MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

TIME COMPASS: O APLICAȚIE DE TIME MANAGEMENT PENTRU ANDROID

LUCRARE DE LICENȚĂ

Absolvent: **Bogdan NANE**

Coordonator ştiinţific: **Șef lucr. ing. Cosmina IVAN**

2012

MINISTERUL EDUCAȚIEI, CERCETĂRII, TINERETULUI ȘI SPORTULUI

UNIVERSITATEA TEHNICĂ
DIN CLUJ-NAPOCA

FACULTATEA DE AUTOMATICĂ ŞI CALCULATOARE
DEPARTAMENTUL CALCULATOARE

DECAN,
Prof. dr. ing. Liviu MICLEA

VIZAT,
DIRECTOR DEPARTAMENT,
Prof. dr. ing. Rodica POTOLEA

Absolvent: **Bogdan NANE**

TIME COMPASS: O APLICAȚIE DE TIME MANAGEMENT PENTRU ANDROID

1. **Enunţul temei:** *Proiectul își propune realizarea unei aplicații pentru dispozitive mobile a cărei funcționalități au la bază concepte de time management și are rolul de a îl ajuta pe individ în planificarea activităților*

2. **Conţinutul lucrării:** *Cuprins, Introducere, Obiectivele proiectului, Studiu bibliografic, Analiză și fundamentare teoretică, Proiectare de detaliu și implementare, Testare și validare, Manual de instalare și utilizare, Concluzii, Bibliografie, Anexe.*

3. **Locul documentării:** Biblioteca Universității Tehnice din Cluj-Napoca.

4. **Consultanţi: Șef lucr. ing. Cosmina IVAN**

5. **Data emiterii temei:** 1 Noiembrie 2011

6. **Data predării:** 20 Septembrie 2012

Absolvent: _____

Coordonator ştiinţific: _____

Declaraţie

Subsemnatul *Bogdan NANE*, student al Facultăţii de Automatică şi Calculatoare, Universitatea Tehnică din Cluj-Napoca, declar că ideile, analiza, proiectarea, implementarea, rezultatele şi concluziile cuprinse în această lucrare de licenţă constituie efortul meu propriu, mai puţin acele elemente ce nu îmi aparţin, pe care le indic şi recunosc ca atare.

Declar de asemenea că, după ştiinţa mea, lucrarea în această formă este originală şi nu a mai fost niciodată prezentată sau depusă în alte locuri sau alte instituţii decât cele indicate în mod expres de mine.

Data: 20 Septembrie 2012

Absolvent: **Prenume NUME**
Număr matricol: 21010930
Semnătura:_____

Cuprins

1 Introducere

1.1 Context general

În ziua de azi omul modern are tot mai multe de făcut, își ia din ce în ce mai multe angajamente atât pe plan profesional cât și personal, însă indiferent de numarul de sarcini pe care și le propune să le îndeplinească, acesta nu poate fi în mai multe locuri în același timp, numarul de acțiuni pe care la poate face în paralel este mic iar timpul pe care îl are la dispoziție este limitat.

În acest context timpul devine o resursă prețioasă deoarece trebuie folosit în mod întelept, în caz contrar individul nu își va putea duce la până la capăt angajamentele în timp util, exisă posibilitatea ca acesta să se piardă în detalii investind timp în activități neimportante și nereușind să le facă pe cele importante și pe lângă toate acestea individul poate deveni stresat datorită repercusiunilor ce le poate avea neîndeplinirea sarcinilor.

1.2 Contextul Aplicației

Pentru a putea folosi cât mai eficient timpul și a reduce factorul de stress asociat cu pierderea acestuia și neatingerea obiectivelor, individul trebuie să își planifice din timp activitățile. În acest scop au apărut o serie de tehnici și metodologii care să-l îndrume astfel încât să-și planifice eficient activitățile și să tină cont de diversele circumstanțe de care depind acestea. Însă, și pentru stăpânirea acestor tehnici este nevoie o serie de de resurse printre care se numără și timp.

Evoluțiile tehnologice privind dispozitivele mobile vin în sprijinul omului modern. În ultima vreme au apărut o serie de dispozitive mobile cu performanțe bune privind puterea de calcul, capacitate de stocare, durată de viață a bateriei, aceste dispozitive având dimensiune redusă și sunt din ce în ce mai frecvent întâlnite la îndemâna utilizatorului. Astfel aceste dispozitive ar putea să vină în sprijinul utilizatorului ajutându-l să-și gestioneze eficient timpul. Acest lucru ar fi posibil prin intermediul unor aplicații care ar implementa principii de *time managemen*t și ar fi ușor de folosit oferind informațiile potrivite la momentul potrivit

1.3 Rezumat

Subiectul lucrării de față constă într-o aplicație proiectată de rula pe dispozitive mobile ce rulează pe sistemul de operare Android versiunea 4, și care, prin funcționalitățile pe care le oferă implementează principii de *time management* utile pentru utilizator ajutandu-l să-și planifice eficient activitățile.

Astfel că în capitolul 2 vor fi prezentate obiectivele pe care le urmărește lucrarea de față, în capitolul 3 vor fi prezentate principalele concepte de time management de care s-a ținut cont în procesul de realizare a aplicației, precum și câteva produse software de time management deja existente pe piață. În capitolul 4 sunt oferite detalii legate de tehnologiile folosite pentru realizarea aplicației precum și use case-urile pe care aceasta ar trebui să le îndeplinească. Detalii lagate de implementare sunt prezentare în capitolul 5 iar în capitotul 6 este prezentat un scenariu de test pentru aplicație urmând ca în capitolul 7 să fie prezentate instrucțiunile de folosire a

aplicației. Iar în cele din urmă, ultimul capitol este prezentată o concluzie a celor realizate precum și câteva direcții de dezvoltare a aplicației.

2 Obiectivele proiectului

În cadrul acestui proiect se urmărește realizarea unei aplicații software proiectată pentru a rula pe smartphone-uri cu Android și care să se modeleze pe principii de *time management.*

Aplicația ar oune la dispoziția utilizatorului următoarele funcționalități:

- Modalitate de grupare a activităților, astfel utilizatorul are posibilitatea de a grupa activitățile proprii în funcție de diverse criterii
- Modalități de vizualizare prin intermediul cărora utilizatorul își poate forma o imagine de ansablu asupra situației sarcinilor pe care le are de îndeplinit. Informațiile obținute de la aceste moduri de vizualizare l-ar ajuta pe utilizator în luarea deciziilor privind planificarea timpului disponibil pentru efectuarea activităților planificate. Aplicația i-ar oferi utilizatorului posibilitatea de a vizualiza situația activităților planificate pentru o anumită zi sau pentru o anumită lună calendaristică
- Un sistem prin intermediul căruia utilizatorul poate asocia un anumit grad de importanță activităților ajutandu-l pe acesta să facă o distincție între activitățile importante și cele mai puțin importante și să își centreze atenția asupra celor importante.
- Asocierea unui context fiecărei activități
- O interfață grafică intuitivă

Conceptul de context care ar putea fi asociat unei activități poate fi vazut ca un punct de interest situat pe o hartă, pentru implementarea acestui concept va fi integrat un serviciu de cartografiere ce va fi folosit pentru redarea și înregistrarea punctelor de interes reprezentând diferitele contexte asociate activităților și de care depinde îndeplinirea acestora.

De asemenea aplicația din discuție va furniza un sistem de notificări în funcție de poziția geografică a utilizatorului la un moment dat. Astfel în cazul în care utilizatorul se află la un moment dat în apropierea unui punct de interes aferent unui context, aplicația îl informa despre acest aspect prin intermediul unei notificări, informandu-l în același timp și despre activitățile care sunt corelate cu punctul de interes respectiv.

Aplicația din discuție o să-i permită utilizatorului să înregistreze sarcinile respectiv activitățile pe care dorește să le întreprindă. Activitățile respective vor fi memorate într-o bază de date stocată local în memoria dispozitivului, astfel utilizatorul nu va trebui să își facă vreun cont online de client sau să depindă de o conexiune la internet pentru a-și accesa datele.

În capitolul următor vor fi detaliate conceptele acitvitate, context și principiile în funție de care se recomandă gruparea activităților din perspectiva unor principii de time management.

3 Studiu bibliografic

În acest capitol vor fi prezentate o serie de concepte și tehnologii revelante pentru domeniul aplicației și de care s-a ținut cont în etapa de implementare a proiectului.

Astfel în cărțile [1] și [2] sunt prezentate concepte și metodologii de time management prin intermediul cărora individul își poate gestiona eficient timpul și astfel să fie mai productiv. Scopul acestora este cel de a îl ajuta pe individ să indentifice activitățile importante și determinandu-l să investească timp pentru soluționarea acestora.

În referința [3] este prezentat un argument în favoarea proiectării aplicațiilor de time management pentru dispozitve mobile precum smartphone-uri deoarece acestea au un grad mai mare de disponibilitate față de unitățile de tip desktop, fiind de multe ori mai la îndemâna utilizatorului decât acestea. Acest aspect le ofera utilizatorilor un grad mare de flexibilitate dându-le posibilitatea de a avea în orice moment o vedere de ansablu asupra tuturor sarcinilor pe care trebuie să se le îndeplinească și de a adauga sarcini noi sau de a efectua actualizări în orice moment în funcție de necesități.Astfel pe baza celor descrise in referința [3] s-a ales proiectarea aplicației pentru o platformă software pentru smartphone-uri.

În articolul [4] este prezentat un studiu de piată privind cele mai populare sisteme de operare pentru dispozitive mobile și răspândirea acestora. Concluzia acestui studiu este ca Android este cel mai răspandit sistem de operare pentru dispozitive mobile, acest rezultat a determinat ca aplicația de față să fie proiectată pentru această platformă.

În referințele [5] [6] [7] este prezentată platforma Android fiind oferite detalii legate de sistemul de operare Android, a componetelor acestuia și a mijloacelor prin care se pot dezvolta aplicații care să ruleze pe acesta.

În referința [5] sunt prezentate diferitele versiuni de Android și facilitățile pe care le ofera fiecare astfel pe baza celor prezentate s-a decis ca platforma de dezvoltare să fie Android versiunea 4 Ice Cream Sandwich deorece este o versiune recentă, stabilă, ce oferă facilități de dezvoltare a interfețelor grafice pentru aplicații.

În referințele [6] si [7] sunt prezentate concepte de care ar trebui să se țină cont în proiectarea și implementarea aplicațiilor pentru Android.

În proiectarea interfeței grafice s-a ținut cont de principiile menționate în referința [5] și care au fost detaliate în secțiunea 5.2.1.

În continuare sunt prezentate conceptele de time management pe baza cărora a fost proiectată aplicația de față.

3.1 Concepte de time management

În cartea [1] este prezentata o metodologie denumita "*Getting Things Done (GTD)"* a cărui rol este cel de a îl ajuta pe individ să își gestioneze eficient timpul alocat pentru indeplinirea diverselor sarcini. La baza metodologiei propusă de autorul stau 2 concepte cheie: conceptul de control și conceptul de perspectivă, metoda fiind menită să îi ofere individului o imagine de ansamblu asupra sarcinilor pe care trebuie să le îndeplinească și să-l ajute să elaboreze o succesiune de acțiuni ce trebuie întreprinse pas cu pas pentru îndeplinirea acestora.

În Time Management (Gestionarea timpului) un rol important îl are gradul de prioritate asociat fiecărei sarcini pe care trebuie să o îndeplinească individul. Metoda descrisă presupune efectuarea unei retrospective săptămânală asupra situației sarcinilor,ce aparțin diverselor nivele de interest ale individului. Perspectivele obținute în urma restrospectivei ar urma apoi să

influențeze prioritățile individului, și prin urmare ar determina și gradul de prioritate asociat fiecărei sarcini ce trebuie îndeplinită de către individ.

De asemenea, pentru fiecare sarcină ar trebui indentificat contextul de care depinde îndeplinirea acesteia, iar sarcinile care au același context asociat ar trebui grupate într-o lista în funcție de acesta.

GTD este prezentată ca o metodă ce se bazează pe înregistrarea pe un suport extern a sarcinilor ce trebuie îndeplinite de către individ precum și a elementelor de care depinde indeplinirea acestora, monitorizarea sarcinilor și obținerea, la nevoie, de informații legate de acțiunile ce trebuie întreprinse pentru indeplinirea sarcinilor la un moment dat. Metoda furnizează atfel un surport extern ce îl ajută pe individ să își aducă aminte ce acțiunit trebuie întreprinse și când trebuie întreprinse pentru a își atinge obiectivele, fiind un element complementar sistemului de antenționare a creierului uman mai puțin eficient.

De multe ori se presupune că individul ajunge în impas în procesul de îndeplinire a sarcinilor datorită unei planificări necorspunzătoare a activităților, de aceea se recomandă ca individul să își planifice în avans activitățile ce trebuie executate pentru îndeplinirea sarcinilor. Acest aspect i-ar ușura astfel situația individului, eliminând o parte din stressul generat găsirea unei soluții pe moment.

În cartea [3] sunt prezentate o serie de principii și metode de time management care aplicate corespunzător l-ar ajuta pe individ să își îndeplinească în mod eficace sarcinile și să își atingă obiectivele propuse.

Conform celor menționate în lucrarea [3] există trei categorii sau generații în funcție de care pot fi grupate uneltele de time management

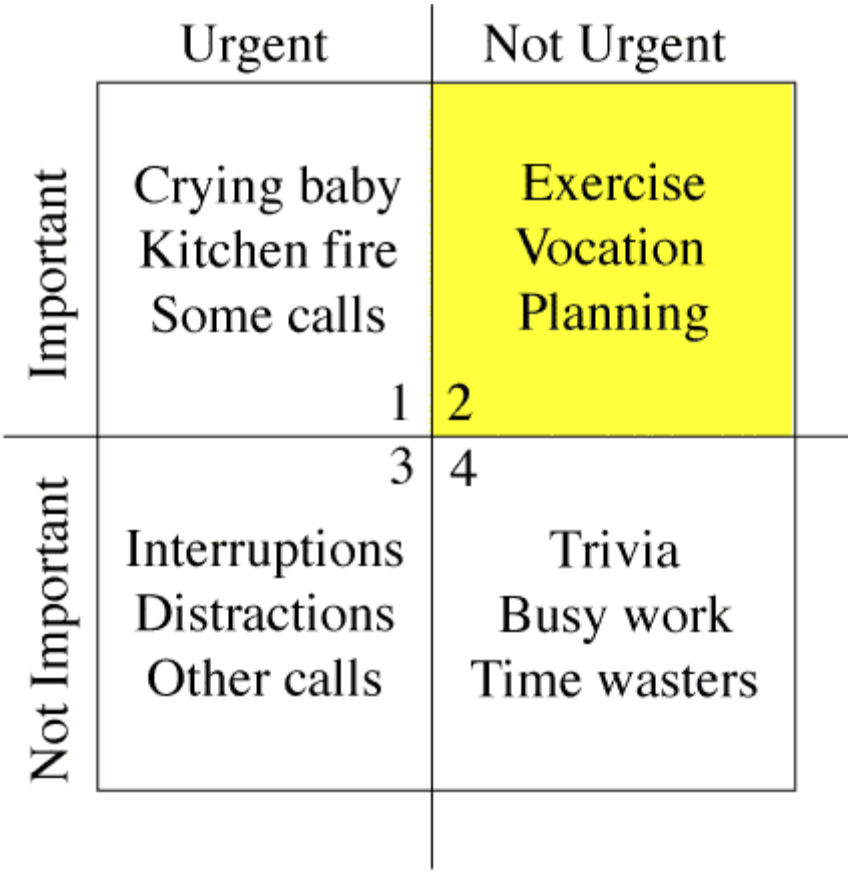
1. De primă generație. Aici intră listele de sarcini, liste în care individul își înregistrează toate activitatile pe care dorește să le întreprindă și gruparea acestora în funcție de diverse criterii.
2. De a doua generație. Mijloacele din aceasta categorie presupun ținerea unei evidențe a activităților individului în funcție de termenul limita la care acestea ar trebui îndeplinite (ex calendar, jurnal)
3. De generatia a treia. Mijloacele din această categorie au la bază sistemul de valori a individului, acesta acordându-le importanță activităților pe care le consideră relevante pentru îndeplinirea obiectivelor importante.

În cadrul cărții[3] autorul recomandă ca sistemul de valori al individului să fie elementul principal în funcție de care acesta ar trebui să stabilească gradul de prioritate sarcinilor pe care trebuie să le îndeplinsească, astfel sarcinile care îl ajută pe individ să își atingă obiectivele importante ar trebui să li primească mai multă atenție, uneori în detrimentul sarcinilor care din punct de vedere a timpului ar fi considerate urgente dar care însa nu sunt la fel de importante.

Activitățile, sarcinile, obiectivele individului ar trebui încadrate într-un sistem cu 2 axe:

1. axa importanței. Poziția de pe această axă arată cât de important este un anumit obiectiv pentru individ sau cât de importantă este îndeplinirea unei anumite sarcini pentru individ.
2. axa timpului. Poziția de pe această axă arată cât de repede ar trebui îndeplinit un anumit obiectiv, o anumită sarcină, sau întreprinsă o anumită activitate.

Prin intersectarea celor 2 axe se formează 4 cadrane:



Figură 3.1 Matricea de priorități [3]

Cadranul 1: Aici sunt încadrate acțiunile importante și urgente și care ar trebui să fie prioritare pentru individ, îndeplinirea acestora având prioritate față de celelalte activități.

Cadranul 2: Aici sunt încadrate acțiunile importante dar care nu sunt urgente, de obicei acestea sunt corespunzatoare scopurilor pe termen lung. Acțiunile din acesta categorie ar trebui să fie urmatoarele ca și prioritate dupa cele din cadranul 1.

Cadranul 3: Aici sunt încadrate acțiunile urgente dar neimportante

Cadranul 4: Aici sunt încadrate acțiunile ce nu sunt urgente și care sunt neimportante. Acest acțiuni de multe ori duc la pierderea timpului și care la nevoie ar putea fi omise în favoarea activităților din celelalte cadrane.

Aceasta matrice de 2x2 mai este cunoscută informal ca și matricea Eisenhower.

3.2 Produse software de time management existente pentru dispozitive mobile

La ora actuală, pe piață exista o gama variata de aplicații care se modeleaza pe diferite aspecte legate de domeniul de time management, în continuare fiind descrise câteva dintre acestea.

MyLifeOrganized

Conform referintei [8] MyLifeOrganized este o aplicație care poate fi instalata pe dispozitivele utilizatorilor și care îi ajuta în gestionarea timpul și activităților. Această aplicație îi ajută pe utilizatori să țină o evindentă a activităților pe care vor să le întreprindă.

Interfața grafică este intuitivă oferind 2 modalități de vizualizare:

- Primul mod de vizualizare este cel de lista de activitati (task-list) unde sunt afișate toate activitățile care necesită atenția utilizatorului la momentul apelării, utilizatorul fiind informat de sarcinile a căror termen limita a expirat recent, care trebuie să se desfășoare în ziua respectivă sau care urmează să se desfășoare în ziua imediat următoare.

Utilizatorul poate bifa sarcinile pe care le-a îndeplinit și astfel acestea automat dispar din lista de sarcini, utilizatorului ramânându-i să se concentreze pe sarcinile rămase.

- Al doilea mod de vizualizare este modul de vizualizare ierarhic.În cadrul acestui tip de vizualizare utilizatorul poate observa toate sarcinile pe care le are de întreprins sub forma unei ierahii.

Prin intermediul acestui mod utilizatorul poate să vadă:

- Dependențele dintre sarcini, relațiile dintre sarcini și sub-sarcini
- Gradul de importanța a sarcinilor
- Un indicator ce arată cât de mult progress a fost făcut în îndeplinirea unei sarcini
- Sarcinile care au fost indeplinie

Aplicația îi permite utilizatorului să definească o structura ierarhica de sarcini, astfel fiecarei sarcini i se pot adăuga mai multe sub-sarcini.

Despre fiecare sarcina utilizatorul poate specifica:

- Contextul de care depinde îndeplinirea sarcinii respective (ex.acasa, la birou)
- Detalii cu ajutorul cărora activitățile sunt încadrate în timp (data de început, data de încheiere)

Există o serie de indicatori cu ajutorul cărora utilizatorul poate specifica cât de importantă, respectiv urgentă, este o anumită activitate.

Aplicația îi oferă posibilitatea utilizatorului de a fi notificat despre anumite activități din lista în funcție de termenul limită a acestora.

Aplicația este disponibila pentru PC, Iphone, Android și oferă posibilitatea de a sincroniza conținutul de pe diversele tipuri de aplicații client prin intermediul unui serviciu de cloud pe baza de cont de utilizator.

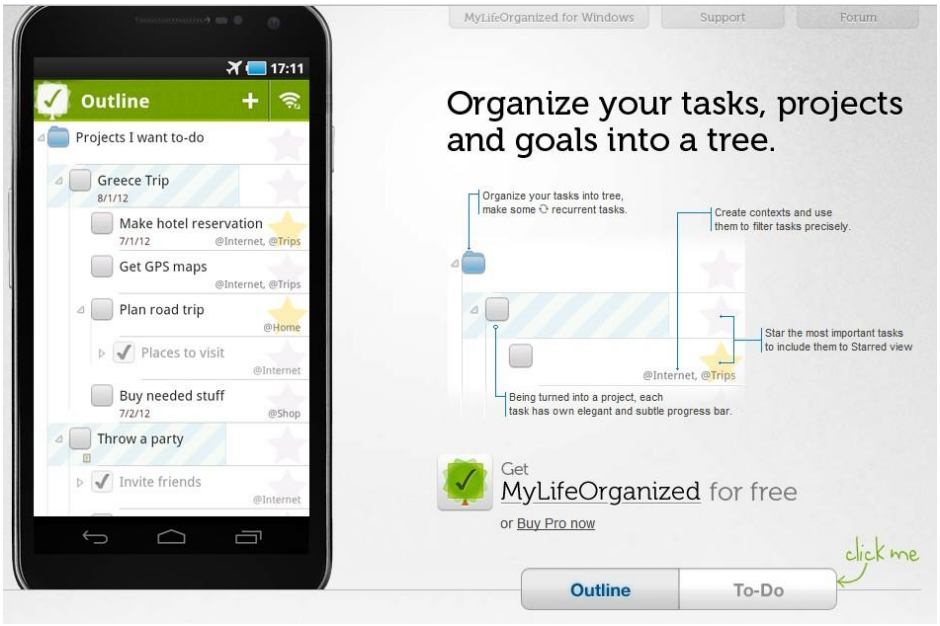
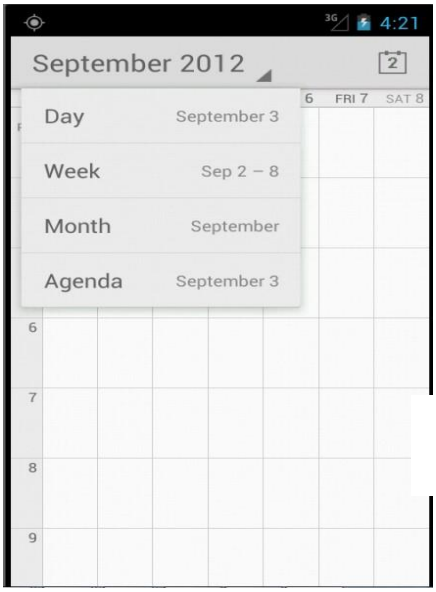


Figura 3.2 Interfață grafică MyLifeOrganized

Google Calendar

Conform referinței [9] Google calendar este un serviciu online oferit de compania Google care le pune la dispoziția utilizatorilor o serie de funcționalități de bază pentru gestionarea timpului și activităților.



Figură 3.3 Intefață grafică Google Calendar pentru Android

Spre deosebire de MyLifeOrganized elementul central al interfeței utilizator este calendarul prin intermediul căruia utilizatorul poate să vadă ce activități are planificate într-o anumita zi.

Serviciul oferă mai multe modalități de vizualizare a activităților:

- În funcție de zi. Acest mod de vizualizare îi informează pe utilizatori despre evenimentele dintr-o anumită zi redându-le sub forma unei liste, elementele ei fiind trecute în ordine cronologică

- În funcție de săptămână. Mod de vizualizare ce le permite utilizatorilor să vizualizeze evenimentele dintr-o anumită săptămână sub forma unui tabel în care coloanele reprezintă zile iar rândurile reprezintă ora din intervalul orar al unei zile.
- În funcție de lună. Acest mod de vizualizare ia forma unei matrici în care fiecare căsuță reprezintă o anumită zi din lună, iar în cadrul casuței apare numele evenimentelor ce au a fost planificate în data respectivă
- Mod agendă. Acest mod de vizualizare ia forma unei liste de activități în care apar toate activitățile în ordine cronologică

Utilizatorul își poate defini propria lista de sarcini iar pentru fiecare sarcină utilizatorul poate defini:

- Data limită până la care trebuie terminată sarcina.
- Câteva informații sub forma de text

De asemenea utilizatorul își poate defini mai multe calendare, fiecare având o evindetă proprie de sarcini.

De asemenea utilizatorii pot face schimb de calendare și să vadă planificările altora sau să adauge activitățile altora la propriul calendar.

Toate aceste functionalități sunt disponibile tuturor utilizatorilor care au cont de Google.

