



UNIVERSITATEA DIN
BUCUREȘTI

FACULTATEA
DE
MATEMATICĂ ȘI INFORMATICĂ



Lucrare de disertație

“PLATFORMĂ ANDROID PENTRU STUDENȚII FMI”

ABSOLVENT

Vîlceanu Sonia-Nicoleta

COORDONATOR ȘTIINȚIFIC

Prof. Dr. Ipate Florentin

București, septembrie 2023

REZUMAT

Proiectul dezvoltat este o aplicație mobilă pentru dispozitivele cu sistemul de operare *Android*, elaborată utilizând platforma *Android Studio*.

Tema a fost aleasă pornind din nevoile identificate în propria activitate profesională studențească. O variantă mobilă a platformei decomisionate *Moodle* din cadrul vechiului site web *fmi.unibuc.ro*, și a actualei platforme *carnet.unibuc.ro*, ar aduce o disponibilitate mai rapidă și mai simplă a informațiilor universitare ce sunt puse la dispoziție studenților.

Aplicația facilitează experiența studențească prin furnizarea notificărilor la încărcarea sau actualizarea notelor și cursurilor din orar, împreună cu datele evenimentelor calendaristice academice și examenelor de către personalul didactic. Totodată, se pot urmări anunțuri și activități de actualitate de la secretariat, la fel și oferte de voluntariat din cadrul Asociației Studenților de la Matematică și Informatică, alături de oportunități de practică și de internship-uri.

Cum în zilele contemporane fiecare persoană are accesibilitate aproape constantă la cel puțin un dispozitiv mobil datorită portabilității facilitate de dimensiunile mici ale acestora, crește astfel preferința utilizatorilor pentru o alternativă mobilă de prezentare a informațiilor universitare.

Răspândirea largă a dispozitivelor mobile aduce de la sine și o gamă largă de materiale de informare educațională, accesibile atât în mediu online cât și fizic, care ne pot îndruma în conceperea unei aplicații pentru acestea.

Crearea aplicației își are ca scop și facilitarea acumulării de cunoștințe noi în domeniul aplicațiilor *Android*, peste progresarea aptitudinilor deja existent deținute în domeniul general al programării.

Pentru această inițiativă vom contura procesul de concepere și creare a unei astfel de aplicații, etapele parcurse și obstacolele întâmpinate alături de soluțiile întrebuintate.

ABSTRACT

The developed project is a mobile application for *Android* operating system devices, crafted using the *Android Studio* platform.

The theme was chosen based on the needs identified in my own professional student activity. A mobile version of the decommissioned *Moodle* platform of the old *fmi.unibuc.ro* website, and the current *carnet.unibuc.ro* platform, would provide faster and easier access to university information available for students.

The application enhances the student experience by providing notifications when uploading or updating grades and class schedules, along with academic calendar events and exams by the teaching staff. Additionally, it can also track current announcements and activities from the secretariat, as well as volunteer opportunities from the Mathematics and Computer Science Student Association, along with internship and placement opportunities.

As nowadays every person has almost constant accessibility to at least one mobile device, due to the portability facilitated by their small size, this increases user preference for a mobile alternative in presenting university information.

The widespread use of mobile devices also brings forth a wide range of educational information materials, accessible in both online and physical environments, which can guide us in designing an application for them.

The creation of the application also aims to facilitate the acquisition of new knowledge in the field of *Android* applications, building upon existing skills in the general field of programming.

For this initiative we will outline the process of designing and creating such an application, the steps taken and challenges encountered along with the employed solutions.

CUPRINS

Introducere.....	6
Capitolul 1: Noțiuni Introductive.....	8
Limbajul Java.....	9
Obiectivul lucrării.....	10
Rezumat contribuție proprie.....	10
Etapale dezvoltării.....	10
Capitolul 2: Noțiuni teoretice.....	12
2.1 Emulatoare.....	12
2.2 Elemente ale interfeței grafice.....	13
2.3 Activități și Fragmente.....	14
2.3.1 Activități.....	14
2.3.2 Fragmente.....	17
2.3.3 Transmiterea datelor între activități.....	18
2.4 Metode de persistență a datelor.....	18
2.4.1 Shared Preferences.....	19
2.4.2 SQLite și SQLiteHelper.....	21
2.4.3 SQLite și Room.....	24
Capitolul 3: Contribuția proprie în implementare.....	27
3.1 Structura bazei de date.....	27
3.2 Organizarea interfeței.....	29

3.2.1 Meniul de autentificare.....	29
3.2.2 Secțiunile meniului principal.....	29
Carnet.....	30
Orar.....	32
Calendar Academic.....	33
Anunțuri și Activități.....	35
Internship și Voluntariat.....	37
Informații generale.....	38
Adăugare cont nou.....	39
3.2.3 Bara de opțiuni secundare.....	40
Căutare.....	41
Profil.....	41
Notificări.....	41
Deconectare.....	43
Capitolul 4: Dificultăți și îmbunătățiri.....	44
4.1 Dificultăți întâmpinate.....	44
4.2 Îmbunătățiri.....	45
Concluzie.....	47
Bibliografie.....	48

INTRODUCERE

Evoluția aplicațiilor mobile a fost o călătorie transformativă în peisajul digital. Pornind de la aplicații de bază care ofereau funcționalități rudimentare, cererea pentru experiențe mai sofisticate și mai centrate pe utilizator s-a extins rapid. Aplicațiile mobile au depășit sfera comunicării și a divertismentului pentru a cuprinde diverse sectoare, cum ar fi sănătatea, finanțele, educația și multe altele.

O aplicație mobilă oferă acces rapid și ușor la informații și caracteristici importante direct de pe smartphone-ul sau tableta utilizatorului. Această comoditate elimină necesitatea de a deschide un browser web și de a naviga către un site, făcându-l mai accesibil din mers.

Multe aplicații *Android* sunt concepute pentru a funcționa offline, permițând utilizatorilor să acceseze anumite funcții și conținut chiar și atunci când nu au o conexiune activă la internet. Acest lucru poate fi deosebit de util pentru sarcini precum citirea articolelor sau navigarea hărților.

Aplicațiile *Android* pot fi personalizate pentru a se alinia cu preferințele utilizatorului. Adesea, utilizatorii pot ajusta setările, personaliza temele și configura aplicația în funcție de nevoile lor.

O aplicație mobilă poate furniza notificări "*push*" utilizatorilor, alertându-i cu privire la actualizări, știri, evenimente sau modificări importante. Acest sistem de notificare în timp real asigură informarea promptă a utilizatorilor fără a verifica în mod activ site-ul web. Utilizatorii își pot seta preferințele pentru a primi actualizări despre subiectele care îi interesează. De asemenea, se pot integra cu alte funcționalități ale dispozitivului, cum ar fi galeria de imagini, email-ul sau calendarul.

Pentru administratorii de site-uri, existența unei aplicații dedicate poate încuraja un angajament mai mare din rândul utilizatorilor. Notificările, funcțiile interactive și accesul offline pot menține utilizatorii mai implicați și mai conectați la informațiile disponibile pe site.

Cele două mari nume din spațiul sistemelor de operare mobile sunt *Android* și *iOS*. *Android*, dezvoltat de *Google*, deține cota de piață majoritară datorită disponibilității sale pe o gamă largă

de dispozitive de la diverși producători. Pe de altă parte, *iOS*, dezvoltat de *Apple*, este cunoscut pentru integrarea sa robustă cu hardware-ul și prezența sa exclusivă pe dispozitivele *Apple*.

Companiile care dezvoltă aplicații mobile, fie că sunt mari sau mici, au nevoie de un set puternic de instrumente pentru a crea aplicații eficiente și complexe. *Android Studio*, conceput de *Google*, este principalul *IDE* (mediu de dezvoltare integrat) pentru crearea de aplicații *Android*. Acesta oferă dezvoltatorilor o suită completă de instrumente pentru proiectarea, codificarea, testarea și depanarea aplicațiilor. Unul dintre avantajele sale majore este integrarea profundă cu ecosistemul *Android*, ceea ce îl face să fie alegerea de bază pentru dezvoltarea de astfel de aplicații. Companiile mari folosesc adesea *Android Studio* datorită familiarității sale și a resurselor extinse disponibile în comunitatea de dezvoltare *Android*.

În timp ce *Android Studio* este conceput special pentru dezvoltarea de aplicații *Android*, alte alternative multi-platformă precum *Flutter* conceput de *Google* și *React Native* conceput de *Facebook*, au câștigat popularitate. Aceste *framework*-uri permit dezvoltatorilor să scrie codul o singură dată și să îl implementeze atât pe platformele *Android*, cât și pe cele *iOS*, reducând timpul și efortul de dezvoltare. Cu toate acestea, este posibil ca acestea să nu ofere același nivel de optimizări specifice platformei ca *Android Studio*.

Din aceste motive am optat pentru acest *IDE* în proiectarea aplicației noastre. *Android Studio* oferă o curbă de învățare mai directă și mai concentrată pentru dezvoltarea aplicațiilor *Android* datorită specializării pe un singur sistem de operare. Astfel, este mult mai puțin copleșitor pentru începători în comparație cu *framework*-urile *cross-platform* care se adresează mai multor platforme.

Organizarea pe capitole a lucrării:

Capitolul 1 - Concepte preliminare, percepție generală asupra structurii, scopul și abstract al contribuției proprii

Capitolul 2 - Concepte teoretice și componente specifice platformei *Android*

Capitolul 3 - Studiu amănunțit pentru contribuția proprie în proiectarea aplicației

Capitolul 4 - Concluzii, dificultăți întâmpinate și posibile îmbunătățiri ce se pot aduce aplicației

CAPITOLUL 1: NOȚIUNI INTRODUCTIVE

Aplicația prezentată în această lucrare este o versiune demo a unei variante complete ce s-ar putea implementa, însă conține toate funcționalitățile de bază necesare și unei aplicații complet finisate pentru a-și îndeplini scopul propus.

Aceasta își ar avea folosința în mare parte în activitatea studenților, aceștia fiind ținta principală a fluxului de informații comunicate, dar ar acoperi un necesar și pentru personalul didactic și didactic auxiliar.

Astfel, vom avea trei tipuri de conturi, unul pentru studenți, unul pentru personalul didactic și unul pentru cel didactic auxiliar sau administrativ. Pe lângă aceasta, am ales să structurăm aplicația în șase secțiuni principale, ce vor avea întrebuințări mai mult sau mai puțin comune pentru cele trei tipuri de conturi. Secțiunile au fost denumite astfel: Carnet, Orar, Calendar, Anunțuri și Activități, Internship și Voluntariat și Informații generale. Aici se adaugă o a șaptea secțiune, denumită Adăugare cont nou, ce este întrebuințată doar contului de administrator.

Contul unic dedicat personalului didactic auxiliar are cele mai multe opțiuni de adăugare și editare asupra aplicației. La sub-secțiunea Anunțuri din Anunțuri și Activități, are dreptul de a adăuga postări scurte, de obicei provenite de la secretariat, precum detalii despre înscrierile la reexaminari și mărimi de note, restituirile de taxe de înscriere pentru admitere sau distribuirile locurilor de cazare în cămin. Asemenea, este și sub-secțiunea Activități, care servește același scop de informare, însă diferența între cele două este dată de scopul conținutului lor, acesta fiind în schimb orientat spre resurse educaționale și activități auxiliare organizate de facultate. Aceasta poate să cuprindă noutăți despre conferințe ale unor firme, prezentări de lucrări de doctorat, susțineri de teze sau prelegeri ale unor asociații, ce ar fi prezentate în cadrul universității.

Tot pe acest model a fost concepută și secțiunea Internship și Voluntariat, unde pot fi postate scurte anunțuri cu detalii privind oferte de practică și internship, aduse de firme către elevii facultății, precum ofertele prezentate în cadrul târgului anual de job-uri al facultății. Sub-secțiunea Voluntariate este dedicată Asociației Studenților de la Matematică și Informatică (ASMI), unde

pot fi încărcate detalii de participare pentru activitățile acestora cu interes pentru studenții din afara asociației.

Administratorii au acces și la secțiunea de Orar unde pot completa sau modifica orele pentru cursurile grupelor, la secțiunea de Carnet, de unde pot bloca sau debloca note în cazul depunerii unei cereri de reexaminare, și secțiunea de Calendar Academic unde pot rezerva evenimente pe zile. Exemple de astfel de evenimente cuprind perioadele de sesiune, de înscriere în anul universitar, de alegere a materiilor opționale, de achitare a taxelor anuale și multe altele.

Conturile de student vor fi notificate cu fiecare ocazie când au loc astfel de schimbări în oricare din secțiunile precedent precizate. Aceștia au capacitatea de a urmări orarul tuturor grupelor la secțiunea Orar, posibilitatea de a vedea Calendarul Academic și citi secțiunile de Anunțuri și Activități și Internship și Voluntariat, chiar având opțiunea să aplice la voluntariatele afișate. Spre deosebire de conturile de administrator, la conturile de studenți secțiunea Carnet este dedicată pentru urmărirea situației școlare la toate disciplinele din cursul anilor universitari.

Aceste secțiuni au întrebuințări și pentru profesori, aceștia putându-și vedea propriile cursuri din orar, putând adăuga note în carnet, seta examene în calendar și date pentru susținerea contestațiilor.

