string de tipul „qsTr”, ce va fi preluat de Qt, și prin intermediul CMakeLists (fig. 38) se va crea un fișier de tip „.ts” ce va afișa stringurile care vor fi traduse in software-ul anterior menționat, după care se vor compila sub formă de intrări „.qm”.

```
# generate .ts if not present
foreach(TS_FILE ${TS_FILES})
    if (NOT EXISTS ${TS_FILE})
        execute_process(COMMAND ${QT_LUPDATE} ${CMAKE_CURRENT_SOURCE_DIR} -ts ${TS_FILE})
    endif()
endforeach
```

Figura 38. Funcție retractare expansiune

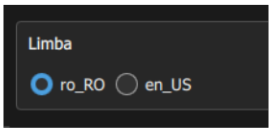


Figura 39. Setare pentru schimbarea limbii în aplicație

Expansiunea de tip „Ghid” conține mai mulți „Itemi” (fig. 40) ce au la bază funcționalitate de citire a fișierelor „.pdf” din interiorul proiectului.

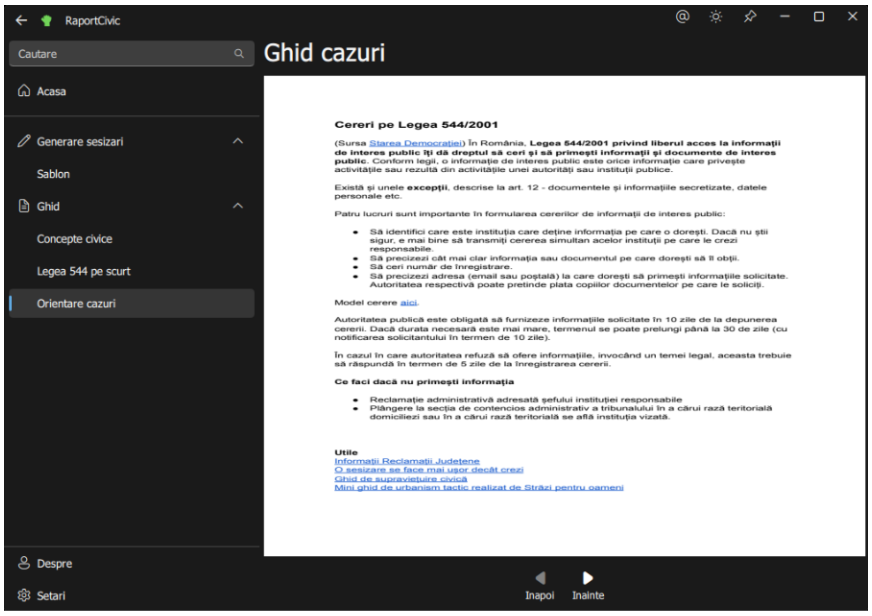


Figura 40. Cititor fișiere format „.pdf”

Clasa qml are importat modulul „QtQuick.Pdf”. Codul conține componenta „PdfDocument”, având proprietate cu calea către sursa fișierului „.pdf”. Această componentă este vitală în încărcarea și afișarea documentelor. Interfața este constituită utilizând un tip „ColumnLayout”, ce plasează vertical itemii conținuți. Tipul moștenitor „ListView” afișează paginile pdf utilizând orientarea orizontală.

Pagina „Acasă” (fig. 29), ce se află sub fișierul „Home.qml”, conține un container de tip „ListModel”, ce are în definire un tip moștenitor „ListElement”, ce are în vedere afișarea de itemi intr-o listă. Lista conține niște elemente dreptunghice, care la apăsarea lor, vor apela funcția aflată sub proprietatea „clicked”. Astfel, se va apela „openUrlExternally”, ce va încărca o pagină externă aplicației, așa cum este definit în proprietatea „url”. De asemenea, se deleagă „com_gallery” ce este o componentă cu rol de funcționare în scrollarea verticală a itemilor.

Scurtăturile, sunt legate în mod dinamic de modulul qml „ItemsMain” cât și de existența unui cont conectat.

Utilizatorii, au la dispoziție meniul de conectare în cont (fig. 41). În cazul în care nu se dispune vreun cont, se poate opta pentru Înregistrare.

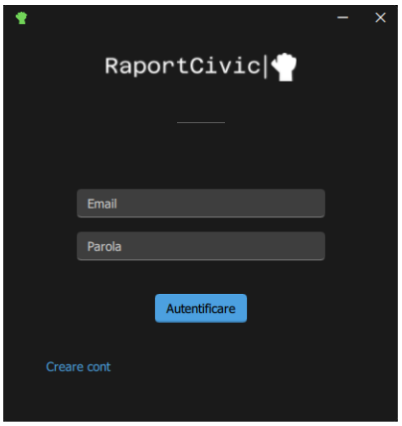


Figura 41. Pagină conectare utilizator

Pagina de înregistrare cât și cea de conectare sunt moștenitoare a clasei „AccountContent” ce au funcții de încărcare pagină, fie „AccountLogin.qml” ori „AccountRegister.qml”, prin intermediul tipului „Loader”, ce poate să încarce fișiere definite, atunci când sunt necesare, în mod dinamic. Există un sistem ce afișează un „infoBar”, tip pop-up în cazul unei erori pentru field-uri, de pildă, parolă („textbox_password.text”) nesigură.

După conectare, se va activa meniul „Utilizator” cu itemul „Profil”. Aici, cetățeanul își va putea insera datele personale în field-urile dedicate (fig. 42).

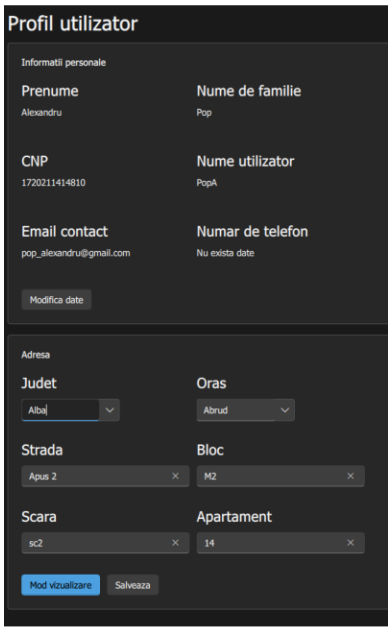


Figura 42. Modificare date utilizator

Pagina permite modificarea sau vizualizarea datelor deja existente. Field-urile au capacitatea de a afișa avertismente dacă input-urile nu sunt corecte. De asemenea, s-a folosit componenta „QComboBox” ce folosind funcția „JsonParser.getAllCounties()”, preia toate județele și orașele disponibile în fișierul „judete.json”.

Pentru generarea unei petiții, se va proceda prin selectarea unui tip dorit din vizualizarea tabelară (fig. 43). Aplicația va reține alegerea, prin intermediul unui singleton specializat.

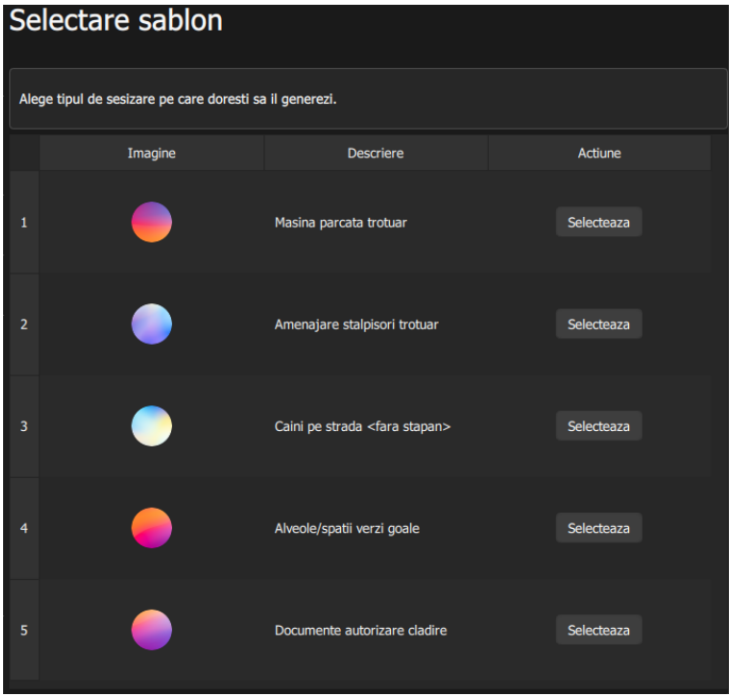


Figura 43. Selectare tabelară

Ulterior, va fi necesară introducerea datelor, dacă acestea nu există, sau se vor folosi deja cele introduse.