Serviciul poate fi accesat folosind un browser web astfel o conexiune la internet este tot timpul necesară, există de asemenea și o aplicație client pentru Google Calendar disponibila pentru Android, însă și aceasta necesită cont de utilizare înregistrat la Google. Sincronizarea datelor între aplicația client de pe Android și serviciul Google se face automat.

ToDoMatrix

Conform referintei [3] ToDoMatrix este o aplicație de time management dezvoltată de către compania REXWireless software, disponibilă pentru BlackBerry și Iphone. Prin facilitățile pe care le oferă aplicația se modelează atât pe metodologia de time management *Getting Things Done* precum si cea de *First Things First*.

Similar cu celelate 2 produse menționate anterior această aplicație îi dă utilizatorului posibilitatea de a își defini propriile activități, de a le încadra în timp, utilizatorul putând specifica pentru fiecare activitate durata estimată, data limită până la care ar trebui îndeplinită precum și contextul de care depinde îndeplinirea activității respective.

Spre deosebire de celelalte aplicații descrise mai sus, prin intermediul aplicației utilizatorul își poate defini propria structură de directoare în funcție de care își poate grupa activitățile. Aplicația oferă utilizatorului diverse modalități de vizualizare a activităților, într-o manieră mult mai elaborată decât în produsele menționate anterior. Vizualizarea activităților se poate face în funcție de diferite criterii, precum zi, context, activități care sunt în curs de desfășurare, activități cărora le-a expirat data limită și multe altele. De asemenea sunt oferite functionalități de căutare și sortare a activităților în funcție de diferite criterii.

Printre functionalitățile oferite de ToDoMatrix se numără și un sistem de notificări care să-i aducă aminte de activitățile cărora ar trebui să le acorde atenție la un moment dat.

Toate datele utilizatorului sunt stocate local în memoria dispozitivului iar interfața grafică este ușor configurabilă fapt ce îi permite utilizatoului să stabilească gradul de complexitate și volumul de informație oferit de interfața grafică a aplicație.

De asemenea aplicația mai pune la dispoziția utilizatorului o funcționalitate denumită quick review wizzard prin intermediul căreia utilizatorul își poate face o retrospectivă și să

obțină o imagine de ansamblu asupra situației activităților planificate, imagine care l-ar ajuta să se organizeze mai eficient în viitor. Acest element nefiing prezent la celelalte aplicații descrise

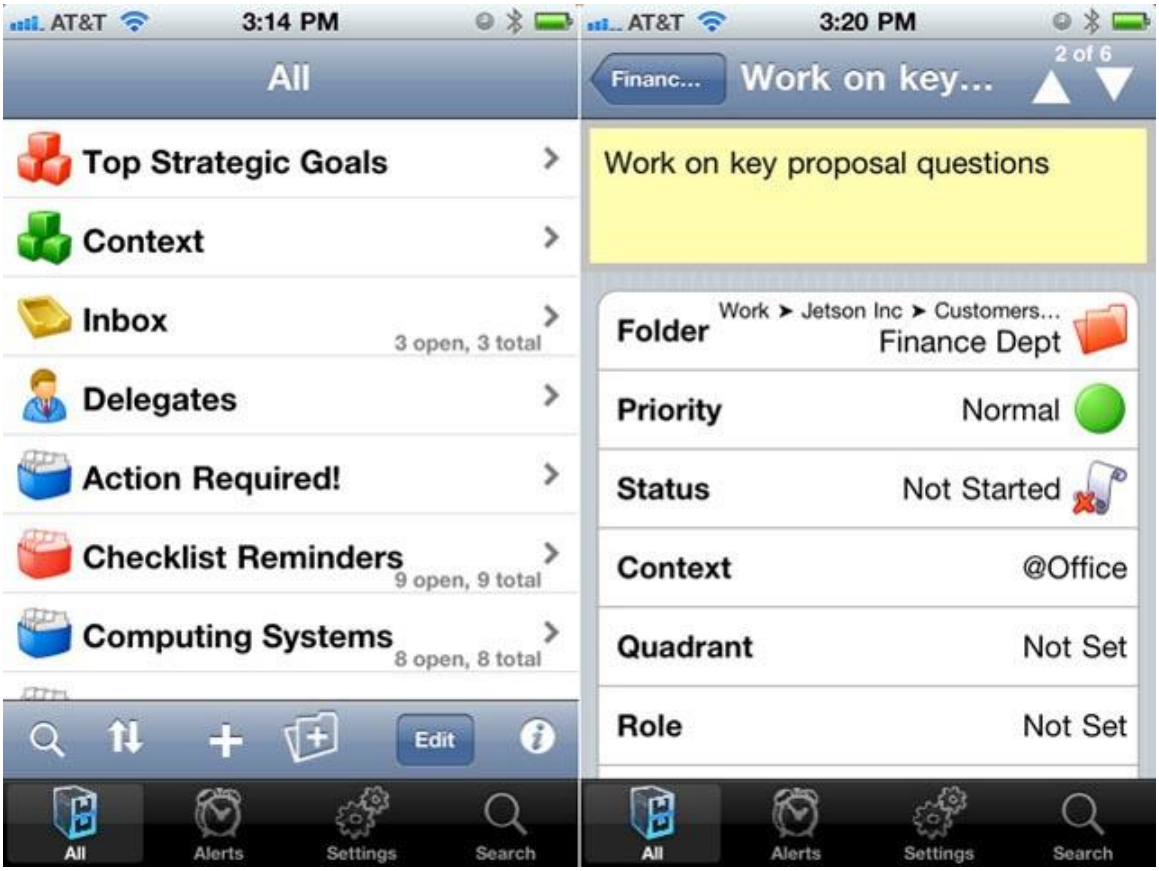


Figura 3.4 Interfață grafică ToDoMatrix (iPhone)

.

În urma prezentării produselor de mai sus se pot deduce următoarele:

- Google Maps are o funcționalitate redusă comparativ cu celelalte 2 produse însă este ușor de folosit iar utilizatorul are tot timpul o imagine de ansamblu asupra activităților ce sunt planificare într-o luna datorită componentei de calendar.
- MyLifeOrganized oferă functionalități acceptabile, oferind mai multe funcționalitate decât Google Calendar deși interfata grafică nu este la fel primitore dar are la bază principii solide de time management inspirate din metodologia "*Getting Things Done"*
- ToDoMatrix este o soluție foarte apropiată de o solutie completî implementând atât elemente ce țin de motdologia "*Getting Things Done"* precum și "*First Things First"* însă interfața grafică e complexă.

Aplicația ce face subiectul acestei lucrări va fi similara cu MyLifeOrganized implementând o mare parte din elementele ce țin de metodologia "Getting Things Done", însă va fi similară și cu ToDoMatrix deoarece va avea și elemente din "First Things First"iar la interfața grafică va avea o componentă de calendar similara cu cea din Google Calendar deoarece îi oferă utilizatorului o imagine de ansamblu într-o manieră intruitivă.

În următorul capitol vor fi prezentate principalele cazuri de utilizare la care va răspunde aplicația din discuție precum și tehnologiile folosit pentru realizarea acesteia.

4 Analiză şi fundamentare teoretica

4.1 Tehnologii utilizate

În cele ce urmează vor fi detaliate tehnologiile folosite în realizarea proiectului.

Proiectul a fost dezvoltat folosind tehnologii și tool-uri open-source, bine documentate, larg răspândite și care au în spate comunități de utilizatori și dezvoltatori preocupați de utilizarea eficientă și dezvoltarea continuă a acestora.

Aceasta alegere prezinta urmatoarele avantaje:

- Cost redus privind dezvoltarea aplicației, nefiind necesar achitarea unor taxe de licențiere pentru tehnologiile folosite;
- Există support din partea comunităților care susțin tehnologiile respective cum ar fi tutoriale, forum-uri de discuții pe diverse tematici (erori, bune practici) și access la diverse resurse precum librării specializate.

Aplicația a fost proiectată pentru dispozitive mobile ce rulează sistemul de operare ***Android*** prin urmare, aceasta a fost realizată folosind *Android* ***SDK***. Mediul de programare ales este ***Eclipse IDE***. Codul sursă a fost scris în limbajul ***JAVA*** iar pentru implementarea anumitor functionalități ale aplicației au fost folosite servicii oferite de *Google* prin intermediul *Google* ***API***.

Detalii legate de tehnologiile utilizate precum și justificarea folosirii acestora în realizarea aplicației vor fi detaliate în cele ce urmează.

4.1.1 Android

Android un produs open source creat de *Google* iar de dezvoltarea sa contiună ocupandu-se *Open Handset Alliance* [6] și constă într-o colecție de componente software ce cuprinde un sistem de operare, middleware și aplicații cheie, destinat dispozitivelor mobile, printre care și smartphone-uri, cu scopul de a putea folosi eficient resursele acestora în oferirea diverselor funcționalități utilizatorilor.

Caracteristici Android

Printre elementele caracteristice ale sistemului Android se numără urmatoarele[6] :

- *Platformă open-source*. Android este un produs open source, distribuit sub licența Apache License versiunea 2 (cu excepția nucleului de Linux care se află sub licență GPL versiunea 2), o licența destul de permisivă ce ofera libertatea de a copia, distribuii și modifica continutul în mod liber fără nici un cost de licențiere, rămânând la alegerea dezvoltatorilor dacă distribuie sursele modificate sub aceeași licența sau nu. Singurul element din Android care face excepție de la aceasta regula este nucleul de Linux care se afla sub licență GPL versiunea 2 ce presupune că orice modificare a surselor trebuie sa fie făcută publică și ditribuită în continuare gratuit sub licenta GPL versiunea 2.
- Portabilitatea rulării pe o gamă largă de hardware curente şi viitoare. Toate programele sunt scrise în Java şi executate pe maşina virtuală Dalvik, existând posibilitatea portării

codului pe ARM, x86 şi alte arhitecturi. Interfețele utilizator pot fi adaptate la orice rezoluție pe care o pot avea dispozitivele.
- Arhitectură bazată pe comopenete ce permite reutilizarea componentelor în diverse aplicații precum și înlocuirea acestora.
- Ofera suport pentru grafică 2D şi 3D utilizand OpenGL ES 1.0
- Posibilitatea de a stoca datele sub forma unor baze de date de tip SQLite.
- Suport pentru formate media uzuale audio, video, și imagini (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF)

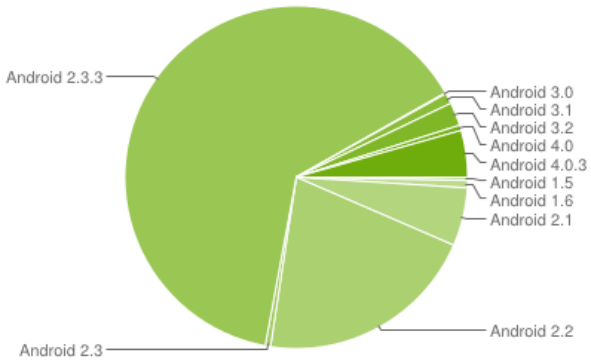
De asemenea *Android*-ul pune la dispoziția devoltatorilor o serie de unelte utile pentru dezvoltarea aplicațiilor precum un emulator, unelte pentru debugging, pentru măsurarea performanțelor aplicațiilor, și posibilitatea de intregrare cu *Eclipse IDE*

Android este dezvoltat continuu, fiecare versiune lansată aduce îmbunătățiri la diverse componente deja existente precum și elemente și functionalități noi care sa utilizeze cât mai bine resursele fizice ale dispozitivelor.

Versiuni

Fiecare versiune de Android lansată reprezintă un nivel API (API level) și este un criteriu important de care trebuie ținut cont în procesul de proiectare a aplicațiilor pentru această platformă. Astfel, dacă o aplicație este proiectată pentru un nivel API mare corespunzatoare unei versiuni recente de *Android* atunci aceasta nu va putea funcționa pe dispozitive ce rulează versiuni anterioare iar dacă o aplicație este proiectată pentru un nivel API mic, aceasta va putea rula pe mai multe dispozitive ce au instalate versiuni superioare de Android însă nu vor beneficia de toate facilitățile oferite de versiunile superioare.[5]

Versiunea 4, precum şi versiunile superioare, de Android face disponibile aplicaţiilor pentru smartphone-uri o serie de elemente ce înainte se regăseau doar în versiunea 3 (o versiune destinată strict tabletelor), elemente ce permit realizarea unor interfețe grafice flexibile și performante și care îmbunătățesc experiența utilizatorului precum și alte elemente care îmbunatațesc performanța aplicațiilor. Pe baza acestui criteriu s-a ales Android 4.03 –Ice Cream Sandwich corespunzator nivelului API 15 ca și platformă țintă
În tabelul 4.1 sunt enumerate toate versiunile de Android precum și răspândirea în rândul dispozitivelor ce rulează pe această platformă.



Figură 4.1 Grafic distribuție versiuni Android (preluat de pe http://developer.android.com)

Tabel 4.1 Versiuni de Android (preluat de pe http://developer.android.com)

Platformă	Denumire	API Level	Răspândire

1.5	Cupcake	3	0.2%
1.6	Donut	4	0.4%
2.1	Eclair	7	3.7%
2.2	Froyo	8	14%
2.3 - 2.3.2	Gingerbread	9	0.3%
2.3.3 - 2.3.7		10	57.2%
3.1	Honeycomb	12	0.5%
3.2		13	1.6%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.1%
4.0.3 - 4.0.4		15	20.8%

Vedere de ansamblu

Android este structurat, conform figurii 4.2, pe mai multe nivele, fiecare nivel depinzând de functionalitățile oferite de nivelul inferior acestuia.

Figura 4.2 Structura Android

Nucleu de linux (linux kernel)

La baza arhitecturii Android se află un nucleu de Linux 2.6 care asigură functionalităţi de bază ale sistemului precum gestionarea sistemului de fişiere, gestionarea proceselor, a memoriei, a elementelor ce țin de rețelistică și a driverelor.[6]

Librării

Al doilea nivel al arhitecturii constă într-un set de librării C/C++ ce stau la baza funcționării diverselor componente ale sistemului Android.
Printre aceste librarii se numara:

- Un subset al librariei standard C (libc) special adaptată pentru dispozitive mobile pe care rulează Linux.
- O serie de librării media ce oferă suport pentru formate audio şi video uzuale precum MPEG4, H.264, MP3, AAC, AMR, JPG şi PNG
- Suport pentru grafica 2D (funcționalități oferite de către motorul grafic SGL) şi 3D (funcționalități oferite de libraria OpenGL ES 1.0)
- FreeType -o librărie ce asigură redarea fonturilor vectoriale sau de tip bitmap
- SQLite-**Baza de date cu ajutorul căreia se pot stoca datele persistente**.

SQLite

SQLite este o bibliotecă, scrisă în ANSI-C, în cadrul căreia se găsește implementarea unui motor de **baze de date (database engine)** tranzacțional, de sine-stătător, care nu necesită server sau configurări speciale pentru a rula.

În SQLite citirea, respectiv scrierea datelor se face direct pe disk. O bază de date completă SQL cu mai multe tabele, indexi, triggere şi vederi, va fi stocată pe disk sub forma unui singur fişier. Formatul fişierelor în care este stocată o baza de date SQLite este conceput astfel încat să fie cross-platform facilitând portarea bazei de date între sistemele 32-bit şi 64-biţi sau între arhitecturi big-endian si little-endian.

SQLite este o bibliotecă compactă. Având toate caracteristicile activate, dimensiunea bibliotecii poate fi mai mică de 350KiB, (depinzând de de platforma ţintă şi setările de optimizarea compilatorului). În cazul în care funcţile opţionale sunt omise, dimensiunea bibliotecii SQLite poate fi mai mică de200KiB.
Astfel SQLite devine motor SQL ideal pentru dispozitive cu memorie limitată, precum telefoane mobile, PDA-uri, playere MP3.

Proiectul SQLite este administrat de o echipă internaţională de dezvoltatori care continuă să extindă capacităţile SQLite cu scopul de a spori spori fiabilitatea şi performanţele acestuia menţinând în acelaşi timp compatibilitate cu interfaţa publică, sintaxa SQL, baze de date şi formatul de fişier.

Tipurile de date ce pot fi folosite in SQLite sunt TEXT (echivalentul tipului de date String din Java), INTEGER (echivalentul tipului de date long din Java) şi REAL (echivalentul tipului de date double din Java) însa nu se face validarea automată a tipurilor de date înainte ca acestea să fie introduse în baza de date.[10]

În Android se regăsește SQLite versiune 3, fiecare baza de date SQLite fiind salvată în memoria dispozitivului în directorul
`DATA/data/NUMELE_APLICATIEI/databases/NUMELE_BAZEI_DE_DATE`.

SDK-ul de Android pune la dispoziția devoltatorilor o serie de clase java prin intermediul cărora pot fi acesat SQLite și efectuate diverse operații asupra bazelor de date. Aceste clase se regăsesc în pachetele `android.database` și `android.database.sqlite`

Din punct de vedere legal SQLite este încadrat în domeniului public făcând astfel posibilă utilizarea codului în orice scop (atât comercial cât și privat) fără nici o constrângere din punct de vedere legal[7].

Android runtime

Android include un set de librării care susţin o mare parte din funcţionalitatea pusă la dispoziţie de limbajul de programare *Java*. Mai exact pune la dispoziție un set restrâns de functionalități oferite de *Java SE 1.4* din care au fost excluse elementele ce țin de swing și awt.

Elementele .class sunt convertite de systemul *Android* in fieisere .dex (*Dalvik* executable,un tip special de bytecode optimizat pentru a rula pe dispozitive mobile necesitând puține resurse și consum mic de energie). Fisierele *.dex* sunt rulate de către mașina virtuala *Dalvik*. Fiecărei aplicații fiindu-i alocate o instanța a mașinii virtuale *Dalvik*[6]

Application Framework

Acest nivel ofera dezvoltatorilor toate funcționalitățile necesare utilizării resurselor oferite de sistem precum și dezvoltarea de aplicații care să le folosească. Este organizat pe componente astfel încat să permită ca functionalitățile unei aplicații să poată fi utilizate de către alte aplicații cu condiția de a respecta constrângerile de securitate impuse de framework.

Tot în cadrul acestui nivel al ierarhiei sistemului se pot gasi o serie de servicii și mecanisme ce stau la baza funcționării tuturor aplicațiilor precum:

- **un set bogat de componente UI** ce pot fi folosite de către dezvoltator în proiectarea interfețelor grafice pentru aplicații.
- **Activity Manager :** controlează ciclul de viaţă al aplicaţiilor, modul de comunicare dintre acestea, precum și navigarea între aplicații.
- **Resource Manager :** controlează accesul la resursele non-cod, cum ar fi: string-uri, layout-uri, imagini etc.
- **Content providers :** Aceste componente permit ca datele unei aplicații să fie folosite de către alte aplicații
- **Notification Manager :** permite diferite modalități de notificare a utilizatorilor.

Applications

Ultimul nivel este reprezentat de aplicațiile propriuzise, create de către dezvoltatori și care pot fi folosite de către utilizatorii finali.

Androidul vine implicit cu o serie de aplicații pre-instalate însa alte aplicații pot fi descărcate de pe Android Market sau alte surse online.

O aplicație constă într-un fișier *.apk* (**a**ndroid **p**ac**k**age) și conține, de obicei, 3 componente [<4>]:

• *Fisierul executabil Dalvik* – reprezintă codul sursă Java compilat pentru a obține executabilul si care e rulat de masina virtuala *Dalvik*

• *Resurse* – orice nu este cod sursă este o resursă. Aplicația poate să conțină imagini, fișiere audio/video și numeroase fișiere XML care descriu layout-ul, etc.

• *Librării native* – Opțional, aplicația poate contine librării native, C/C++, care sunt incluse în fișierul .apk

După cum am menționat mai sus, aplicaţiile Android sunt scrise în limbajul de programare *Java*. Codul compilat împreună cu resursele statice (imagini, texte etc.) sunt stocate cu ajutorul unui instrument numit appt într-un pachet *Android* cu extensia *.apk*.(**a**ndroid **p**ac**k**age) *Android* este un sistem multi utilizator în care fiecare aplicație e vazută ca un utilizator (fiecărei aplicații fiindu-i atribuit un ID de utilizator unic de către sistem), astfel fiecare aplicație instalată ruleaza într-un proces *Linux* separat. Fiecare proces are propria rulează o instant a maşinii virtuale *Dalvik*, astfel că aplicaţiile *Android* rulează izolat una faţă de cealaltă. Un alt avantaj al aplicaţiilor *Android* este acela că o aplicaţie poate folosi elemente ce aparțin altor aplicaţii.

Aplicaţiile *Android* sunt formate din componente slab cuplate, legate printr-un fisier *manifest*.xml în care sunt descrise fiecare componentă şi modul în care acestea interacţionează[6].

Elemente componente

Există şase componente de bază care care sunt folosite pentru construirea unei aplicaţii, componente ce vor fi discutate individual în paragrafele urmatoare.

Activity

Componentele de tip *Activity* sunt componentele responsabile cu partea de prezentare a aplicațiilor *Android*. Aceste tip de componentă este responsabilă de furnizarea unei interfețe grafice utilizatorului precum și de captarea, respectiv procesarea comenzilor provenite de la utilizator prin intermediul interfeței grafice.

O aplicație poate avea în componența ei una sau mai multe componente de tip Activity în funcție de design, una distre acestea fiind marcată ca și "Activitatea" principala a aplicației, aceasta fiind primul obiect de tip *Activity* care va fi lansat în execuție după pornirea aplicației.

Fiecare obiect de tip *Activity* are un ciclu de viața compus din mai multe stări. Datele privind stările diferitelor componente sunt reținute într-o structură de date denumită *stivă de activități* (back-stack) și este gestionată de sistem. Această structură de date este o structura de tip FIFO, în capul stivei situându-se obiectul de tip *Activity* activ la un moment dat. Când utilizatorul interacționează cu un alt obiect de tip *Activity*,acesta ajunge în capul stivei, beneființd de resursele sistemului iar obiectul precedent trece la un nivel inferior pe stivă, devenind inactiv și neconsumând resurse. Când utilizatorul dorește să revină la componenta *Activity* precedentă, componenta pe care era centrat utilizatorul va fi scoasă din stiva și distrusă iar componenta precedentă va reveni în capul stivei.

Un obiect de tip Activity se poate afla în una din cele 3 stări:

- starea activa. Componenta de tip *Activity* se află în prim plan (se situeaza în capul stivei de activităţi) utilizatorul interacționand direct cu aceasta.
- stare de așteptare. Componentele aflate in aceasta stare nu interactioneaza cu utilizatorul insa sunt vizibile acestuia, fiind în continuare atașată manager-ului de ferestre. Toate datele legate de starea *acestor obiecte* sunt păstrate în memorie. Însă în cazul în care sistemul are nevoie de resurse acestea vor fi distruse pentru a elibera resurse.
- oprită. Componentele aflate în această stare nu interacționeaza cu utilizatorul și nici nu sunt vizibile acestuia. Toate datele legate de starea *activității* sunt păstrate în memorie. Însă în cazul în care sistemul are nevoie de mai multe resurse activitățile din această stare pot fi distruse pentru a elibera resurse.

Se pot specifica operațiile care să se execute la tranziția dintre diversele stări prin implementarea unor metode de callback furnizate de clasa Activity și care sunt apelate de sistem în momentul tranziției [6]

Elemente de interfata grafică

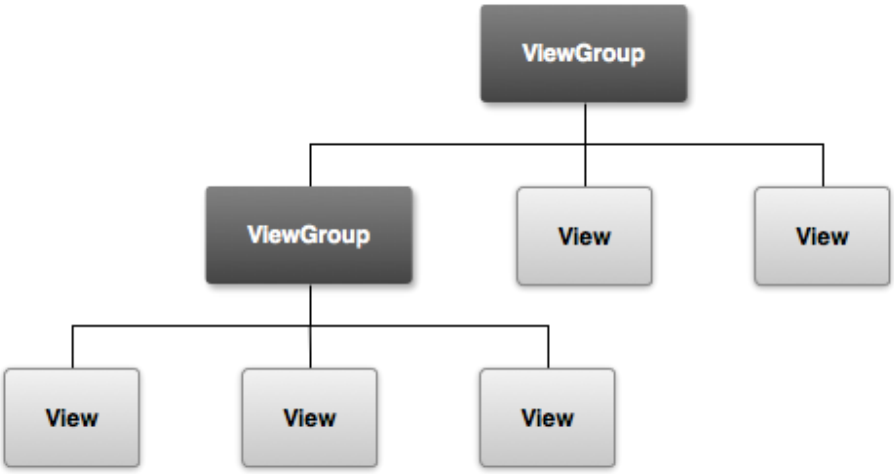
View si ViewGroup

Interfețele grafice sunt alcătuite dintr-o serie de obiecte de tip *View* si *Viewgroup* organizate sub forma unei structuri ierarhice conform figurii 4.4.

Un obiect de tip *View* este un obiect care are o reprezentare grafică și cu care utilizatorul poate interacționa

Un obiect de tip *ViewGroup* este un obiect ce grupeaza mai multe elemente de tip View si le poate afișa simultan.

Android pune la dispoziție o gamă variată de subclase ce extind *View* și *ViewGroup* și care pot fi folosite în construirea interfețelor grafice. De asemenea dezvoltatorii au posibilitatea de a își creea propriile elemente grafice personalizate extinzand clasele *View* respectiv *ViewGroup*.