Gruparea țintă pentru care aplicația noastră este concepută sunt studenții și profesorii de la domeniul Informatică din cadrul Facultății de Matematică și Informatică, atât programul de studiu Informatică al licenței cât și toate programele de master. Aplicația s-a limitat doar la aceste categorii întrucât o extindere a scopului aplicației pentru a acoperi și celelalte specializări ar fi creat o arie de operare mult prea largă pentru a putea fi administrată consistent de o versiune demo a aplicației.

Limbajul Java

În domeniul *Android Studio*, aplicațiile pot fi scrise în două limbaje de programare proeminente: *Java* și *Kotlin*. *Java*, un limbaj bine stabilit, este de mult timp fundamental în dezvoltarea aplicațiilor *Android*, iar utilizarea sa pe scară largă este subliniată de o serie de avantaje. Familiaritatea sa, adeseori datorată predării sale în timpul cursurilor universitare, îl poziționează ca o alegere naturală pentru dezvoltatori, în special pentru începători. Datorită popularității ridicate

a acestui limbaj, resursele ample, tutorialele și documentația disponibile pentru *Java* în contextul dezvoltării *Android*, oferă un sistem de sprijin solid pentru cursanții și dezvoltatorii independenți.

Cu toate acestea, merită menționat faptul că *Java* vine cu unele limitări, spre exemplu, lipsa unor anumite caracteristici moderne întâlnite în limbaje precum *Kotlin*. *Kotlin* apare ca o alternativă cu sintaxa sa concisă, mecanismele de siguranță contra null și caracteristicile moderne, cum ar fi funcțiile de extensie și clasele de date. Cu toate acestea, decizia de a utiliza *Java* pentru aplicația noastră se bazează pe familiaritatea existentă cu acest limbaj. Această familiaritate adunată din timpul studiilor universitare, poate simplifica procesul de dezvoltare, permițând să aplicăm cunoștințele dobândite la crearea aplicației *Android*.

Obiectivul lucrării noastre este de a concepe o aplicație *Android*, utilizând platforma *Android Studio* și limbajul *Java*, ce ar servi atât studenților cât și personalului didactic pentru a concentra și ușura accesul la datele universitare, anunțuri educaționale și resurse academice.

Contribuția proprie poate fi rezumată la proiectarea completă a aplicației, de la conceptualizare până la implementare, pornind de la conturarea teoretică a structurii bazei de date, continuând cu planificarea interfeței grafice vizuale, urmată de implementarea propriu-zisă, asigurând coeziunea elementelor vizuale cu cele de infrastructură și mai departe cu cele de stocare a datelor. Ulterior s-a asigurat fiabilitatea și funcționalitatea eficientă și completă prin numeroase teste asupra componentelor elaborate.

Etapele dezvoltării

1. Deciderea asupra unei palete de culori, importarea unor *asset*-uri grafice deja existente în librăriile *Android Studio* (denumite *Drawables*, mai concis iconițele pentru meniuri și opțiuni) și crearea de *asset*-uri reutilizabile de-a lungul proiectului, pentru a conferi o consistență aspectului aplicației.
2. Implementarea unui pagini de înregistrare și autentificare în aplicație pentru utilizatori.

3. Crearea unui meniu principal ce va conține secțiunile aplicației: Carnet, Orar, Calendar Academic, Anunțuri și Activități, Internship și Voluntariat, Informații Generale și Adăugare cont nou.
4. Crearea unui bari superioare ce conține opțiuni de importanță secundară: Profil, Notificări, Căutate și Delogare.
5. Crearea bazei de date și a tabelor pentru stocarea informațiilor și implementarea claselor necesare pentru manipularea datelor.
6. Implementarea secțiunilor din meniul aplicației, respectându-se următoarea ordine pentru fiecare în parte: aranjarea elementelor interfeței grafice în pagină pentru toate cele trei tipuri de conturi, integrarea funcționalităților propuse, conectarea acestora cu baza de date și apoi testarea acestora.
7. Extinderea bazei de date prin crearea de tabele și coloane noi pentru a satisface evoluția constantă a sarcinilor ce dorim să le realizăm.
8. Finisare, ajustări sporadice ale interfeței, bug fixing și adăugarea unor funcționalități minore.

CAPITOLUL 2: NOȚIUNI TEORETICE

2.1 Emulatoare

Un emulator în contextul dezvoltării *Android*, este un program software care reproduce comportamentul și funcționalitățile unui dispozitiv *Android* fizic pe un computer gazdă. Acesta permite dezvoltatorilor să simuleze diverse configurații de dispozitive *Android*, dimensiuni ale ecranului și niveluri API, ajutând la testarea și depanarea aplicațiilor *Android* fără a fi nevoie de un dispozitiv fizic^[1]. Emulatorul *Android* furnizat de *Android Studio* oferă un mediu virtual pentru rularea aplicațiilor, suportând atât emularea bazată pe software, cât și cea bazată pe hardware.

Emulatoarele din *Android Studio* pot fi clasificate în două tipuri, în funcție de tehnologia care stă la baza lor. Emulatoarele bazate pe hardware utilizează resursele hardware ale computerului gazdă pentru a emula un dispozitiv *Android*, oferind o aproximare mai apropiată de comportamentul dispozitivului real. În schimb, emulatoarele pe bază de software replică componentele software ale unui dispozitiv *Android*, oferind adesea o mai bună flexibilitate și gestionare a resurselor^[2].

Caracteristicile acestora cuprind capacitatea de a emula o gamă largă de configurații de dispozitive și niveluri API, ceea ce face ca testarea compatibilității să fie mai ușor de gestionat. Acestea oferă instrumente de depanare, funcții de monitorizare a performanței și opțiuni de simulare a rețelei, facilitând identificarea și rezolvarea problemelor din codul aplicației^{[1][2]}.

Printre avantajele utilizării emulatoarelor se numără rentabilitatea acestora, deoarece elimină necesitatea de a achiziționa mai multe dispozitive fizice pentru testare. În plus, acestea oferă timpi de implementare mai rapizi, permițând lansarea și testarea instantanee a aplicației^[1].

Cu toate acestea, emulatorii prezintă anumite dezavantaje. Este posibil ca aceștia să nu reproducă cu exactitate comportamentul dispozitivului real, ceea ce duce la variații în ceea ce privește performanța, experiența utilizatorului și problemele de compatibilitate. Emulatoarele pot consuma resurse de sistem semnificative, putând provoca conflicte cu alte aplicații care rulează.

Avantajele testării pe dispozitive reale cuprind evaluarea interacțiunilor tactile și funcționalitatea camerei. Depanarea pe un dispozitiv real poate descoperi probleme care ar putea să nu iasă la suprafață pe un emulator, asigurând o evaluare mai precisă a experienței utilizatorului.

Cu toate acestea, testarea pe dispozitive reale prezintă și dezavantaje. În primul rând, poate fi costisitor să se obțină diverse dispozitive fizice pentru a testa diferite configurații. Mediul de testare este și el mai puțin controlat, configurațiile software putând varia de la dispozitiv la dispozitiv.

2.2 Elemente ale interfeței grafice

Structura și aspectul interfeței cu care utilizatorul va interacționa într-o aplicație *Android* sunt definite de fișierele de layout XML, fișiere scrise în Extensible Markup Language. Ele oferă o modalitate declarativă de a descrie dispunerea elementelor de interfață, relațiile și atributele acestora^[3].

Fișierele XML au o structură ierarhică în care elementele interfeței grafice sunt cuibărite unele în altele. Elementul rădăcină este, de obicei, un element recipient de *layout*, cum ar fi `<LinearLayout>` (dispune elementele interfeței grafice într-o secvență liniară orizontală sau verticală), `<RelativeLayout>` (poziționează elementele *UI* în raport unul față de celălalt sau față de elementul părinte) sau `<ConstraintLayout>` (utilizează constrângeri pentru a defini pozițiile elementelor și relațiile dintre ele)^{[4][6]}. *Layout*-urile pot include atribute pentru a defini proprietăți precum lățimea, înălțimea, marginile sau alinierea^{[5][6]}.

Elemente des utilizate de UI în proiectul nostru:

- *Button*: Declanșează acțiuni atunci când este acționat prin atingere sau click^[7]
- *FloatingActionButton*: Are aceeași funcționalitate ca butoanele simple, însă sunt plasate separat deasupra *layout*-ului. De obicei sunt folosite pentru o acțiune primară din pagină^[8]
- *Checkbox*: Permite utilizatorilor să facă selecții binare^[9].
- *TextView*: Casetă de text ce se afișează pe ecran^[10].

- *EditText*: Casetă de text ce acceptă date introduse de utilizator^[11].
- *ScrollView*: Activează derularea conținutului din interiorul său^[12]
- *TabLayout*: Furnizează file pentru navigare^[13].
- *Spinner*: Afișează o listă derulantă de opțiuni^[14].
- *ViewPager*: Facilitează navigarea prin glisare orizontală^[15].
- *MaterialCardView*: Îmbunătățește atractivitatea vizuală și interacțiunea cu containerele de tip card.^[16]
- *ImageView*: Afișează imagini de tip *Bitmap* sau *Drawable*^[17]
- *TableLayout*: Organizează conținutul în rânduri și coloane^[18].
- *AppBarLayout*: un *LinearLayout* vertical ce oferă navigare flexibilă și integrare dinamică a antetului^[19].
- *DrawerLayout*: Simplifică navigarea cu meniuri glisante accesate din lateralele ecranului^[20].
- *NavigationView*: Oferă opțiuni de navigare, folosit des în asociere cu un fișier de *layout* de tip meniu^[21].

2.3 Activități și Fragmente

2.3.1 Activități

Activitățile reprezintă principalul punct al organizării interfeței grafice și a interacțiunii utilizatorului cu aceasta. În esență, o activitate reprezintă un ecran singular în cadrul unei aplicații, servind ca o fereastră prin care utilizatorii acționează asupra conținutului și funcționalităților^[22]. Această modularizare permite dezvoltatorilor să structureze aplicațiile în componente distincte, oferind un flux logic și intuitiv pentru utilizatori.

Clasa *Activity* are o importanță semnificativă în cadrul unei aplicații *Android*, constituind un aspect fundamental al modelului de aplicație al platformei în ceea ce privește lansarea și asamblarea activităților. Spre deosebire de abordările de programare în care aplicațiile sunt declanșate prin

intermediul unei metode *main()*, *Android* urmează o abordare distinctă. Codul se inițializează în cadrul unei instanțe de activitate prin invocarea unor metode de *callback* desemnate care corespund unor faze distincte ale ciclului de viață al acesteia^[23].

În cadrul sistemului *Android*, activitățile sunt orchestrate prin stive de activități. La începerea unei noi activități, aceasta ocupă de obicei o poziție deasupra stivei existente, preluând rolul activității active și operaționale din acel moment. Activitatea care a precedat-o se află în mod constant sub ea în cadrul stivei și își recapătă importanța doar atunci când activitatea nou inițiată își încheie execuția. Interfața poate afișa fie o singură stivă de activități, fie numeroase, fiecare dintre acestea fiind distinct observabilă pe ecran^[24].

O activitate poate trece prin patru diferite etape sau stări: activă, vizibilă, oprită și eliminată, iar fiecare dintre acestea este marcată de metode ale ciclului de viață, cum ar fi: *onCreate()*, *onStart()*, *onResume()*, *onPause()*, *onStop()* și *onDestroy()*. Această secvență de stări reflectă domeniul de aplicare al unei activități, cuprinzând crearea, vizibilitatea, interacțiunea și, în cele din urmă, încetarea acesteia^{[23][24]}.

Funcțiile cheie din ciclul de viață al activităților:

onCreate(): Această funcție este invocată la instanțierea unei activități sau a unui fragment. Ea servește ca locație ideală pentru sarcinile de inițializare și pentru configurarea interfeței de utilizator, asigurând un start fără probleme pentru componentă^{[23][24]}.

onStart(): La finalizarea metodei *onCreate()*, activitatea trece în starea "*Started*", marcând vizibilitatea sa pentru utilizator. Acest *callback* încapsulează pregătirile finale ale activității care urmează să fie deschisă^[23].

onPause(): În momentul în care o activitate este pe cale să treacă în fundal, funcția *onPause* oferă posibilitatea de a salva modificările *UI* (*user interface*) și de a elibera resurse, contribuind la gestionarea eficientă a acestora^{[23][24]}.

onResume(): Această funcție este invocată atunci când o activitate se reia dintr-o stare de pauză. Este utilizată pentru a actualiza elementele *UI* și pentru a restabili operațiunile, asigurându-se că aplicația se află într-o stare de răspuns pentru utilizator^[23].

onStop(): Atunci când o activitate nu mai este vizibilă pentru utilizator, se declanșează funcția *onStop*^[23]. Această etapă este deseori utilizată pentru eliberarea resurselor și efectuarea operațiunilor de curățare.

onDestroy(): Funcția *onDestroy* marchează încheierea ciclului de viață al unei activități. Este utilizată pentru curățarea finală și pentru dealocarea resurselor înainte ca activitatea să fie eliminată^[23].