În continuare, cetățeanul este dus în pagina cu rol de completare a petiției (fig. 44). Se pot introduce descrieri adiționale, se poate selecta tipul generării, cu transmitere automată prin email sau generare text copiabil și atașamente („redirecționarea externă” nu atașează în mod automat atașamentele prin web, datorită limitărilor fizice, putând doar prin client e-mail specializat).

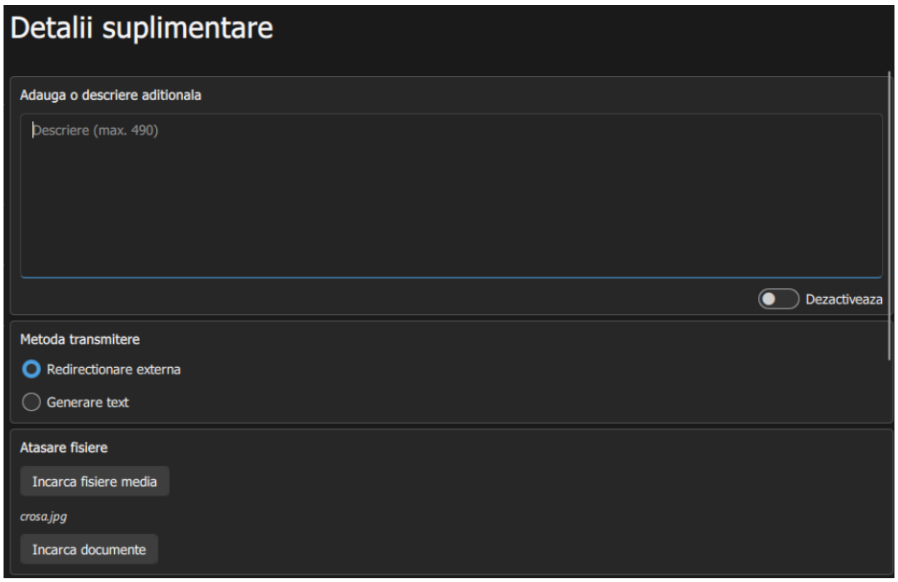


Figura 44. Introducere date suplimentare (pt. 1)

Mai apoi, acesta poate alege, dacă dorește să completeze adresa evenimentului ce leagă petiția (fig. 46). Există mai multe metode disponibile, fie prin extragere EXIF, ce utilizând o librărie externă, se pot afla coordonatele gps din imagine (dacă este cazul), fie cu pinul central de pe harta open-source integrată în aplicație, sau în final, introducere manuală a adresei.

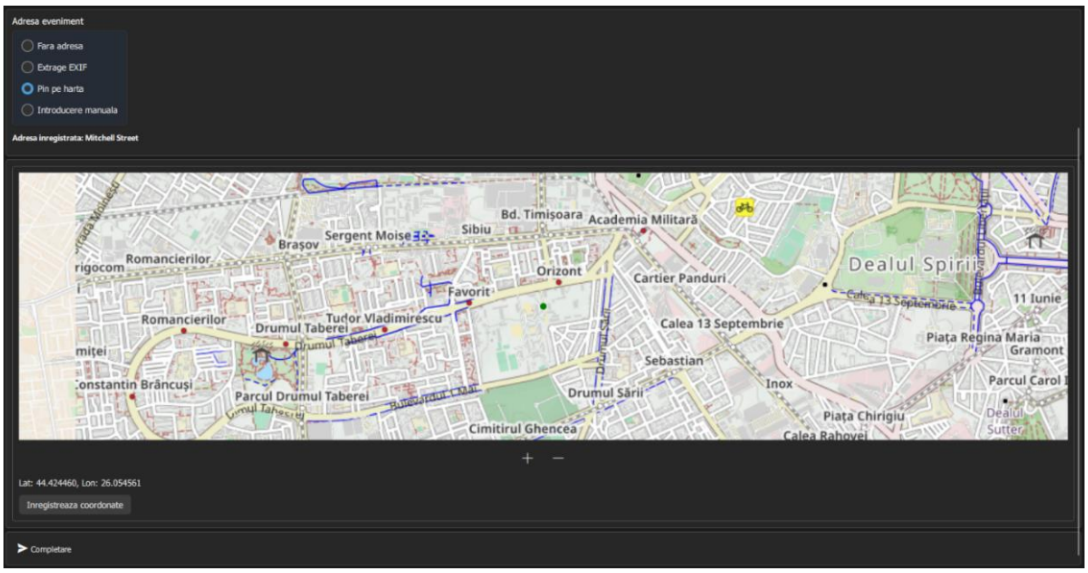


Figura 45. Introducere date suplimentare (pt. 2)

Sumar, funcționalitatea hărții folosește biblioteca „QtLocation” cu tipul „Map”, ce poate afișa interactiv imagini ale globului.

În final, aplicația va genera textul dorit al petiției, sub forma unui „tab_view” (fig. 46). Textul poate fi scrollat dar și copiat pentru transmiterea manuală prin intermediul e-mail-ului.

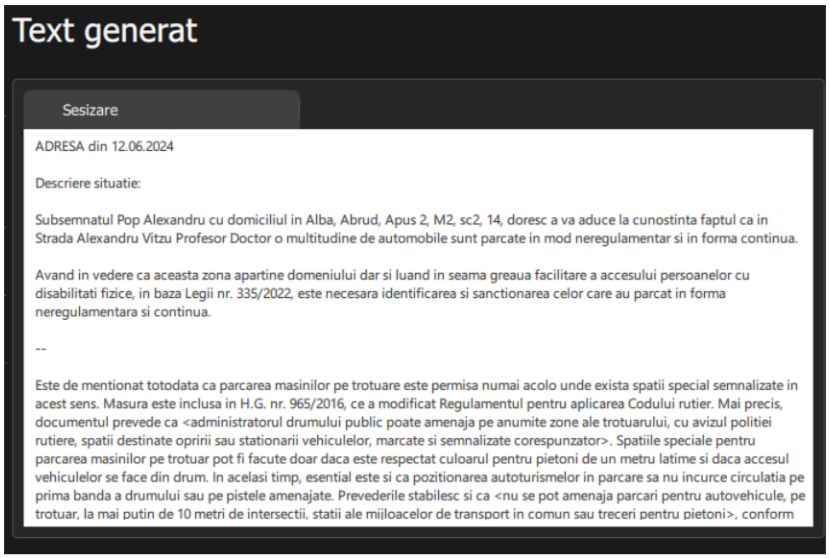


Figura 46. Vizualizare text petiție generat

3.2.2 Implementarea modulului back-end prin intermediul Qt Core și C++

Pentru început, este important de notat că limbajul C++ este îmbunătățit prin intermediul Qt Core. Un concept extrem de important în cadrul Qt C++ este tehnica „Signals & Slots”. În domeniul dezvoltării interfețelor grafice, este un aspect obișnuit existența unei notificări la schimbarea unui widget atunci când un altul are parte de alterări. În imaginea de ansamblu, obiectele pot să stabilească un canal de comunicare între ele.

Unele seturi framework conțin „callback-uri”, pentru a întreprinde comunicarea. Un tip „callback” este reprezentat de un pointer transmis unei funcții. Se folosește ca o anumită funcție să proceseze informația unui eveniment întâmplat. Aceste „callback-uri” nu sunt întotdeauna la îndemână și pot provoca probleme. Qt oferă un substitut pentru „callback-uri”, anume, semnale și sloturi (fig. 47). Acestea sunt predefinite în framework. Atunci când un eveniment se petrece, un „signal” este rezultat. Se poate extinde prin crearea de subclase și „signal-uri” personalizate. De asemenea, un „slot” este o subrutină drept reacție la un anumit „signal”. Tot în aceeași idee, ele sunt predefinite, însă este un obicei comun să se definească propriile „sloturi”.