Figură 4.3 Structura ierarhică a interfețelor grafica în Android (preluată de pe http://developer.android.com)

Există un obiect central de tip *ViewGroup* care ține referințe către restul elementelor de tip *View* sau *ViewGroup*.

Există un element de tip *ViewGroup* cu rol de părinte iar fii acestuia pot fi obiecte de tip View sau chiar *ViewGroup*, care la rândul lor pot avea fii de tip *View* sau *ViewGroup*. Dimensiunea arborelui poate crește în funcție de cat de complex este proiectată interfața grafică, însa este recomandat să se păstreze cât mai simplă posibil deoarece mărimea arborelui afectează performanța [5].

Interfața grafică a unei aplicații Android poate fi realizată fie programatic din cod fie definită în cadrul unui fișier XML care este încărcat de către o componentă de tip Activity

Action Bar

Action Bar sau bara de acțiuni este un element vizual introdus în *Android* versiunea 3.0 (API level 11) cu scopul de a îmbunătăți experiența utilizatorului. Acest element ia forma unei bare orizontale a cărui conținut se schimbă în funcție de poziția utilizatorului în cadrul aplicației.

Printre facilitățile oferite de *Action Bar* se numără

- informarea vizuală a utilizatorului în legatură cu poziția acestuia în cadrul aplicației la un moment dat
- pune la dispoziția utilizatorului o serie de opțiuni în funcție de poziția acestuia de în aplicațe, opțiuni care în versiuni de Android anterioare 3.0, ar fi localizate într-un meniu de opțiuni.
- Pune la dispoziția utilizatorului diverse modalități de navigare prin conținutul aplicației.

Componenta de tip *ActionBar* furnizează un API prin intermediul căruia pot fi controla diverse aspecte ale acesteia în cadrul aplicației din care face parte .

App widget

O componente de tip *App Widget* poate fi privită ca o aplicație în miniatură ce oferă posibilitatea utilizatorului de a vizualiza sau interacționa cu anumite elemente ce țin de conținutul aplicație. O astfel de componentă poate fi folosită de componente de tip *App Widget Ho*st printre care se numără și *Home Screen-ul de Android*

Fragment

În scopul creării unor interfețe grafice mai dinamica și flexibile, odata cu Android 3.0 (nivel API 11) a fost introdusă o nouă componentă denumită Fragment.
Un obiect de tip Fragment poate fi privit ca o copomentă ce modelează o anumită parte din comportamentul aplicației la nivelul interfeței utilizator oferite de un obiect al clasei Activity, fiecare Fragment având definit propriul comportament față de interacțiunea cu utilizatorul.

Un obiect de tip *Activity* poate avea în componența interfeței sale grafice mai multe obiecte de tip Fragment iar un obiect de tip Fragment poate fi proiectat astfel încat să poată fi refolosit ca și parte componenta a interfeței grafice în mai multe obiecte de tip Activity.

Un obiect de tip *Fragment* are propriul ciclul de viața (a se consulta figura 4.6), similar cu cel al unui obiect de tip Activity însă acesta e strans legat de ciclul de viața a obiectului de tip *Activity* în care este încorporat, astfel dacă acesta e în starea de așteptare atunci și toate componentele sale de tip *Fragment* vor fi în aceeași stare, dacă aceasta este distrusă atunci și toate componentele sale de tip *Fragment* vor fi distruse.

Orice operații efectuare asupra obiectelor de tip *Fragment* (adaugare, ștergere,modificare)sunt văzute ca niște tranzacții denumite *Fragment Transactions* iar fiecare tranzacție poate fi trecută în stiva de operații ale *Activității* (back-stack) ceea le permite utilizatorilor să acceseze diferite stări anterioare ale ciclului de viața ale *Fragmentelor*.

Un *Fragment* poate avea propria interfață grafică definită într-un fisier *.xml*. Interfața grafică a fragmentului va fi înserată în ierarhia de *View*-uri ale activității din care face parte.

Un *Fragment* poate fi adaugat la interfata grafica a unui obiect de tip *Activity* fie folosind tag-ul <*fragment*> în cadrul fișierului layout.xml al acestuia fie adaugandu-l din cod la un *ViewGroup*.

O componentă de tip *Fragment* este foarte similară cu una de tip *Activity* ca și structură,în cadrul lor regăsind aceleași metode *onCreate,onStart*().*onPause*(),*onStop*(), însă au și câteva metode specifice.

Fragmentele au fost concepute astfel încă să ofere suport în realizarea unor interfețe grafice dinamice și flexibile care pot fi modificate în timpul rulării, fără a necesita multe modificări la nivelul ierarhiilor de vederi ale activităților

Gestionarea fragmentelor se face cu ajutorul unei entități denumire *FragmenteManager* care permite urmatoarele operații:

- Găsirea anumitor fragmente pe baza de ID
- Găsirea anumitor fragmente pe bază de tag
- Executarea operațiilor de tip fragmente Tranzaction

Un avantaj pe care îl au fragmentele este cel că pot fi cu usurință adăugate, eliminate, modificate ca răspuns la acțiunile utilizatorului.

Aceste acțiuni se pot realiza sub forma unor tranzacții prin folosirea api-ului de fragment transaction

Tranzactiile de tip *fragmente tranzaction* se pot folosi prin intermediul obiectului de tip *FragmentManager*

Servicii

Un serviciu (o componentă de tip *Service*) este o componentă lipsită de interfață grafică și care execută operații pe fundal și a cărui ciclu de viață este independent de cel al altor componente.

O componentă de tip *Service* poate fi pornit de către alte component și odata pornit, serviciul respectiv își execută independent sarcinile pe care le are de făcut, chiar daca componenta care l-a pornit inițial este distrusă.

Un serviciu odată pornit rulează în mod implicit în cadrul firului de execuție principal al aplicației, executând operațiile pe care le are de executat, dupa aceea distrugându-se. *Serviciile* au un ciclu de viață, simplificat ce este controlat în cea mai mare parte de către dezvoltator și nu de către sistem.

Content provider (furnizor de conținut)

O component de tip *Content Provider* este un obiect din cadrul unei aplicații *Android* care face ca anumite date, din cadrul aplicației,să fie disponibile altor aplicații.

Datele partajate pot fi: fișiere audio, video,imagini, alte tipuri de fișiere precum și date stocate într-o baza de date *SQLite*.

Pentru a crea o astfel de componentă, trebuie extinsă clasa *ContentProvider*. Această clasă oferă un set de metode pentru expunerea și prelucrarea datelor, similare cu cele utilizate pentru baze de date. O parte din elementele native ale sistemului *Android* conțin componente de tip *ContentProvider* făcând disponibile altor aplicații date precum, datele gestionare de manager-ul de contacte a dispozitivului și altele.

Pentru a putea accesa datele, oferite de un anumit furnizor de conținut, trebuie folosit un obiect de tip *ContentResolver*. Un *ContentResolver* este un obiect client ce furnizează operații CRUD (create, read, update și delete) asupra datelor oferite de către un *ContentProvider* dintr-o anumita aplicație. Toate elementele ce țin de comunicarea dintre procese (procesul aplicației ce conține *ContentProvider*-ul și procesul aplicației ce solicită datele) sunt gestionate automat de către sistem.

Receptori de anunțuri (Broadcast Receivers)

Un obiect de tip *Broadcast Receiver* este o componentă care răspunde la mesaje de tip broadcast.

Mesajele de tip broadcast pot fi transmise de către sistem, pentru a notifica aplicațiile despre anumite modificari ale parametrilor sistemului (precum nivelul memoriei, bateria disponibilă), sau de către aplicații pentru a notifica alte aplicații despre anumite evenimente

precum terminarea descărcării unui fișier. În funcție de conținutul mesajelor de *broadcast* anumite aplicații pot reacționa la evenimentele anunțate.

Pentru a putea implementa o astfel de componenta trebuie extinsă clasa *BroadcastReceiver* iar mesajele de broadcast sunt de obicei obiecte de tip *Inten (intenții).* Acest tip de componentă nu posedă o interfață grafică proprie, însa pot comunica cu utilizatorul prin trimiterea de notificari către bara de stare (*satus bar*).

Acest sistem de mesaje de broadcast și receptori de broadcast poate fi vazut ca un mecanism de tip publish-subscribe sau o implementare a șablonului Observer.

Intent (Intenții)

O "intenție", sau un obiect de tip *Intent*, conține informații despre operațiile pe care ar trebui să le facă o anumită componentă destinație pe un anumit set de date. Cu ajutorul obiectelor de tip *Intent* este posibilă comunicarea, în timpul rulării, cu diverse componente aflate fie în interiorul aceleiași aplicații fie sunt localizate în alte aplicații. Printre componentele ce pot fi activate prin intermediul obiectelor de tip *Intent* se numară obiecte de tip *Activity, Service* și *BroadcastReceiver*.

Există 2 tipuri principale de intenții:*intenții implicite* și *intenții explicite*.

In cadrul *intențiilor explicite* este specificat obiectul distinație care să realizeze prelucrările cerute (fie ca e de tip *Activity*, *Service* sau *BroadcastReceiver*).

In cazul *intențiilor explicite*, rămane pe seama sistemului sarcina de a selecta aplicația corespunzatoare care să raspundă la solicitatea din cadrul *Intent*-ului.Elementele din *Intent* care sunt analizate sunt cele din categoriile action, data, și category. In acest caz sistemul ține cont de *filtrele de intenție* declarate în fișierul *manifest* a fiecarei aplicații. Un *filtru de intenții (intent filter)*specifica tipul de *intenții* pe care le poate prelucra o anumita aplicație.

Fisierul manifest Android

Fiecare proiect *Android* include un fişier xml denumit *AndroidManifest*.xml, stocat în directorul rădăcină al proiectului. In acest fişier sunt descrise componentele folosite în aplicaţie şi alte informaţii referitoare la permisiuni sau librării ce trebuie legate de aplicaţia curentă. Se poate defini versiunea de *Android* folosită de aplicaţie, resursele și drepturile de care are nevoie aplicația pentru a putea rula. Structura acestui fişier impune prezenţa unui tag rădacină <*manifest*> urmat de tag-ul <*application*> în interiorul căruia vor fi descrise componentele care alcătuiesc aplicaţia.

Procese si fire de executie in Android

Atunci când e necesară pornirea unei aplicații, sau a unei anumite componente dintr-o aplicație *Android* pornește automat un nou proces *Linux* cu un singur fir de execuție pentru componenta respectivă. Acest proces este cunoscut și sub numele procesul principal (procesul main) al aplicației. În cazul în care a fost pornit un proces pentru o componentă aparținând unei aplicații anume, atunci orice cerere ulterioară de a rula alte componente apartinând aceleași aplicații vor fi executate în același process nefiind creat unul nou.

Procese

Android încearcă să mențină procesele în stare de funcționare pe cât de mult posibil, acestea fiind oprite doar în cazul în care sistemul are nevoie să recupereze resurse pentru a executa operațiile importante, însă în cazul în care sistemul are nevoie de resurse, vor fi oprite procesele mai puțin importante în favoarea celor importante
In funcție de importanța acestora exista 5 tipuri de procese

- Procese de tip *foreground*. Acest nivel de importață este asociat proceselor în care rulează componentele ce se află în prim plan și cu care utilizatorul interacționează direct la un moment dat.Exista un numar mic de astfel de procese la un moment dat și sunt oprite doar în ultimă instanță atunci când starea resurselor e critică.
- Procese *vizibile*.Acest tip de procese găzduiesc componente care nu sunt în prim plan dar care însă sunt vizibile pentru utilizator.
- Procese de tip *service*. Acest tip de procese conțin componente de tip Service care ruleaza în fundal efectuând diverse operații
- Procese de fundal. Acest tip de procese conțin componente care nu mai sunt vizibile utilizatorului. La un moment dat pot exista multe astfel de procese, evidența acestora fiind păstrată într-o lista ordonată în funcție de cât de recent au fost accesate componentele acestora. Astfel că un proces ce conține un obiect de tip *Activity* care a fost folosit recent va fi oprit ultimul process de tip background oprit în cazul în care sistemul are nevoie să recupereze resurse.E indicat să fie implementate corect metodele de *callback* ce țin de ciclul de viața din cadrul fiecărei componente, pentru a preveni o eventuală pierdere a datelor în cazul în care procesul acestora este oprit.
- Procese goale. Acest tip de procese nu conțin nicio componenta activă, având doar rol de caching. Rolul lor fiind cel de a îmbunătăți timpul de lansare a unei noi component fară a carea un nou proces.

Importanța unui proces este de dată de gradul de importanță a componentelor ce rulează în acesta.

Importanța proceselor mai este data și de interdependența dintre ele. Astfel că dacă, de un anumit proces depinde un alt proces cu un grad rididat de importanță atunci procesului respectiv i se va asocia cel puțin același grad de importanță.

Fire de executie

Implicit fiecărei aplicații îi este asociat un singur fir de executie cunoscut ca și firul principal de execuție (main thread). Toate instanțierile de obiecte respectiv prelucrări de date sunt executate pe acest fir de execuție. Tot în cadrul acestui fir de execuție au lor operațiile ce vizează interacțiunile dintre elementele de *Android UI toolkit* (components ce apartin pachetelor android.widget si android.view și aplicația propriu-zisă, de aceea acest fir de execuție mai este supranumit și *UI thread*.

Există o problemă în acest cazul operațiilor solicitante deoarece ar afecta performanța aplicației și ar putea duce la blocarea ei.

In cazul operațiilor mai complicate se poate implementa un obiect de tip *Handler* care apoi să fie asociat unui nou fir de execuție sau se pot folosi obiecte de tip *AsyncTask*.

Asynctask

Obiecte de tip *AsyncTask* pot fi folosite pentru a efectua prelucrări asincrone asupra obiectelor apartinând interfeței grafice. Operațiile sunt executate în cadrul unor fire de execuție

care rulează pe fundal iar rezultatul operațiilor este publicat în elementele corespunzatoare situate în *UI Thread.*

Pentru a folosi un astfel de obiect trebuie extinsă clasa *AsyncTask* și implementată metoda *doInBackground*(). Operațiile specificate în aceasta metodă vor fi executate în cadrul unor fire de execuție ce rulează în fundal.

Loader(incarcator)

Un obiect de tip *Loader* este un obiect ce facilitează încărcarea asincrona a datelor în cadrul unui obiect de tip *Activity* sau *Fragment.*

Acest tip de obiect este disponibil odata cu versiune 3.0 de *Android* și este special conceptul pentru obiecte de tip *Activity* și *Fragment* pentru a putea încarca date din diverse surse de date în mod asincron fără ca această operație să fie executate pe UI thread și afecteze performanța aplicației.

Un obiect de tip *Loader* odată creat monitorizează sursa de date actualizând automat interfața grafică în cazul modificării datelor.

4.1.2 Google Maps

Google Maps este un serviciu web de cartografiere oferit de *Google Inc*, ușor de folosit și care oferă multe facilități printre care se numără :

- Diverse modalități de vizualizare a zonelor de interes:Imagini satelitare, Street View
- Oferă informații legate de diverse puncte de interes de natura economică, turistică, etc. specifice unei anumite zone geografice.
- Posibilitatea de a genera un traseu între 2 puncte de interes pentru utilizator și de a oferi indicații acestuia de cum să-l parcurgă.
- Interfață grafică intutivă.

4.2 Tool-uri folosite

4.2.1 Eclipse IDE

Eclipse este un mediu de dezvoltare software scris în *Java* care poate fi folosit pentru dezvoltarea aplicațiilor software în diverse limbaje de programare, printre care se numără și *Java*.

Eclipse este structurat sub forma unei ierarhii de plugin-uri ce asigură diferite funcționalități. Aceste funcționalități fiind asigurate fie de către plugin-uri individuale sau de către combinații de plugin-uri, fiecare plugin fiind activat în mod dinamic doar în momentul în care funcționalitatea respectivă este apelată.

Infrastructura de bază din *Eclipse* este asigurată de catre plugin-urile furnizate de *Eclipse RCP*(a se consulta tabelul 4.2).

Tabel 4.1 Structura pluging-urile Eclipse RPC

Platform Runtime	Componentă, implementată folosind framework-ul OSGi, responsabilă de descoperirea dinamică a plugin-urilor și menținerea unei evidențe a acestora în registul platformei. Asigură că plugin-urile sunt încărcate și lansate în execuție atunci când e nevoie de ele.
Resource	Componentă ce definește API-urile necesare creării repsectiv gestionării

management (workspace)	resurselor (precum proiecte, fișiere și directoare) care sunt produse de către tool-uri și salvate în sistemul de fișiere.
Eclipse UI Workbench	Asigură interfața grafică prin intermediul căreia utilizatorul poate accesa functionalitățile oferile de tool-urile instalate, punând la dispoziția acestuia diverse modalități de vizualizare, meniuri și alte elemente prin intermediul cărora utilizatorul poate manipula diverse resurse.
Help system	Componentă ce permite vizualizarea documentațiilor oferite de plugin-uri
Team support	Componentă ce facilitează lucrul în echipă în cadrul proiectelor, oferind functionalități precum gestionarea și versionarea resurselor
Debug support	Componentă ce pune la dispoziție un model de debugger independent de limbaj precum și diverse clase cu ajutorul cărora se pot proiecta tool-uri de debugging.
Other utilities	Alte tipuri de plugin-uri care opereaza asupra resurselor, precum execuția operațiilor de build în funcție de specificațiile menționate într-un fișier de configurare XML.

De asemenea este posibilă extinderea functionalităților oferite de *Eclipse* prin adaugarea de noi pluginuri.

In prezent există mai multe versiuni de Eclipse iar pentru realizarea proiectului a fost folosit Eclipse 3.6 Helios deoarece este versiunea minimă compatibilă cu plugin-ul de *ADT* (detaliat in secțiunea 4.1.3.3)

4.2.2 Android SDK

Android SDK reprezintă pachetul de componente software și unelte cu ajutorul cărora se pot dezvolta aplicații pentru dispozitive mobile ce rulează pe Android.
SDK-ul de Android este compus dintr-o colecție modulară de pachete care pot fi descărcate separat prin intermediul componentei *SDK manager*.

În tabelul 4.2 sunt menționate pachetele importante.

Tabel 4.2 Componente din Android SDK

Pachet	Descriere	Localizare în cadrul SDK-ului
SDK Tools	Cuprinde unelte necesare depanării și testării aplicațiilor alături de alte utilitare necesare dezvoltării aplicațiilor.	`<sdk>/tools/`
SDK Platform-tools	Conține unelte specifice anumitor platforme de Android necesare proiectării respectiv depanării aplicațiilor specifice platformei respective.	`<sdk>/platform-tools/`
Documentation	O copie offline actualizată a documentației API-urilor diverselor platforme de Android.	`<sdk>/docs/`

SDK Platform	Există o platfoma SDK pentru fiecare versiune de Android lansată, care poate fi alesa ca și platforma țintă pentru aplicații	`<sdk>/platforms/<android-version>/`
System Images	Fiecare platformă oferă una sau mai multe imagini sistem pentru (ARM sau x86). Aceste imagini sistem sunt necesare emulatorului de Android pentru a putea rula.	`<sdk>/platforms/<android-version>/`
Sources for Android SDK	O copie a codului sursă pentru componentele diverselor platforme de Android.	`<sdk>/sources/`
Samples for SDK	Exemple care ilustrează funcționalitățile diverselor API-uri de Android	`<sdk>/platforms/<android-version>/samples/`
Google APIs	Pachet special ce permite unei aplicații să apeleze servicile oferite de Google prin intermediul Google API. Oferă și o imagine system cu ajutorul căreia pot fi testate ce folosesc Google API	`<sdk>/add-ons/`
Android Support	Librărie suport care poate fi inclusă în aplicații pentru a putea oferi acestora functionalități disponibile unui nivel API superior în cazul în care acestea trebuie să ruleze pe o versiune de Android.	`<sdk>/extras/android/support/`
Google Play Billing	Librărie statică care permite integrarea unui serviciu de taxare în aplicații cu google Play	`<sdk>/extras/google/`
Google Play Licensing	Librărie ce oferă functionalități de verificare a licențelor pentru aplicații atunci când acestea sunt distribuite prin intermediul serviciului Google Play	`<sdk>/extras/google/`

4.2.3 ADT (Android Development Tool)

SDK-ul de *Android* poate fi integrat cu mediul de dezvoltare *Eclipse* prin intermediul unui plugin denumit *Android Development Tool*.

ADT este un tip special de plugin conceput pentru *Eclipse* menit sa faciliteze devolzarea aplicațiilor destinate pentru *Android*. Acesta extinde capabilitățile IDE-ului oferind posibilități

precum crearea de proiecte, proiectarea de interfețe grafice, compilare și generare de APK-uri , instalarea aplicațiilor pe dispozitive sau pe emulator.

Acest tool oferă o serie de perspective prin intermediul cărora devoltatorii pot, în mod vizual accesa diverse tool-uri oferite de *SDK* , monitoriza resurse, seta și controla instanțe ale emulatorului, depana și testa aplicații.

4.2.4 Google APIs Add-On

Google API add-on este o extensie a SDK-ului de Android ce oferă dezvoltatorilor posibilitatea de a integra în aplicațiile lor servicii oferite de *Google*.

In cadrul acestui add-on este inclusă o librărie pentru *GoogleMaps* și alte componente necesare accesării serviciilor oferite de *Google*.

Acest Add-on cuprinde și o imagine sistem cu librăriile necesare rulării,testării și depanării aplicațiilor cu ajutorul Emulatorului.

Pachetul *Google APIs* add-on cuprinde:

- Librăria externă pentru *GoogleMaps*
- Libraria USB Open Accessory (compatibila doar cu API Levels 10 și 12+)
- Imagine sistem Android (cu diverse componente sistem integrate)
- Aplicație demo în care e folosită librăria GoogleMaps denumită MapsDemo
- Documentația aferentă librăriei pentru GoogleMaps.

4.3 Cerințele aplicației

4.3.1 Caracteristici funcționale

Conform principiilor menționate în secțiunea 3.1 reiese că , pentru ca aplicația să satisfacă cerințele unui tool de time management ar trebui să ofere o serie de funcționalitâti specifice care să se modeleze pe domeniul respectiv.Aceste funcționalități vor fi detaliate în cele ce urmează

Pentru început sa luam în considerare următorul scenariu:

Să presupunem că un utilizator are o lista de sarcini sau activități pe care trebuie sa le îndeplinescă într-un timp dat.Fiecare activitate poate avea o anumita semnificație pentru un anumit utilizator în funcție de obiectivul pe care acesta îl poate atinge utilizatorul respectiv prin săvârșirea activității respective, contextul asociat activității respective[CF-1], precum și alte criterii ce țin de preferințele utilizatorului.

Activitățile pot fi de sine stătătoare sau pot fi grupate în funcție de diferite criterii ale utilizatorului[CF-2]. Unele activități au un grand de importanță mai mare față de altele din perspectiva utilizatorului[CF-3] astfel acesta le va acorda atentie și timp și apoi se va ocupa de cele mai puțin importante.

Pentru a își planifica eficient timpul, utilizatorii încearcă să țina o evidența a tuturor activităților[CF- 4], evenimentelor, și uneori apeleaza la o reprezentare vizuala a acestora, folosind diverse tehnici și intrumente pentru le fi mai ușor de urmărit fluxul de activități[CF-5].

De asemenea utilizatorul are nevoie elemente care să-l anunțe când trebuie să întreprindă o anumită activitate.[CF-6]

În tabelul 4.3 sunt menționate principalele caracteristici funcționale ce reies din scenariul descris mai sus.

Table 4.3 Cerințe funcționale

Identificator	Descriere
CF-1	Specificarea contextului activității (timp, loc, durata, descriere)
CF-2	Gruparea activităților
CF-3	Atribuirea unui grand de importanță activităților
CF-4	Stocarea activităților
CF-5	Reprezentare grafică a fluxului de activități
CF-6	Mecanism de notificare a activităților

4.3.2 Caracteristici non-funcționale

Pentru ca aplicația să poată nofica utilizatorul despre diversele evenimente aceasta ar trebui sincronizată cu ceasul sistemului [CNF-1].

Aplicația ar trebui sa fie împărțită pe mai multe module pentru a permite separarea diverselor fucntionalitati si pentru a permite scalabilitatea si extinderea aplicatiei [CNF-2].

Un alt aspect de care ar trebui să se țină cont ar fi folosirea mai multor fire de execuție pentru execuția operațiilor, în caz contrar totul s-ar executa pe firul de execuție principal al aplicației așa cum a fost menționat în secțiunea 4.1.1.3, lucru ce ar afecta performanța aplicației per ansamblu precum și exeperiența utilizatorului[CNF-3].

Tabel 4.4 Caracteristici non-funcționale

identificator	Descriere
CNF-1	Sincrionizarea cu ceasul sistemului
CNF-2	Împărțire pe module
CNF-3	Utilizarea mai multor fire de execuție pentru efectuarea operațiilor

4.3.3 Specificații Dispozitiv

Dispozitivul mobil, pe care ar urma să ruleze aplicația, ar trebui sa respecte o serie de cerințe hardware.

Un astfel de dispozitiv ar trebui să dispună de un modul de *GPS* pentru a putea obtine pozitia utilizatorului la un moment dat[CD-1].

Pentru a putea stoca activitatile utilizatorului, dispozitivul ar trebui să dispuna de o baza de date integrată sau să permita operații de CRUD asupra unor fisiere la nivel local[CD-2].