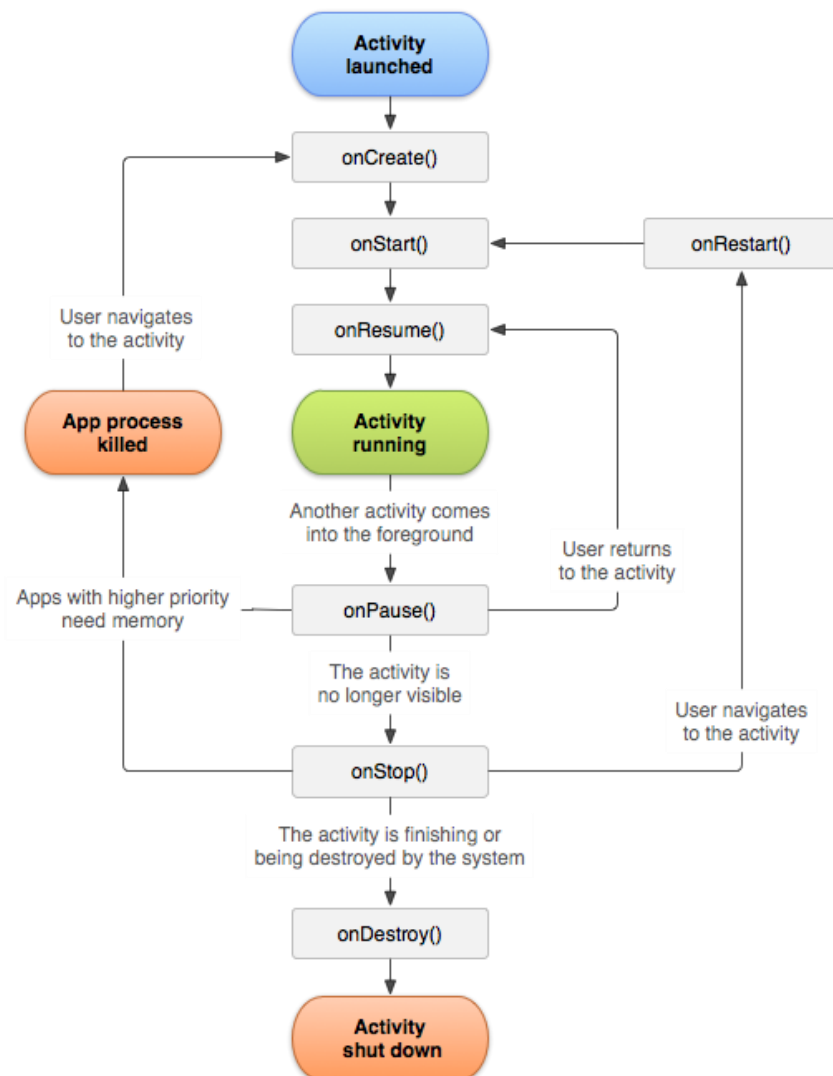


Fig. 1 – Cursul stărilor unei activități^[24]

2.3.2 Fragmente

Un fragment este o componentă modulară ce cuprinde o mică parte din funcționalitatea unei aplicații sau din interfața grafică, care poate fi integrată fără probleme într-o activitate. Clasa *Fragment* prezintă o versatilitate înaltă, adaptându-se la diverse scenarii de utilizare și producând o multitudine de rezultate^{[22][24]}. Aceasta întruchiează o sarcină specifică sau o componentă de interfață care funcționează în contextul unei activități mai ample. Fragmentul și activitatea în care se află sunt strâns legate, făcând fragmentul inseparabil de contextul activității. În timp ce un fragment are propriul său ciclu de viață, acest ciclu de viață rămâne interconectat cu cel al activității asociate. Atunci când activitatea este oprită, nici un fragment conținut în ea nu mai poate fi inițiat, iar atunci când activitatea este eliminată, toate fragmentele afiliate vor fi de asemenea eliminate^[25].

Fragmentele au și ele aceleași funcții cheie utilizate în ciclul de viață: *onCreate*, *onStart*, *onPause*, *onResume*, *onStop*, *onDestroy*^[25], iar, în adiție la acestea, fragmentele utilizează și funcția *onCreateView*. Ea definește prezentarea vizuală a conținutului fragmentului^[25].

Două moduri de abordare predominante apar atunci când se ia în considerare numărul de activități dintr-o aplicație. Abordarea cu o singură activitate presupune utilizarea fragmentelor pentru a compartimenta diferite secțiuni ale interfeței grafice în cadrul unei singure activități. Această alegere de proiectare promovează modularizarea și simplifică navigarea. Pe de altă parte, abordarea cu mai multe activități implică existența unor activități distincte pentru diferite ecrane, facilitând navigarea tradițională. Și în aceasta situație este totuși desemnată o activitate principală, anume cea pe care aplicația o va încărca mereu prima și în care se va deschide^[23]. Pentru aplicația noastră s-a ales a doua modalitate, crearea fiecărei secțiuni a aplicației noastre ca o activitate individuală fiind un flux mult mai natural din punctul de vedere al modului în care utilizatorul ar naviga aplicația noastră.

2.3.3 Transmiterea datelor între activități

Intent-ul este o descriere conceptuală a unei acțiuni care urmează să fie executată, ce oferă un mecanism pentru stabilirea de conexiuni dinamice în timpul execuției între coduri din diverse aplicații sau între diferitele componente ale aceleiași aplicații. Funcția sa predominantă constă în inițierea activităților, servind drept obiect de legătură între aceste activități. Practic, un *Intent* este asemănător unei construcții de date care înglobează o descriere abstractă a unei acțiuni care urmează să fie executată^[26].

Metoda utilizată pentru a face un transfer de variabile între activități este *putExtra()*. Această metodă permite dezvoltatorilor să atașeze informații suplimentare la un obiect *Intent*. Ea grupează date precum șiruri de caractere, numere întregi, booleeni și chiar obiecte mai complexe, cum ar fi instanțele *Parcelable* sau *Serializable*^[27].

2.4 Metode de persistență a datelor

În dezvoltarea *Android*, persistența datelor se referă la metodele utilizate pentru a stoca și gestiona datele în diferite stări ale unei aplicații.

Pe măsură ce aplicația trece prin diferite stări, interacțiuni ale utilizatorilor și configurații ale dispozitivelor, necesitatea de a păstra și gestiona datele în mod consecvent devine extrem de importantă. Prin utilizarea metodelor de persistență a datelor, dezvoltatorii permit aplicațiilor să rețină preferințele utilizatorilor, să mențină conținutul generat de utilizatori și să facă tranziția fără probleme între diferite sarcini și ecrane.

Există diverse tehnici disponibile pentru persistența datelor, iar fiecare dintre acestea este concepută pentru a răspunde unui spectru de cazuri de utilizare și tipuri de date. Fie că este vorba de păstrarea preferințelor utilizatorului, de stocarea fișierelor, de gestionarea datelor structurate sau de schimbul de informații între aplicații, fiecare metodă este adaptată pentru a răspunde unor cerințe specifice.

Metode de persistență a datelor puse la dispoziție în Android Studio:

- *Shared Preferences*: Stocază perechi cheie-valoare simple pentru preferințele și setările utilizatorului^[28].
- Stocare internă: Ajută la protejarea fișierelor de date private specifice aplicațiilor în cadrul dispozitivului^[28].
- Stocare externă: Stocază fișierelor publice pe o memorie externă partajată, cum ar fi un card SD^[28].
- Bază de date *SQLite*: Gestionează eficient datele structurate cu ajutorul capacităților unei baze de date relaționale^[28].
- Furnizori de conținut: Facilitează partajarea securizată a datelor între aplicații^[28].

Pentru aplicația noastră am utilizat *Shared Preferences* și *SQLite*.

2.4.1 Shared Preferences

Shared Preferences oferă o soluție ușoară și eficientă pentru stocarea unor cantități mici de date, în principal sub forma unor perechi cheie-valoare^{[28][29]}. Această abordare de stocare este utilizată cel mai des pentru a păstra preferințele, setările și valorile de configurare simple ale utilizatorului.

Fiecare înregistrare de date este asociată cu o cheie unică, ceea ce facilitează recuperarea și gestionarea cu ușurință. Tipurile de date acceptate includ Integer, String, boolean și float. Datele stocate cu ajutorul *Shared Preferences* persistă dincolo de închiderea aplicațiilor și de repornirea dispozitivului, asigurând continuitatea alegerilor și a setărilor utilizatorului^[30].

Confidențialitatea și securitatea este asigurată deoarece datele stocate în *Shared Preferences* sunt limitate la domeniul privat al aplicației. Spre deosebire de alte metode de stocare, aceste date nu sunt direct accesibile altor aplicații din dispozitiv sau altor utilizatori ai aceleiași aplicații, păstrând integritatea datelor și confidențialitatea utilizatorului^[28].

În spate, *Shared Preferences* utilizează fișiere XML pentru stocarea datelor, ceea ce este deosebit de eficient pentru seturi de date mici.

Exemplu de utilizare Shared Preferences^{[29][30]}:

```
// Obținerea obiectului SharedPreferences
SharedPreferences sharedPreferences =
    getSharedPreferences("Student", Context.MODE_PRIVATE);

//obiectul obținut este unic, toate valorile sunt stocate într-
un singur fișier.

// Obținerea editorului Shared Preferences
SharedPreferences.Editor editor = sharedPreferences.edit();

// Adăugăm date în editor
editor.putString("nume", "Popescu Mihai");
editor.putInt("an", 2);
editor.putBoolean("taxa", false);
editor.putFloat("medie", 9.45);

// Stocare date definite
editor.apply();
```

În timp ce *Shared Preferences* oferă avantaje notabile, ele au și limitări. Nu sunt potrivite pentru manipularea structurilor de date complexe sau pentru gestionarea unor seturi de date mari, din cauza metodei lor de stocare de tip cheie-valoare. În plus, nu dispun de capacitățile de interogare structurată ale unor baze de date mai complexe, precum *SQLite*, iar suportul lor este limitat la tipurile de date de bază. De asemenea, datele din fișierul unic nu sunt criptate, pentru criptarea datelor fiind necesară utilizarea *EncryptedSharedPreferences*^[29].

2.4.2 *SQLite* și *SQLiteHelper*

SQLite este un motor de baze de date *SQL* încorporat care are, de asemenea, proprietățile de a fi autonom, serverless și tranzacțional^[31].

SQLite își are folosința în multitudinea de scenarii ce necesită stocarea structurată a datelor. Conceput pentru gestionarea datelor relaționale, *SQLite* este potrivit pentru aplicațiile care necesită relații complexe de date. De la catalogarea conținutului generat de utilizatori și stocarea setărilor specifice aplicațiilor până la menținerea *cache*-urilor locale și gestionarea autentificării utilizatorilor, *SQLite* oferă un *framework* consistent pentru organizarea și gestionarea diverselor tipuri de informații.

SQLite funcționează ca o bibliotecă de sine stătătoare, fără a necesita o instalare externă sau o configurare a serverului. Este încorporată direct în aplicație, minimizând dependențele și simplificând implementarea.

O altă caracteristică distinctivă a *SQLite* este suportul său tranzacțional. Acesta aderă la proprietățile *ACID* (*Atomicity, Consistency, Isolation, Durability*) ale tranzacțiilor, asigurând integritatea datelor chiar și în cazul unor defecțiuni ale sistemului. Acest nivel de fiabilitate este de înaltă folosință, în special în scenariile în care sunt implicate date critice ale utilizatorilor^{[31][32]}.

Bazele de date *SQLite* sunt structurate în jurul conceptelor de bază ale tabelelor, rândurilor și coloanelor. Conform modelului relațional, fiecare tabel reprezintă o entitate distinctă, în timp ce rândurile din cadrul tabelului corespund înregistrărilor individuale. Coloanele definesc atributele asociate cu fiecare înregistrare. Această abordare structurată facilitează stocarea organizată și eficiența a datelor^[31].

Clasa *SQLiteHelper* funcționează ca legătură între baza de date și aplicație, eliminând o mare parte din complexitatea asociată cu crearea, versiunea și actualizările bazei de date. *SQLiteHelper* ajută la crearea bazei de date inițiale, specificând tablele, coloanele și tipurile de date ale acestora. Pe măsură ce aplicațiile evoluează, bazele de date necesită adesea modificări structurale. *SQLiteHelper* este utilizat pentru a simplifica o astfel de gestiune^[33].

Când vine vorba de securitate, în *Android* baza de date a aplicației este stocată în directorul privat al acesteia. Această configurație asigură securitatea datelor, deoarece această zonă desemnată este inaccesibilă în mod implicit altor aplicații sau utilizatorului^[33].

Exemplu deschidere bază de date^{[34][35]}:

```
public class DatabaseHelper extends SQLiteOpenHelper {  
    private static final String DATABASE_NAME =  
    "RoomDatabase.db";  
  
    private static final int DATABASE_VERSION = 1; // versiunea  
    bazei se schimbă atunci când am facut actualizări la structura  
    sa  
  
    public DatabaseHelper(Context context) {  
        super(context, DATABASE_NAME, null, DATABASE_VERSION);  
    }  
  
    @Override  
    public void onCreate(SQLiteDatabase db) {  
        // Crearea tabelelor  
        db.execSQL("CREATE TABLE Students (id INTEGER PRIMARY  
KEY, name TEXT, email TEXT)");  
    }  
  
    @Override  
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int  
newVersion) {  
        // aici se face actualizarea versiunii bazei  
    }  
}  
  
// pentru a deschide și folosi baza de date  
DatabaseHelper dbHelper = new DatabaseHelper(context);  
SQLiteDatabase db = dbHelper.getWritableDatabase();  
// operațiunile pe baza de date se fac cu obiectul db
```

```
db.close();    // închiderea bazei
```

Cursoarele servesc ca iteratori dinamici, facilitând parcurgerea fără întreruperi prin rezultatele interogărilor din baza de date. Utilitatea cursorilor este evidentă mai ales atunci când se lucrează cu mai multe rânduri de date în cadrul bazelor de date.