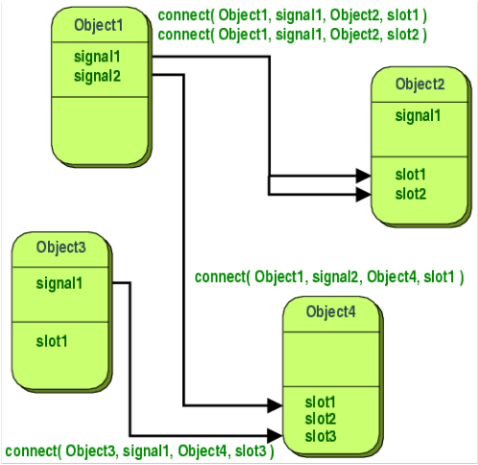


Figura 47. Semnale și sloturi – conexiuni abstracte[8]

Fișierul rădăcină al proiectului, pe partea back-end, este „main.cpp”. Acesta conține toate legăturile între back-end și este chiar și cu rol de inițializator pe parte de front-end.

Codul definește import-urile, sursele precum headerele, face legături în funcție de sistemul de operare, setează datele proiectului, inițializează diverse instanțe, baza de date, expune clase C++ sub formă de module, înspre QML, cu ajutorul funcției template „qmlRegisterType” dar și expunerea specifică, globală, la nivel de signleton C++ în mediul QML, prin intermediul metodei „engine.rootContext()->setContextProperty”.

Proiectul Qt „RaportCivic” deține un set destul de consistent în ceea ce privește managerele și utilitarele. Ele sunt de obicei împărțite în „headers” și „implementations”. Astfel că, un prim exemplu îl constituie clasa „ProfileValidator”. Este capabilă de validarea și transmiterea validității a mai multor elemente, precum, nume prin intermediul metodei „validateName” (fig. 48), adrese, telefoane e-mailuri și chiar cnpuri.

[8] https://doc.qt.io/qt-6/signalsandslots.html

```
QString ProfileValidator::validateName(const QString &name, bool isFirstName) const
{
    QString trimmedName = name.trimmed();
    if (trimmedName.isEmpty()) {
        return QObject::tr("Numele nu poate fi gol.");
    }

    if (isFirstName) {
        static const QRegularExpression firstNameRegex("^[a-zA-Z]+(\\s[a-zA-Z]+)*$");
        if (!firstNameRegex.match(trimmedName).hasMatch()) {
            return QObject::tr("Prenumele poate continue numai litere si spatii.");
        }
    } else {
        static const QRegularExpression lastNameRegex("^[a-zA-Z]+$");
        if (!lastNameRegex.match(trimmedName).hasMatch()) {
            return QObject::tr("Numele de familie poate contine doar litere.");
        }
    }

    return QString(); // passed validation
}
```

Figura 48. Validator nume

Numele este validat prin tehnica referinței și există un tip boolean care certifică dacă ceea ce primește metoda este numele de familie sau prenume. Funcția returnează un „QString”, care fie este gol sau un mesaj ce are o eroare, dacă numele este invalid. Prin intermediul „Regex”, elimină spații suplimentare și verifică dacă numele este gol. Pentru prenume verifică dacă are mai multe cuvinte (multiple prenume) și sunt separate prin spații și pentru numele de familie, verifică doar dacă conține litere.

Clasa „DbManager” implementează managementul tuturor operațiilor ce sunt legate de baza de date a utilizatorului. Are metode pentru adăugarea și verificarea utilizatorilor, actualizarea timpilor la logare, extragerea ID-urilor și înregistrările datelor asociate cu utilizatorii. Se folosește biblioteca „QSqlQuery” pentru executarea comenzilor ce țin de partea SQL dar și „QVariantMap” pentru returnarea rezultatelor datelor.

În situația dată, un exemplu ar fi metoda „createAddress” (fig. 49) care este concepută să insereze noi înregistrări ale adreselor în tabela „Address” a bazei de date. Se iau în considerare câțiva parametrii precum: „citizenId”, „county”, „city”, „street”, „building”, „entrance” și „apartment” care sunt detaliile adresei ce vor fi adăugate.

Inițializarea se face prin „result” de tipul „QVariantMap” ce înmagazinează rezultatul operațiunii. Totodată, obiectul „query” de tipul „QSqlQuery” este creat, fiind asociat cu baza de

date. Declarația „INSERT” a părții SQL adaugă noi înregistrări în tabela „Address”. „CURRENT_TIMESTAMP” este în uzul „last_modification”, ce înregistrează în mod automat o înregistrare a timpului la data inserării. „bindValue” este utilizat pentru a lega fiecare parametru în spațiul corespunzător al „query-ului SQL”, prevenind și atacurile cu injecție SQL. Metoda „exec” este apelată și execută „query-ul”. Dacă este finalizată cu succes, operațiunea returnează „success” de tip „true”, în „map”. Se creează un „QVariantMap” imbricat denumit „data” ce înmagazinează o parte din datele inserate iar rezultatul este în final, returnat.

```
QVariantMap DbManager::createAddress(int citizenId, const QString &county, const QString &city,
                                     const QString &street, const QString &building,
                                     const QString &entrance, const QString &apartment)
{
    QVariantMap result;
    QSqlQuery query(db);
    query.prepare("INSERT INTO Address (citizen_id, county, city, street, building, entrance, apartment, last_modification) "
                  "VALUES (:citizenId, :county, :city, :street, :building, :entrance, :apartment, CURRENT_TIMESTAMP)");
    query.bindValue(":citizenId", citizenId);
    query.bindValue(":county", county);
    query.bindValue(":city", city);
    query.bindValue(":street", street);
    query.bindValue(":building", building);
    query.bindValue(":entrance", entrance);
    query.bindValue(":apartment", apartment);

    if (!query.exec()) {
        result["success"] = false;
        result["error"] = query.lastError().text();
        return result;
    }

    result["success"] = true;
    result["data"] = QVariantMap{{"citizenId", citizenId}, {"county", county}, {"city", city}};
    return result;
}
```

Figura 49. Metoda „createAddress” a clasei „DbManager”

Clasa „AccountManager” este însărcinată cu managementul sesiunilor de conectare, interfațarea cu „DbManager” fiind necesară pentru operațiunile bazei de date. Totodată, salvează în mod opțional și datele utilizatorului conectat folosindu-se de „AccountStorage’. Când utilizatorul dorește să se conecteze, prin intermediul metodei „loginAccount” se va face verificarea datelor cu ajutorul metodei externe „DbManager:verifyUser”. Dacă verificarea a rezultat a fi validă, „AccountManager” actualizează condiția, astfel utilizatorul are proprietățile „isLoggedIn”, i se conferă „userEmail” și „accountId”. Totodată, se verifică dacă datele sunt înmagazinate, prin metoda „loadAccount”. Dacă contul a fost găsit, utilizatorul va fi conectat în mod direct, actualizând data conectării și se va face rost de „id-ul” contului. De asemenea, metoda „logoutAccount” elimină în mod complet condițiile, precum informațiile utilizatorului, starea conectării și așa mai departe.

Fișierul de tip antet „singleton.h” (fig. 50) specifică o clasă Singleton bazată pe un șablon care garantează existența doar a unei singure instanțe globale a unei clase cu acces universal pe tot parcursul execuției proiectului. Șablonul de clasă Singleton utilizează o metodă membră static „getInstance()” pentru a acorda acces în instanța unică a „template-ului”. Instanța statică este inițiată atunci când metoda „getInstance()” este utilizată pentru prima dată și rămâne în uz pe întreg parcursul, ciclul de viață al programului, stabilind o garanție ca obiectul în mod unic să fie returnat indiferent de invocările viitoare. De asemenea, „constructorul” și „destructorul” sunt protejate, ceea ce în mod clar, poate să evite instanțierea, ori, după caz, eliminarea, în cale directă a obiectelor aparținătoare. Tot în aceeași idee se aplică și evitarea „copierii” și „asignării”. Header-ul este folosit, în mai multe implementări ale claselor proiectului, putând fi considerat de fel, ca un tip de centralizator uniform pentru managementul instanțelor unice.

```
#ifndef SINGLETON_H
#define SINGLETON_H

/* single instance with global acceess */

template <typename T>
class Singleton {
public:
    static T& getInstance() {
        static T instance; // static instance
        return instance; // same object
    }

protected:
    // prevent direct instantiation
    Singleton() {}
    virtual ~Singleton() {}

private:
    // explicitly delete to disallow copying or assigning
    Singleton(const Singleton&) = delete;
    Singleton& operator=(const Singleton&) = delete;
};

#endif // SINGLETON_H
```