Sistemul de operare care va rula pe dispozitiv va fi *Android* versiunea 4.03. Asa cum a fost menționat în sectiunea 4.1.1.2 *Android* dispune de *SQLite* care ar satisface cerința de stocare a datelor menționată mai sus însă introduce o costrângere suplimentară și anume dispozitivul respectiv ar trebui sa fie un smartphone bazat pe arhitectura ARM sau x86 capabil să ruleze *Android 4.03 Ice Cream Sandwich* [CD-3].

Tabel 4.5 Speficații despozitiv

Identificator	Descriere
CD-1	GPS
CD-2	Modalitate de stocare a informației la nivel local (bază de date sau fișiere)
CD-3	Dispozitiv bazat pe arhitectura ARM sau x86 capabil să ruleze Android 4.03 Ice Cream Sandwitch

4.3.4 Factorul uman

Pentru ca aplicația să fie utilă utilizatorului, aceasta ar trebui să fie ușor de utilizat de către acesta, nefiind nevoie de un număr prea mare de pași pentru accesarea diverselor functionalitati [UF-1], iar informația să fie prezentată utilizatorului într-o manieră ușor de înțeles.

Interfața unei astfel de aplicații ar trebui să atragă utilizatorul, să fie intuitivă și usor de folosit [UF-2].

Tabel 4.6 Caracteristici non-functionale ce tin de factorul uman

identificator	Descriere
REQ-UF-1	Numar mic de pasi pentru accesarea functionalitatilor
REQ-UF-2	Interfata placuta si intuitiva

4.4 Cazuri de utilizare

Cazurile de utilizare vor fi redate prin intermediul diagramelor use-case specifice standardului UML 1.x

Din cele menționate în sectiunea 4.2.1 reiese că aplicația are trebui să îi permită utilizatorului să efectueze operații de tip CRUD asupra activităților și de asemenea să îi pună la dispoziție o serie de mijloace prin care:

- Să prioritizeze activități
- Să grupeze activități
- Să îi ofere diferite mijloace de vizualizare care să îl ajute în luarea anumitor decizii privind activitățile cărora ar trebui să le ofere atenție.

Cazurile de utilizare identificate pentru aplicația din discuție sunt urmatoarele:

1. Adaugarea unei noi acitivăți la lista de activități din memorie
2. Editarea unei activități deja existente
3. Ștergerea unei activități deja existente
4. Vizualizarea activităților în funcție de zi
5. Vizualizarea zilelor din lună în care sunt planificare activități
6. Vizualizarea matricei Eisenhower pentru activități
7. Vizualizeaza activităților asociate unei anumite etichete
8. Adaugarea unei noi etichete la lista de etichete deja existente în memorie
9. Editarea unei etichete deja existente
10. Ștergerea unei etichete deja existente
11. Adaugarea unui nou obiectiv pe hartă la lista de obiective deja existente în memorie
12. Editarea unui obiectiv deja existent

13. Ștergerea unui obiectiv deja existent
14. Asocierea unui obiectiv de pe hartă cu contextul unei activități planificate de utilizator
15. Vizualizarea hărții de obiective
16. Notificarea utilizatorului în momentul în care acesta se află în apropierea unui obiectiv.

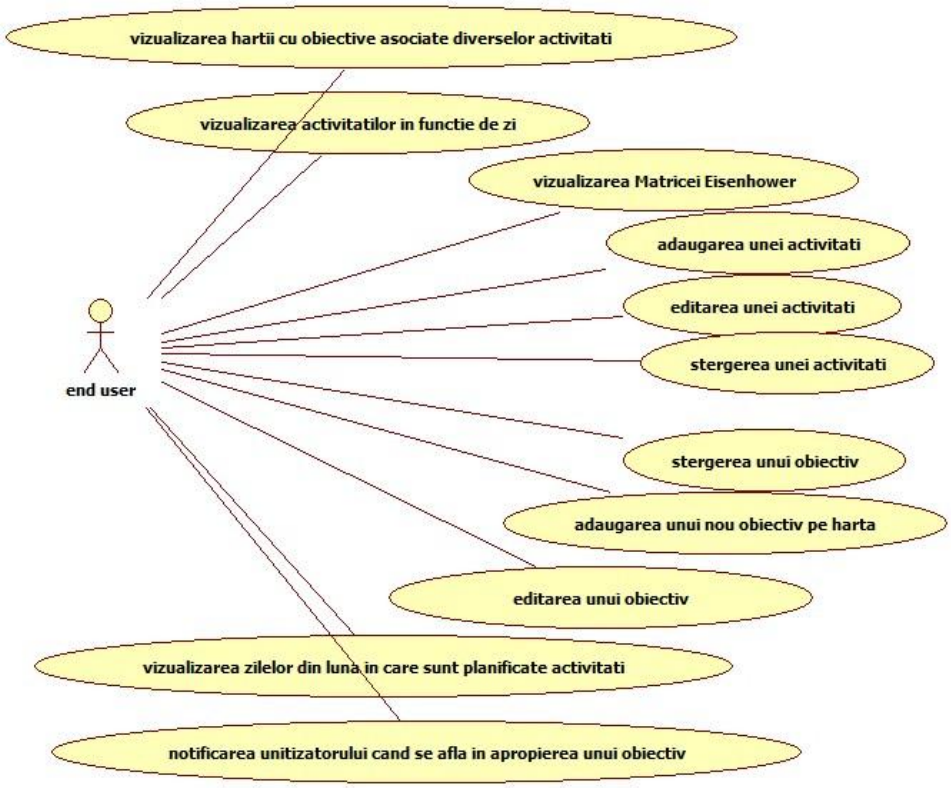


Figura 4.4 Diagrama cazurilor de utilizare

Titlu: Adaugarea unei noi activități (Cazul de utilizare 1)
Descriere:

Permite utilizatorului să adauge o nouă activitate la lista de activități deja existente în memorie.

Actor: utilizatorul final.
Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul trebuie să se afle în unul din urmatoarele moduri de vizualizare:
   - Viziualizare activități în funcție de zi
   - Vizualizate zile din lună
   - Vizualizare activități în funcție de etichetă
   - Modul de vizualizare principal

Pași:

1. Utilizatorul alege optiunea "*add new event*" din meniul de opțiuni
2. Aplicația afișează modul de vizualizare corespunzator adaugării unei noi activități.
3. Utilizatorul specifică informațiile legate de activitate
   - Denumire
   - Alege o etichetă din lista de etichete deja existente. **Include** (vizualizare etichete)

- Alege un obiectiv din lista de obiective deja existente reprezentând contextul de care depinde indeplinirea aplicației. **Include** (vizualizare listei de obiective)
- Atribuire grad de importanță din cele puse la dizpoziție de aplicație . **Include** (vizualizare prioritate)
- Specificare dată.
- Notițe
- Status prin alegerea unei opțiuni din cele puse la dispoziție de aplicație . **Include** (vizualizare status)

și apasă butonul "*save"*

4. Aplicația crează o nouă activitate cu datele definite și o salvează în memorie.

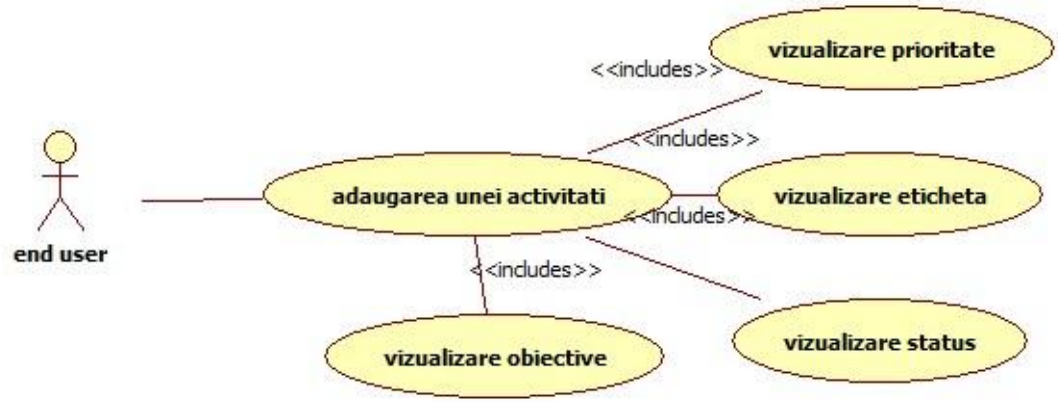


Figura 4.5 Diagrama use case pentru adaugarea unuei noi activitati

Titlu: Vizualizare prioritate (Cazul de utilizare 1.1)
Descriere:

Aplicația pune la dispoziția utilizatorului colecția de grade de prioritate stocate în memorie și care pot fi folosite de către utilizator pentru specificarea importanței unei activități.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul trebuie să se afle în modul de vizualizare corespunzător adăugării unei noi activități.

Pași:

1. Aplicația încarcă din memorie colecția de elemente semnificând gradele de prioritate ce pot fi asociate activităților
2. Aplicația pune la dispoziția utilizatorului datele încărcate de la pasul anterior.

Titlu: Vizualizare etichetă (Cazul de utilizare 1.2)
Descriere:

Aplicația pune la dispoziția utilizatorului colecția de etichete stocate în memorie și care pot fi folosite de către utilizator pentru a eticheta activități.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Uitilizatorul trebuie să se afle în unul din urmatoarele moduri de vizualizare:
   - Adăugarea unui nou evenimente
   - Vizualizarea activităților în funcție de etichetă.

Pași:

1. Aplicația încarcă colecția de etichete stocate în memorie.
2. Aplicația pune la dispoziția utilizatorului datele încărcate de la pasul anterior.

Descriere:

Aplicația pune la dispoziția utilizatorului colecția de etichete stocate în memorie și care pot fi folosite de către utilizator pentru a eticheta activități.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Uitilizatorul trebuie să se afle în unul din urmatoarele moduri de vizualizare:
    - Adăugarea unui nou evenimente
    - Vizualizarea activităților în funcție de etichetă.

Pași:

1. Aplicația încarcă colecția de etichete stocate în memorie.
2. Aplicația pune la dispoziția utilizatorului datele încărcate de la pasul anterior.

Titlu: Vizualizare obiective (Cazul de utilizare 1.3)

Descriere:

Aplicația pune la dispoziția utilizatorului o colecție de elemente semnificând obiectivele ce pot fi asociate contextului activității.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul trebuie să se afle în modul de vizualizare corespunzător adăugarii unei noi activități

Pași:

1. Aplicația încarcă din memorie colecția de elemente semnificând obiectivele de pe hartă ce pot fi asociate contextului activității definite de utilizator.
2. Aplicația pune la dispoziția utilizatorului datele încărcate de la pasul anterior.

Titlu: Vizualizare listei de activități pentru o anumită zi (Cazul de utilizare 4)

Descriere:

Vizualizarea de către utilizator a tuturor activităților planificate și înregistrate de aplicație, care ar trebui să se desfășoare la o anumită dată specificată.

Actor: utilizatorul final.

Precondiții:

1. Aplicația trebuie să fie pornită

Pași:

1. Utilizatorul navighează către modul de vizualizare principal
2. Aplicația încarcă din memorie activitățile planificate pentru data curentă.
3. Aplicația lansează modul de vizualizare principal și afisează o lista cu toate activitățile încărcate la pasul anterior, pentru fiecare eveniment sunt afișate informații precum:
    - Titlu
    - Ora
    - Eticheta
    - Status
    - Gradul de importanță asociat

Extension

a. Utilizatorul dorește să vadă activitățile planificate pentru ziua precedentă.

Precondiții:

1. Aplicația trebuie să fie pornită
2. **2.** Utilizatorul trebuie să se afle în modul de vizualizare a activităților în funcție de zi

Pași:

1a. utilizatorul apasă butonul corespunzător opțiunii de afișare a activităților ce au loc în ziua precedentă datei curente.

2a. Sistemul calculează noua dată

3a. Sistemul încarcă colecția de evenimente din memorie planificate pentru data calculată la pasul anterior și o afișează utilizatorului.

b. Utilizatorul dorește să vadă evenimentele planificate pentru ziua următoare

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul trebuie să se afle în modu de vizualizare a activităților în funcție de zi

Pași:

1b. utilizatorul apasă butonul corespunzător opțiunii de afișare a activităților ce au loc în ziua următoare datei curente.

2b. Sistemul calculeaza noua dată.

3b. Sistemul încarcă colecția de evenimente stocate în memorie și care sunt planificate pentru data calculată la pasul anterior și o afișează utilizatorului.

c. Utilizatorul dorește să vadă evenimentele planificate pentru o anumită zi din lună

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul trebuie să se afle în modul de vizualizare a zilelor din lună în care au fost planificate activități.

Pași:

1c. Utilizatorul dă click pe ziua din lună pentru care dorește să vadă activitățile planificate.

2c. Sistemul calculeaza noua dată.

3c. Sistemul încarcă colecția de evenimente stocate în memorie și care sunt planificate pentru data calculată la pasul anterior și o afișează utilizatorului.

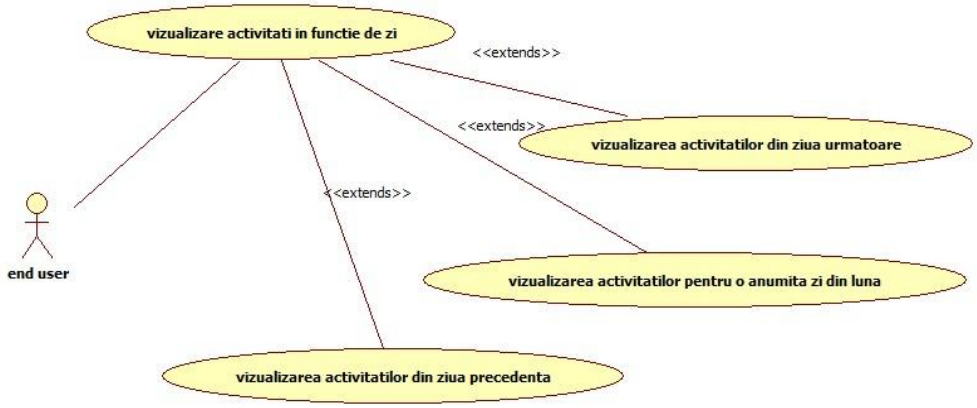


Figura 4.6 Diagrama use-case pentru afisarea evenimentelor in functie de zi

Titlu: Vizualizarea activităților în funcție de etichete (Cazul de utilizare 7)

Descriere:

Vizualizarea de către utilizator a tuturor activităților care au o anumită etichetă

Actor: utilizatorul final.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul se află în modul principal de vizualizare

Pași:

1. Utilizatorul apasă butonul "manage events"
2. Aplicația afișează modul de vizualizare corespunzător
3. Aplicația afișează toate etichetele stocate în memorie. Include (vizualizare etichetă)
4. Utilizatorul alege o etichetă.
5. Aplicația încarcă activitățile din memorie care sunt etichetate cu eticheta de la pasul anterior.
6. Pentru fiecare activitate sunt afișate informații precum
    - Titlu
    - Ora
    - Eticheta
    - Status
    - Gradul de importanță asociat

Titlu: Vizualizare zilelor din luna în care sunt planificate activități (Cazul de utilizare 5)

Descriere:

Vizualizarea de către utilizator a zilelor dintr-o anumită lună, în care sunt planificate activități.

Actor: utilizatorul final.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul se afla in modul principal de vizualizare

Pași:

1. Utilizatorul alege opțiunea *detailed view*
2. Aplicația lansează modul de vizualizare corespunzător, afișând toate zilele din luna curentă și marcând cu o anumită culoare zilele în care sunt planificate activități (culoarea fiind aleasă în funcție de cea mai importantă activitate ce are loc în ziua respectivă)

Extension

a. Utilizatorul dorește să vadă zilele din luna precendentă în care sunt planificate activități.

Precondiții:

1. Aplicația trebuie să fie pronită
2. Utilizatorul se află în modul de vizualize detailed view.

Pași:

1a. Utilizatorul apasă butonul corespunzător afișării zilelor din luna anterioară lunii curente.

2a. Aplicația calculează noua dată calendaristică.

3a. Aplicația încarcă activitățile din baza de date planificate pentru luna calculată la pasul anterior.

4a. Aplicația actualizează interfața grafică afisând zilele din luna precedentă și marcându-le corespunzător pe cele care au activități planificate.

b.Utilizatorul dorește să vadă zilele din luna următoare în care sunt planificate activități.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul se află în modul de vizualize detailed view.

Pași:

1b. Utilizatorul apasă butonul corespunzător afișării zilelor din luna următoare lunii curente.
2b. Aplicația calculează noua dată calendaristică.
3b. Aplicația încarcă activitățile din baza de date planificate pentru luna calculată la pasul anterior.
4b. Aplicația actualizează interfața grafică afisând zilele din luna precedentă și marcându-le corespunzător pe cele care au activități planificate.

Titlu: Vizualizare hărții cu obiective (Cazul de utilizare 15)

Descriere:

Vizualizarea de către utilizator a unei hărți pe care sunt marcate puncte de interes relevante pentru activitățile planificate

Actor: utilizatorul final.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul se afla in modul principal de vizualizare

Pași:

1. Utilizatorul alege opțiunea *go to map.*
2. Aplicația lansează modul de vizualizare corespunzător
3. Aplicația încarcă din memorie punctele de interes asociate activităților
4. Aplicația afisează o hartă pe care sunt evidențiate punctele de interes încărcate la punctul anterior.
5. Utilizatorul dă click pe un punct de interes
6. Aplicația afișează adresa asociata cu punctul respectiv precum și numarul de activități asociate acelui punct.

Titlu: Vizualizare Matricea EisenHower pentru activități (Cazul de utilizare 6)

Descriere:

Permite utilizatorului să vizualizeze matricea Eisenhower pentru diverse activități deja existente

Actor: utilizatorul final.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Utilizatorul trebuie să fie situat în unul din următoarele moduri de vizualizare:
   - Vizualizare evenimente in functie de zi
   - Vizualizare zile din lună
   - Vizualizare evenimente în funcție de etichetă

Pași:

1. Utilizatorul alege opțiunea “view Eisenhower” din meniul de opțiuni
2. Aplicația afișează modul de vizualizare corespunzător populându-l cu activitățile din memorie
3. Utilizatorul specifică gradul de prioritate dorit
4. Aplicația afișează sub forma unei liste toate activitățile ce au asociatate gradului de prioritate respectiv

Extension

a. Utilizatorul dorește să vizualizeze matricea Eisenhower pentru activitășile planificate într-o anumită zi.

Preconditii:

1. Utilizatorul se află în modul de vizualizare evenimente în funcție de zi

Pași:

2aa. Aplicația încarcă activitățile din memorie planificate pentru ziua respectivă.
2ab. Aplicația afișează modul de vizualizare corespunzător populandu-l cu activitățile de la pasul anterior.

b. Utilizatorul dorește să vizualizeze matricea Eisenhower pentru activitățile planificate într-o anumită lună.

Precondiții:

1. Utilizatorul se află în modul de vizualizare a zilelor din lună în care au fost planificate evenimente.

Pași:

2ba. Aplicația încarcă activitățile din memorie planificate pentru luna respectivă.
2bb. Aplicația afișează modul de vizualizare corespunzator populându-l cu activitățile de la pasul anterior.

c. Utilizatorul dorește să vizualizeze matricea Eisenhower pentru evenimentele asociate unei anumite etichete.

Precondiții:

1. Utilizatorul se află în modul de vizualizare a activităților în funcție de etichetă/

Pași:

2ca. Aplicația încarcă activitățile din memorie ce prezintă o anumită etichetă.
2cb. Aplicația afișează modul de vizualizare corespunzator populându-l cu activitățile de la pasul anterior.

Titlu: Noficarea utilizatorului când aceste se află în apropierea unui obiectiv. (Cazul de utilizare 16)

Descriere:

Utilizatorul primește o notificare în momentul în care acesta se află în apropierea unui punct de itneres asociat contextului unei activități planificate în ziua respectivă.

Actor: utilizatorul final.

Precondiții:

1. Aplicația trebuie să fie pornită
2. Trebuie să existe puncte de interes înregistrate în baza de date și asociate activităților planificate în ziua respectivă

Pași:

1. Aplicația calculează poziția utilizatorului
2. În cazul în care acesta se află în apropierea unui punct de interes transmite o notificare

În noțiunile prezentate în acest capitol au fost prezentată platforma care va fi folosită pentru proiectarea aplicației din discuție și anume Android precum si principalele cazuri de utilizarea la care ar trebui să răspundă aplicația urmând ca în capitolul următor să fie detaliat modul de

implentare a aplicației și componentele de Android la care s-a apelat pentru realizarea diverselor functionalități

5 Proiectare de detaliu și implementare

În acest capitol vor fi tratate aspecte ce țin de implementarea aplicației oferindu-se detalii legate de structura aplicație, principalele module precum și componentele și serviciile folosite pentru realizarea functionalităților descrise în capitolul anterior.

5.1 Arhitectură conceptuală

Aplicația de față este proiectată pe baza unei arhitecturi pe 3 nivele (3-tier architecture) unde functionalitațile elementelor de pe un nivel depind de functionalitățile componentelor aflate pe nivelul inferior acestuia.

Cele 3 nivele sunt următoarele.

- Presentation
- Businesslogic
- DataAccess

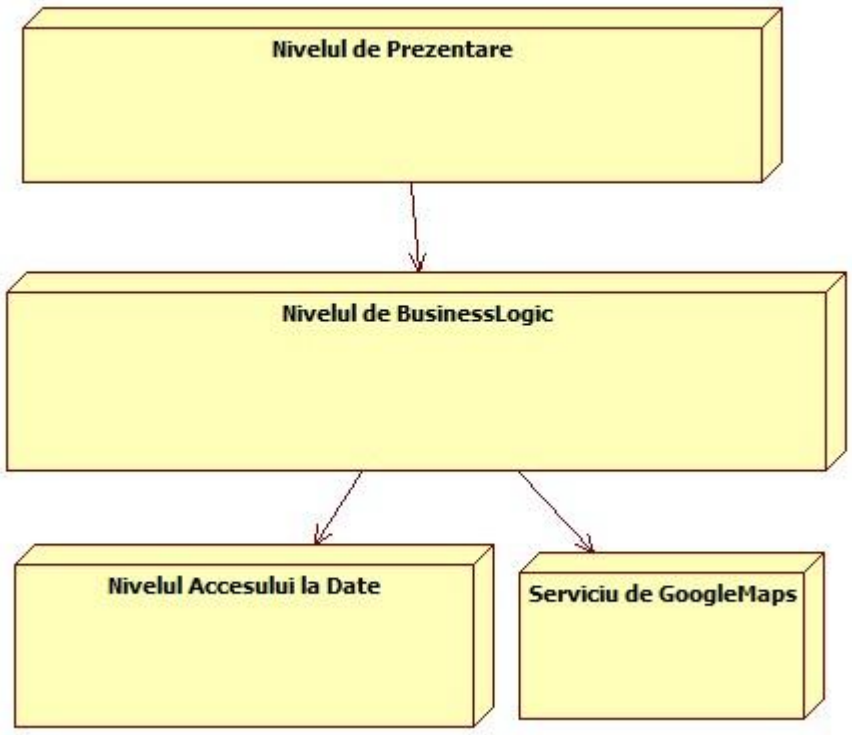


Figura 5.1 Diagrama conceptuală

5.2 Nivelul de prezentare (Presentation)

La acest nivel se afla toate componentele responsabile de interacțiunea utilizatorului cu aplicația.

În cadrul aplicației acest nivel este reprezentat de un pachet cu obiecte ce extind componentele de tip Activity și Fragment ale platformei Android.

5.2.1 Principii de design

Nivelul de prezentare a fost proiectat ținând cont de o serie de principii de design a aplicațiilor Android, promovate de Google [5].

Conform acestor principii elementul central al aplicațiilor ar trebui să fie conținutul pe care acestea îl ofera utilizatorului și în acest scop ar trebui să existe o ierarhie de moduri de vizualizare care să ofere utilizatorului o imagine de ansablu asupra conținutului.

Este indicat ca modul de prezentare a informației să fie structurat pe 3 nivele (conform figurii 5.2) și anume:

- Nivelul vederilor de nivel înalt

Acest nivel de vizualizare e conceput să le ofere utilizatorilor o vedere de ansamblu asupra conținutului oferit de aplicație precum și opțiuni de navigare prin care aceștia ar putea intra în detaliu în cadrul conținutului. La acest nivel pot fi încadrate interțetele grafice ce sunt puse la dispoziția utilizatorului imediat dupa pornirea aplicației.

- Nivelul vederilor de tip categorie

In cadrul acestui nivel utilizatorul poate vedea datele din conținutul aplicației grupate pe categorii distincte.

- Nivelul vederilor de detaliu

În cadrul acestui nivel utilizatorii au posibilitatea de a vizualiza elemente de detaliu și de a manipula conținutul aplicației.