Exemplu petru utilizarea cursorilor^[33]:

```
public class DatabaseHelper extends SQLiteOpenHelper {

    // metodă pentru a extrage datele din bază
    public Cursor getAllStudents() {
        SQLiteDatabase db = getReadableDatabase();
        String[] projection = {"id", "name", "email"};
        Cursor cursor = db.query("Students", projection, null,
null, null, null, null);
        return cursor;
    }
}

// apel într-o activitate sau fragment
DatabaseHelper dbHelper = new DatabaseHelper(context);
Cursor cursor = dbHelper.getAllStudents();

if (cursor != null && cursor.moveToFirst()) {
    do {
        int studentId =
cursor.getInt(cursor.getColumnIndex("id"));
        String studentName =
cursor.getString(cursor.getColumnIndex("name"));
```

```

        String studentEmail =
        cursor.getString(cursor.getColumnIndex("email"));

        // putem procesa datele extrase
    } while (cursor.moveToNext());
    cursor.close();
}

```

2.4.3 SQLite și Room

Room este o bibliotecă de persistență concepută pentru a eficientiza procesul de lucru cu bazele de date *SQLite*, oferind o serie de caracteristici și optimizări care simplifică stocarea, recuperarea și manipularea datelor. *Room* prezintă un strat de abstractizare care operează deasupra bazelor de date *SQLite*, făcând legătura între interogările *SQL* brute și gestionarea complexă a instanțelor de baze de date^[37].

De remarcat este faptul că *Room* utilizează adnotări pentru a defini entitățile, interogările și relațiile din baza de date. Prin utilizarea adnotării *@Entity*, o clasă este desemnată ca entitate, fiind asociată fără probleme cu un tabel din baza de date. Fiecare variabilă din cadrul clasei corespunde unei coloane din cadrul tabelului^{[38][39][45]}.

Pe lângă declararea entităților, *Room* introduce conceptul de obiecte de acces la date (*Data Access Objects* - *DAO*). *DAO*-urile sunt interfețe adnotate care delimitează interacțiunea dintre aplicație și baza de date. Metodele din cadrul acestor interfețe sunt adnotate cu interogări *SQL*, încapsulând operațiunile de date într-un mod organizat^{[38][40][43]}.

Crearea bazei de date în se face prin intermediul adnotării *@Database*. Această adnotare desemnează o clasă ca deținător al bazei de date, orchestrând generarea unei instanțe *singleton*^{[38][45]}.

Un avantaj semnificativ oferit prin intermediul adnotărilor sale este prezența verificărilor la compilare pentru interogări și pentru accesarea datelor^{[37][45][46]}.

Observatorii, un concept fundamental în dezvoltarea *Android*, sunt cuplați de biblioteca *Room* cu *LiveData*, o clasă de suport de date care este observabilă. *LiveData* servește la transpunerea

modificărilor din sursa de date de bază (în cazul nostru o bază de date *Room*) și comunicate automat componentelor *UI*, asigurând o interfață grafică receptivă care reflectă fluctuațiile de date în timp real^{[41][42][45]}.

Pe măsură ce datele se modifică în baza de date *Room*, *LiveData* emite notificări către observatorii săi, orchestrând actualizări automate ale interfeței grafice. Acest lucru elimină necesitatea de declanșare manuală a actualizării interfeței grafice ori de câte ori sursa de date evoluează^{[42][43]}.

Operațiunile executate prin intermediul *Room* sunt de natură asincronă, având ca scop principal evitarea blocării interfeței, întârzierile sau lipsa de reacție. Atunci când este invocată o operațiune a bazei de date, firul de execuție al interfeței rămâne liber, permițând utilizatorilor să navigheze fără probleme prin aplicație în timp ce *Room* execută sarcina solicitată în fundal. La finalizare, rezultatele sunt retransmise înapoi către firul interfeței^{[44][47]}.

Exemplu utilizare Room:

```
// în clasa DAO

@Query("SELECT * FROM Students WHERE email = :email")
    LiveData<Student> getStudentByEmail(String email);

// apel într-o activitate sau fragment
studentModal.getStudentByEmail(emailHolder).observe(CarnetActivi
ty.this, new Observer<Student>() {

    @Override
    public void onChanged(Student student) {
        if (student != null){
            // putem procesa aici date provenite
            din obiectul student
        }
    }
});
```

Migrațiile din *Room* servesc drept mecanism critic pentru a gestiona modificările în schema bazei de date pe măsură ce aplicațiile evoluează. O migrare a schemei devine necesară atunci când se fac modificări în structura bazei de date, cum ar fi modificarea tabelelor, adăugarea sau eliminarea de coloane sau schimbarea tipurilor de date. Migrațiile asigură tranziția fără întreruperi a datelor și păstrează integritatea acestora atunci când versiunea aplicației unui utilizator nu mai corespunde cu versiunea curentă a schemei. Prin definirea căilor și strategiilor de migrare, suntem asigurați că datele rămân intacte în timp ce baza se adaptează la schimbările structurale^[48].

TypeConverter este o componentă ce permite traducerea fără probleme între tipurile de date complexe și formele lor corespunzătoare compatibile cu baza de date. Această componentă facilitează conversia câmpurilor cu tipuri de date neprimitive sau nestandardizate, care nu sunt suportate nativ de *SQLite*, în formate care pot fi stocate și recuperate din baza de date. Tot ce este nevoie este o clasă specializată pentru definirea logicii de conversie personalizate și adnotarea metodelor sau câmpurilor cu *@TypeConverter*.

CAPITOLUL 3: CONTRIBUȚIA PROPRIE ÎN IMPLEMENTARE

Contribuția proprie concretă în cadrul aplicației constă în crearea meniurilor ce vor alcătui interfața grafică vizuală și a bazei de date *SQLite* ce va face conexiunea între acestea și elementele de infrastructură.

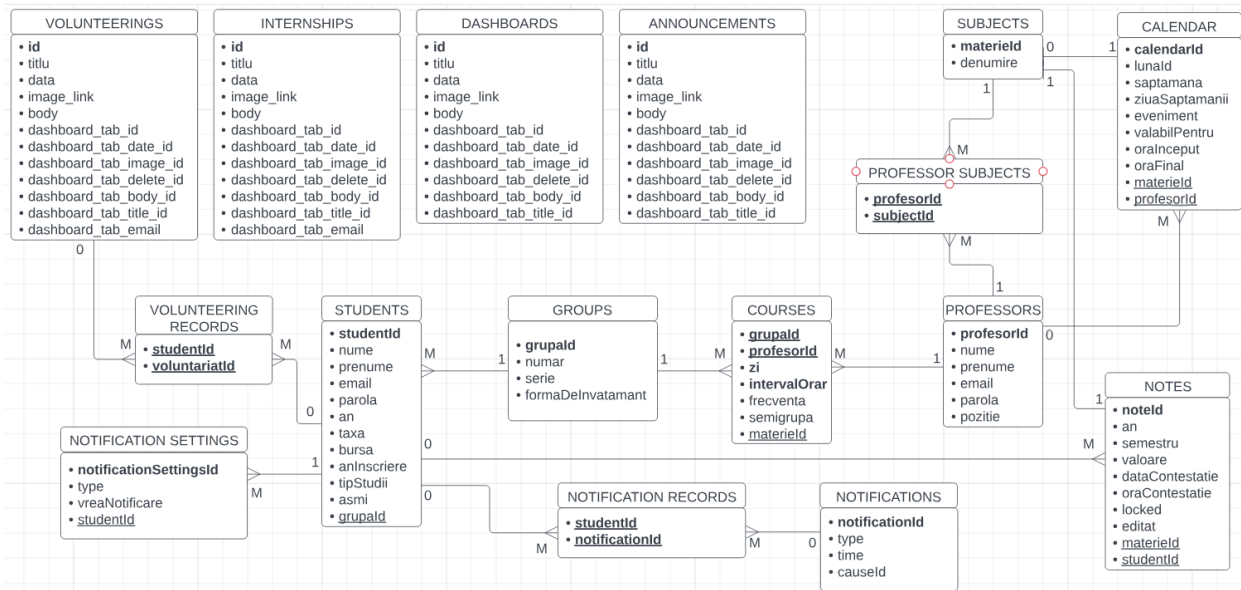
În cadrul interfeței grafice, casetele *TextView* utilizate au fost formate în trei moduri diferite, în funcție de rolul pe care îl vor împlini: titlu, scop informativ și scop descriptiv, acesta din urmă venind mereu în pereche cu o casetă *EditText*, *Spinner*, sau *Checkbox*. Asemenea, pentru casetele *EditText* s-a respectat aceeași formatare indiferent de meniu, pentru a avea un aspect consistent de-a lungul întrebuințării lor. Prin formatare se înțeleg atributele elementului XML, mai concret, atribute precum: *textSize*, *textStyle*, *textColor*, *background* (*@drawable/lavender_border*), *backgroundTint*, *padding*, *margins*, *gravity* etc.

3.1 Structura bazei de date

Atât construcția cât și interogarea tabelelor au fost făcute utilizând în parte egală *Room* și *SQLiteHelper*. Versiunea bazei de date de la finalul implementării nu este aceeași cu versiunea acesteia de la început. Baza de date a evoluat ca scop și a fost extinsă ori de câte ori cerințele aplicației au necesitat aceasta. Actual baza conține 16 tabele:

- Students - conține conturile de studenți și de admin din aplicație
- Groups - conține toate grupele de la licență și master
- Courses - tabel de legătură pentru relația *many-to-many* dintre studenți și profesori, conține cursurile din orarele grupelor
- Professors - conține conturile de profesori din aplicație
- Subjects - conține toate materiile ce sunt predate la specializarea Informatică

- ProfessorSubjects - tabel de legătură pentru relația *many-to-many* dintre profesori și materiile predate
- Calendars - conține toate evenimentele din calendarul academic
- Notes - conține toate notele obținute de studenți în cadrul examenelor
- Notifications - conține toate notificările create de diversele operații ce trimit notificări
- NotificationRecords - ține evidența notificărilor vizualizate sau șterse de studenți
- VolunteeringRecords - ține evidența voluntariatelor la care s-au înscris studenți
- NotificationSettings - conține preferințele pentru notificări ale tuturor studenților
- Dashboards - conține informațiile pentru fiecare articol din fila Activități
- Announcements - conține informațiile pentru fiecare articol din fila Anunțuri
- Internships - conține informațiile pentru fiecare articol din fila Internship-uri
- Volunteerings - conține informațiile pentru fiecare articol din fila Voluntariate



Pentru interogarea tabelor au fost create clase *DAO*, *Repository* și *Modal* pentru a permite lucrarea cu tipul de date *LiveData* și pentru a permite operații asincrone.

La fiecare modificare nou adusă structurii bazei de date, fie schimbarea tipului de date stocat într-o coloană, fie schimbarea numelui unei coloane, chei primare sau străine, fie adăugare de tabele noi, s-a creat o nouă clasă de migrare pentru a actualiza structura bazei de date astfel încât să corespundă cu structura prezentă în clasele specific adnotate cu *@Entity*.

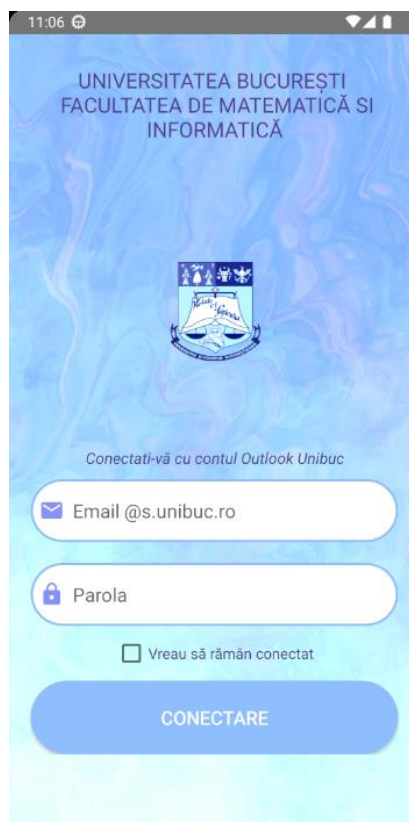
Multe dintre câmpurile de date ce le dorim stocate sunt de tip *LocalTime* și *LocalDate*, însă *SQLite* nu acceptă aceste tipuri de date complexe în mod nativ. Prin urmare, am circumvins această problemă prin convertirea acestor tipuri complexe într-un format pe care *SQLite* îl poate înțelege, anume *String*, cu ajutorul a două clase *TypeConverter*.