Figura 50. singleton.h template

Clasa „PasswordHasher” (fig. 51) are în vedere demonstrarea și utilizarea algoritmului denumit „Argon2id”, o funcție de tip hasher ce a fost totodată și premiat în cadrul „PHC” pentru calitățile sale, aceea fiind, sumar, o competiție dedicată acest instrumente pentru „password hashing”. Se folosește biblioteca externă „libsodium”, ce manageriază un amalgam de evenimente precum criptări, semnături, decriptări, desigur, hashingul parolelor și așa mai departe. Scopul clasei „PasswordHasher” este de a asigura gestionarea în siguranță a parolelor alături de conturile ce vor fi create de „AccountManager” împreună cu „DbManager”. Clasa are un constructor de

inițializare și în plus, două funcții principale, care se enumeră: „hashPassword” dar și „verifyPassword”. Funcția „hashPassword” utilizează funcția „crypto_pwhash_str” din interiorul „libsodium” pentru a încerca transformarea dintr-o simplă parolă de tip test string într-un hash extrem de sigur. Algoritmul prezintă un nivel ridicat de securitate datorită rezistenței sale la atacurile bazate pe „GPU” dar și prin natura „memory-hard”. Funcția, printre altele, utilizează niveluri de praguri ale operațiunii și utilizării memoriei, regăsind un echilibru între eficiență dar și securitate. Parola, care a fost convertită într-un hash, este salvată într-un vector de stringuri tip „QByteArray”. Ulterior, după finalizarea eliminării oricăror caractere de drept nule, acesta este returnat ca un string al clasei „QString”. Totodată, funcția „crypto_pwhash_str_verify” determină dacă o parolă de tip text string simplu corespunde cu parola trecută prin procesul de hashing anterior.

```
PasswordHasher::PasswordHasher(QObject *parent) : QObject(parent) {
}

QString PasswordHasher::hashPassword(const QString &password) {

    /* hashing the password using argon2id
     * https://doc.libsodium.org/password_hashing
     * https://github.com/P-H-C/phc-winner-argon2
     */
    QByteArray hashedPassword(crypto_pwhash_STRBYTES, '\0');
    if (crypto_pwhash_str(hashedPassword.data(), password.toUtf8().data(), password.length(),
                          crypto_pwhash_OPSLIMIT_INTERACTIVE, crypto_pwhash_MEMLIMIT_INTERACTIVE) != 0) {
        qFatal("Out of memory");
    }

    // trim null characters while converting
    return QString::fromUtf8(hashedPassword.constData(), hashedPassword.indexOf('\0'));
}

bool PasswordHasher::verifyPassword(const QString &hashedPassword, const QString &password) {
    return crypto_pwhash_str_verify(hashedPassword.toUtf8().data(), password.toUtf8().data(),
password.length()) == 0;
}
```

Figura 51. Unealta „PasswordHasher”

Utilitarul denumit „ExifExtractor” (fig. 52) definește în mod simplu un proces de extragere a datelor de poziție, adică coordonatele GPS, din fotografiile de tip „jpeg”, dacă este cazul, folosind implementarea bibliotecii „TinyEXIF”, ce este o bibliotecă compactă, scrisă in C++, care preia eficient metadatele Exif. Este o subclasă a „QObject”. Metoda „extractExifLocation” deschide fișierul de imagine furnizat utilizând „QFile”. Dacă, în cazul în care, fișierul e gol, se transmite o eroare. Datele imaginii sunt preluate și stocate într-un string „QByteArray”. Mai departe, se creează o instanțiere a „TinyEXIF::ExifInfo” ce înglobează datele EXIF. Funcția „parseFrom”

extrage metadatele EXIF din matricea de octeți. Se va verifica existența și corectitudinea datelor folosind metoda „hasLatLon()”, iar dacă sunt disponibile, se vor converti într-un format șir.

```
...
QByteArray data = file.readAll();
file.close();

TinyEXIF::EXIFInfo exifInfo;
qDebug() << "File read successfully, size:" << data.size();

int parseResult = exifInfo.parseFrom(reinterpret_cast<const unsigned char*>(data.data()), data.size());
qDebug() << "EXIF parse result:" << parseResult;

if (parseResult == 0) { // verify if lat and long exists
    if (exifInfo.GeoLocation.hasLatLon()) {
        double latitude = exifInfo.GeoLocation.Latitude;
        double longitude = exifInfo.GeoLocation.Longitude;
...
```

Figura 52. Extractor „Exif”

Clasa „ReportBridge” este responsabilă pentru gestiunea datelor petițiilor. Oferă metode de „getter” și „setter” pentru mai multe proprietăți, cum ar fi „selectReportId”, „firstName”, „lastName” și enumerarea poate continua. Setterul poate oferă semnale atunci când există schimbări de valori, permițând interfeței să se actualizeze în mod independent.

Utilitarul denumit „ReportJsonManager” (fig. 53) produce petiții cuprinzătoare, utilizând șabloanele definite „JSON” dar și inputuri ale utilizatorilor. Are funcții specifice, importând biblioteci ale claselor precum „QJsonDocument” (specializare în citire/scriere) dar și „QJsonArray” (încapsulare vector „string json”) ce ajută în analizarea fișierelor JSON și generarea de petiții. Funcția „readJsonObject” analizează un fișier definit JSON și preia conținutul în clasa tip listă de obiecte „chei-valori” denumită „QJsonObject”. Funcția „generateReport” acceptă un ID al petiției, în funcție de alegerea utilizatorului în partea front-end, și preia șablonul JSON ce corespunde, combinând diversele submodule reutilizabile, precum „introduction.json”, „og27.json” și lista poate continua. Este implicat „readJsonObject” pentru preluarea șablonului și a acelor reutilizabile alese. Ulterior, se vor înlocui „placeholderele” (substituenții) șablonului cu datele reale din instanța singleton „ReportBridge”.

```
...
QString ReportJsonManager::generateReport(int mainTemplateId) {
    QString report;
    QString mainTemplateFile;

    // Map the ID to corresponding templates
    switch (mainTemplateId) {
    case 1:
        mainTemplateFile = ":/qt/qml/src/assets/data/report_templates/car_park_sidewalk.json";

...

    QJsonObject law544Json = readJsonObject(":/qt/qml/src/assets/data/report_templates/reusable/l544_2001.json");

    ReportBridge& data = ReportBridge::getInstance();

    // Process the main template based on the data and sub-modules
    QJsonArray mainTemplateArray = mainTemplateJson.value("template").toArray();
    for (const QJsonValue& value : mainTemplateArray) {
        QString section = value.toString();
...

        // replace the placeholders in the section
        if (!addressComponents.isEmpty()) {
            section = section.replace("{{building}} {{entrance}} {{apartment}}", addressComponents);
        } else {
            section = section.replace("{{building}} {{entrance}} {{apartment}}", "");

            // comma handle
            section = section.replace(", ,", ",");
            section = section.replace(",  ,", ",");
            section = section.replace(",,", ",");
            section = section.replace(" ,", "");
        }
...

  QJsonObject ReportJsonManager::readJsonObjectPrivate(const QString& fileName) {
    QFile file(fileName);
    if (!file.open(QFile::ReadOnly)) {
        qDebug() << "Failed to open file:" << fileName;
        return QJsonObject();
    }

    QJsonParseError error;
    QJsonDocument doc = QJsonDocument::fromJson(file.readAll(), &error);
    if (error.error != QJsonParseError::NoError) {
        qDebug() << "Failed to parse JSON:" << error.errorString();
        return QJsonObject();
    }

    return doc.object();
}
```

Figura 53. Manager de module JSON

3.2.3 Nivelul de acces la date folosind SGBD-ul relațional PostgreSQL

Baza de date relațională de tip PostgreSQL comunică cu aplicația prin intermediul stabilirii conexiuni aflate în fișierul rădăcină back-end, anume „main.cpp” (fig. 54).

```
QSqlDatabase db = QSqlDatabase::addDatabase("QPSQL");
    db.setHostName("pg-093jdnasd-armin-4op3f.b.aivencloud.com");
    db.setPort(19099);
    db.setDatabaseName("db_raportcivic");
    db.setUserName("avnadmin");
    db.setPassword("AVNS_o23idjioH-X7pO23dB");
    db.setConnectOptions("requiressl=1;sslmode=require;sslrootcert=ca.pem");
    if (!db.open()) {
        qDebug() << "Failed to connect to the database: " << db.lastError().text();
        return -1;
    } else {
        qDebug() << "Connected succesfully to the database!";
    }
```

Figura 54. Stabilire conexiune cu baza de date

În această secțiune componentă a fișierului „main.cpp”, se face o conexiune la o bază de date PostgreSQL utilizând clasa „QSqlDatabase” ce, are specific, întrebuințarea cu conexiunile la bazele de date. Este nevoie de definirea driver-ului, predefinit și existent, ori compilat de utilizator, care se derivă din clasa abstractă „QSqlDriver”, ce se ocupă cu diversele tipuri de baze de date. În acest caz, a fost introdus „QPSQL” adică, clasa „QPSQLDriver” ce stabilește, în mod normal, o conexiune de tipul PostgreSQL. Parametrii de conectare sunt, de pildă, „HostName-ul”, „Port-ul”, „databaseName”, „UserName-ul” și „Password”. În plus, parametrii conexiunii SSL sunt definiți pentru a oferi o conexiune sigură prin utilizarea „requiressl” cu valoarea „true”, „sslmode=require”, ce înseamnă că se cere ca datele să fie criptate și că valoarea de securitate este verificată de rețea înspre „server-ul” dorit. Este necesar un certificat „ca.pem”, ce se află în directorul rădăcină.