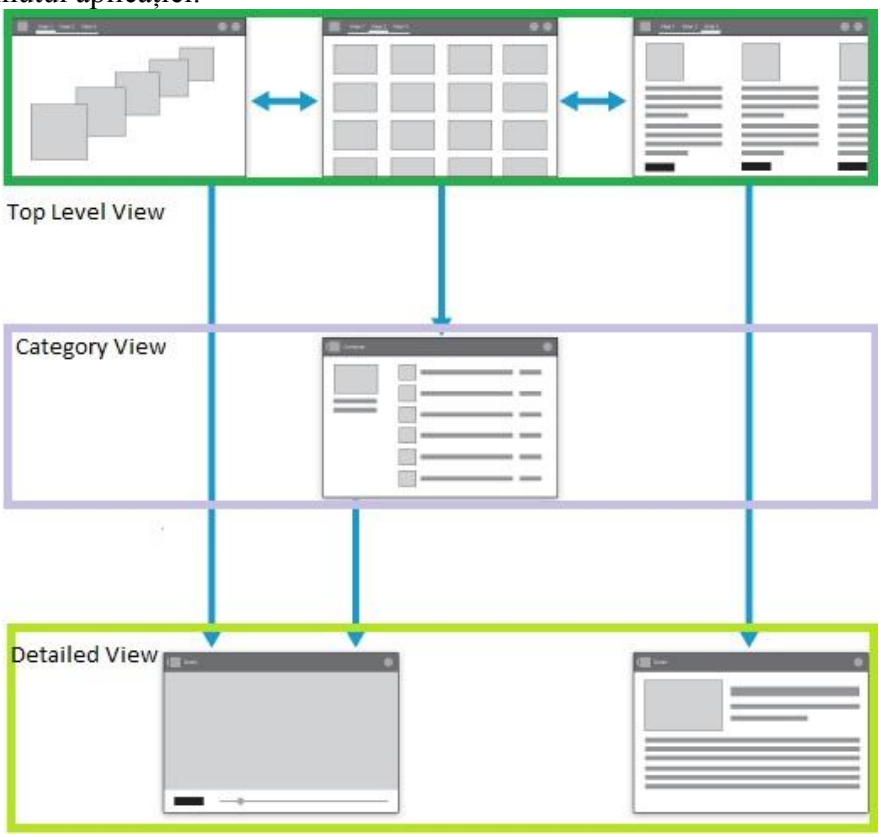


Figura 5.2 Structura modurilor de vizualizare

Bara de Acțiuni (Action Bar)

În aplicațiile destinate pentru Android 3.0 sau versiuni superioare se recomandă folosirea elementului **Action Bar** pentru :

- A pune la dispoziția utilizatorului modalități facile de navigare prin conținut.
- A oferi acces la diverse funționalități ale aplicației. Nu este recomandat totuși ca toate functionalitățile să fie puse la dispoziție în Action Bar, ci doar cele care respecta criteriul FIT (Frequent, Important, Tyical), adică funcționalitățile care sunt fie importante, fie des folosite sau tipice, la care utilizatorul se așteaptă să găsească în cadrul aplicației.
- A oferi un mod de personalizare a aplicației având loc dedicat pentru logo-ul și numele aplicației.

5.2.2 Implementare concretă

În cadrul aplicației, nivelul de prezentare este un pachet cu clase java ce extind clasa *Activity* sau *Fragment*, fiecare clasă fiind responsabilă de gestionarea interfeței grafice a unei anumite părți ale aplicației.

DayViewFrament este o componenta de tip Fragment care afișează activitățile planificate de utilizator în funcție de o anumită zi. Pe lângă faptul că extinde clasa *Fragment*, aceasta implementeaza și interfața *LoaderManager.LoaderCallbacks*, interfață necesara folosirii obiectelor de tip *Loader*, obiecte cu ajutorul se pot executa, pe alte fire de execuție decât cel implicit, operații de încărcarea a datelor din baza de date.

Printre metodele surprascrise a clasei *Fragment* se numară *OnActivityCreated(),* această metodă e suprascrisă astfel încat să comunice obiectului de tip *Activity*, căreia îi este atașat, faptul că are propriul meniu de opțiuni care ar trebui afișat în *Action Bar*, și de asemenea, tot în cadrul acestei metode se obține o intanță a *LoaderManager*-ului din *Android* care va fi folosit pentru gestionarea obiectelor de tip *Loader* utilizate.

Interfața grafică oferiă de această componentă se încadrează la nivelul vederilor *de nivel de detaliu*. Șablonul acestei interfețe este definit în fișierul *day_view_layout.xml*, fișier este folosit în metoda *onCreateView()* pentru a crea interfața grafică și a obține referințe către diversele componente grafice.

În cadrul acestei clase mai sunt definite și urmatoarele obiecte:

- *changeDayListener*, acesta este un obiect de tip *OnClickListener* atașat butoanelor prin care utilizatorul poate schimba data calendaristică, și are rolul de a actualiza lista de activități planificate în cazul în care utilizatorul schimbă data calendaristică.
- *eventListClickListener*, acesta este un obiect de tip *OnItemClickListener* atașat obiectului de tip *ListView* reprezentând lista de activități și se activează în momentul în care utilizatorul apasă pe una din activitățile din lista generând un obiect de tip *Intent* care lansează o componentă de tip *Activity* cu ajutorul căreia utilizatorul poate afla mai multe detalii legate de activitatea dorită.

MainActiviy este o clasă java ce extinde clasa *Activity* fiind chiar componenta de tip *Activity* principală a aplicației, ce va fi activată la lansarea aplicației. Aceasta este responsabilă de generarea interfeței de *nivel înalt* cu care va interacționa utilizatorul la lansarea aplicației.

Șablonul interfeței grafice este este definit în fișierul *top_view.xml* care va fi încărcat prin apelarea metodei *setContextView*() în cadrul metodei *onCreate*(), metodă specifică clasei *Activity*.

Tot în cadrul metodei *onCreate*() sunt obținute referințe către butoanele de navigare către celelalte moduri de vizualizare și le sunt atașate obiecte de tip *listener* corespunzătoare care să lanseze modurile de vizualizare selectate de utilizator

În cadrul clasei *MainActivity* este definit *buttonListener* un obiect de tip *OnClickListener* și atașat butoanelor de navigare.

Rolul acestui obiect este de a genera obiecte de tip *Intent* responsabile de lansarea în execuție a componentelor de tip *Activity* responsabile de modurile de vizualizare oferite de aplicație:

- Optiuni de navigare
- ManageEvents
- DetailedView
- ObjectiveMapview
- Help

CalendarView este o componenta de tip *Fragment* care îi permite utilizatorului să vizualizeze zilele dintr-o anumită lună, marcându-le corespunzător pe cele în care utilizatorul are planificate activități.

Interfața grafică oferită de această componentă se situează la nivelul vederilor de detaliu datorită conținutului oferit de aceasta. Această interfață este definită în fișierul *calendar.xml*, fișier încărcat în cadrul metodei *onCreateView*() specifica clasei *Fragment*, tot în cadrul acestei metode sunt obținute referințe către restul elementelor grafice cu care utilizatorul poate interacționa și cărora le sunt atașate obiecte de tip *listener* corespunzătoare ce vor reacționa în urma interacțiunii cu utilizatorul.

Interfața grafică pune la dispoziția utilizatorului butoane prin care acesta poate alege să vadă situația de pe luna precedentă sau luna urmatoare, aceste butoane au atașate obiecte de tip *OnClickListener* responsabile cu actualizarea conținutului în cazul în care utilizatorul schimbă data calendaristică.

În cadrul acestei clase este definită interfața *onDaySelectedListener* care va fi folosită pentru a comunica cu obiectul de tip *Activity* căruia îi va fi atașat. Informațiile schimbate prin intermediul interfeței sunt de date de tip *java.util.calendar* în funcție de care se va face afișarea datelor.

În interfața grafică, situația zilelor dintr-o lună este redată vizual prin intermediul unei componente de tip *GridView* unde fiecare element reprezintă o zi din lună. Conținutul este gererat de un obiect de tip *CalendarAdapter*. Obiectul de tip *GridView* are atașat un obiect de tip *OnItemClickListener* care are rolul de a lansa un mod de vizualizare prin intermediul căruia utilizatorul poate vedea toate activitățile ce au loc în ziua aleasă de acesta.

PerspectiveActivity este o componenta de tip *Activity* ce pune la dispoziția utilizatorului un mod de vizualizare în timp a activitatilor planificate. Astfel conținutul oferit utilizatorului de către această componentă o încadrează la nivelul *vederilor de tip categorie*.

Clasa *PerspectiveActivity* extinde clasa *Activity* suprascriind o serie de metode.

- *onCreate*() .În cadrul acestei metode se obține o referință către *ActionBar* prin metoda *getActionbar*(). Odată obținută referinta către *ActionBar* aceasta este adaptată pentru a corespunde cerintelor aplicației astfel *ActionBar* este setat să ofere un mod de navigare tip lista de opțiuni prin apelarea metodei *actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);*, punând la dispozitia utiliztorului 2 optiuni:
  - day view – opțiune ce îi oferă utilizatorului posibilitatea de a vizualiza toate activitățile planificate pentru o anumita zi.Funcționalitatea aceastei opțiuni este asigurat de un obiect de tip *DayViewFragment* care va fi atașat la *ActionBar*
  - month view – opțiune ce îi permite utilizatorului să vizualizeze zilele din lună marcându-le corespunzător pe cele în care sunt planificate activități. Funcționalitatea

aceastei opțiuni este asigurată de un obiect de tip *CalendarView* care va fi atașat la *Action Bar*.

De asemenea în cadrul acestei clase se găsește definit și un obiect de tip *OnNavigationListener* care este atașat la *Action Bar* și are rolul de actualiza interfața grafică afisând fragmentul corespunzător selecției utilizatorului din lista de opțiuni de navigare.

Schimbarea fragmentelor se face prin tranzacții de tip *FragmentTranzactions* . aceste tranzacții sunt definite în cadrul metodei *onNavigationItemSelected* a obiectului de tip *OnNavigationListener*

Pe lângă faptul ca *PerspectivActivity* extinde clasa *Activity* aceasta mai implementează și interfețele *OnDaySelectedListener* și *DayViewCommnunication* definite în cadrul claselor *CalendarView.java* respectiv *DayViewFragmen.javat* folosite pentru a face schimb de informații între fragmenele atașate.

De asemenea în cadrul acestei clase este implementată și metoda *onCreateOptionsMenu*() care încarcă meniul definit în *top_menu.xml* și face vizibile în *Action* Bar opțiunile acestuia.

Optiunile disponibile la nivelul acestei componente sunt:

- Add new event- opțiune ce permite utilizatorului să adauge o nouă activitate în baza de date
- view Eisenhower matrix – încadreaza activitățile planificate în matricea Eisenhower[2].

Tot în cadrul acestei clase este implementată și metoda *onOptionsItemSelected*().Această metodă este apelată în momentul în care utilizatorul alege o opțiune din meniul de opțiuni și are rolul de a crea un obiect de tip *Intent* care să pornească componentele de tip *Activity* corespunzătoare fiecărei opțiuni.

ManageEventFragment este o componentă de tip *Fragment* care îi permite utilizatorului să vizualize toate activitățile asociate unei anumite etichete.

Interfața grafică oferită de acestă componentă îi oferă utilizatorului o listă cu toate etichetele definite de acesta și stocate în baza de date și o listă cu activități, listă automat populată în funcție de eticheta aleasă de utilizator din lista de etichete.

Interfața grafică oferită de acestă componentă este încadrată la *nivelul vederilor de detaliu.*

Interfața grafică este definită în fișierul *labels_fragment_layout.xml,* fișier încărcat în cadrul metodei *onCreateView().* Tot în cadrul acestei metode sunt obținute referințe către elementele vizuale ale interfeței grafice.

Operația de încărcare a etichetelor este executată pe un fir de execuție separat de cel implicit, aceast aspect este asigurat de următorul mecanism:

În interiorul clasei *ManageEventsFragment* este definit un obiect *labelStringLoader* care implementează interfața *LoaderManager.LoaderCallbacks<ArrayList<String>>,*în definiția acestui obiect sunt suprascrise metodele *onCreateLoader* și *onLoadFinish* astfel încat să folosească un obiect de tip *LabelLoader* pentru a executa pe un fir de execuție separat operația de încărcare a etichetele din baza de date și a pune datele in *spinnerAdapter,* obiect de tip adapter asociat listei de etichete.

Obiectul *labelStringLoader* este apoi folosit de către *LoaderManager* pentru a inița acest mecanism de citire a etichetelor prin urmatorul apel *getLoaderManager().initLoader(0, null, labelStringLoader);*

Operația de citire a activităților utilizatorului din baza de date sunt executate pe un fir de execuție separat, aceasta aspect fiind asigurat de un mecanism similar cu cel descris mai sus astfel că:

In interiorul clasei *ManageEventsFragment* este definit un obiect *agendaEventLoader* care implementează interfața *LoaderCallbacks<ArrayList<AgendaEvent>>* ,în definiția acestui

obiect sunt suprascrise metodele *onCreateLoader* și *onLoadFinish* astfel încât să folosească un obiect de tip *EventsLoader* pentru a executa pe un fir de execuție separat operația de încărcare a etichetelor din baza de date și a pune datele în *eAdapter,* obiect de tip EventsAdapter asociat unui obiect ListView reprezentând lista de activități.

Obiectul *agendaEventLoader* este apoi folosit de către *LoaderManager* pentru a inița acest mecanism de citire a etichetelor prin urmatorul apel *getLoaderManager().initLoader(0, null, agendaEventLoader);*

De asemenea există definit un obiect *labelSpinnerSelectListener* de tip *OnItemSelectedListener* care apoi este atașat listei de etichete, fiind responsabil de schimbarea conținutul listei de activități în funcție de eticheta aleasă de utilizator la un moment dat, acest lucru fiind realizat prin apelul *getLoaderManager().restartLoader(1, b, agendaEventLoader);* în cadrul metodei *onItemSelected()* suprascrise în cadrul definirii acestui obiect.

Acest componentă de tip *Fragment* are propriul meniu a cărui opțiuni vor fi vizibile în *Action Bar* când utilizatorul interacționează cu aceasta.

Meniul este definit în *top_menu.xml* și va fi încarcat în cadrul metodei onCreateOptionsMenu(), iar în cadrul metodei *onOptionItemSelected()* vor fi definite acțiunile ce trebuie executate pentru fiecare opțiune selectată de utilizator.

Optiunile disponibile în cadrul interfeței grafice oferită de această componentă sunt:

- Add new event- opțiune ce permite utilizatorului să adauge o nouă activitate în baza de date
- view Eisenhower matrix – încadrează activitătile ce sunt eticheteate cu eticheta aleasă la un moment dat de utilizator în matricea eisenhower.

Tot în cadrul acestei clase este implementată și metoda *onOptionsItemSelected*(). Această metodă este apelată în cazul în care utilizatorul alege o opțiune din meniul de opțiuni și are rolul de a crea un obiect de tip Intent care să pornească componentele de tip *Activity* corespunzătoare opțiunii alese.

ManageEventActivity este o componentă de tip *Activity* ce le oferă utilizatorilor o evidență a tuturor activităților din baza de date și mijloace de gestionare a acestora.
Astfel, prin conținutul oferit, această componentă poate fi încadrată la *nivelul vederilor de tip categorie.*

ManageEventActivity extinde clasa *Activity* și surpascrie metoda *onCreate*(), unde obține o referință către *ActionBar* prin apelul *getActionBar*(); și o adaptează setandu-i modul de navigare la tab-uri de navigare prin metoda *setNavigationMode(ActionBar.NAVIGATION_MODE_TABS*); și atasând fragmentele corespunzătoare fiecărui tab.

Tab-urile afișate de *Action Bar* sunt:

- Tasks- unde utilizatorul poate vedea toate activitățile în funcție de etichete. Functionalitățile corespunzătoare acestui mod de vizualizare sunt asigurate de Fragmentul *ManageEventsFragment*
- Labels- unde utilizatorul poate vedea toate etichetele. Functionalitățile acestui mod de vizualizare sunt asigurate de fragmentul *LabelsFragment.*

În cadrul clasei *ManageEventActivity* se găsește definiția unei clase interioare *TabListener* ce implementează interfața *ActionBar.TabListener* și care este folosită pentru a afișa componenta de tip *Fragment* corespunzatoare tab-ului ales de utilizator la un moment dat. Afișarea propriu-zisă a componentelor de tip *Fragment* se face folosind tranzacții de tip *FragmentTranzaction* definite în cadrul implementărilor metodelor interfeței *ActionBar.TabListener.*

EisenhowerMatrixActivity este o componentă de tip *Activity* care îi permite utilizatorului să vizualize o colecție de activități planificate sub forma unei matrici Eisenhower, activitățile fiind încadrate în diversele cadrane ale matricei în funcție de gradul lor de importanță[2].

Activitățile asupra cărora se va aplica matricea Eisenhower vor fi încărcate din baza de date în functie de criteriile menționate în obiectul de tip Intent folosit pentru pornirea acestei componente. Astfel matricea Eisenhower poate fi aplicată pe activitățile planificate într-o anumita zi, luna sau care posedă o anumită etichetă, aceste criterii fiind menționate în obiectul de tip *Intent* folosit pentru a porni această componentă și care sunt preluate în cadrul metodei *onCreate*().

Încărcarea efectiva a activităților în funcție de criteriile date se face prin intermediul unui obiect de tip *Loader* și anume *EventsLoader*.

Clasa *EisenHowerMatrixActivity* extinde clasa *Activity* și implementeaza interfața *LoaderManager.LoaderCallbacks<ArrayList<AgendaEvent>>* a carei metode sunt necesare pentru accesare *LoaderManagerului* și folosirea obiectelor de tip *Loader*.

AddEventActivity este o componentă de tip *Activity* ce îi oferă utilizatorului o intefață grafică prin intermediul căreia poate adăuga noi activități în baza de date. De asemenea această componentă poate fi folosită pentru a oferi mai multe detalii asupra activităților utilizatorului precum și posibilitatea de a efecuta modificări asupra acestora, acest aspect depinzând de datele conținute în obiectul de tip *Intent* folosit pentru lansarea acestei componente, datele din obiectele de tip *Intent* fiind procesate în cadrul metodei *onCreate*().

Interfața grafică îi permite utilizatorului să definească urmatoarele detalii legate de o activitate

- Denumire
- Dată calendaristică la care să fie planificată
- Etichetă
- Grad de importanță
- Detalii
- Status

Utilizatorul poate atribui activității o etichetă dintră-o listă de etichete. Această listă de etichete este încărcată din baza de date folosind un obiect de tip *LabelsLoader*.

În cadrul clasei *AddItemActivity* există o referință către servicul *TableModuleService,* serviciu folosit pentru a salva în baza de date activitatea definită sau modificată de către utilizator.

Pe lângă faptul ca *AddItemActivity* extinde clasa *Activity* aceasta implementează și interfața *LoaderManager.LoaderCallbacks<ArrayList<String>>* necesară folosiri *LoaderManager*-ului și a componentei de tip *LabelsLoader*.

Interfața grafică oferită de această componentă poate fi încadrată *la nivelul vederilor de detaliu* datorită conținutului oferit de aceasta. Șablonul acestei interfețe grafice este definiti în cadrul fișierului *new_task_form.xml,* fișier încărcat în cadrul metodei *onCreate().*

AddLabelActivity este o componentă de tip *Activity* ce îi oferă utilizatorului o interfață grafică prin intermediul căreia poate adăuga noi etichete în baza de date. Această componentă poate de asemenea să fie folosită și pentru a oferi mai multe detalii asupra etichetelor definite de utilizator precum și posibilitatea de a face modificări asupra acestora, acest aspect depinzând de datele conținute de obiectul de tip *Intent* folosit pentru a lansa această componentă.

Interfața grafică îi permite utilizatorului să definească urmatoarele detalii legate de o activitate

- Denumire

- Detalii

În cadrul clasei *AddLabelActivity* există o referință către servicul *TableModuleService* care este folosit pentru a salva în baza de date eticheta definită sau modificată de către utilizator.

Interfața grafică oferită de această componentă poate fi încadrată la nivelul vederilor de detaliu, datorită conținutul oferit de aceasta.

AddMapObjectiveActivity este o componentă de tip *MapActivity* ce îi oferă utilizatorului o intefață grafică prin intermediul căreia poate adăuga noi obiective pe hartă reprezentând puncte de interest pentru utilizator și pot fi folosite ca și context în definirea unor activităților.Aceste puncte de interest sunt stocate în baza de date locală. De asemenea această componentă poate fi folosită pentru a oferi mai multe detalii legate de punctele de inters deja definite de către utilizatorul precum și posibilitatea de a efecuta modificări asupra acestora, acest aspect depinzând de datele conținute în obiectul de tip *Intent* folosit pentru lansarea acestei componente, datele din obiectele de tip *Intent* fiind procesate în cadrul metodei *onCreate*().

Această componentă extinde clasa *MapActivity*, o clasă din pachetul *com.google.android.maps* furnizat de modulul de Google API. *MapActivity* este o clasa java similară cu cu *Activity* ce oferă în plus facilități de redare și manipulare a hărții oferite de serviciul *GoogleMaps*.

Interfața grafică îi permite utilizatorului să definească urmatoarele detalii legate de o activitate

- Denumire
- Detalii pe care dorește să le specifice utilizatorul legate de punctul de interest
- Poziție pe hartă

Funționalitatea de specficare a poziției noului punct pe hartă este asigurată de un obiect de tip *AddNewObjectiveOverlay* ce se aplică peste hartă furnizată de serviciul *GoogleMaps*, înregistrează locul specificat de utilizator prin atingerea hărții și convertește datele obținute în coordonate geografice care sunt apoi stocate în baza de date.

De asemenea pe langă noul obiectiv definit, utilizatorul are posibilitatea de vedea și punctele de interest definite anterior, acest fapt fiind asigurat de un obiect de tip *CustomItemizedOverlay*. Acest obiect încarcă din baza de date coordonatele punctelor definite și le redă pe hartă. Această componentă prezintă de asemenea și o referință către un serviciul *TableModuleService* ce va fi folosit pentru persistarea datelor.

De asemenea este folosit și componenta de LocationManager din Android pentru a reda poziția curentă a utilizatorului.

Prin funționalitățile ce le oferă utilizatorului interfața grafică furnizată de această componentă se încadrează la nivelul *vederilor de detaliu* iar șablonul acestei interfețe grafice este definiti în cadrul fișierului *map_view.xml,* fișier încărcat în cadrul metodei *onCreate().*

ViewObjectiveActivity este o componentă de tip *MapActivity* ce îi oferă utilizatorului o intefață grafică prin intermediul căreia poate vizualiza toate punctele de interes ce reprezintă un context pentru activitățile planificate într-o anumită zi sau care prezintă o anumită etichetă, în funcție de alegerea utilizatorului. În cadrul acestei componente sunt definite o serie de obiecte de tip listener responsabile de interpretarea acțiunilor utilizatorului. Astfel există un obiect de tip *OnCheckedChangeListener* care va fi atașat grupului de butoane prin intermediul căruia utilizatorul va putea selecta criteriul în funcție de care să vizualizeze punctele de interes, opțiunile disponibile fiind:

- În funcție de etichetă. În urma alegerii acestei opțiuni i se pune la dispoziția utilizatorului o serie de componente grafice ce îi permite acestui să selecteze o etichetă din lista de

etichete și în funcție de aceasta să populează harta cu punctele de interes ce reprezintă contextul asociat activităților ce au eticheta respectivă. Pentru redarea punctelor de interest corespunzătoare etichetei alese de utilizator este folosit un obiect de tip *OnItemSelectedListener* atasat *spinner*-ului de unde acesta poate alege etichetele.

- În funcție de zi. Utilizatorul alege o anumită data celendaristică iar componenta afișează pe hartă toate obiectivele ce reprezintă puncte de interes pentru activitățile planificate la data respectivă. Pentru redarea punctelor de interest corespunzătoare datei specificate de utilizator este folosit un obiect de tip *OnClickListener* atașat butoanelor din interfața grafică prin intermediul cărora utilizatorul specifică data calendaristică.

În cadrul acestei clase sunt folosite și trei component de tip Loader (*LabelLoader,EventLoader* și *MapObjectiveStringLoader*) pentru a încărca din baza de date etichetele utilizatorului, activiățile definite de acesta precum și punctele de interes corespunzătoare.

Redarea efectiva a obiectivelor de pe hartă este folosit un obiect de tip *CustomItemizedOverlay*
Șablonul acestei interfețe grafice este definiti în cadrul fișierului *view_objectives_layout.xml*, fișier încărcat în cadrul metodei *onCreate().*

De asemenea acestă componentă pune la dispoziția utilizatorului un meniu cu următoarele opțiuni:

- Add new map objective. Opțiune prin intermediul căreia utilizatorul își poate defini un nou punct de interes
- View all map objectives. Opțiune prin intermediul căreia utilizatorul poate vedea o listă cu toate punctele de interes definite.

Meniu este definit în fișierul *map_options_menu.xml* ce este încărcat în cadrul metodei *onCreateOptionsMenu()*, iar acțiunile ce trebuie executate pentru fiecare opțiune sunt definite în cadrul metodei *onOptionsItemSelected()*

MpObjectiveListActivity este o componentă de tip *ListActivity* responsabilă de afișarea punctelor de interes definite de utilizator și stocate în baza de date sub forma unei liste.

Încărcarea punctelor este realizată de o componentă de tip *MapObjectivesLoader* ce execută operația de citire din baza de date pe un alt fir de execuție. De asemenea este definit un obiect de tip *OnItemSelectedListener* ce este atașat listei și în cazul în care utilizatorul alege un anumit element din listă, lansează un mod de vizualizare ce îii oferă mai multe detalii legate de elementul respectiv.