3.2 Organizarea interfeței

3.2.1 Meniul de autentificare

Acesta este meniul în care aplicația se deschide dacă utilizatorul nu a optat să îi fie reținută autentificarea din aplicație. În cele două casete de tip *EditText* utilizatorul trebuie să introducă email-ul instituțional, după cum este specificat și într-un *TextView* plasat deasupra, împreună cu parola pentru a avea acces în aplicație. Email-urile instituționale sunt cele puse la dispoziție de facultate la începerea activității universitare, utilizate pentru conectarea prin intermediul *Outlook*. Acestea au specific numele de domeniu *@s.unibuc.ro* pentru studenți și numele de domeniu *@unibuc.ro* pentru cadrele didactice.

Dedesubt se află un *Checkbox* care poate fi bifat dacă utilizatorul dorește să rămână conectat la următoarea redeschidere a aplicației. În acest sens, se evită conectarea repetitivă în cazul în care aplicația este accesată mereu de pe același dispozitiv și, astfel, implicit de același utilizator.



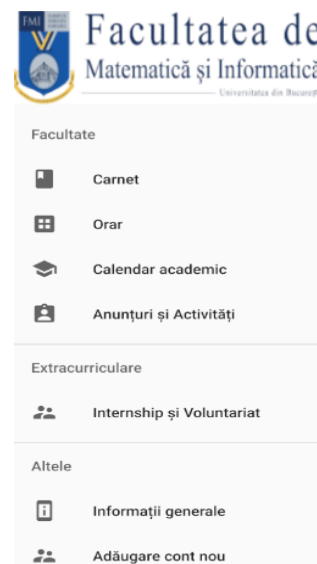
La acționarea butonului pentru conectare se va verifica dacă email-ul introdus este înregistrat în baza de date și dacă parola acestuia este introdusă corect. Dacă nu, un mesaj corespunzător este afișat, care va solicita reintroducerea corectă a datelor de conectare.

3.2.2 Secțiunile meniului principal

Fiecare din secțiunile aplicației: Carnet, Orar, Calendar Academic, Anunțuri și Activități, Internship și Voluntariat, Informații generale și Adăugare cont nou, au în parte diverse caracteristici ce variază în funcție de tipul de cont cu care utilizatorul este conectat. Diferențele pot fluctua de la doar câteva elemente XML, până la o întreagă reamplasare a elementelor interfeței grafice.

Meniul principal al aplicației este accesibil din fiecare dintre secțiunile aplicației, dar nu din paginile adiacente secțiunilor sau paginile ce se deschid din bara de opțiuni secundare.

Interfața meniului a fost creată utilizând un *NavigationView* în asociere cu un fișier *layout* de tip meniu care dictează aspectul grafic al secțiunilor. Acesta a fost apoi plasat într-un *DrawerLayout* pentru a putea fi restrâns și glisat din laterală stângă a paginii.



Carnet

Spre deosebire de restul secțiunilor ce vor fi detaliate mai jos, aceasta are o întrebuințare vast diferită pentru fiecare dintre tipurile de cont oferite de aplicație.

Studentii au posibilitatea de a-și urmări situația școlară, putându-și vizualiza notele separate pe ani și semestre. Alterarea între acestea se face prin selecție în lista celor două *Spinnere* plasate la începutul paginii. În continuarea acestora este prezent un tabel cu două coloane ce reprezintă materia și nota finală obținută la materia respectivă.

Cadrele didactice pot atribui note studenților în această secțiune. Din cele două elemente *Spinner* se poate selecta acum în schimb materia predată (întrucât un cadru didactic poate preda mai multe materii) și grupa la care acesta dorește să încarce notele în carnet. Tabelul generat sub acestea va avea acum coloanele reprezentând numele studentului și nota atribuită.

La acționarea butonului de adăugare se va deschide o fereastră în care se va cere să se introducă data și ora contestațiilor. După acționarea butonului final se va verifica dacă toate câmpurile au fost completate corespunzător cu validările definite și, dacă nu, va fi afișat un mesaj sugestiv. În cazul materiilor ce sunt notate cu calificative precum Practică, Deontologie Academică, Pregătirea Lucrării de Disertație, Pregătirea Lucrării de Licență și Limba Engleză, validarea nu va permite introducerea de text cu alt format decât “Admis/Respins”.

The screenshot shows a mobile application interface with a blue header bar containing a menu icon, a user profile icon, a notification bell, and a share icon. Below the header, there are three selection options: "Alege profesorul:" with the value "Andronescu Zaharia", "Alege materia" with the value "Sisteme de baze de date", and "Alege grupa" with the value "131". Below these options is a table with the following data:

STUDENT	
Adrianescu Dorian	7
Andranetcu Sorin	3
Cartescu Dorina	7
Catrinelu Alin	8
Codrescu Lucian	9
Corteanu Adina	10
Doinaru Casian	5
Merescu Cristina	6
Odorinescu Constantin	7
Paunaru Andrada	8
Sarmeanu Mihaela	9

At the bottom of the screen is a blue button labeled "DEBLOCHEAZA".

Câmpurile pentru note ale tabelului vor deveni inaccesibile, ne mai putând fi editate până la data contestațiilor. Urmând data aceasta, câmpurile de note vor deveni accesibile pentru maximum două zile, pentru a-i da timp profesorului să încarce notele reevaluate, după care vor fi blocate din nou, fie că acesta le-a actualizat fie că nu. Bineînțeles, și după actualizarea notelor de către profesor notele vor fi imediat blocate. Din acest punct, starea notelor nu va mai putea fi modificată fără intervenția unui administrator. Această dezvoltare a fost efectuată cu scopul de a evita orice eventuale tentative frauduloase.

Contul de administrator are posibilitatea de a schimba starea notelor, putând atât bloca cât și debloca. Scopul acestuia este de a interveni în cazul de reexaminari, reevaluări sau dacă cadrul didactic a greșit la înregistrarea notei. Interfața specifică administratorului mai are un *dropdown Spinner* în plus, pentru a putea selecta cadrul didactic la care trebuie intervenit asupra stării notelor.

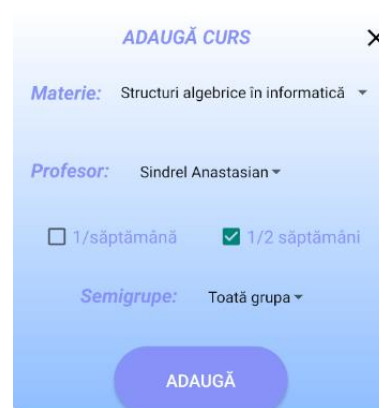
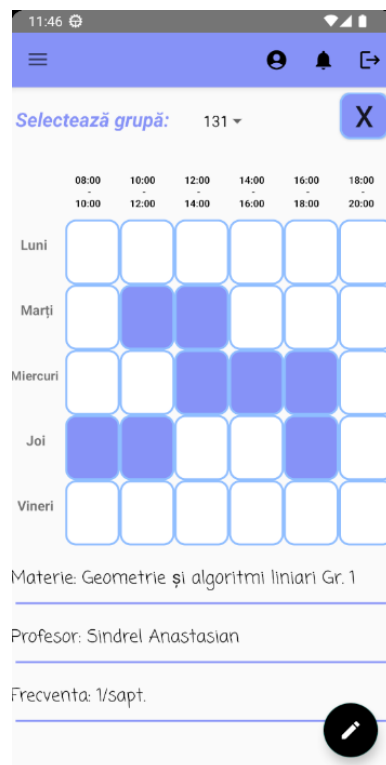
Orar

Această secțiune are ca scop afișarea cursurilor, seminariilor și laboratoarelor, împreună cu diverse detalii ce le privesc precum zilele în care acestea sunt ținute, intervalele orare în care sunt cuprinse, materia ce este prezentată în cadrul cursului, profesorii care predau cursurile respective, frecvența cu care este ținut și împărțirea pe grupe a studenților dacă este cazul.

În partea de sus a paginii avem un *dropdown Spinner* pentru selectarea grupei pentru care dorim să fie afișate cursurile și un buton pentru a elimina toate cursurile atribuite.

În urmarea să se afle un tabel ale cărui coloane sunt reprezentate prin intervale orare de la 08:00 până la 20:00, iar rândurile prin zilele săptămânii, de luni până vineri. Celulele tabelului sunt regăsite în două culori cu semnificații diferite: alb, indicând că poziția respectivă este liberă, și mov, indicând că un curs ocupă intervalul respectiv.

Administratorii sunt singurele conturi cu posibilitatea de a adăuga cursuri în orar. La atingere sau click pe o celulă albă a tabelului se va deschide o fereastră pentru a selecta datele descriptive ale cursului. Dintr-un *dropdown* se selectează materia ce dorim să facă obiectul cursului. Următorul *dropdown*, cel pentru alegerea profesorului care o va predă, va fi încărcat doar cu profesorii ce predau materia precedent selectată. În urmărire, se află două *Checkbox*-uri ce reprezintă frecvența cu care orele vor fi predate, o dată pe săptămână sau o dată la două săptămâni, și un *dropdown* final din care se selectează dacă ora va fi ținută cu o grupă întreagă sau doar cu o anumită semigrupă. La acționarea butonului de adăugare, celula ce a fost selectată își va schimba culoarea pentru a semnala că acea fereastră orară a fost ocupată. În dedesubtul tabelului este acum vizibilă o secțiune informativă cu datele inserate. La click pe oricare din secțiunile colorate se vor afișa în secțiunea informativă datele despre cursul respectiv, împreună cu un buton de tip *FloatingActionButton* pentru editarea informațiilor.



Fereastra ce se deschide prin acționarea acestuia este asemănătoare cu cea de adăugare a unei ore noi, însă câmpurile sunt deja încărcate cu detaliile cursului respectiv.

Studentii și profesorii au doar posibilitatea de a urmări orele, indiferent de grupă. Atunci când interfața este deschisă cu un cont de student, grupa selectată pentru care orarul va fi afișat este grupa din care studentul face parte. Atare, pentru conturile cadrelor didactice, interfața va fi deschisă pe orarul utilizatorului. Pe deasupra, aceasta versiune a interfeței mai prezintă un buton în partea de sus a paginii, prin intermediul căruia profesorul poate să vadă din nou orarul propriu, dacă între timp a dorit să navigheze și orarul grupelor.

Am ales să nu impunem accesul studenților de vizualizare strictă doar la orarul grupei proprii sau profesorului doar la orarul propriu, întrucât, spre exemplu, master-ul, necesită înrolarea la cursuri opționale, ce fac parte din curriculum-ul altor programe de studii.

Calendar Academic

În interfața Calendarului Academic avem organizate pe luni evenimentele academice din timpul anului universitar. Acestea sunt afișate în celulele unui tabel ale cărui coloane reprezintă zilele săptămânii de luni până duminică.

La fel ca în cazul orarului, zilele care sunt ocupate cu evenimente sunt diferit colorate pentru a le evidenția. De altfel, zilele ce aparțin lunii precedente și lunii următoare sunt marcate ca inaccesibile, atât printr-o culoare diferită cât și prin faptul că sunt neinteractive. Fiecare celulă conține numărul zilei calendaristice.

În calendar se pot salva diverse tipuri de evenimente: termene limită pentru depunerea diplomelor de licență, pentru depunerea actelor de înscriere, de plățirea a taxei de școlarizare, examene, etc.



Atât administratorii cât și profesorii au drepturi de adăugare și editare a evenimentelor, însă profesorii vor avea posibilitatea doar de adăugare a examenelor, pe când administratorilor le va fi rezervată posibilitatea de a adăuga oricare alt tip de eveniment. De altfel, este permisă înregistrarea unui număr multiplu de examene în aceeași zi, pe când alte evenimente sunt restricționate la doar unul per dată calendaristică.

Asemenea implementării din pagina secțiunii Orar, la interacționarea cu o celulă goală se va deschide o fereastră unde se va cere completarea detaliilor pentru evenimentul pe care dorim să îl înregistrăm. Pentru administratori va fi vizibilă o caseta *EditText* ce dorește introducerea denumirii evenimentului alături de un *dropdown Spinner* care va selecta pentru cine este valabil această informație: Licență, Master, Restanțieri Licență sau Restanțieri Master. Categoria selectată va fi de astfel notificată asupra modificărilor făcute. Informațiile introduse vor fi afișate într-un *LinearLayout* cu două casete *TextView*, în regiunea aflată sub tabel și totodată vor deveni vizibile două butoane de tip *FloatingActionButton*, unul pentru ștergerea evenimentului și unul pentru editarea sa. La apăsarea butonului de editare, fereastra va fi din nou deschisă și deja încărcată cu informațiile anterior inserate.

Utilizatorii cadre didactice pot accesa orice celulă a tabelului pentru a adăuga examene, nu trebuie să se rezume la celulele marcate ca libere, din motiv ce este permisă inserarea de examene multiple pentru aceeași dată calendaristică. La atingere sau click pe o celulă liberă va fi deschisă, asemenea administratorului, fereastra de inserare. La atingere sau click pe o celulă ocupată, vor fi întâi afișate informații asupra evenimentelor înregistrate în data respectivă, în cazul în care scopul acțiunii este doar citirea acestor detalii. Alături va apărea un buton de tip *FloatingActionButton* ce prin acționare va deschide fereastra de inserare.