Baza de date PostgreSQL, menționată anterior, este găzduită de Aiven.io (fig. 55), printr-un pachet de gratuitate, acesta fiind un furnizor de servicii cloud gestionare care eficientizează implementarea și întreținerea bazei de date. Baza de date este găzduită pe infrastructura „DigitalOcean” din regiunea Frankfurt, Germania. Aiven.io oferă disponibilitate robustă, scalabilitate, măsuri de securitate riguroase și întreținere automată, garantând astfel gestionarea rapidă a activităților precum „backup-urile”, „update-uri” și chiar și „patch-uri” de securitate. DigitalOcean oferă arhitectura fundamentală, fiind renumit pentru capacitatea excepțională de stocare în tip „SSD”, rețea de încredere și metodele de redundanță a datelor.

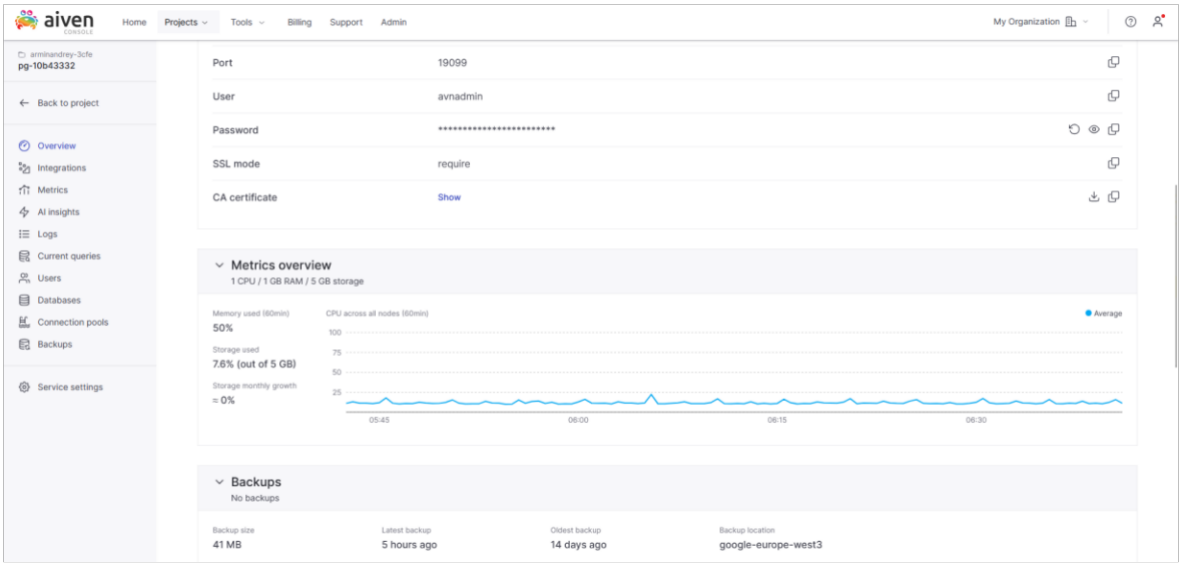


Figura 55. Panou de control cloud Aiven.io

Prezentare generală a schemei bazei de date PostgreSQL

1. Tabelul „Account”

Acest tabel, denumit „Account” (fig. 56) este conceput special pentru a conține informații esențiale ale conturilor de utilizator.

„account_id” servește drept cheie primară, de tip serial a tabelei „Account” și este o valoare întreagă, ce se auto-incrementează.

„email” este utilizat în sensul reținerii adresei poștei electronice, email-ul utilizatorului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 100 caractere. Există impusă o restricție unică pentru garantarea inexistenței a două conturi ce dețin aceeași adresă de email folosită. Coloana este obligatorie, nu poate fi nulă.

„password_hash” este utilizat pentru păstrarea criptării parolelor utilizatorilor. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 100 caractere. Coloana este obligatorie, nu poate fi nulă.

„creation_date” înregistrează, printre altele, data exactă a creării contului utilizator. Valoarea implicită definită în câmp este setată la data curentă („CURRENT_DATE”). Coloana este obligatorie, nu poate fi nulă.

„last_login” este utilizat, de asemenea, pentru a reține momentul cel mai recent, în sensul datării, la care s-a conectat utilizatorul. Datele sunt de tip „TIMESTAMP”. Coloana poate fi nulă.

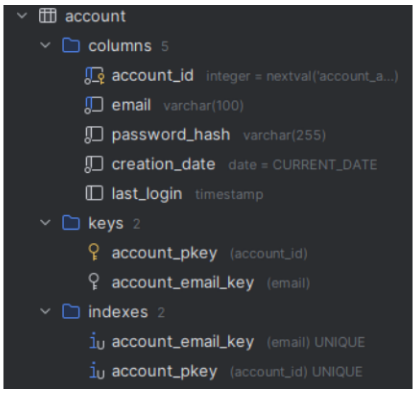


Figura 56. Tabela „Account”

2. Tabelul „CitizenData”

Tabelul în cauză (fig. 57), conține informații personale ale cetățenilor care sunt asociate cu conturile acestora. Fiecare persoană este conectată la un cont din tabelul „Account”, printr-o conexiune cu cheie străină.

„citizen_id” din tabel, servește cu rolul de cheie primară, de tip serial al tabelei. Este o valoare întreagă, ce se auto-incrementează.

„account_id” este o intrare cu cheie străină, de tip întreg, și face referire la coloana „account_id” din tabelul „Account”. Legătura dintre „Account” dar și „CitizenData” are de fel, o relație de tipul „1:1”. Coloana este obligatorie, nu poate fi nulă.

„last_name” este folosit pentru conținerea numelui de familie al cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 50 caractere. Coloana este obligatorie, nu poate fi nulă.

„first_name” este folosit pentru conținerea prenumelui cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 50 caractere. Coloana este obligatorie, nu poate fi nulă.

„personal_numeric_code” conține codul numeric personal, adică CNP-ul exclusiv al cetățeanului. Există restricția de unicitate, ce se asigură că nu există doi cetățeni cu același cod. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 15 caractere. Coloana este obligatorie, nu poate fi nulă.

„username” se utilzează pentru înregistrarea numelui de utilzator al cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 50 caractere. Coloana poate fi nulă.

„phone” are scopul înregistrării numărului de telefon al cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 20 caractere. Coloana poate fi nulă.

„last_modification” este utilizat, totodată, pentru a a marca cel mai recent moment, în sensul datării, la care s-a efectuat modificarea datelor cetățeanului. Datele sunt de tip „TIMESTAMP”. Coloana poate fi nulă.

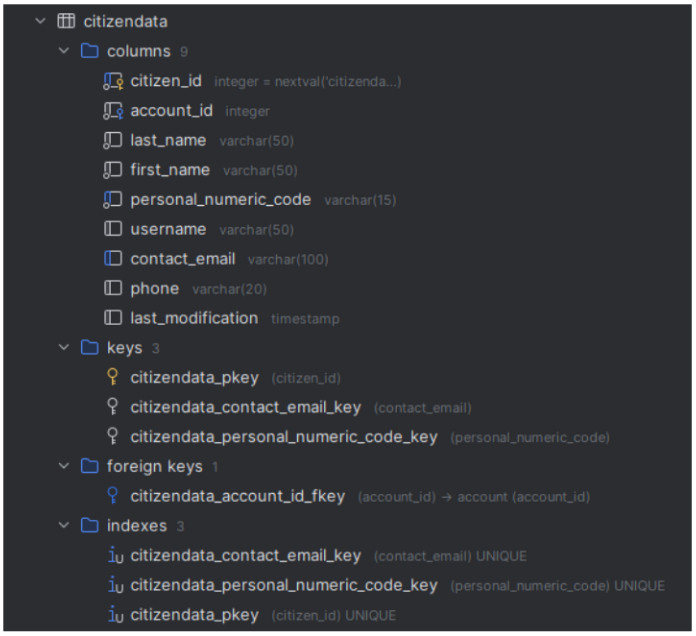


Figura 57. Tabela „CitizenData”

3. Tabelul „Address”

Tabelul în fapt (fig. 58), conține informații legate de specificul adreselor utilizatorilor. Fiecare dintre înregistrări ale „CitizenData” se asociază în mod direct cu un cetățean distinct.