Diagrama de clase corespunzătoare acestui modul se poate consulta în Anexa 1.

5.3 Nivelul de BusinessLogic

La acest nivel sunt definite componente ce implementează logica aplicației.

Acest modul constă într-un pachet de clase java ce prelucrează datele stocate în baza de date pentru a obține informații relevante pentru utilizator. O parte din componentele definite în acest modul sunt proiectate astfel încât să ruleze operațiile solicitante, precum accesul la baza de date, pe fire de execuție separate decât cel implicit cu scopul de îmbunătății performanțele aplicației reducând riscul blocării acesteia.

LabelsLoader este o componentă de tip *AsyncTaskLoader* folosită de componentele situate la nivelul de prezentare pentru obține din baza de date etichetele definite de utilizator.

Această clasa extinde clasa *AsyncTaskLoader* din Android, clasa ce asigură că operația de încărcare a datelor din baza de date va fi executată pe un fir de execuție separat de cel principal.

Metoda importantă a acestei clase este metoda *loadInBackground()*, este o metodă a clasei AsyncTaskLoader, suprascrisa astfel încât să apeleze componenta *EventTableGateway* pentru obținerea datelor din baza de date.

ObjectivesLoader este o componentă de tip *AsyncTaskLoader* folosită de componentele situate la nivelul de prezentare pentru obține din baza de date punctele de interes definite de utilizator.

Această clasa extinde clasa *AsyncTaskLoader* din Android, clasa ce asigură că operația de încărcare a datelor din baza de date va fi executată pe un fir de execuție separat de cel principal.

Metoda importantă a acestei clase este metoda *loadInBackground()*, este o metodă a clasei AsyncTaskLoader, suprascrisa astfel încât să apeleze componenta *EventTableGateway* pentru obținerea datelor din baza de date.

EventsLoader este o componentă de tip *AsyncTaskLoader* folosită în cadrul componentelor de la nivelul de prezentare pentru încărcarea din baza de date a activităților definite de utilizator.

Aceasta clasă extinde clasa *AsyncTaskLoader* din Android, clasa ce asigura ca operația de încărcare a datelor din baza de date va fi executată pe un fir de execuție separat de cel principal.

Metoda importantă a acestei clase este metoda *loadInBackground()*, este o metodî suprascrisă a clasei *AsyncTaskLoader*, în cadrul acestei metode este apelată componenta *EventTableGateway* pentru a obține datelor necesare din baza de date. Această componentă asigură încărcarea activităților utilizatorului în funcție de diferite criterii precum zi, luna, etichetă, toate aceste criterii fiind definite într-un obiect de tip *OperationType* transmis ca parametru constructorului clasei.

EventAdapter este o componenta de tip **ArrayAdapter** folosită la prelucrearea obiectelor de tip **AgendaEvent** astfel încât să poată fi redate de către elemente vizuale precum *ListView*.

Această componentă creează un obiect de tip *View* pentru fiecare **AgendaEvent** în care sunt afișate informații legate de acesta precum denumirea, eticheta, ora la care trebuie să se desfăsoare, și statusul, și colorând corespunzător fundalul componentei *View* în functie de gradul de importanță(culorile fiind rosu, corespunzător priorității urgent și important, galben pentru important dar neurgent, portocaliu pentru urgent dar neimportant, și verde pentru neurgent si neimportant). Obiectele de tip *View* create sunt apoi atașate componentelor de tip *ListView* responsabile de redarea vizuala a datelor.

EventAdapter extinde clasa *ArrayAdapter* și suprascrie metoda *getView*() implementând mecanismul de creere a obiectelor de tip *View* corespunzătoare.

CalendarAdapter este o componentă de tip *BaseAdapter* folosită în redarea calendarului. Rolul acetei componente fiind cel de a genera colecția de elemnte de tip *View* corespunzătoare zilelor dintr-o luna și marcandu-le corespunzător pe cele în care utilizatorul are planificate activități. Elementele de tip *View* generate vor fi apoi utilizate de către componenta de tip *GridView* din clasa CalendarView pentru redarea grafică a calendarului. Pentru fiecare obiect de tip View corespunzător unei zile din luna este afișat numărul acesteia din luna și i se asociază o culoare în functie activitatea cu cea mai mare prioritate planificată în ziua respectivă

Metode importante:

- setUp()- metodă în cadrul căreia se apelează un obiect de tip EventTableModule pentru a obține din baza de date zilele din luna în care sunt planificate evenimente.
- refeshDays() metoda folosită pentru a actualiza conținutul
- getView()- metodă a clasei BaseAdapter și suprascrisă astfel încât să includă mecanismul de formare a obiectelor de tip View corespunzătoare

TableModuleService este o componenta de tip *Service* ce rulează în fundal și asigură componentelor de la nivelul de prezentare functionalități precum operații de creere,actualizare și ștergere pentru obiecte de tip *MapObjective, AgendaEvent* și *Label*.

În cadrul acestei clase sunt definite două clase interioare ce extind clasa *AsyncTask* și sunt folosite pentru a executa pe fire de execuție separate operațiile ce presupun acces la baza de date. Cele două clase sunt *MyTask*, responsabilă de operațiile ce țin de obiecte de tip *AgendaEvent* și *MyLabelTask*, responsabilă de operațiile ce țin de obiecte de tip *Label*.
În ambele clase este suprascrisă metoda *doInBackGround*() astfel încât să folosească un obiect de tip *EventTableGateway* pentru a efectua operațiile necesare asupra bazei de date. Tipurile de operații executate la un moment asupra datelor de tip *AgendaEvent* și *Label* sunt definit de asemenea in interiorul metodei *doInBackGround* și depind de conținutul obiectelor de tip *OperationType* transmise ca parametru.

UpdateServiceWidget este o componentă de tip *Service* ce rulează în fundal și are rolul de a actualiza conținutul afișat de *widget*-urile aplicației.

Această clasă extinde clasa *Service*, suprascriind metoda *onStart*() astfel încât să genereze un obiect de tip *RemoteView* ce afișează data curentă precum și numărul de activități planificate de utilizator pentru data respectivă. Obiectul de tip *RemoteView* generat este apoi transmis widget-urilor aplicație pentru a fi afișat.

Datele care vor fi afișate de widget sunt generate în cadrul metodei *analyze*() apelate în cadrul metodei *onStart*(). În cadrul metodei *analyze*() este folosit un obiect de tip *EventTableGateway* pentru obținerea datelor necesare din baza de date asupra cărora se vor face prelucrările corespunzătoare obținerii informațiilor dorite.

ProximityAlertService este o componentă de tip *Service* cu ajutorul căreia este implementat un sistem de notificări ce se declanșează în momentul în care utilizatorul se află în apropierea unui obiectiv de pe hartă corelat cu o activitate planificată în ziua respectivă. Această componentă este lansată la pornirea aplicație, folosește un obiect de tip *EventTableGateway* pentru a prelua toate activitățile planificate în ziua respectivă, găsește obiectivele de pe hartă asociate acestora și folosește *LocationManager*-url pentru a seta un proximty alert pentru punctul respectiv printr-un apel al metodei *addProximityAlert().* Când utilizatorul se află în apropierea unui punct de interes pentru care a fost setat un proximity alert *LocationManager*-ul va trimite un mesaj către componenta *OnProximityReceiver*.

OnProximityReceiver este o componentă de tip *BroadcastReceiver* care are rolul de a intercepta mesajele transmise de *LocationManager* în momentul în care utilizatorul se află în apropierea unui punct de interest și a transmite informația respectiva utilizatorului
Diagrama de clase pentru nivelul de BusinessLogic poate fi găsită la Anexa 2.

5.4 Pachetul Data

Acest pachet conține componente reprezentând structurile de date folosite de componente situate pe cele 3 nivele (Prezentare, BusinessLogic și DataAccess). Clasele java din acest pachet modelează concepte de bază relevante pentru logica aplicației.

Clasa **AgendaEvent** se modelează pe conceptul de activitate planificată de utilizator la un moment dat, acest concept poate fi văzut ca și o înregistrare în agenda utilizatorului care să îi aducă aminte de un anumit eveniment de care trebuie să țină cont sau de o anumită acțiune pe care trebuie să o întreprindă la un moment dat. În definiția acestei clase se găsesc urmatoarele atribute ce sunt caracteristice unei activități planificate de utilizator, precum și metodele de tip getter și setter corespunzatoare:

- Name, reprezintă numele asociat activității respective, poate fi văzut ca și titlul intrării în agenda utilizatorului
- Priority, denotă prioritatea pe care o are pentru utilizator acțiunea respectivă
- Duration, durata acțiunii estimată de către utilizator
- Status, reprezintă stadiul în care se află activitatea, din perspectiva utilizatorului
- Label, etichetă asociată de către utilizator

Clasa **AgendaLabel** modelează conceptul de etichetă cu ajutorul căreia pot fi grupate mai multe activități ale utilizatorului în categorii în funcție de preferințele acestuia.

În cadrul clasei sunt definite urmatoarele atribute precum și metodele de tip getter și setter corespunzatoare:

- Title- denumirea etichetei definită de utilizator
- Description- camp în care utilizatorul poate defini o o descriere a etichetei

Clasa **MapObjective** se modelează pe conceptul de punct de interes. Acesta este un punct geografic ce poate fi relevant pentru contextul unei anumite activități definite de utilizator.

Această clasă extinde clasa *GeoPoint* din pachetul *com.google.android.map* furnizat de componenta Google Api, astfel încât să poată stoca denumirea și câteva detalii specificate sub forma text de către utilizator referitoare la punctul de interes

5.5 Nivelul de DataAccess

La nivelul acestei componente sunt definite elemente responsabile de accesul datelor stocate în fișierul ce reprezintă baza de date SQLite și procesarea lor astfel înât să poată fi folosită de către componentele nivelului de *BusinessLogic*.

EventTableGateway este o clasă java ce extinde clasa *SQLiteOpenHelper* din *Android* responsabilă de creerea, actualizarea și gestionarea bazei de date.

5.5.1 Baza de date

Datele definite de utilizator sunt salvate și stocate în cadrul unei baze de date *SQLite* generată de aplicație și stocată local pe dispozitiv.

Elementele Bazei de date sunt:

- *Event_table*

- *Labels_table*
- *MapObjectives_Tabel.*

Tabela *Events_table* .In cadrul acestei tabele sunt stocate înregistrări reprezentând activități planificate de utilizator.

Tabela are urmatoarele câmpuri:

- _id- câmp în care se va stoca o valoare numerică reprezentând codul unic al activității
- Name- câmp în care se va stoca o valoare tip șir de caractere reprezentând denumirea activității
- Description- câmp în care se va stoca o valoare de tip șir de caractere reprezentând o descriere a activității utilizatorului.
- Starttime- câmp în care se va stoca o valoare de tip Double reprezentând data la care este planificată activitatea, exprimată in milisecunde
- Priority- câmp în care se va stoca o valoare numerice reprezentând prioritatea asociată activității respective
- Label câmp în care se va stoca o valoare de tip șir de caractere reprezentând denumirea etichetei asociate activității

Tabelul *Labels_Tabel*.

În cadrul acestei tabele sunt stocate înregistrări reprezentând etichete cu ajutorul cărora se pot grupa activitățile definite de utilizator în categorii.

Tabelul are urmatoarele câmpuri

- _id- câmp în care se va stoca o valoare numerică reprezentând codul unic al etichetei
- Title-- câmp în care se va stoca o valoare detip șir de caractere reprezentând denumirea etichetei
- Description- câmp în care se va stoca o valoare de tip șir de caractere reprezentând o descriere a etichetei

Intre tabela Label_Tabel și Events_Label există o relație de tip 1 la n deoarece o etichetă din tabela Label_Tabel poate fi asociată mai multor activitati din tabela Events_Label. Această legătură marchează apartenența activităților la o singură categorie.

Tabelul *MapObjectives_Tabel*.

În cadrul acestei tabele sunt stocate înregistrări reprezentând puncte geografice ce pot fi relevante indeplinirii unei anumite activități devenind astfel un punct de interes relevant în contextul îndeplinirii acestia.

Tabel are urmatoarele câmpuri

- _id- câmp în care se va stoca o valoare numerică reprezentând codul unic al obiectivului
- Description- câmp în care se va stoca o valoare de tip șir de caractere reprezentând o descriere a obiectivului
- Lat- câmp în care se va stoca o valoare numerică reprezentând latitudinea la care este situat punctul respectiv
- Long - câmp în care se va stoca o valoare numerică reprezentând latitudinea la care este situat punctul respectiv
- Title-- câmp în care se va stoca o valoare detip șir de caractere reprezentând denumirea etichetei

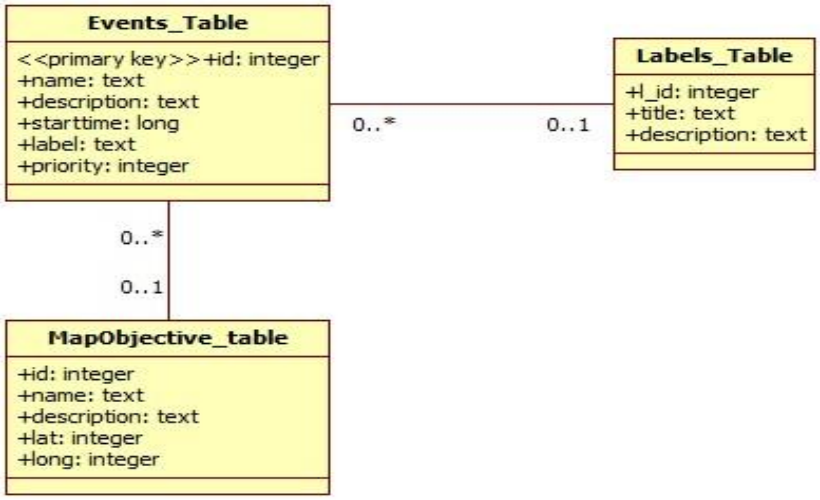


Figura 5.3 Diagrama Bazei de date

5.6 Interacțiunea dintre componente

În cele ce urmează vor fi descrise interacțiunile dintre diversele componente ce au loc în scopul satisfacerii cazurilor de utilizare, aceste interacțiuni vor fi descrise folosind diagrame de secvențiere ce respectă standardul UML 1.x

Pentru satisfacerea cazului de utilizare de adaugare a unei noi activități la baza de date au loc următoarele interacțiuni dintre componente:

Când utilizatorul alege opțiunea *add event,* este creat un obiect de tip Intent care va fi folosit pentru pornirea componentei AddEventActivity. Odată pornită, este stabilită legătura cu serviciul TableModuleService, apoi este folosit un obiect de tip LabelsLoader, care la randul sau face apel la un obiect de tip EventTableGateway pentru a încărca etichetele din baza de date.

Dupa ce utilizatorul introduce datele trebuie să apese butonul save pentru a le persista. Odată apasat butonul save este creat un obiect de tip AgendaEvent cu datele furnizate și trimis către componenta de tip TableModuleService pentru a îl scrie în baza de date. În cadrul componentei TableModuleService este folosit un obiect de tip MyTask pentru a executa operațiile pe un alt fir de executie decât cel principal. Operația executată pe firul de executie respectiv este un apel către un obiect EventTableGateway pentru a salva obiectul AgendaEvent creat, aceasta ultima component asigura scrierea datelor în baza de date.

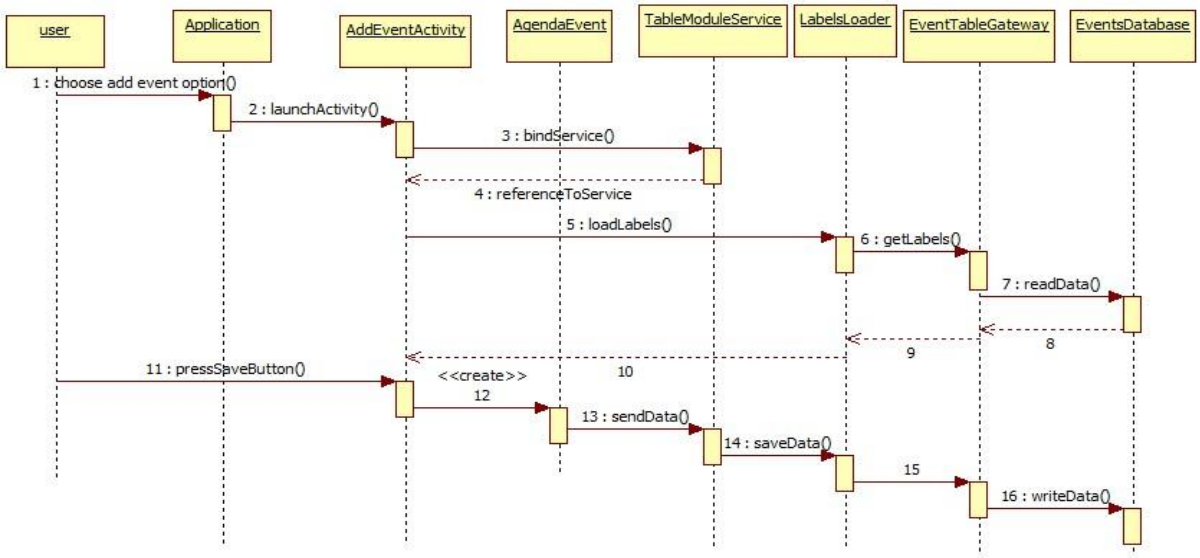


Figura 5.4 Diagramă de secvențiere pentru adaugarea unei noi activități

Pentru satisfacerea cazului de utilizare de actualizarea unei activități au loc următoarele interacțiuni dintre componente:

Când utilizatorul selectează o activitate din lista de activități, este creat un obiect de tip Intent care va fi folosit pentru pornirea componentei AddEventActivity, obiectul Intent continând obiectul de tip AgendaEvent corespunzător activității selectate. Odată pornită componenta AddEventActivity, aceasta stabilește legătura cu serviciul TableModuleService. Este folosit obiectul de tip LabelsLoader pentru a încarca etichetele din baza de date, de asemenea este actualizată și interfața grafică populând câmpurile cu date din obiectul de tip AgendaEvent primit.

După ce utilizatorul face modificări, acesta trebuie să apese butonul update pentru a persista modificările făcute.

Odată apăsat datele sunt salvate în obiectul de tip AgendaEvent care este trimis către componenta TableModuleService pentru a îl scrie în baza de date, în cadrul serviciului este apelat un obiect de tip MyTask pentru a executa operațiile pe un alt fir de execuție decât cel curent, operația executata pe firul de execuție respective este un apel către un obiect de tip EventTableGateway pentru actualizarea datelor, acesta din urmă realizând scrierea datelor din baza de date.

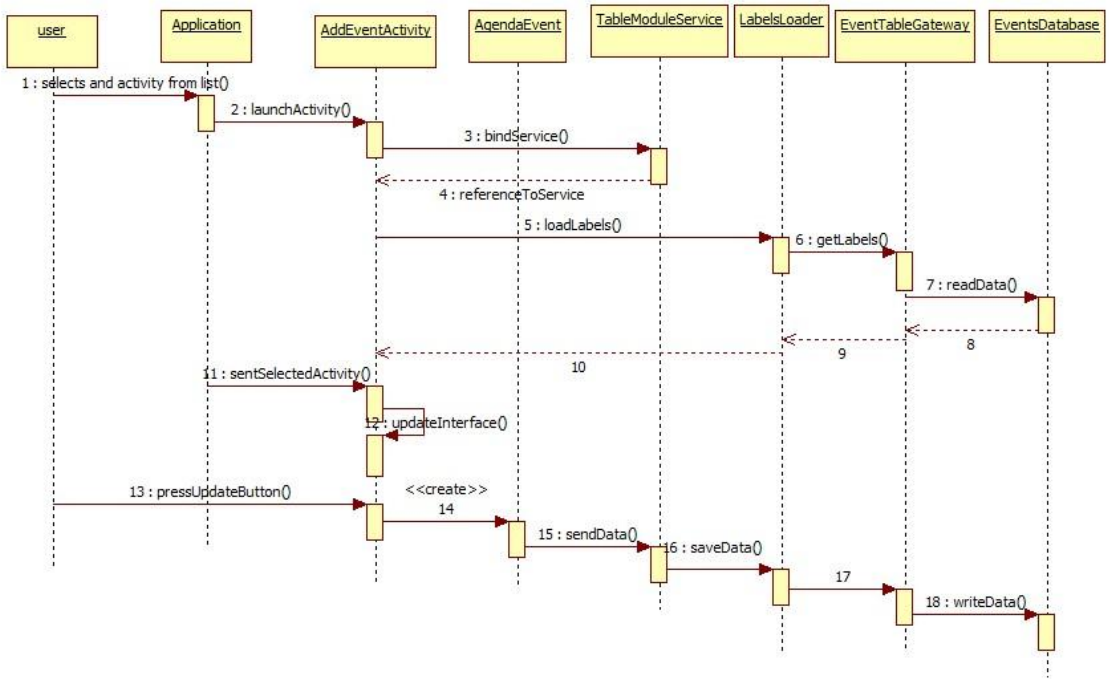


Figura 5.5 Diagrama de secventiere pentru modificarea unei activitati

Pentru satisfacerea cazului de utilizare de definire a unei noi eticehte și adaugarea ei la baza de date au loc următoarele interacțiuni dintre componente:

Când utilizatorul alege opțiunea AddLabel, este creat un obiect de tip Intent care va fi folosit pentru pornirea componentei AddLAbelActivity. Odată pornită, este stabilită legătura cu serviciul TableModuleService.

După ce utilizatorul introduce datele acesta trebuie sa apese butonul save pentru a persista datele. Odată apăsat butonul save este creat un obiect de tip AgendaLabel cu datele furnizate ți trimis către componenta TableModuleService pentru a îl scrie în baza de date. În cadrul serviciului este folosit un obiect de tip MyLabelTask pentru a executa operațiile pe un alt fir de execuție decât cel principal, operația executată pe firul de execuție respectiv este un apel către un obiect EventTableGateway pentru a salva obiectul AgendaLabel creat, această ultima componentă asigură scrierea datelor în baza de date.

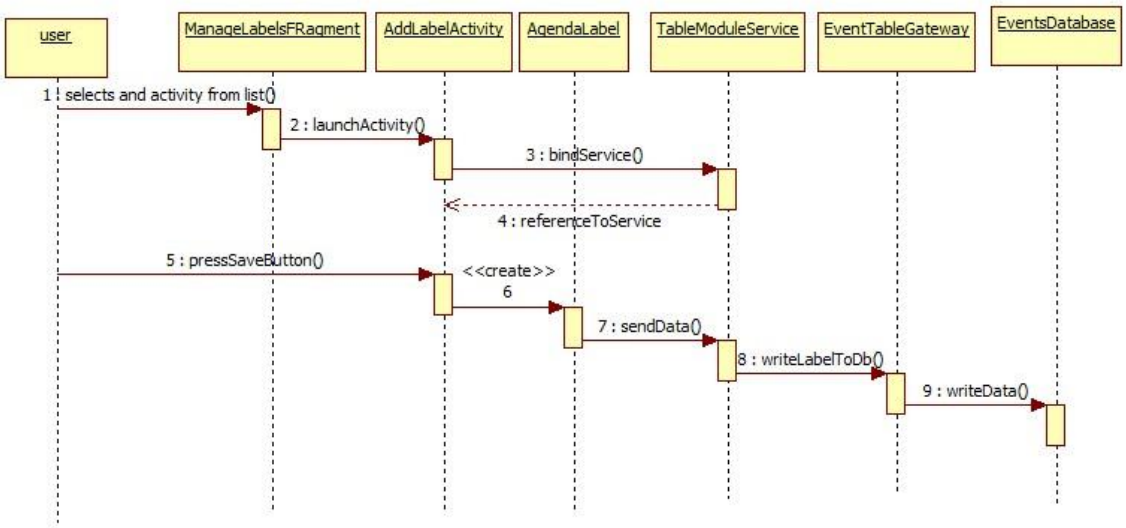


Figura 5.6 Diagrama de secventiere pentru adaugarea unei noi etichete

Pentru satisfacerea cazului de utilizare de modifcare a unei etichete deja exsitente au loc următoarele interacțiuni dintre componente:

Când utilizatorul alege o etichetă din lista de etichete pusă oferită de interfața grafică a componentei ManageLabelsFragment, este creat un obiect de tip Intent care va fi folosit pentru pornirea componentei AddLabelActivity, obiectul de tip Intent conține și obiectul de tip AgendaLabel corespunzător etichetei selectate. Odată pornită, este stabilită legătura cu componenta TableModuleService iar interfața grafică este actualizată populând câmpurile cu date din obiectul de tip AgendaLabel primit.