Fereastra apărută va cere alegerea materiei, grupei și a intervalului orar în care examenul urmează a fi desfășurat. *Dropdown*-ul cu materii va conține doar materii care sunt predate de profesorul respectiv. Iar *dropdown*-ul din care se aleg grupele pentru care va fi valabil examenul, conține doar grupe ce au cursuri înregistrate în orar ca fiind predate de profesorul respectiv. La acționarea

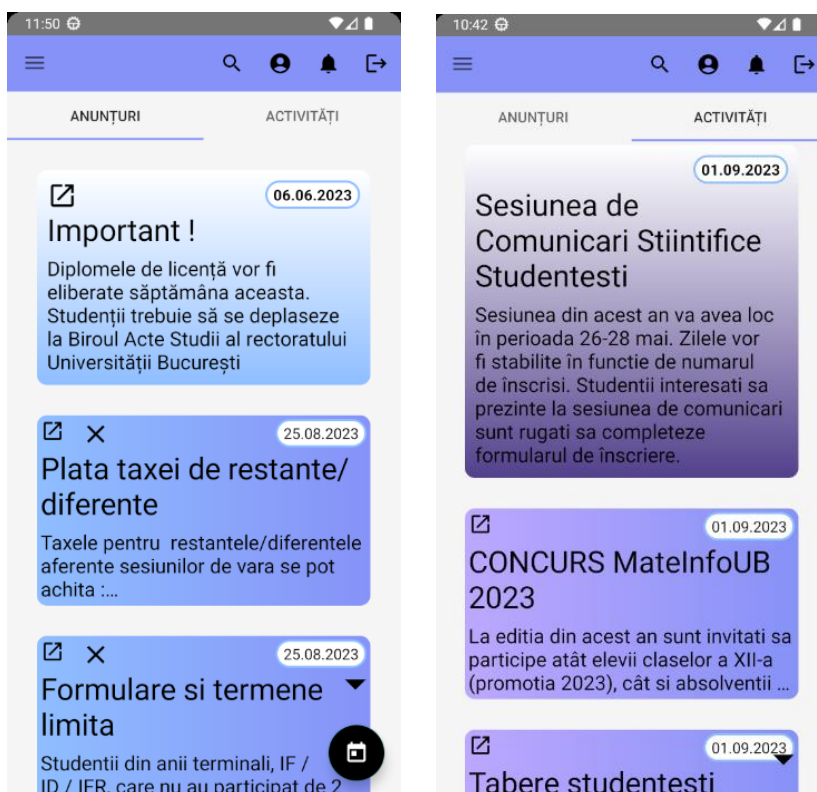
butonului de adăugare va fi verificat dacă cadrul didactic sau grupa aleasă mai au examene în aceeași zi, într-un interval orar suprapus. În caz afirmativ, se va afișa un mesaj corespunzător iar examenul nu va putea fi adăugat până datele nu vor fi validate. Când examenul va fi inserat cu succes în calendar, în dedesubtul tabelului vor fi afișate informațiile completate. Cum într-o zi pot avea loc mai multe evenimente, am optat pentru un conținut derulant *ScrollView*. În acesta, fiecare eveniment va avea rezervat un *LinearLayout*, ce la rândul său va conține în ierarhia mai multe elemente. Printre acestea se numără trei casete *TextView* cu numele evenimentului (mereu "Examen"), grupa care va susține examenul și intervalul orar al acestuia și două butoane, unul de ștergere și unul de editare. Acestea două butoane vor fi vizibile utilizatorului strict pentru examenele adăugate de sine, nefiindu-i permis să modifice examenele înregistrate de alte cadre didactice.

Anunțuri și Activități

Această secțiune este cea care se deschide după ce un utilizator s-a autentificat cu succes. De asemenea, acesta este și fereastra în care aplicația se va deschide dacă utilizatorul a optat să rămână conectat și pentru viitoarele deschideri ale aplicației.

Meniul este împărțit în două fragmente, fiecare plasat în propria filă de tip *ViewPager*, iar accesarea acestora se face printr-un *TabLayout* cu două elemente *TabItem* plasate sub bara de opțiuni secundare. Cele două sub-secțiuni se pot accesa alternativ prin operația de glisare laterală sau atingere pe fila respectivă.

În acest meniul avem înșirate unul sub altul mai multe



articole, fiecare conținând informații despre un anunț sau despre o activitate de interes. Pentru a naviga articolele în mod derulant, acestea au fost plasate într-un *ScrollView*. Deși aspectul lor este asemănător, separarea dintre anunțuri și activități s-a făcut în funcție de relevanță și obiectiv. Sub-secțiunea pentru anunțuri ar fi dedicată anunțurilor de la secretariat, acestea fiind de o importanță mai ridicată pentru studenți. Sub-secțiunea de activități este rezervată pentru diverse conferințe ținute de firme sau prelegeri ale persoanelor din domeniu, concursuri, activități ale unor organizații etc.

Fiecare articol este compus din patru elemente: data la care articolul a fost postat, un titlu sugestiv, conținutul propriu-zis și un link către articolul postat în prealabil pe site, pentru a fi accesat de utilizator dacă dorește detalii adiționale. Detalii precum imagini sau tabele, deoarece acestea nu ar putea fi introduse din aplicație datorită modului de input strict text pentru conținutul unui articol.

Textul fiecărui articol este prezentat scurtat în pagină, pentru a facilita derularea prin cât mai multe articole cât mai repede, deoarece nu fiecare articol prezentat se află în interesul utilizatorului. Pentru citirea conținutului articolului în întregime, la atingere sau click pe un articol se poate deschide o pagină nouă, unde conținutul complet este vizibil alături de data și titlu. Datele au fost transmise către această pagină prin metoda *putExtra()* a unui obiect *Intent*.

Link-ul către site poate fi accesat din lista principală a articolelor, prin acțiune pe iconița sugestivă a fiecăruia dintre articole.

Primul dintre articolele din pagină este cotate cu cea mai mare relevanță pentru utilizatori, iar plasarea sa la început are ca scop creșterea vizibilității. La deschiderea paginii acesta este restrâns pentru a atrage atenția asupra sa, față de celelalte articole care prezintă un aspect identic. La acționare prin click sau atingere, acesta se desfășoară, având acum un aspect asemănător cu celelalte articole, însă nefiind restricționat din punct de vedere al lungimii.

Contul de administrator este singurul cu permisiunea de a crea și șterge articole în această secțiune. Astfel acestuia îi mai apare în partea dreapta-jos a ecranului un buton de tip *FloatingActionButton*. La acționarea acestuia se deschide o casetă "Creare articol nou" în care acesta poate introduce titlul, conținutul articolului și link-ul către site. Ștergerea se face prin butonul atașat fiecărui articol în sine. Pe lângă acestea, tot administratorul are permisiunea de a desemna un articol ca fiind cel

de importanță maximă pentru a fi plasat la începutul listei de articole. Acest prim articol nu poate fi șters, dar acest lucru nu este o necesitate din moment ce acesta are un conținut editabil.

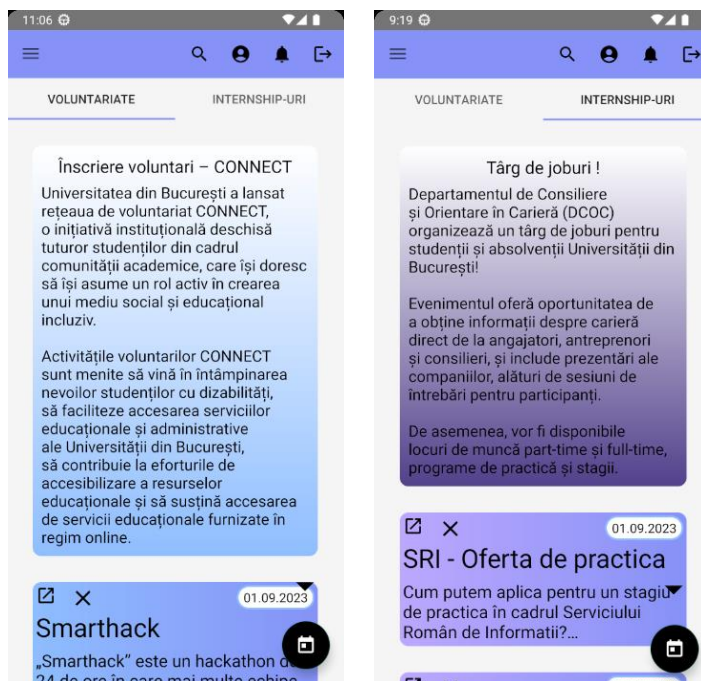
Cele mai noi articole adăugate sunt plasate la sfârșitul listei. În scopul de a trece rapid prin lista de articole până la cele noi apărute, avem un buton special de tip *FloatingActionButton* aflat în partea dreapta-jos, care va derula până la final.

Pentru cele două butoane din această pagină s-a ales tipul *FloatingActionButton* deoarece acesta "plutește" deasupra conținutului paginii. Altfel, dacă s-ar fi folosit tipul *Button*, acesta ar fi fost fixat în pagină între elementele de *layout* între care a fost declarat în ierarhia XML. Situația aceasta nu ar fi fost o problemă dacă pagina era de asemenea fixă. Însă noi avem nevoie de conținut derulabil datorită numărului mare și variabil de articole.

Internship și Voluntariat

Aproape analog secțiunii prezentate mai sus când vine vorba de aspect, este și secțiunea Internship și Voluntariat. Scopul articolelor este sugerat de numele sub-secțiunilor prezente în aceasta: Internship și Voluntariat. Pe de altă parte, acestea mai prezintă niște opțiuni în plus. În Voluntariat, dându-se click pe un articol pentru a se deschide pagina extinsă de detalii, observăm încă o opțiune, aceea de a ne înscrie la un voluntariat. Odată înscrisi vom primi și opțiunea să renunțăm dacă nu mai prezentăm interes pentru voluntariatul respectiv. Aceste opțiuni sunt valabile doar pentru conturile de student.

Pentru conturile de administrator, mai există un câmp de tip *EditText* ce trebuie completat atunci când postăm articole în această secțiune: Email. În cazul internship-urilor acesta ar reprezenta



mail-ul recrutorului și va apărea și în pagina extinsă de detalii a unui articol pentru a exista o metodă de contact între studenți și recrutori. În cazul voluntariatelor, acest email trebuie să aparțină unui student din cadrul Asociației Studentilor de la Matematică și Informatică și va fi utilizat pentru a notifica persoana respectivă atunci când alți studenți se vor înscrie la un voluntariat pe care îl administrează.

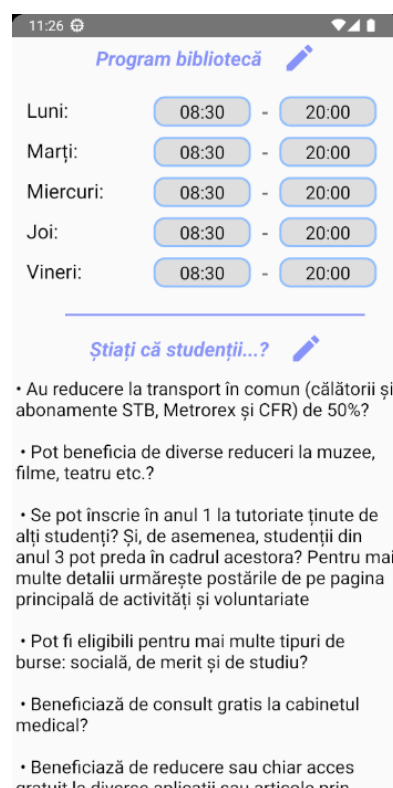
Cum sub-secțiunea Voluntariat este dedicată în mare parte studenților din cadrul ASMI, conturile astfel înregistrate ca făcând parte din Asociație primesc drept de adăugare și ștergere a articolelor. Pentru studenții din cadrul Asociației care au în administrare voluntariate, la pagina extinsă de detalii va apărea și o listă cu studenții care s-au înscris la voluntariatul respectiv.

Informații Generale

În aceasta secțiune se găsesc informații de uz general pentru studenți precum: zilele de decontare ale abonamentelor pentru transportul public, intervalul orar în care secretariatul, biblioteca și cantina sunt deschise, programul de lucru al cabinetului medical. Tot aici mai sunt făcute cunoscute și o serie de înștiințări asupra diverselor beneficii aduse prin statutul de student.

Pentru conturile de studenți și cadre didactice, această secțiune este restricționată în scop pur vizual, nu există elemente interactive. Pentru conturile administrative însă, fiecare sub-secțiune a paginii este editabilă. Alături de fiecare titlu al unei sub-secțiuni se află un element *Button*, ce prin acționare va debloca elementele de tip *EditText* aflate sub acesta pentru a permite interacționarea. Iconița butonului se va schimba într-un “check mark”, iar prin apăsarea acestuia se vor salva modificările introduce și vor fi reblocate câmpurile editabile, fapt ce este și evidențiat prin schimbarea culorii acestora.

Întrucât fluxul de datele ce trebuie persistat este de un nivel scăzut, iar structura lor nu este nici adecvată pentru stocarea în tabele în baza de date, s-a optat pentru folosirea *SharedPreferences*.



La finalul pagini, vizibil doar conturilor administrative, se află o scurtă opțiune de adăugare a unui student în cadrul ASMI, făcând folosință de un *EditText* și un *Button*.