„address_id” prezent, servește cu rolul de cheie primară, de tip serial al tabelei. Este o valoare întreagă, ce se auto-incrementează.

„citizen_id” este un element de cheie străină, de tip întreg, și se conectează, printr-o referință, la coloana „citizen_id” din tabelul „CitizenData”. Legătura dintre „Address” precum și cea din „CitizenData” este marcată ca o relație ce are valoare „1:1”. Coloana este obligatorie, nu poate fi nulă.

„county” înglobează județul în care se află adresa cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 100 caractere. Coloana poate fi nulă.

„city” reprezintă conținerea orașului unde locuiește cetățeanul. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 100 caractere. Coloana poate fi nulă.

„building” păstrează numărul sau codul clădirii, după caz, blocului, din adresa cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 20 caractere. Coloana poate fi nulă.

„entrance” coloana are uzul de a prezenta numărul sau numele intrării, după caz, a scării în adresa cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 20 caractere. Coloana poate fi nulă.

„apartment” poate să țină numărul apartamentului, după caz, reședinței cetățeanului. Datele sunt de tip „VARCHAR” cu o lungime maxim permisă de 20 caractere. Coloana poate fi nulă.

„last_modification” este utilizat, totodată, pentru a a marca cel mai recent moment în sensul datării, la care s-a efectuat modificarea datelor adresei cetățeanului. Datele sunt de tip „TIMESTAMP”. Coloana poate fi nulă.

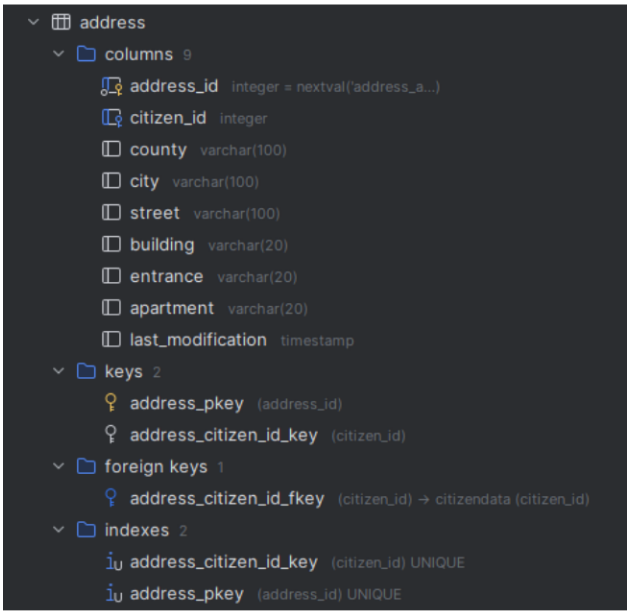


Figura 58. Tabela „Address”

Diagramă cu relațiile bazei de date

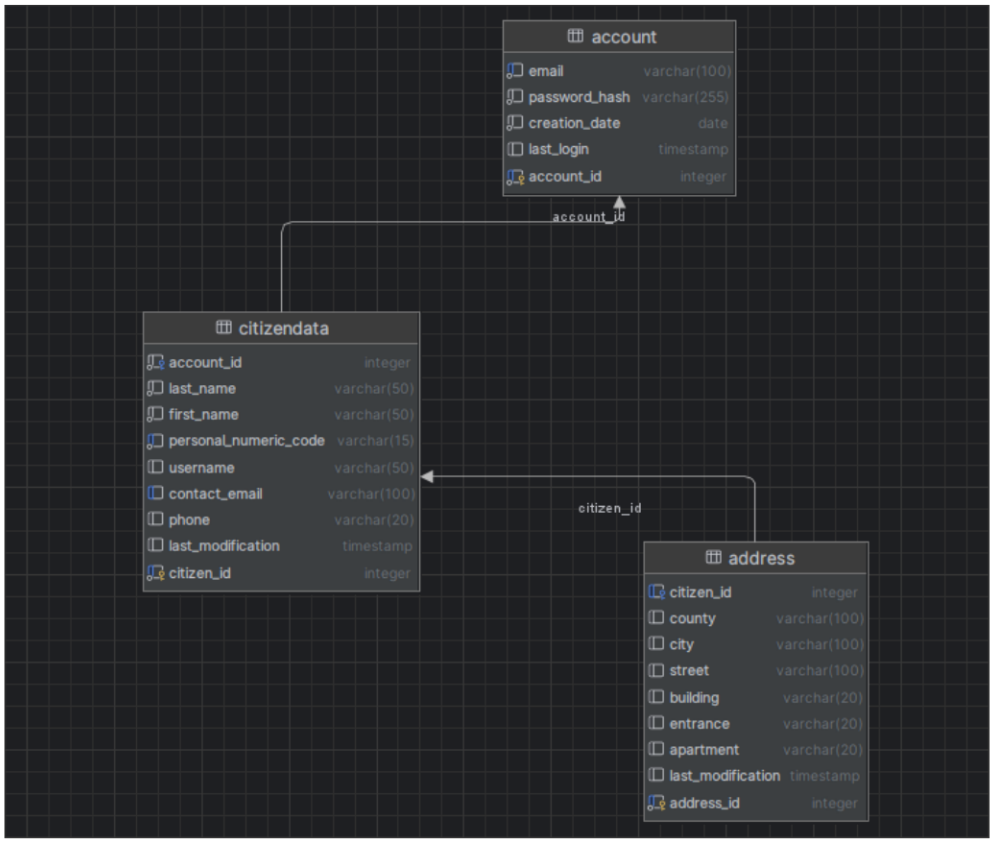


Figura 59. Diagrama bazei de date

3.3 Analiza sistemului – model analist

Prezenta diagramă de activitate (fig. 60), afișează descriptiv în sens vizual, fluxul de operare „in and out” al sistemului, pornind de la prima pagină a aplicației, până la finalizarea și trimiterea sesizării.