Dupa ce utilizatorul actualizează datele, acesta trebuie să apese butonul update pentru a persista schimbările. Odată apasat butonul modificările sunt salvate în obiectul de tip AgendaLabel care apoi e trimis către componenta TableModuleService pentru a îl scrie în baza de date. În cadrul componentei TableModuleService este folosit un obiect de tip MyLabelTask pentru a executa operațiile pe un alt fir de execuție decât cel principal, operația executată pe firul de executie respective apel catre un obiect EventTableGateway pentru a salva obiectul AgendaLabel creat, această ultimă componentă asigură scrierea datelor în baza de date

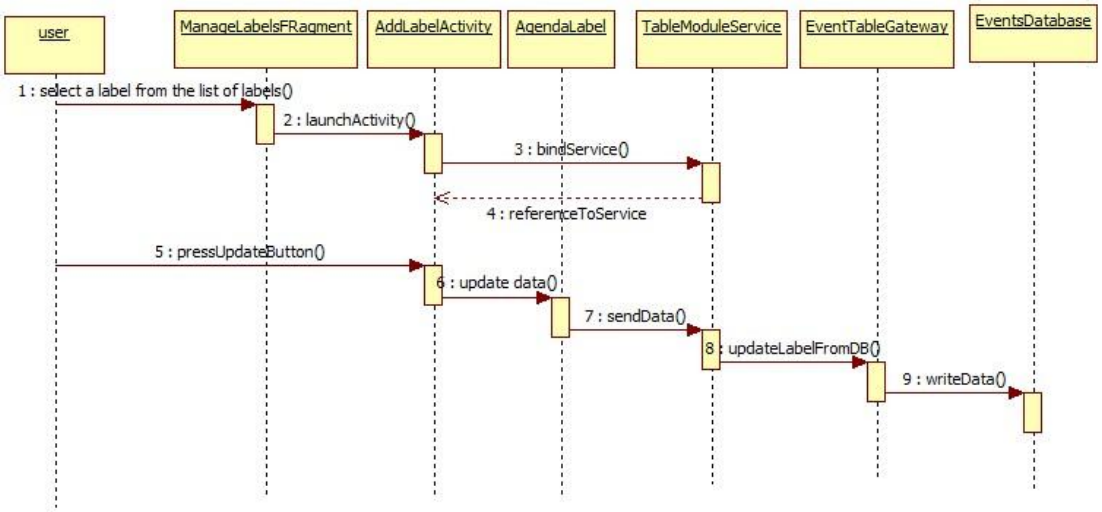


Figura 5.7 diagrama de secventiere pentru actualizarea unei etichete

Pentru satisfacerea cazului de utilizare de vizualizarea activitaților în funție de etichetă au loc următoarele interacțiuni dintre componente:

În modul principal de vizualizare gestionat de componenta MainActivity utilizatorul apasă butonul *manage events*. La apasărea butonului este creat un obiect de tip Intent ce va porni componenta ManageEventsActivity. La initializarea componentei ManageEventsActivity este creată o componenta de tip ManageEventsFragment care apoi va fi atașată la ActionBar. La inițializarea componentei ManageEventsFragment este folosit un obiect de tip LabelsLoader pentru încarcarea etichetelor și EventsLoader pentru încarcarea evenimentelor din baza de date.

Când utilizatorul alege o etichetă din lista de etichete obiectul de tip OnItemSelectedListener atașat acestei liste va trimite eticheta aleasă de către utilizator către obiectul de tip EventsLoader pentru a încărca din baza baza de date toate activitățile etichetate cu eticheta respectivă. Activitățile odată încărcate de către EventsLoader vor fi preluate de către un obiect de tip EventsAdapter responsabil de redarea lor în lista de activități a interfeței grafice.

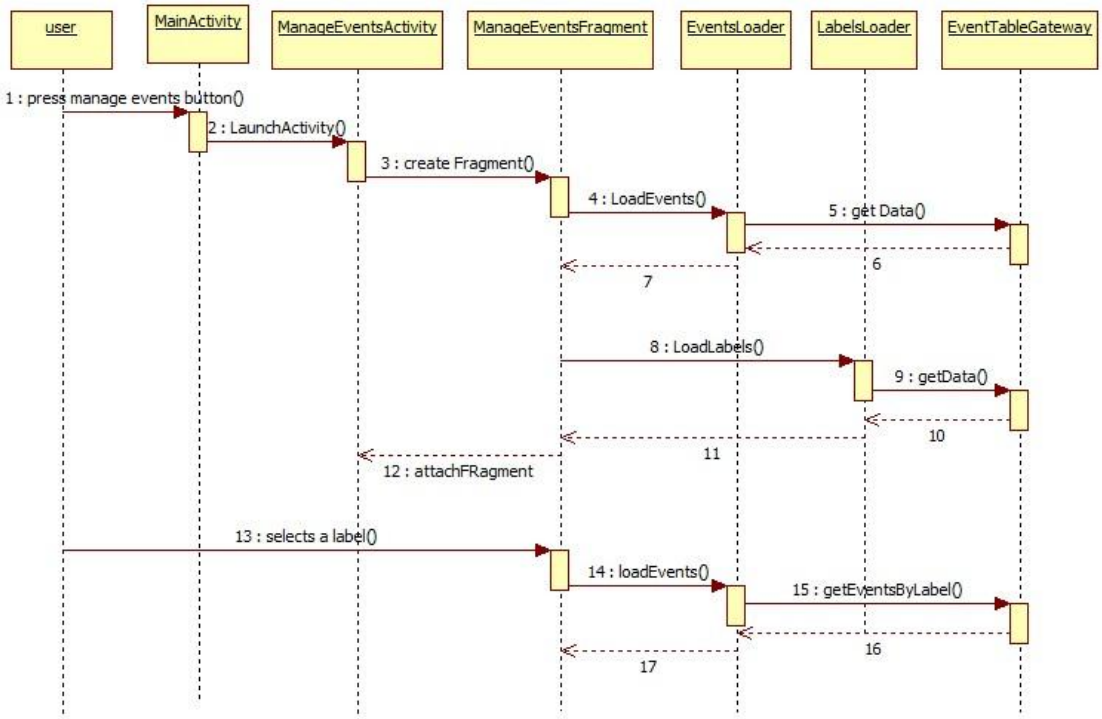


Figura 5.8 Diagrama de secventiere pentru vizualizarea activitatilor in functie de eticheta

Pentru satisfacerea cazului de utilizare de vizualizarea activitaților în funție de matricea Eisenhower au loc următoarele interacțiuni dintre componente:

Când utilizatorul alege opțiunea de vizualizare a matricei Eisenhower este creat un obiect de tip Intent care va lansa componenta EisenhowerMatrixActivity, tot în cadrul acestei componente de tip Intent se vor memora și criteriile în funcție de care să fie preluate din baza de date activitățile asupra cărora ar urma să fie afișate în matricea Eisenhower. Odată pronită componenta EisenhowerMatrixActivity aceasta va prelua datele legate de selecția activităților din obiectul de tip Intent primit și le va trimite unui obiect de tip EventsLoader pentru a le încarca din baza de date. Obiectul de tip EventsLoader va executa un apel catre un obiect de tip EventTableGateway pentru preluarea activităților utilizatorului în funcție de criteriile transmise, apelul va fi executat pe un fir de execuție separat de cel principal. Odată obținute, datele vor fi

transmise unui obiect de tip EventsAdapter responsabil de redarea grafica a activităților și care e atasat obiectului de tip ListView din interfața grafică.

Activitățile utilizatorului pot fi preluate din baza de date in functie de urmatoarele criterii

- Zi
- Luna
- Etichetă

Aceste criterii depind de poziția utilizatorului în aplicație în momentul în care acesta apelează opțiunea, criteriul este înregistrat în obiectul de tip Intent folosit pentru pornirea componentei. Interfața grafică furnizata de EisenhowerMatrixActivity cuprinde un grup de butoane de tip Radio button prin intermediul căreia se pot vizualiza activitățile cu un anumit grad de prioritate. Astfel exista 4 butoane în grupul de butoane, 1 pentru fiecare grad de prioritate. Când utilizatorul alege unul din butoanele respective,obiectul de tip OnCheckChangedListener atașat grupului de butoane înregistrează alegerea utilizatorului, caută în lista de activități pe cele ce au gradul de prioritate respectiv si actualuzează interfața grafică afisând activitățile găsite.

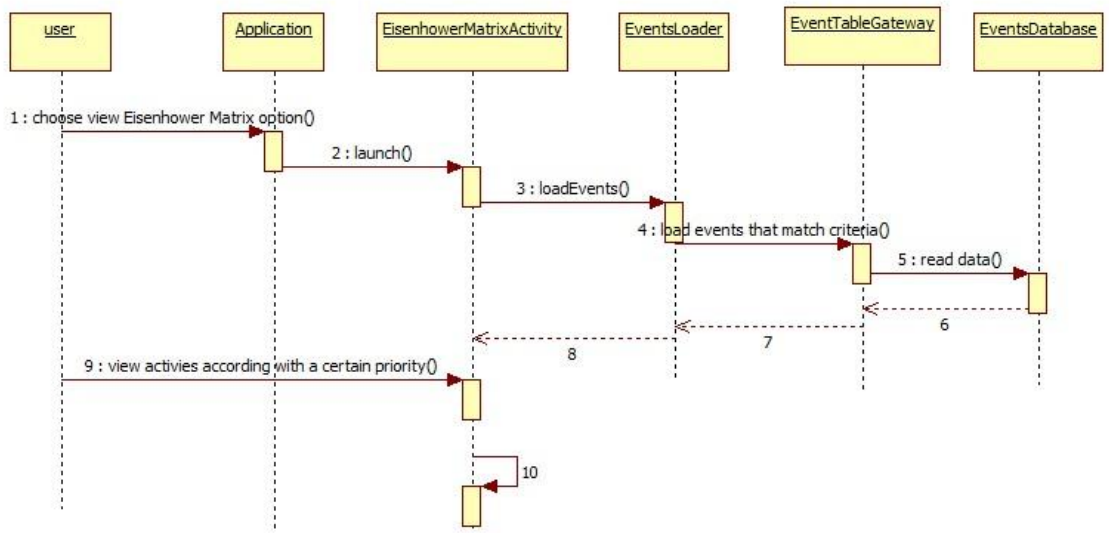


Figura 5.9 Diagrama de secventiere pentru vizualizarea activitatilor sub forma unei matrici eisenhower

Pentru satisfacerea cazului de utilizare de vizualizarea activitaților în funcție zi au loc următoarele interacțiuni dintre componente:

Componenta DayViewFragment este responsabilă de afișarea activităților planificate într-o anumită zi. Această componentă este atasată unei componente de tip Activity sau la ActionBar, în momentul atasării aceasta preia de la componeta de tip Activity părinte data calendaristica curentî sub forma unui obiect de tip java.util.calendar în funcție de care va căuta activitățile din baza de date. Datele vor fi transmise unui obiect de tip EventsLoader care va face un apel catre un obiect de tip EventTableGateway pentru a prelua datele din baza de date, apelul către obiectul de tip EventTableGateway se va executa pe un fir de execuție separat de cel principal iar datele obținute vor fi memorate într-un obiect de tip EventsAdapter care va fi atașat unei componente de tip ListView din interfața grafică reprezentând lista de activități programate pentru ziua respectivă.

are utilizatorul dorește să vizualizeze activitățile planificate pentru ziua anterioră sau ziua urmatoare obiectul de tip OnClickListener atașat butoanelor din interfața grafică va modifica data calendaristica în functie de alegerea utilizatorului și va apela la obiectul de tip EventsLoader pentru a obține activitătile din baza de date în funcție de noua dată calendaristică generată. După obținerea noilor date se va actualiza interfața grafică.

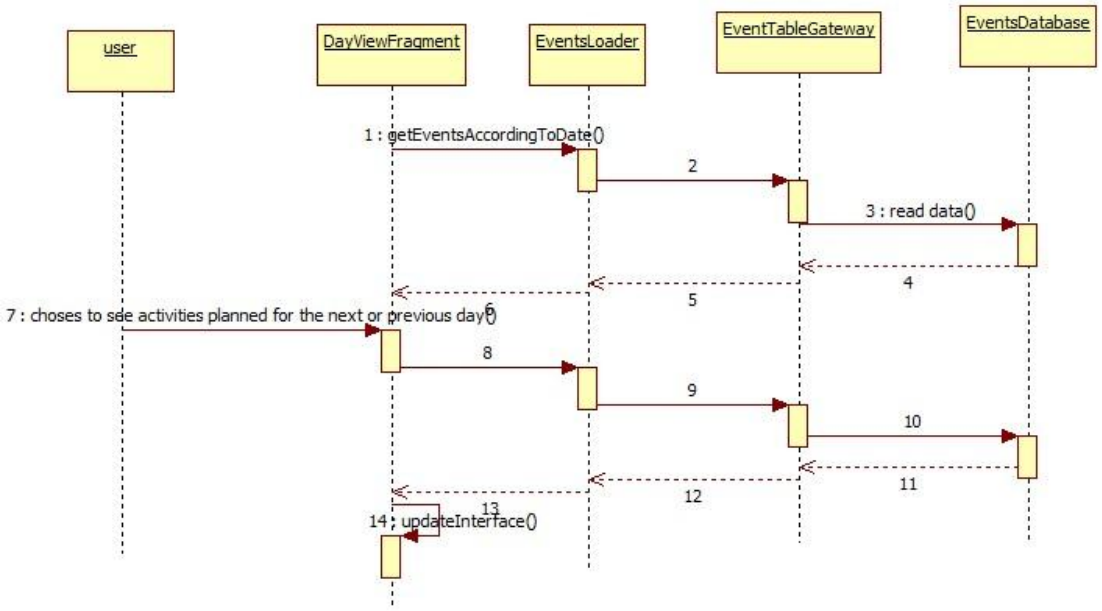
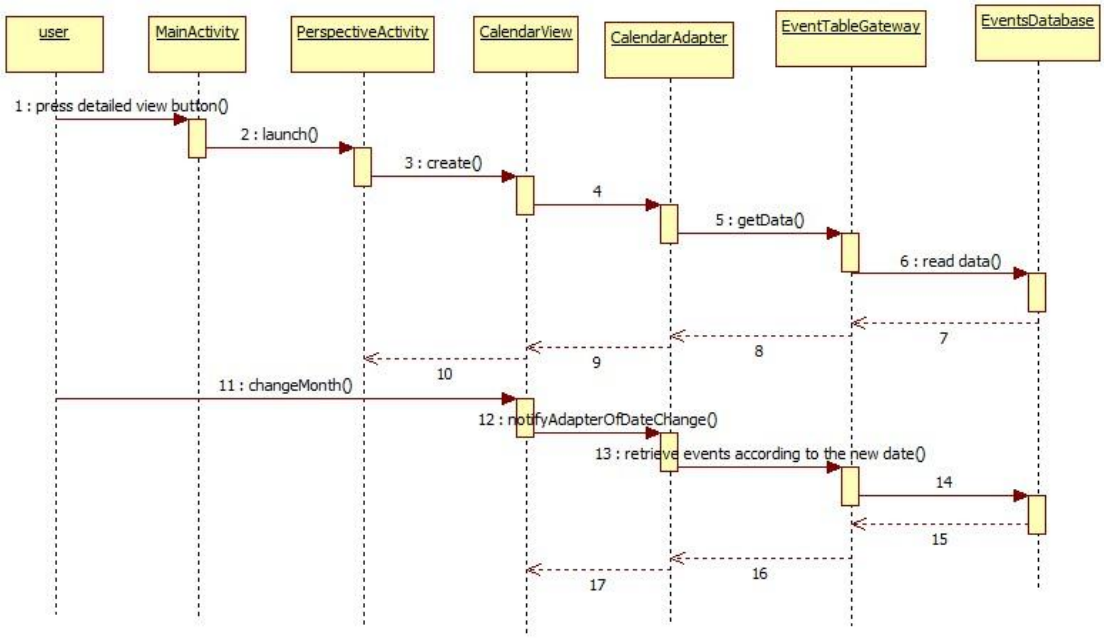


Figura 5.10 Diagrama de secventiere pentru vizualizarea activitatilor in functie de zi

Pentru satisfacerea cazului de utilizare de vizualizarea a zilelor din lună în care sunt planificate activități au loc următoarele interacțiuni dintre componente:

Pentru a vizualiza zilele din luna în care sunt planificate activități, utilizatorul apasă butonul cu textul *detailed view*. După apăsarea butonului obiectul de tip OnClickListener atașat acestuia va crea un obiect de tip Intent care va fi folosit pentru lansarea componentei PerspectiveActivity. Odată lansată, componenta PerspectiveActivity va crea un obiect de tip CalendarViewFragment care va fi atașat la ActionBar.Componenta de tip CalendarViewFragmentcare va afișa informația dorită în funcție de data calendaristică obținută de la sistem și transmisă acestei componente de către PerspectiveActivity. În procesul de inițializare a componentei CalendarViewFragment va fi creat un obiect de tip CalendarAdapter care va fi responsabil de preluarea informațiilor din baza de date și redarea acesteia în mod corespunzător. Obiectul de tip CalendarAdapter va apela la un obiect de tip EventTableGateway pentru a obține din baza de date activitățile utilizatorului în funcție de data calendaristică.
In cazul în care utilizatorul dorește să vizualizeze situația cu zilele în care are planificate activități pentru luna următoare sau cea anteriorara obiectul de tip OnClickListener atașat butoanelor din interfata grafică va schimba data calendaristică în funcție de alegerea utilizatorului și va determina obiectul de tip CalendarAdapter să își actualizeze conținutul în funcție de noua dată.



Figrua 5.11 Diagrama de secventiere pentru vizualizarea zilelor din luna in care sunt planificate activitati

Atfel au fost prezentate principalele componente ale aplicației precum și interacțiunea dintre acestea în scopul indeplinirii cazurilor de utilizare urmând ca în următorul capitol să fie prezentat un scenariu de test ce ar urma să ilustreze comportamentul aplicației

6 Testare şi validare

SDK-ul de Android pune la dispoziția dezvoltatorilor o serie de elemente cu ajutorul cărora dezoltatorii își pot testa aplicațiile.

Printre aceste elemente se numără

- Android Virtual Device manager, o unealtă cu ajutorul căreia se poate crea Android Virtual Device-uri (AVD) ce emulează dispozitive având anumite configurații hardware și care pot fi folosite ca și mediu de test pentru rularea aplicațiilor.

- Mecanism de logare cu ajutorul căruia se poate realiza înregistrarea diferitelor evenimente ce apar pe parcursul ciclului de viață a unei aplicații Android. Acest mecanism poate fi setat să transmită loguri la consola din Eclipse în momentul în care au loc schimburi de informații în urma interacțiunilor dintre utilizator și componentele aplicației sau dintre componentele aplicație.

Pentru testarea aplicației de față a fost folosit un AVD cu următoarea configurație:

- Sistem de operare Android 4 cu componenta de Google Api
- Modul de GPS
- Memorie RAM 512 mb
- Display LCD

Testarea componentei de notificare în funcție de poziția utilizatorului

Pentru testarea acestei componente au fost urmați următorii pași:

1) Este creat un obiect de test de tip MapObjective având coordonatele *46.768751;23.590526* corespunzător *Pieței Unirii din Cluj-Napoca*.(figura 6.1)

Figura 6.1 Locul indicat pe harta de către obiectul de test de tip MapObjective

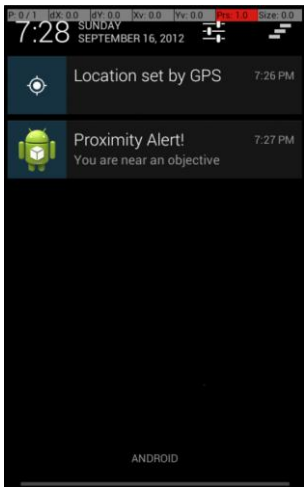


Figura 6.2 Notificarea transmisă de aplicatie

2) Este creat un un obiect de test de tip AgendaEvent care va fi planificat să aibe loc la data curentă și care va avea la rubrica *context* obiectul definit la pasul anterior.

3) Sunt furnizate AVD-ului coordonatele *46.766253; 23.583902*, reprezentând poziția unui utilizator situat pe *Strada Clinicilor din Cluj-Napoca* care inițial se află departe de punctul de interes.
4) Sunt furnizate AVD-ului coordonatele 47.678707; 23.590259 reprezentând poziția unui utilizator situat în apropierea punctului de interest. În momentul sesizării schimbării coordonatelor aplicația va lansa o notificare precum cea din figura 6.2

Modalitate de testarea

Pentru a testa o parte a funcționalităților oferite de aplicația de față s-a apelat la un scenariu de test în care au fost furnizate date de test i-ar logurile aplicației precum și rezultatele furnizate de interfața grafică au fost folosite pentru pentru a verifica validitatea operațiilor rezultatelor obținute.

Date de test

Pentru a testa aplicația sunt furnizate următoarele date de test inspirate de activitățile care au fost efectuate în vederea realizării proiectului de licentă precum și activitățile personale care s-au suprapus uneori.

Etichete:

- *"Licenta doc", o* etichetă cu care vor fi marcate activitățile care urmăresc realizarea documentației aferente lucrării de diplomă
- "*Licenta app*", o etichetă cu care vor fi marcate activitățile care urmăresc realizarea aplicației lucrării de diploma.

Activități:

- *„Capitolul X",* o activitate ce presupune scrierea capitolului X din documentația proiectului de diploma. Activitatea va fi planificată în ziua de Joi 30.08.2012, va avea gradul de prioritate "*Non-urgent but important*" și va fi etichetat cu "*Licenta doc*"
- „*Predare*", o activitate ce presupune predarea lucrării de licență la secretariatul facultății. Această activitate va fi planificată în ziua de Joi 20.09.2012, va avea gradul de prioritate "*urgent and important*" și va fi etichetă cu "*Licenta doc"*
- "*Componenta X",* o activitate ce presupune realizarea componentei x din proiectul de licență. Această activitate va fi planificată în ziua de Joi 10.09.2012, va avea gradul de prioritate "*Non-urgent but important*" și va fi etichetat cu "*Licenta app"*
- "*Film*", o activitate ce presupune vizionarea unui film. Aceasta activitate va fi planificată în ziua de Joi 10.09.2012 și va avea gradul de prioritate "*non-urgent si neimportant*"

Scenariu de test

În prima etapă se vor adauga etichetele, pentru această etapă se va folosi funcționalitatea de adaugare etichete oferită de componenta AddLabeActivity (figura 6.5)

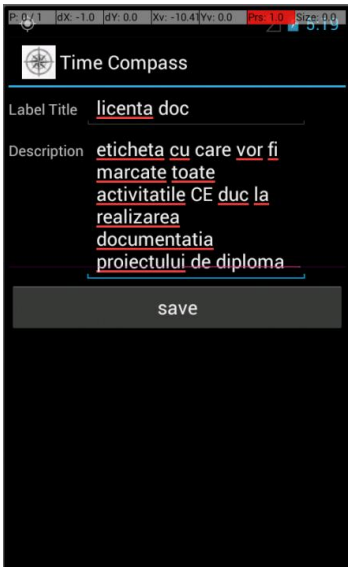


Figura 6.4 Interfata grafica corespunzatoare adaugarii unei noi etichete

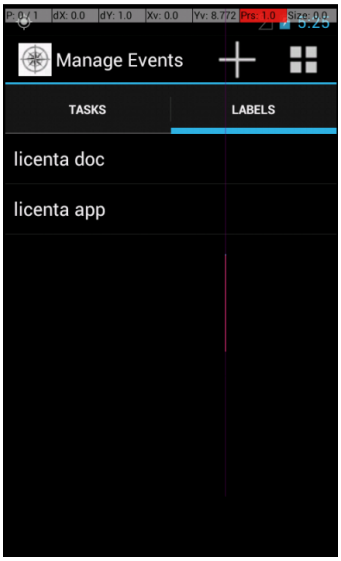


Figura 6.3 Vizibilitatea Etichetelor create

În momentul în care se apasă butonul *Save* în consolă va apărea următoarele mesaje

09-17 17:19:48.168: D/EventTableGateway(1496): Adding new label to database with name=licenta doc
09-17 17:19:49.868: D/EventTableGateway(1496): A new label has been added to the database:licenta doc eticheta cu care vor fi marcate toate activitatile CE duc la realizarea documentatia proiectului de diploma
09-17 17:19:49.868: D/Table Module Service> Label AsyncTask(1496): Label added with name=licenta doc
09-17 17:24:57.671: D/EventTableGateway(1496): Adding new label to database with name=licenta app
09-17 17:24:59.789: D/EventTableGateway(1496): A new label has been added to the database:licenta app eticheta cu care vor fi marcate activitatile CE duc LA realizarea proiectului de diploma
09-17 17:24:59.789: D/Table Module Service> Label AsyncTask(1496): Label added with name=licenta app
09-17 17:25:05.880: D/EventTableGateway(1496): Retrieving all Labels from database

Aceste loguri ilustrază fluxul de operații ce au loc în vederea înregistrării unei etichete

1. Crearea obiectului de tip *AgendaLabel*
2. Pasarea obiectului de tip *AgendaLabel* către componenta TableModuleService
3. Execuția componentei *AsyncTask* din *TableModuleService* repsonsabilă de memorarea etichetei
4. Execuția de către *AsyncTask* pe un fir de execuție separate a unui apel către *EventTableGateway* pentru persistarea datelor în baza de date.

După adăugarea celor 2 etichete acestea vor deveni disponibile în aplicație, un loc în care pot fi văzute este Tab-ul de *Labels* din interfața *ManageEvents (figura 6.5)*. Procesul de încărcare a etichetelor poate fi văzut în logurile de mai jos:

09-17 17:25:05.918: D/EventTableGateway(1496): Retrieving all Labels from database

09-17 17:25:08.078: D/EventTableGateway(1496): Retrieved 2 labels
09-17 17:25:08.088: D/Labels Loader(1496): Loaded all from DB

Aceste loguri ilustrează cum un obiect de tip LabelsLoader execută pe un fir de execuție separat un apel de citire a tuturor etichetelor din baza de date.