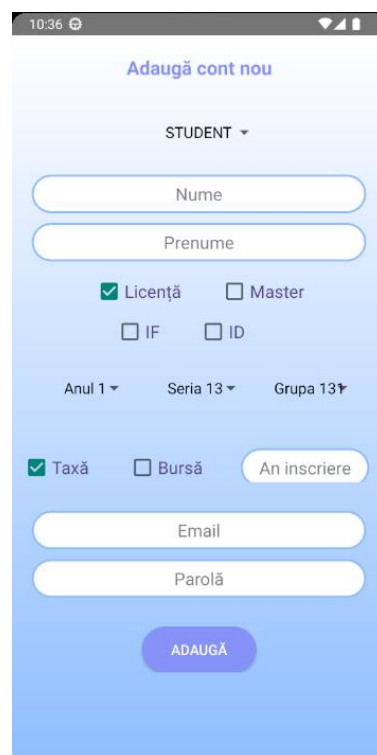
Adăugare Cont Nou

Acest meniu este o secțiune strict dedicată conturilor de administrator pentru a înregistra conturi noi de utilizator în aplicație. Astfel, la conectare cu un cont de student sau de profesor aceasta secțiune nu va fi vizibilă în meniul principal.

Cum aplicația utilizează trei tipuri de conturi, s-au creat opțiuni caracteristice de configurare pentru fiecare în parte. La începutul pagini am plasat un *Spinner* cu cele trei opțiuni, iar în funcție de selecție, vom încărca diferite elemente grafice, cele mai ample fiind cele ale studenților.

Reamintim că studenții ale căror conturi se pot crea din aplicație sunt doar cei înscriși la domeniul Informatică, specializarea Informatică pentru licență și domeniul Informatică pentru master.

Pentru conturile de student avem casete *EditText* pentru inserarea numelui și prenumelui, urmate de două *Checkbox*-uri pentru tipul de studii la care studentul este înscris: licență sau master. În continuare sunt alte două *Checkbox*-uri ce reprezintă forma de învățământ și trei *dropdown*-uri de tip *Spinner* cu anul, seria și grupa.



Informațiile încărcate în aceste elemente sunt determinate de selecțiile anterior făcute în următorul fel:

Dacă "Licență" a fost bifat atunci vor fi vizibile un *Checkbox* "IF" și unul "ID". *Spinner*-ul pentru ani va fi încărcat cu trei valori, întrucât atât durează programul de studiu al specializării Informatică. În funcție de selecția făcută pentru forma de învățământ, valorile din *dropdown*-urile de serie și grupa vor varia de asemenea. Nu există serie și avem doar două grupe dacă s-a bifat *Checkbox*-ul "ID", iar dacă s-a bifat "IF" se vor încărca seriile, în funcție, bineînțeles, și de anul

selectat. Dacă se selectează "Anul 1", se vor încărca doar seriile 13 și 14. Mai departe grupele vor fi încărcate în funcție de selecția seriei. Pentru seria 14 va exista posibilitate de selecție doar pentru grupele 141, 142, 143 și 144.

Procesul este același și în cazul bifării tipului de studii "Master". Pentru aceasta va dispărea *Checkbox*-ul "ID" și va apărea în schimb un *Checkbox* "IFR". *Dropdown*-ul pentru ani nu va mai avea acum decât două valori, iar seria nu va mai exista.

În continuare se află două *Checkbox*-uri: Taxă și Bursă și o casetă *EditText* pentru anul în care studentul a fost înscris la facultate.

La final, ultimele două elemente sunt casete *EditText* pentru inserarea email-ului (strict în format @s.unibuc.ro) și a parolei. Toate conturile au fost create cu parola standard parola.fmi123.

Pentru conturile de profesor avem și aici casete *EditText* pentru nume, prenume, email și parolă cum avem și la student. Pe de altă parte, mai avem un *Spinner* pentru selectarea gradului didactic al acestuia: Asistent, Lector, Conferențiar sau Profesor; și un *MaterialCardView* pentru selectarea mai multor discipline ce sunt predate de profesorul respectiv.

Pentru conturile de administrator există doar două câmpuri de tip *EditText* pentru email și parolă, întrucât alte informații adiționale nu își au folosința în întrebuințarea contului la nivelul curent al aplicației.

3.2.3 Bară de opțiuni secundare

Bara de opțiuni vine mereu alături de meniul principal, fiind astfel vizibilă din aceleași pagini ale aplicației din care este și meniul. Aceasta conține următoarele opțiuni: Căutare, Profil, Notificări și Deconectare.

Când se navighează spre una din opțiunile ce au pagină proprie, se folosește bineînțeles un obiect *Intent* pentru a lansa activitatea, dar este trimis și un parametru prin metoda *putExtra()*, care ne indică pagina din care venim pentru ca butonul de întoarcere să știe ce activitate va trebui să lanseze când va fi acționat.

Bara a fost creată utilizând un element *MaterialToolBar* ce conține un fișier de layout de tip meniu cu indicații pentru poziționarea iconițelor opțiunilor. Acesta este plasat într-un element *AppBarLayout*.

Căutare

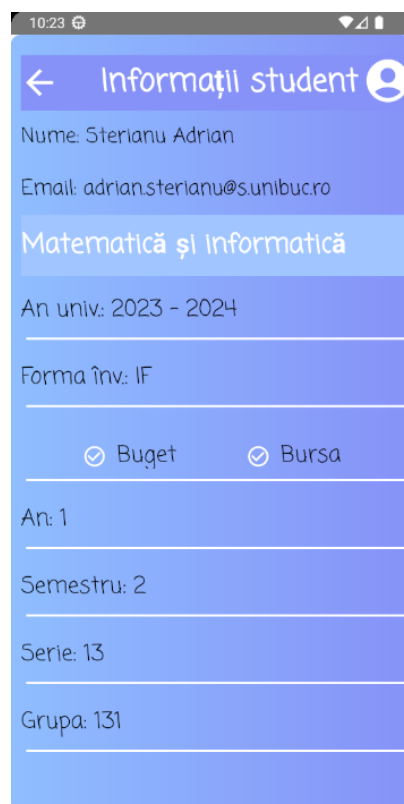
Opțiunea de căutare apare doar în două secțiuni: Anunțuri și Activități și Internship și Voluntariat; deoarece ar fi redundantă în oricare din celelalte secțiuni ale aplicației. În aceasta se pot introduce cuvinte cheie, fie din titlu, fie din conținutul articolelor. Conținutul paginii va fi derulat până în punctul în care cuvintele cheie au fost găsite. Scopul său este de a naviga mai ușor lista considerabilă de articole ce o poate avea aplicația.

Profil

În această pagină se află detalii despre utilizator, aceleași ce au fost salvate și la înregistrarea contului în aplicație de către administrator. Astfel, pentru studenți sunt vizibile informații despre: anul universitar în care aceștia sunt înscriși, forma de învățământ, dacă se află la taxă sau la buget, dacă sunt beneficiari de bursă, anul curent în care se află, semestrul, seria și grupa. Pentru profesori sunt vizibile grupele la care aceștia predau, alături de materiile predate. Pentru administratori este afișată o listă de împuterniciri pe care aceștia le au în cadrul aplicației.

Notificări

Această opțiune este destinată în prezent doar studenților, întrucât nu am găsit până în momentul actual întrebunțări și pentru conturile de cadre didactice și didactice auxiliare. Astfel, notificările sunt vizibile doar acestui tip de cont. În partea de sus, pagina are un buton de revenire la meniul precedent din care a fost accesat și o iconiță care accesează câteva setări prin care utilizatorii pot să își personalizeze notificările în funcție de preferințe.



←	Informații student	👤
Nume: Sterianu Adrian		
Email: adrian.sterianu@s.unibuc.ro		
Matematică și Informatică		
An univ: 2023 - 2024		
Forma înv: IF		
<input checked="" type="radio"/> Buget <input checked="" type="radio"/> Bursa		
An: 1		
Semestru: 2		
Serie: 13		
Grupa: 131		

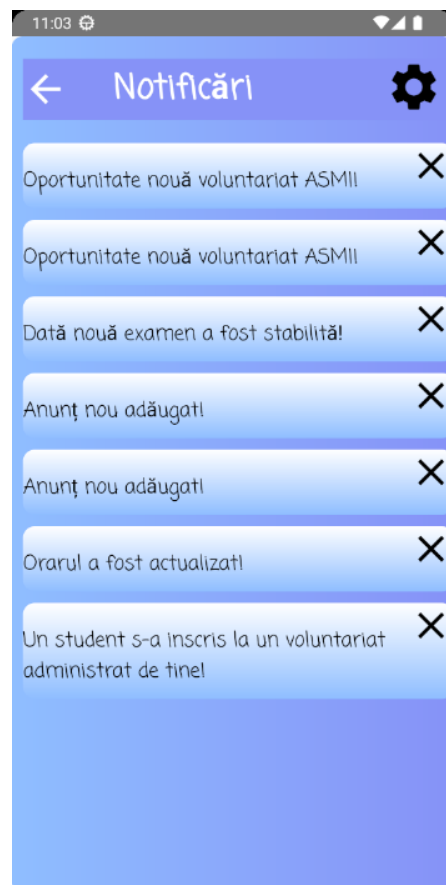
În continuare, dacă utilizatorul are notificari, acestea vor fi înșirate unele sub altele. Fiecare notificare este un *LinearLayout* cu orientare orizontală, ce are în ierarhia sa un *TextView* și un *Button*, acestea reprezentând o scurtă descriere, respectiv un buton de ștergere. Dacă însă nu există notificări, va fi afișat un mesaj *TextView* corespunzător: "Nu aveți notificări".

Notificările sunt de multiple tipuri, în funcție de acțiunea care le declanșează. Tipul notificării va determina descrierea conținută în aceasta, care este standard pentru acțiunea ce a declanșat-o. Notificări sunt trimise studenților atunci când se adaugă articole de orice natură, când se actualizează sau adăugă cursuri noi în orar, când un profesor a adăugat note în carnet sau a actualizat notele deja existente în cazul contestațiilor sau reexaminărilor, când calendarul academic este modificat cu evenimente noi sau când un profesor stabilește data unui examen. În plus, studenții înscriși în ASMI mai pot primi notificări și atunci când alți studenți optează să se înroleze la unul din voluntariatele ce le administrează.

La atingere sau click pe notificare se va deschide pagina acțiunii ce a declanșat-o. Spre exemplu, dacă studentul a fost notificat că a fost actualizat orarul său, atunci notificarea va face redirect către secțiunea orar.

În setările notificărilor este prezentă o listă de opțiuni, fiecare în parte formată dintr-un *TextView* și un *Checkbox*, reprezentând tipurile de notificări ce sunt trimise utilizatorului. Setările implicite au toate opțiunile bifate pentru ca studentul să fie informat asupra tuturor actualizărilor din cadrul aplicației. Dacă însă consideră că o anumite secțiune nu se află în interesul său, acesta poate debifa opțiunea aferentă. Bineînțeles, studenții participanți ai ASMI au o opțiune în plus pentru a opri notificările declanșare de voluntarii noi înscriși.

O dată ce o notificare a fost ștearsă de utilizator sau utilizatorul a dat click pe aceasta, notificarea nu va mai apărea la următoarea deschidere a filei Notificări.



Deconectare

Aceasta permite deconectarea fie în scop de schimbare a contului de utilizator, fie din motive de securitate, pentru ca aplicația să nu mai rețină contul cu care utilizatorul a fost autentificat, în cazul în care a fost bifată opțiunea de reținere conectare. Această opțiune va face redirect către meniul de autentificare.

CAPITOLUL 4: DIFICULTĂȚI ȘI ÎMBUNĂTĂȚIRI

4.1 Dificultăți întâmpinate

De-a lungul implementării au fost întâlnite mai multe obstacole, ce au fost rezolvate prin metode alternative de implementare. Cele mai des întâmpinate greutăți au fost cele de aranjare în pagină a elementelor grafice ale interfeței. Butoane, *TextView*-uri, *LiniarLayout*, *RelativeLayout* etc. adeseori nu puteau fi orientate decent astfel încât să aibă o încadrare plăcută din punct de vedere vizual în vecinătatea altor elemente grafice. Pentru aceste impedimente se găsea însă relativ rapid o rezolvare prin reamenajarea interfeței grafice datorită flexibilității pe care ne-o permitem când vine vorba de aspectul acesteia. Pe lângă aceasta, numărul mare și variat de tipuri de elemente grafice ne facilitează posibilitatea de înlocuire a elementelor al căror plasament nu ne este benefic, cu altele ce au aceeași întrebuințare.

Lipsa de experiență în domeniul *Android* a fost un factor ce a împiedicat dezvoltarea constantă și fluidă a anumitor funcționalități. Cel mai proeminent exemplu este cel al dezvoltării generării programative a unui articol nou. Când butonul de adăugare este acționat, elementele grafice ale articolului nou ce va fi generat sunt asamblate în cod *Java* și trebuie conectate atât între ele cât și la celelalte elemente grafice deja existente. Deși articolele sunt plasate liniar cu o orientare verticală, din lipsa cunoștințelor bine definite asupra modului în care diferitele tipuri de *layout* operează, am folosit *RelativeLayout* în schimb de *LiniarLayout*, fapt ce a necesitat salvarea în bază a id-urilor tuturor *layout*-urilor din pagină, împreună cu id-urile elementelor ce le conțin. Altfel, atât la crearea lor cât și la încărcarea lor din baza de date și-ar fi pierdut poziționarea liniară, deoarece *RelativeLayout* își determina poziția prin orientarea după plasamentul elementului precedent.