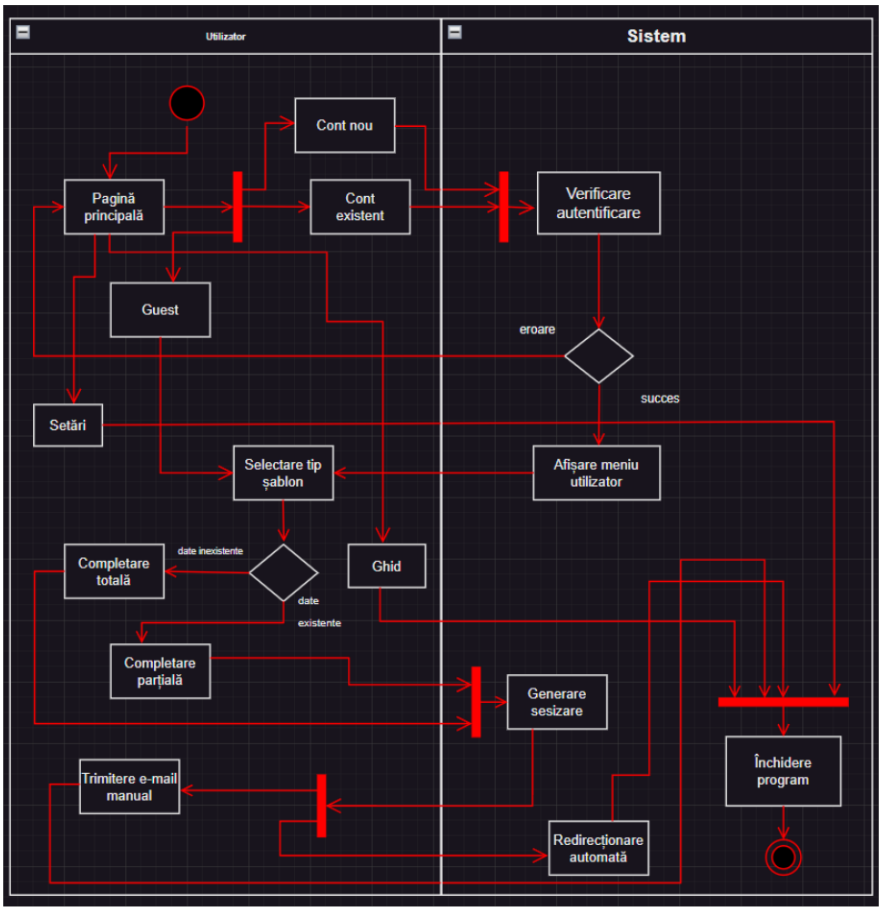


Figura 60. Diagrama „cazului de activitate” a sistemului

4 Evaluarea și testarea

4.1 Testarea și depanarea implementării

Aplicația are integrate diverse module de debugging, fie pentru partea de back-end prin intermediul claselor de tip „QDebug”, fie cea de front-end, de pildă „console.log()” pentru JavaScript. În utilizarea aplicației finale, nu s-au raportat evenimente problematice, orice neregulă fiind remediată imediat.

Proiectul este capabil printr-o bibliotecă de logging, să afișeze detalii tehnice ale sistemului (fig. 61).

```
11:03:03: Starting C:\Users\ArminC\Desktop\LucruQT\RaportCivic\bin\Debug\RaportCivic.exe...
2024/06/13 11:03:03.943[Info][Log.cpp:166][21784]:==================================================
2024/06/13 11:03:03.943[Info][Log.cpp:167][21784]:[Name] "RaportCivic_Core"
2024/06/13 11:03:03.943[Info][Log.cpp:168][21784]:[Path] "C:\\Users\\ArminC\\Desktop\\LucruQT\\RaportCivic\\bin\\Debug\\RaportCivic.exe"
2024/06/13 11:03:03.943[Info][Log.cpp:169][21784]:[QtIteration] 6.7.1
2024/06/13 11:03:03.944[Info][Log.cpp:171][21784]:[ProcessId] "19240"
2024/06/13 11:03:03.944[Info][Log.cpp:175][21784]:[DeviceInfo]
2024/06/13 11:03:03.944[Info][Log.cpp:176][21784]:  [DeviceId] "041c0bd4-8687-4b16-97fa-c77e9bbdd570"
2024/06/13 11:03:03.944[Info][Log.cpp:177][21784]:  [Manufacturer] "Windows >=11 build 22631"
2024/06/13 11:03:03.944[Info][Log.cpp:178][21784]:  [CPU_ABI] "x86_64"
2024/06/13 11:03:03.944[Info][Log.cpp:179][21784]:[LOG_LEVEL] 4
2024/06/13 11:03:03.944[Info][Log.cpp:180][21784]:[LOG_PATH] "C:/Users/ArminC/AppData/Local/ArminC/RaportCivic/log/RaportCivic_Core_20240613.log"
2024/06/13 11:03:03.944[Info][Log.cpp:181][21784]:==================================================
2024/06/13 11:03:03.954[Debug][main.cpp:67][21784]:Succesfully initialized libsodium!
2024/06/13 11:03:04.368[Debug][main.cpp:82][21784]:Connected succesfully to the database!
```

Figura 61. Caracteristici sistem

În același plan, compilatorul proiectului (fig. 62), nu raportează nicio eroare tehnică, rezultând că aplicația rulează în conformitate.

```
Compile Output                                  Filter
    Generated 44 translation(s) (44 finished and 0 unfinished)
Updating 'component_ro_RO.qm'...
    Generated 0 translation(s) (0 finished and 0 unfinished)
    Ignored 44 untranslated source text(s)
[2/54 4.9/sec] Updating translations
Scanning directory 'C:/Users/ArminC/Desktop/LucruQT/RaportCivic/src'...
Updating 'RaportCivic_en_US.ts'...
    Found 250 source text(s) (0 new and 250 already existing)
Updating 'RaportCivic_ro_RO.ts'...
    Found 250 source text(s) (0 new and 250 already existing)
Updating 'C:/Users/ArminC/Desktop/LucruQT/RaportCivic/src/lang/RaportCivic_en_US.qm'...
    Generated 248 translation(s) (248 finished and 0 unfinished)
    Ignored 2 untranslated source text(s)
Updating 'C:/Users/ArminC/Desktop/LucruQT/RaportCivic/src/lang/RaportCivic_ro_RO.qm'...
    Generated 0 translation(s) (0 finished and 0 unfinished)
    Ignored 250 untranslated source text(s)
[3/54 4.4/sec] Automatic MOC and UIC for target RaportCivic
[4/53 5.4/sec] Running AUTOMOC file extraction for target RaportCivic
[5/53 6.1/sec] Running moc --collect-json for target RaportCivic
[6/7 1.6/sec] Building CXX object src/CMakeFiles/RaportCivic.dir/dev/util/Log.cpp.obj
[7/7 1.1/sec] Linking CXX executable C:\Users\ArminC\Desktop\LucruQT\RaportCivic\bin\Debug\RaportCivic.exe
11:03:03: The process "C:\Qt\Tools\CMake_64\bin\cmake.exe" exited normally.
11:03:03: Elapsed time: 00:07.
```

Figura 62. Output compilator

De asemenea, în modul debugging (fig. 63) al proiectului in IDE, nu se raportează nicio eroare, fie pe partea de cod back-end sau QML. Este de menționat că biblioteca de actualizare nu este în totalitate activă, proiectul neavând un sistem tip Git setat pentru extragerea datelor referitoare la versiune.

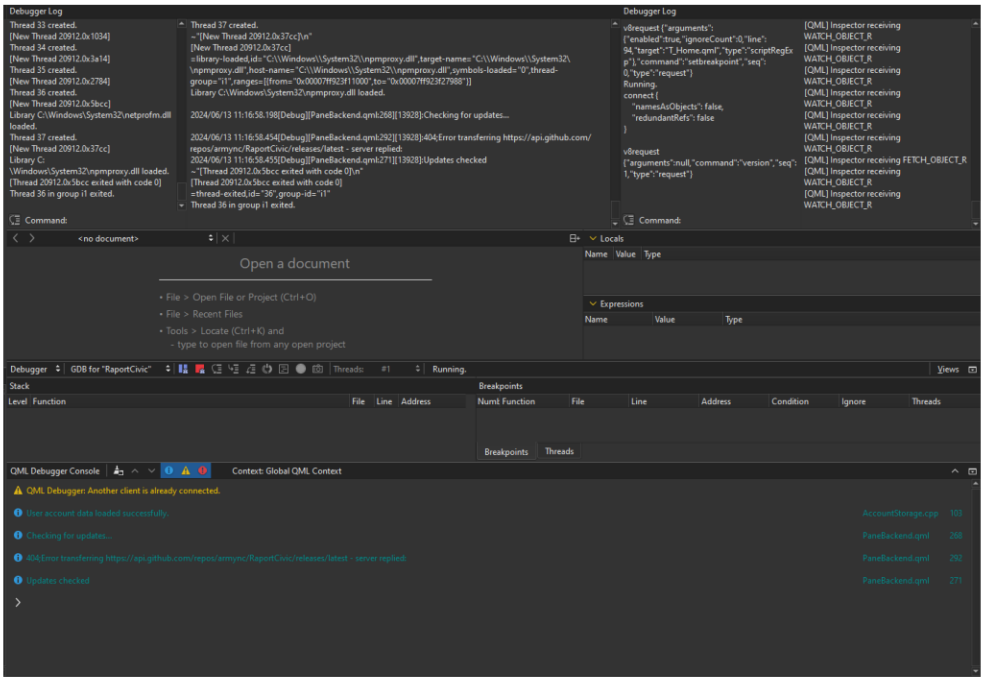


Figura 63. Output debugger, inclusiv QML

În cauzalitatea intervenirii unei defecțiuni, este posibil ca aplicația să raporteze „crash-ul” (fig. 64), unde utilizatorul poate observa log-urile raportate precum cele în consola Qt. Totuși, acestea sunt relativ minimale.

Figura 64. Crash aplicație

4.2 Performanța și scalabilitatea

Proiectul în majoritatea rulărilor a funcționat în mod eficient. S-au mai observat pe alocuri anumite încărcări de memorie datorate compilării ce îl preceda. De cele mai multe ori, o rulare adițională a aplicației gata compilată, a rezultat în degajarea problemelor existente.