Un alt impas întâlnit a fost la interogarea bazei de date cu *framework*-ul *Room*, întrucât calitatea sa de a rula operațiile asupra bazei de date în mod asincron poate cauza dificultăți mari. Prin definiție, operațiile asincron nu sunt rulate în ordinea în care codul a fost scris, ceea ce poate cauza probleme dacă trebuie folosite imediat datele extrase din bază. Astfel, codul ce are nevoie de datele extrase va trebui mutat în interiorul interogării și apoi utilizat în combinație cu un observer pentru

a lucra mereu cu date actualizate. Însă dacă în urma unei interogări este nevoie și de rularea unei operații de insert, update sau delete, observer-ul va fi declanșat, re-rulând tot codul din interiorul său, inclusiv din nou operația de actualizare a bazei care iar va declanșa observer-ul ce va continua tot așa într-o buclă infinită. Cum în interiorul observer-ului se făceau și actualizările asupra interfeței grafice, accesul în aplicație la meniuri, butoane sau casete text devenea complet blocat.

Această complicație nu a avut alternativă decât prin renunțarea la implementarea interogărilor și actualizărilor bazei prin intermediul *Room*, folosindu-se în schimb *SQLiteHelper*.

O ultimă problemă întâmpinată care nu a avut însă repercursiuni ample, s-a petrecut în etapele inițiale de testare. Aceasta a fost imposibilitatea de a transfera baza de date definită local pe un dispozitiv fizic la rularea aplicației. Dar datorită alternativei oferite de *Android Studio* de a rula aplicația într-un emulator, problema a fost ocolită.

4.2 Îmbunătățiri

Pe lângă optimizările ce pot fi aduse de o redactare mai clară a codului și de o eficientizare a implementărilor funcționalităților, aplicația poate beneficia și de un număr ridicat de opțiuni pentru a înlesni experiența utilizatorului.

1. Domeniul aplicației ar poate fi extins pentru a acoperi și celelalte specializări din domeniul Facultății de Matematică și Informatică, eventual și mai amplu pentru a încorpora și celelalte facultăți din cadrul universității.

2. Fereastra de adăugare a articolelor are posibilități destul de limitate pentru editarea conținutului, iar utilizatorilor le-ar putea veni în ajutor niște opțiuni de formatare a textului cu diverse stiluri precum bold, italic sau underline pentru a putea evidenția paragrafe sau cuvinte cheie. Totodată, eventualități pentru adăugarea de imagini, link-uri ancoră sau tabele, ar putea fi puse în practică pentru o mai bună ilustrare a conținutului.

3. Natura actuală a implementării aplicației nu permite trimiterea de notificări în timp real, motiv pentru care s-a dedicat o pagină strict pentru acestea. Această abordare a decurs din imposibilitatea

de a testa o astfel de funcționalitate fără posesia de două dispozitive mobile fizice. Ridicarea bazei la nivelul unui server pentru a putea aplica acest concept ar fi o îmbunătățire care ar elibera pe de o parte bara de opțiuni secundare, iar, pe alta, ar spori experiența utilizatorului, devenind astfel în temă cu modul în care o notificare a unei aplicații mobile ar trebui să se comporte.

4. Articolele din fila de Internship a secțiunii Internship și Voluntariat conțin în momentul actual doar un mail pentru contactarea angajatorului ce a trimis anunțul către direcția facultății. S-ar putea construi un formular minimal de înscriere în interiorul articolului, într-un mod asemănător articolelor din fila Voluntariat. Aplicația locală de email din dispozitivul mobil ar putea fi deschisă din aplicația noastră, cu subiectul, destinatarul și un template standard deja completat în care studentul este prezentat, dându-se informații de interes precum anul curent sau specializarea. Mai departe am mai putea integra o opțiune de încărcare a actelor de identitate și a CV-ului în aplicație, în pagina de Profil, pentru a putea atașa aceste documente de interes pentru angajator.

5. Datele utilizatorilor stocate într-o astfel de aplicație nu sunt statice. Aplicația trebuie să beneficieze de o dinamicitate la schimbarea anilor universitari. Spre exemplu, un student ce a fost anul acesta înscris în primul an, va trece în anul doi, iar datele profilului său trebuie actualizate. La fel, studenții ce acum se află în anul trei, va trebui să fie eliminați din baza de date a aplicației. În momentul de față, singura astfel de funcționalitate dinamică este cea a generării zilelor calendarului, dar aceasta nu este un beneficiu ce influențează direct utilizatorul. Pe lângă cele deja menționate, propunem adăugarea unui calcul de medie anuale în funcție de numărul de credite al fiecărei materii și de nota finală la materia respectivă. Scopul este de a decide dacă studentul a trecut sau nu în anul următor, dacă acesta se va încadra la buget sau la taxă și dacă are posibilitatea de a prinde cazare în căminul facultății.

6. Secțiunea de Informații generale ar putea beneficia de mai multe noțiuni informative precum detalii despre completarea cererilor pentru cazarea în cămin și informații despre obținerea burselor sociale și de merit.

CONCLUZIE

Aplicația noastră mobilă Android, deși aflată în momentul de față încă într-o versiune demonstrativă, consider ca și-a atins pe deplin scopul propus prin implementarea sa.

Alegerea temei pentru această aplicație a fost inspirată din necesitățile identificate în propria activitate studențească, căutând să aducă îmbunătățiri semnificative în accesul la informațiile universitare pentru comunitatea studențească.

Transformarea platformelor existente pentru accesarea informațiilor universitare într-o versiune mobilă, reprezintă o evoluție firescă pentru a răspunde cerințelor studenților într-un mod mai rapid și mai intuitiv. În cadrul aplicației s-a realizat necesarul de opțiuni pentru o dezvoltare ce oferă utilizatorilor o modalitate simplă și eficientă de a naviga în mediul academic și de a-și gestiona progresul și activitățile.

Pe de altă parte, procesul de dezvoltare al acestei aplicații a oferit și oportunitatea de a ne dezvolta abilitățile și cunoștințele în domeniul aplicațiilor Android. Prin această inițiativă, ne-am angajat într-un proces de învățare continuă și ne-am extins competențele, dezvoltându-ne astfel personal și profesional.

În timpul dezvoltării aplicației, dificultățile întâmpinate au avut un rol semnificativ în dezvoltarea abilităților și înțelegerii mai profunde a procesului de dezvoltare a aplicațiilor Android. Aceste provocări nu doar că au testat competențele noastre, dar au și oferit o oportunitate de a învăța și de a ne îmbunătății cunoștințele în mod activ.

Cum procesul de dezvoltare nu se încheie odată cu lansarea inițială, ci este un ciclu continuu de inovare și îmbunătățire, aplicația noastră va fi deschisă către viitoare progrese și optimizări, pentru a continua să rafineze experiența utilizatorilor.

BIBLIOGRAFIE

[1] - Run apps on the Android Emulator. <https://developer.android.com/studio/run/emulator>.

Accesat: 12.08.2023

[2] –Create and manage virtual devices. <https://developer.android.com/studio/run/managing-avds>. Accesat: 12.08.2023

[3] – Create XML layouts for Android – Read and understand XML.

<https://developer.Android.com/codelabs/basic-android-kotlin-training-xml-layouts#2>. Accesat: 14.08.2023

[4] - Create XML layouts for Android – Complete the rest of the layout.

<https://developer.Android.com/codelabs/basic-android-kotlin-training-xml-layouts#5>. Accesat: 14.08.2023

[5] - Create XML layouts for Android –Build the layout in XML.

<https://developer.android.com/codelabs/basic-android-kotlin-training-xml-layouts#3>. Accesat: 14.08.2023

[6] – Layouts in Views. <https://developer.android.com/develop/ui/views/layout/declaring-layout>.

Accesat: 14.08.2023

[7] – Add buttons to your app.

<https://developer.Android.com/develop/ui/views/components/button>. Accesat: 14.08.2023

[8] – FloatingActionButton.

<https://developer.android.com/reference/com/google/android/material/floatingactionbutton/FloatingActionButton>. Accesat: 14.08.2023

[9] – Checkbox. <https://developer.android.com/reference/android/widget/CheckBox>. Accesat:

14.08.2023

[10] – TextView. <https://developer.android.com/reference/android/widget/TextView>. Accesat:

14.08.2023

- [11] – EditText. <https://developer.android.com/reference/android/widget/EditText>. Accesat: 14.08.2023
- [12] – ScrollView. <https://developer.android.com/reference/android/widget/ScrollView>. Accesat: 14.08.2023
- [13] – TabLayout.
<https://developer.android.com/reference/com/google/android/material/tabs/TabLayout>. Accesat: 14.08.2023
- [14] – Spinners. <https://developer.android.com/develop/ui/views/components/spinner>. Accesat: 14.08.2023
- [15] – ViewPager.
<https://developer.android.com/reference/androidx/viewpager/widget/ViewPager>. Accesat: 14.08.2023
- [16] – MaterialCardView.
<https://developer.android.com/reference/com/google/android/material/card/MaterialCardView>. Accesat: 14.08.2023
- [17] – ImageView. <https://developer.android.com/reference/android/widget/ImageView>. Accesat: 14.08.2023
- [18] – TableLayout. <https://developer.android.com/reference/android/widget/TableLayout>. Accesat: 14.08.2023
- [19] – AppBarLayout.
<https://developer.android.com/reference/com/google/android/material/appbar/AppBarLayout>. Accesat: 24.08.2023
- [20] – DrawerLayout.
<https://developer.android.com/reference/androidx/drawerlayout/widget/DrawerLayout>. Accesat: 24.08.2023

[21] – NavigationView.

<https://developer.android.com/reference/com/google/android/material/navigation/NavigationView>. Accesat: 24.08.2023

[22] – Fragments. <https://developer.android.com/guide/fragments>. Accesat: 15.08.2023

[23] – Introduction to activities. <https://developer.android.com/guide/components/activities/intro-activities>. Accesat: 15.08.2023

[24] – Activity. <https://developer.android.com/reference/android/app/Activity>. Accesat: 15.08.2023

[25] – Fragment. <https://developer.android.com/reference/android/app/Fragment>. Accesat: 16.08.2023

[26] – Intent. <https://developer.android.com/reference/android/content/Intent>. Accesat: 16.08.2023

[27] – Intents and intent filters. <https://developer.android.com/guide/components/intents-filters>. Accesat: 16.08.2023

[28] – Data and file storage overview. <https://developer.android.com/training/data-storage>. Accesat: 17.08.2023

[29] – Save simple data with SharedPreferences. <https://developer.android.com/training/data-storage/shared-preferences>. Accesat: 17.08.2023

[30] – SharedPreferences.

<https://developer.android.com/reference/android/content/SharedPreferences>. Accesat: 17.08.2023

[31] – SQL primer. <https://developer.android.com/courses/extras/sql-primer>. Accesat: 19.08.2023

[32] – SQLite is transactional. <https://www.sqlite.org/transactional.html>. Accesat: 19.08.2023

[33] – Save data using SQLite. <https://developer.android.com/training/data-storage/sqlite>. Accesat: 19.08.2023

[34] – SQLiteDatabase.

<https://developer.android.com/reference/android/database/sqlite/SQLiteDatabase>. Accesat: 19.08.2023

[35] – SQLiteOpenHelper.

<https://developer.android.com/reference/android/database/sqlite/SQLiteOpenHelper>. Accesat: 21.08.2023

[36] – Cursor. <https://developer.android.com/reference/android/database/Cursor>. Accesat: 21.08.2023

[37] – Save data in a local database using Room. <https://developer.android.com/training/data-storage/room>. Accesat: 23.08.2023

[38] – Database. <https://developer.android.com/reference/kotlin/androidx/room/Database>. Accesat: 23.08.2023

[39] – Entity. <https://developer.android.com/reference/kotlin/androidx/room/Entity>. Accesat: 23.08.2023

[40] – Dao <https://developer.android.com/reference/kotlin/androidx/room/Dao>. Accesat: 23.08.2023

[41] - Observer. <https://developer.android.com/reference/androidx/lifecycle/Observer>. Accesat: 23.08.2023

[42] – LiveData. <https://developer.android.com/reference/androidx/lifecycle/LiveData>. Accesat: 23.08.2023

[43] - InvalidationTracker.Observer.

<https://developer.android.com/reference/kotlin/androidx/room/InvalidationTracker.Observer>. Accesat: 23.08.2023

[44] – Room. <https://developer.android.com/jetpack/androidx/releases/room>. Accesat: 23.08.2023

[45] – androidx.room. <https://developer.android.com/reference/androidx/room/package-summary>. Accesat: 23.08.2023

[46] – Query. <https://developer.android.com/reference/androidx/room/Query>. Accesat: 23.08.2023

[47] – Write asynchronous DAO queries. <https://developer.android.com/training/data-storage/room/async-queries>. Accesat: 23.08.2023

[48] – Migrate your Room database. <https://developer.android.com/training/data-storage/room/migrating-db-versions>. Accesat: 23.08.2023