Se poate observa că aplicația nu afectează simțitor resursele disponibile, existând o diferență minimală, chiar în marja de eroare. În fapt, consumul memoriei RAM este cu puțin mai ridicat iar resursele procesorului sunt în același prag.

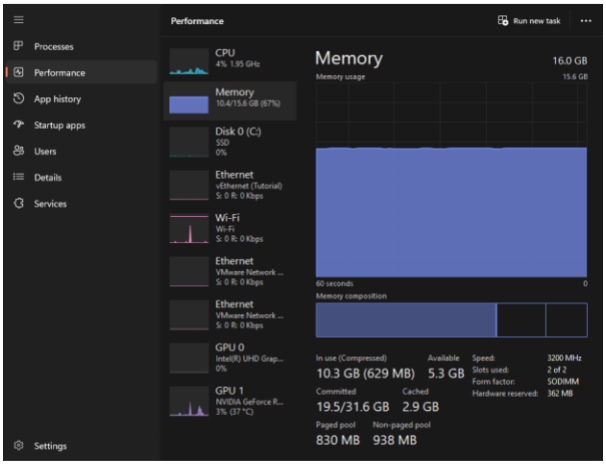


Figura 65. Nivel resurse inițial

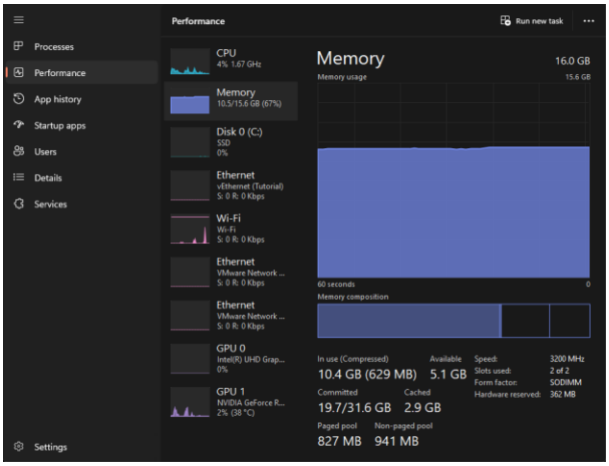


Figura 66. Nivel resurse la rulare

4.3 Analiza securității

- Vulnerabilitatea PostgreSQL „CVE-2023-5869” afectează securitatea întrucât „permite utilizatorilor autentificați ai bazei de date să execute cod arbitrar prin lipsa verificărilor de depășire (tr. «overflow check») în timpul modificării valorii vectorului SQL. Problema este cauzată de depășirea a unui număr întreg în timpul modificării vectorului, unde un utilizator de la distanță poate declanșa depășirea prin furnizarea de date special concepute. Se poate astfel, executa un cod arbitrar pe sistemul țintă, permițând utilizatorilor să scrie octeți arbitrari în memorie și să citească extensiv memoria serverului”[9].

[9] https://www.cvedetails.com/cve/CVE-2023-5869/

Gradul de vulnerabilitate este reprezentat de scorul „8.8” (valoare ridicată) conform „NIST” și „Red Hat, Inc.”. Afectează versiunile cuprinse între „11.0” și „16.0”.

- Vulnerabilitatea Qt „CVE-2023-34410” afectează „validarea certificatului pentru TLS nu ia întotdeauna în considerare dacă rădăcina unui lanț (tr. «root of chain») este un certificat CA configurat”[10].
  Gradul de vulnerabilitate este reprezentat de scorul „7.5” (valoare ridicată), conform „NIST”. Afectează versiunile mai mici decât „5.15.15”, cele încadrate între „6.0.0” și „6.2.10”, dar și cele între „6.3.0” și „6.5.3”.

- Vulnerabilitatea Qt „CVE-2023-32763” afectează „clasa «QTextLayout» întrucât când un fișier SVG cu o imagine în interiorul acestuia este redat, poate fi declanșată o depășire a memoriei (tr. «buffer overflow»)”[11].
  Gradul de vulnerabilitate este reprezentat de scorul „7.5” (valoare ridicată), conform „NIST”. Afectează versiunile mai mici decât „5.15.15”, cele încadrate între „6.0.0” și „6.2.9”, dar și cele între „6.3.0” și „6.5.1”.

[10] https://www.cvedetails.com/cve/CVE-2023-38197/
[11] https://www.cvedetails.com/cve/CVE-2023-32763/

5 Concluzii

Crearea unei aplicații multiplatformă, prin implementarea unei singure soluții cu tematica unui sistem de tip portal generare sesizări și petiții, denumit concret „RaportCivic”, aduce un progres semnificativ în ceea ce reprezintă domeniul civic, promovând astfel participarea civică și forța de reacție a societății în raport cu problemele de zi cu zi.

Proiectul permite cetățenilor să contribuie activ la îmbunătățirea comunităților lor, oferind o interfață plăcută și extrem de elegantă vizual, cu o simplicitate a navigării, ce permite utilizatorului să stabilească un mecanism eficient al raportării către autorități. Acest fapt, încurajează responsabilitatea și deschiderea cetățenilor, invocând un sentiment de datorie comună față de cauză.

Pe parcursul dezvoltării și implementării proiectului, la o astfel de scală, au fost prezente câteva inadvertențe, dificultăți tehnice cât și probleme de proiectare, printre care se enumeră confidențialitatea și securizarea datelor, imprimarea unui design eficient și trasarea unui proces ce este cât mai rapid și complet, în raport cu timpului consumat. Însă, avantajele, precum bunăstarea și satisfacția comunității rezultate surclasează fără orice dubiu, în raport de echivalență, resursele fizice alocate în proiectare.

Proiectul a constituit integrarea și adaptarea framework-ului Qt, ce a adus în prim-plan o dezvoltare corespunzătoare, fără prea mult efort. Utilizarea limbajului C++ în planul inferior, a inoculat aplicației paradigma unei eficiențe crescute, permițând utilizarea proiectului în toate scenariile posibile.

În plus, adaptarea unui sistem cloud, Aiven.io, ce conține baza de date PostgreSQL, a permis, de asemenea, independența proiectului față de încă o componentă suplimentară, dar, în același timp având cele mai eficiente resurse, ce garantează funcționalitatea permanentă a soluției actuale.

Privind în perspectivă, pot exista mai multe modalități posibile de lărgire și îmbunătățire a funcționalităților aplicației prezente. Prin combinarea unui sistem de asistență digital, de tip „AI

chat”, cu un sistem de comunicare cu voluntari, ori chiar un forum integrat, devine posibilă extinderea eficienței de lucru a cetățenilor si activiștilor civici, făcând procesul de raportare mult mai complet din punctul de vedere al calității.

O altă înșiruire de caracteristici, de această data, tehnice, ce pot îmbunătăți aplicația pot fi cele precum implementarea sistemului de resetare a parolei, adăugarea printr-un serviciu de tip e-mail API cu „SMTP”, chiar și în același plan, confirmarea înregistrării de utilizator, adăugarea unui panou de control pentru administrator, afișare de statistici pentru utilizatori și statistica completă care poate fi observată tot de administrator, crearea unui Git ce permite apelarea pentru verificarea versiunii, implementarea a mai multor componente predefinite în meniurile aplicației, dar și îmbogățirea listei de cazuri și șabloane existente.

Pentru a ține pasul cu mediul ce e în continuă schimbare, este vital ca proiectul să fie cât mai flexibil și receptiv la integrarea de soluții îndrăznețe, astfel servind drept o unealtă completă pentru reducerea diferențelor între autoritățile locale și cetățeni, promovând un mediu mai deschis și cât mai receptiv pentru o societate contemporană cât mai modernă.

6 Bibliografie

Berik I. Tuleuov & Ademi B. Ospanova. (2024). *Beginning C++ Compilers: An Introductory Guide to Microsoft C/C++ and MinGW Compilers*. Apress.

Google. (f.a.). *What is a relational database?* https://cloud.google.com/learn/what-is-a-relational-database

Jürgen Bocklage-Ryannel, Cyril Lorquet, & Johan Thelin. (2021). *The Qt 6 Book*. The Qt Company.

Luca Ferrari & Enrico Pirozzi. (2023). *Learn PostgreSQL*. Packt.

Rafal Swidzinski. (2022). *Modern CMake for C++*. Packt.