Tabul de *Tasks* oferă funcționalitatea de vizualizare a tutuora activităților care au o anumită etichetă astfel în momentul în care se alege spre exemplu eticheta "*Licenta doc"* în consolă va apărea următorul mesaj:

09-17 19:44:45.588: D/EventTableGateway(1496): Retrieving all events that have been labeled with label=licenta doc
09-17 19:44:45.598: D/EventTableGateway(1496): Retrieved 2 events
09-17 19:44:45.618: D/Event Loader(1496): Loaded 2 events with the label licenta doc

Iar interfața grafică va arăta conform figurii 6.7

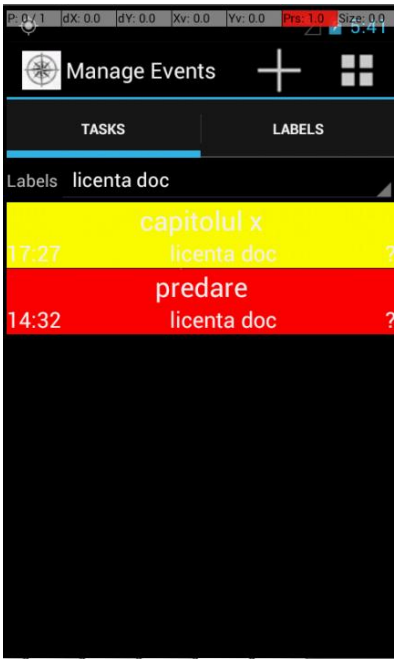


Figura 6.6 Afisarea Activitatilor in functie de eticheta

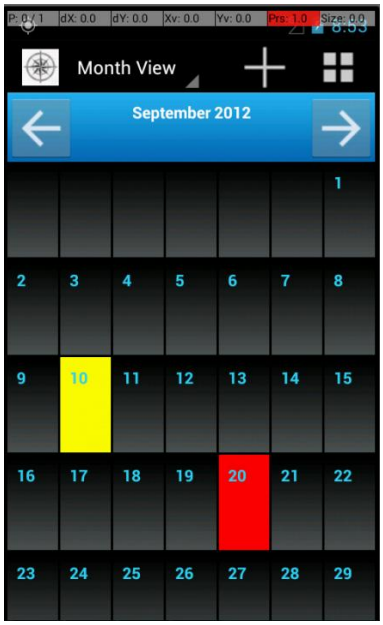


Figura 6.5 Vizualizarea activitatilor in functie de luna

Aceste loguri ilustrează cum cele 2 evenimente etichetate cu *"Licenta doc"* sunt încărcate din baza de date de către componenta *EventTableGateway*, operația fiind executat pe un fir de execuție separat de către *AgendaEventLoader*.

O altă etapă din scenariul de test constă în activarea modului de vizualizare a activităților planificate într-o anumită lună oferită de componenta *MonthViewFragment*

În momentul activării acestui mod de vizualizare în consolă va apărea urmatorul mesaj

09-17 19:49:55.518: D/PerspectiveActivity(1496): MonthViewFragment is focused
09-17 19:49:56.038: D/EventTableGateway(1496): Retrieving AgendaEvents planned to take place in September 12
09-17 19:49:56.168: D/EventTableGateway(1496): Retrieved 3 events

Iar interfața grafică va arăta conform figurii 6.6

Logurile de mai sus arată cum elementul de *EventTableGateway* extrage din baza de date cele 3 activități planificate să aibe loc în luna septembrie 2012

Iar când se da click pe componenta din grid view corespunzatoare zilei de 10 septembrie 2012 se va lansa componenta de tip *DayViewFragment* responsabilă de afisarea activitatilor planificate în ziua respectivă consola afisând

09-17 20:00:44.488: D/DayViewFragment(1496): Intantinating an Event Loader to load events which will take place on Monday
09-17 20:00:44.488: D/DayViewFragment(1496): (10/09/12)
09-17 20:00:44.520: D/EventTableGateway(1496): Retrieving AgendaEvents planned to take place on (10/09/12)
09-17 20:00:44.538: D/EventTableGateway(1496): Retrieved 2 events
09-17 20:00:44.758: D/Event Loader(1496): Loaded 2 events planned to take place on Monday
09-17 20:00:44.758: D/Event Loader(1496): (10/09/12)
09-17 20:00:44.771: D/DayViewFragment(1496): Retrieved 2 events from Event Loader

Corespunzător operației de inițializarea a obiectului de tip *AgendaEventLoader* care execută pe un fir de execuție separat un apel către componenta *EventTableGaterway* pentru a preluat din baza de date activitățile planificate pe data 10.09.2012. Iar în urma operațiilor executate se obțin 2 evenimente identice cu cele introduse inițial în baza de date.

Figura 6.7 Returnarea activitatilor in functie de zi

7 Manual de instalare și utilizare

În cele ce urmează vor fi prezentate principalele resurse necesare pentru rularea aplicației precum și pașii pe care utilizatorul ar trebui să îi facă pentru a folosi cum trebuie aplicația.

7.1 Manual de instalare

Cerințe System:

Telefon ce are instalat Android 4 cu componentă de Google API.

Aplicația necesită urmatoarele permisiuni:

- Permisiune de folosire a internetului
- Permisiune de folosite a GPS-ului.

Toate aceste permisiuni sunt menționate în fișierul *manifest.xml* iar la instalarea aplicației utilizatorul va fi întrebat dacă acceptă sau nu ca aplicația să primească permisiunile cerute, în cazul în care acesta refuză aplicația nu se va mai instala.

Permisiunile descrise în manifest.xml:

```
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
    <uses-permission android:name="android.permission.VIBRATE"/>
```

Instalarea aplicației:

Există 3 modalități de instalare a aplicației, fiecare aplicându-se în circumstanțe diferite.

1. Descărcarea aplicației de pe Android Market.

Acestă metodă se aplică în cazul în care aplicația a fost publicată pe Android Market

Pași de urmat de către utilizator:

a) Utilizatorul intră în aplicația de android market(această aplicație vine preinstalata pe orice dispozitiv cu Android)
b) Este selectată aplicația dorită
c) Se apasă butonul install
d) Este afișată o notificare legată de permisiunile pe care le solicită aplicația pentru a rula și care trebuie aprobate de utilizator pentru ca aplicația să se instaleze, în caz contrar se renunță la instalare.

2. Instalarea aplicatiei prin folosind Google Play [10]

Acestă metodă se aplică în cazul în care:

- Aplicația este publicată pe Google Play
- utilizatorul are cont de Google și un dispozitiv compatibil înregistrat și asociat contului respectiv

Pași de urmat de către utilizator

a) Utilizatorul navighează pe site (fie de pe PC fie de pe dispozitiv)
b) Găsește aplicația dorită
c) Alege să o instaleze

3. Instalarea prin cablu USB

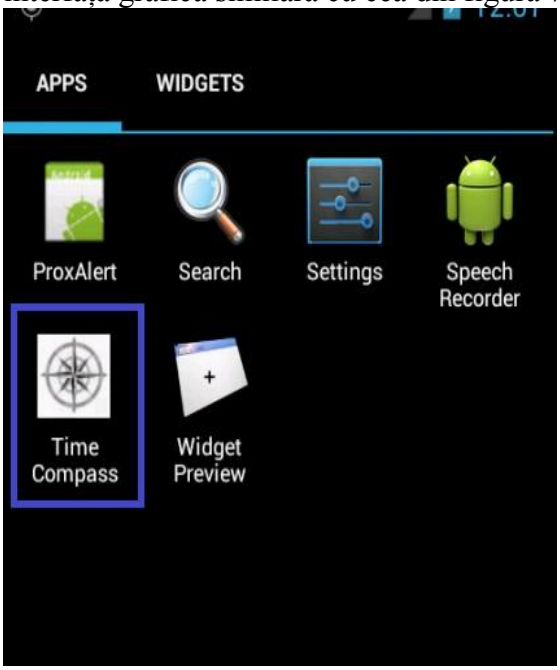
Acest pas se aplică în cazul în care:

- utilizatorul are fișierul *.apk* al aplicației pe PC
- utilizatorul are instalat *Android SDK*
- utilizatorul are instalat driverul de USD
- utilizatorul are următoarele setări de dispozitiv:
  - *Settings-> Application Settings* unde se bifează "*Unknown Sources."*
  - *Settings->*"*SD Card*" and "*Phone Storage*," unde se activează optiunea "*Disable Use for USB Storage*"

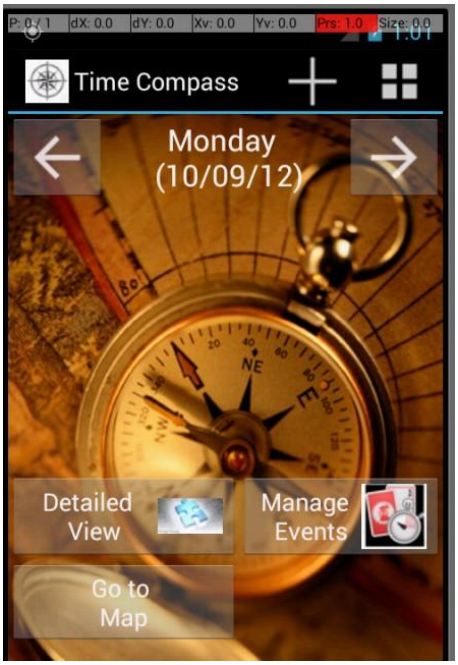
Pași de urmat de către utilizator:

a) Deschide o linie de comandă
b) tastează <1>/<2>*.apk* unde <1> reprezintă calea absolută de pe disk a fișierului *.apk* iar <2> reprezinta numele aplicației, în acest caz *TimeCompass*

În cazul în care procesul de instalare s-a derulat cu success ar trebui să apară o icoană a aplicației în ecranul principal similar ca în figura 7.2 iar când aplicația va fi pornită va aparea o interfață grafică similară cu cea din figura 7.1



Figură 7.2 Icoana aplicației în ecranul principal



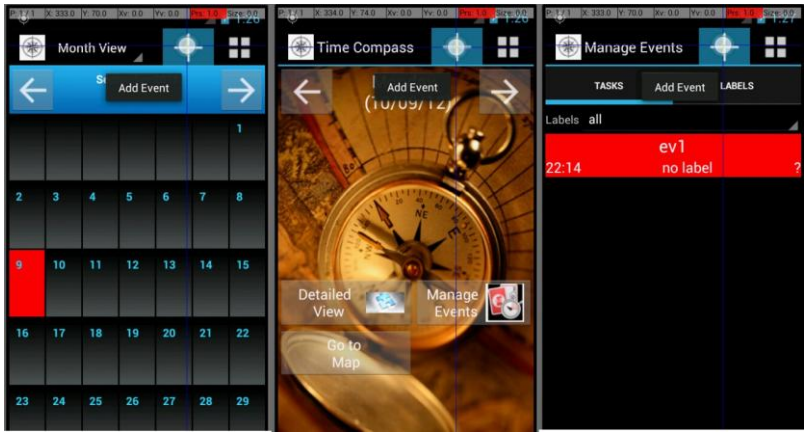
Figură 7.1 Fereastra principală a aplicației

7.2 Manual de utilizare

Definirea unei noi activități

Pentru a defini o nouă activitate utilizatorul trebuie să aleagă opțiunea de *"add event"* pusă disponibilă în bara de acțiuni. Această opțiune este diponibilă in

- Modul de vizualizare principal
- Modul de vizualizare generat in urma apasarii butonului *Detailed View*
- Modul de vizualizare generat in urma apasarii butonului *Manage Events* tabul *Tasks*



Figură 7.3 Disponibilitatea optiunii Add Event

În momentul alegerii opțiunii de către utilizator aplicația va genera o interfață grafică similară cu cea din figura 7.4 în care utilizatorul poate defini o activitate nouă activitate.

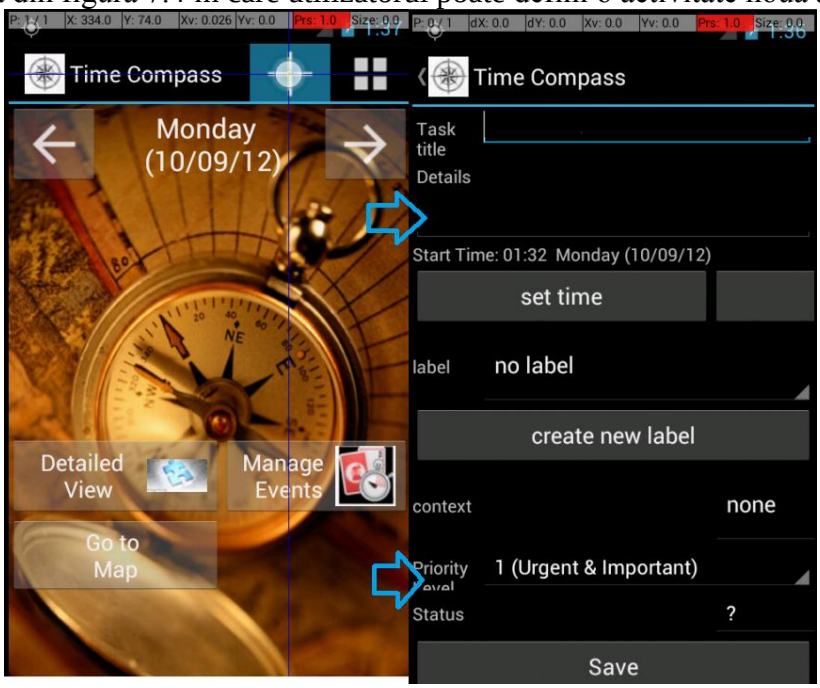
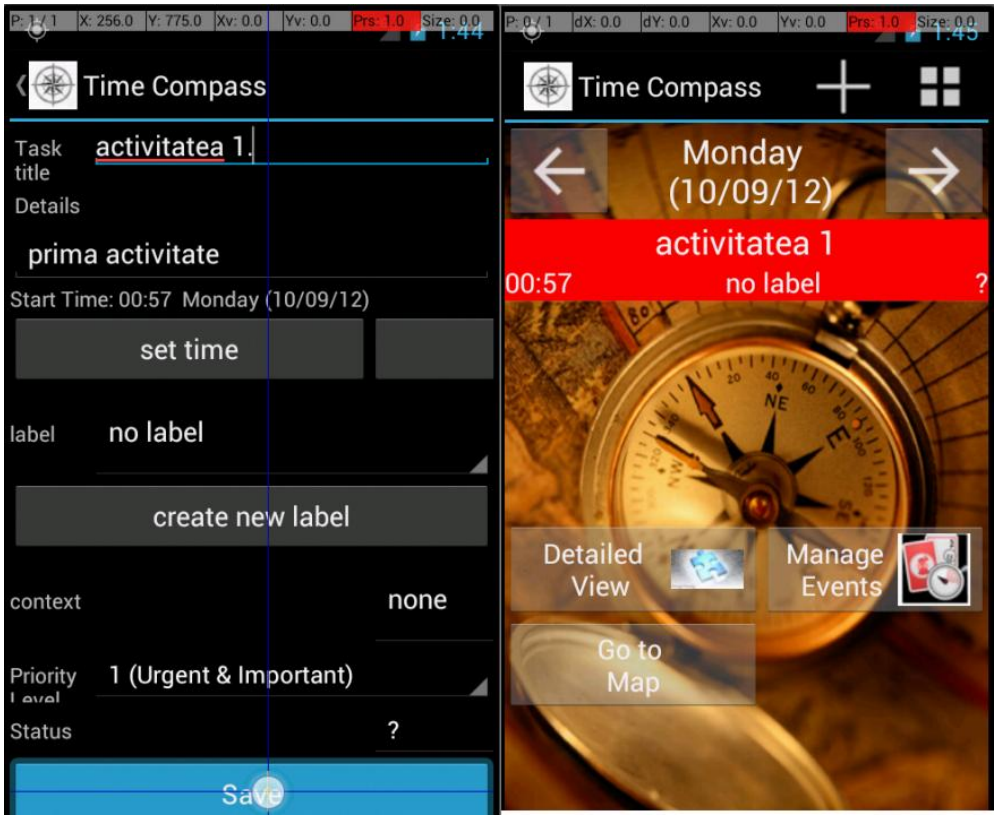


Figura 7.4 Tranziția către interfața grafică pentru adaugarea unei noi activități

După ce utilizatorul introduce datele trebuie apăsat butonul save pentru a persista datele introduse, În caz de succes activitatea ar trebui sa fie vizibilă în modul de vizualizare a activitatilor in funcție de zi la data calendaristică mentionată

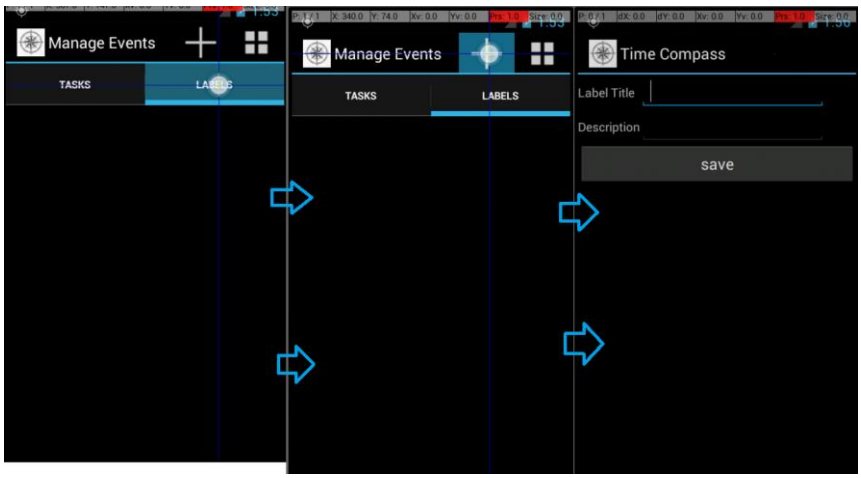


Figură 7.5 Adaugarea cu succes a unei noi activitati

Definirea unei noi etichete

Pentru a defini o nouă etichetă utilizatorul trebuie să aleagă opțiunea de *"add new label"* pusă disponibilă în bara de acțiuni în cadrul modului de vizualizare generat în urma apăsării butonului *Manage Events* tabul *Tasks* (a se vedea figura 7.6)sau apăsând butonul de *create new label* din modul de vizualizare corespunzător definirii unei noi activități.

După introducerea datelor trebuie apăsat butonul *save* pentru a persista datele. În cazul în care datele au faost salvate cu success noua etichetă va deveni disponibilă (a se consulta figura 7.7)



Figură 7.6 Adaugarea unei noi etichete

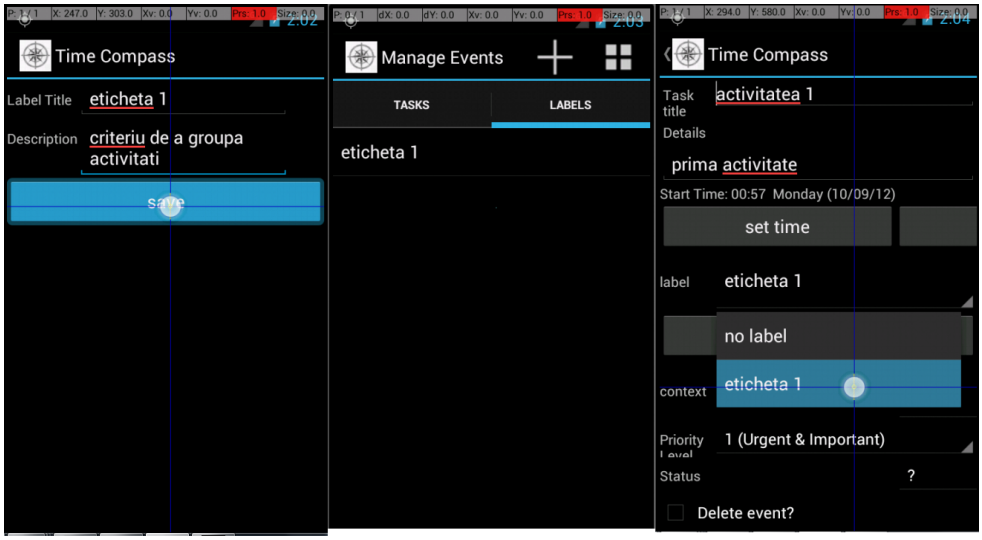


Figura 7.7 Salvarea unei noi etichete

Editarea unei activitati deja existente

Pentru a edita o activitate deja existentă se alege activitatea dorită din lista de activitati prin click și este lansat un o interfață grafică similară cu cea din figura 7.8 unde se pot actualiza datele legate de activitate. Pentru a persista schimbările făcute trebuie apăsat butonul *update.* Similar se procedează și în cazul actualizării unei etichete deja existente

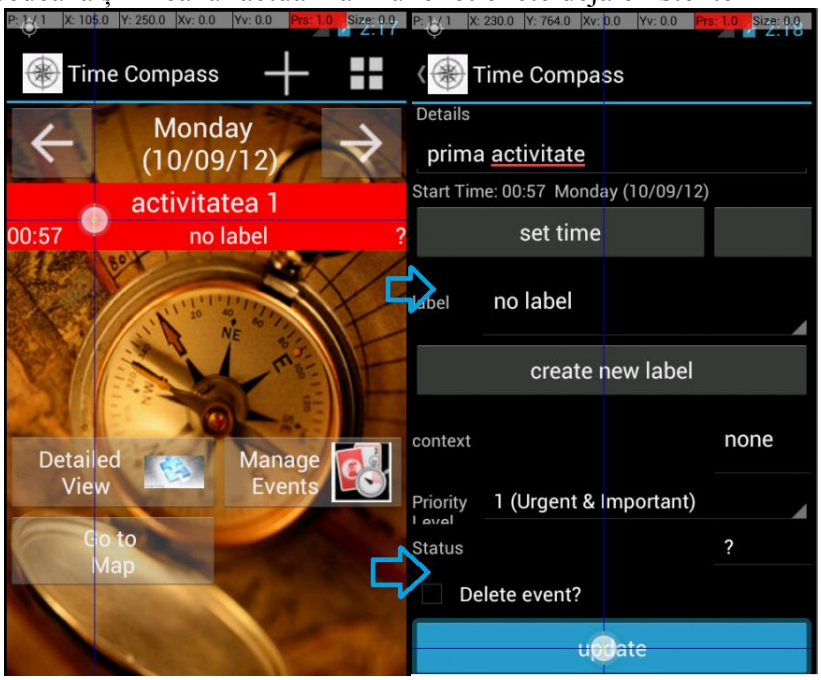


***Figura 7.8* Actualizarea unei activitati deja existente**

Vizualizarea activităților în funcție de zi

Acest mod de vizualizare constă într-o listă de activități planificate pentru o anumită dată, atribuindu-le o culoare în funcție de gradul de prioritate asociat (roșu pentru *urgent & important* galben pentru *neurgent dar important*, portocaliu pentru *urgent dar neimportant* si verde pentru *neurgent & neimportant*). În partea superioară a listei situându-se un text ce reprezintă data calendaristică în functie de care sunt afișate activitățile precum și 2 butoane cu ajutorul cărora se poate schimba data respectivă.

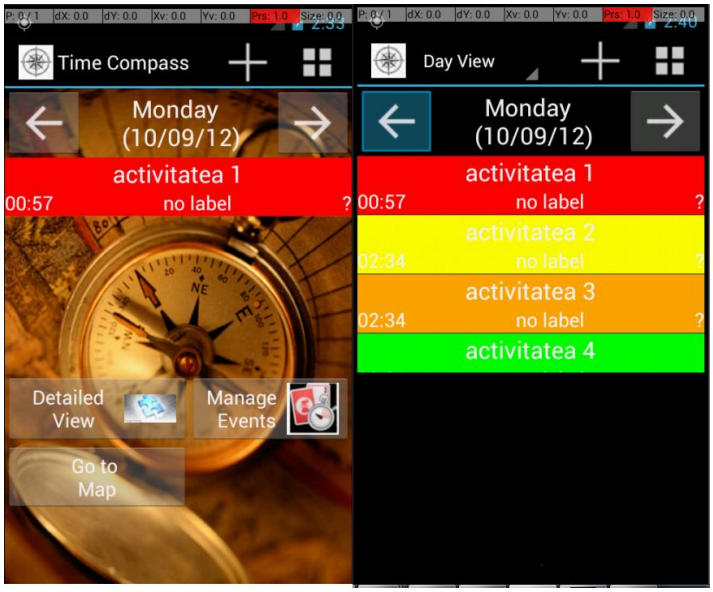


Figura 7.9 Vizualizarea activitatilor in functie de zi

Vizualizarea activităților în funcție de lună

Pentru a vizualiza activitățile planificate într-o anumită lună trebuie apăsat butonul *Detailed View*. Prin apăsarea butonului respectiv se generează o interfată grafică similară cu cea din figura 7.10 în care sunt evidențiate zilele în care sunt planificate activități. Zile în care sunt planificate activități sunt evidențiate cu o culoarea aleasă în funcție de care de activitatea cu cea mai mare prioritate din ziua respectivă. Luna calendaristică pentru care se vizualizează zilele cu activități poate fi schimbată cu ajutorul butoanelor din dreptul textului cu numele lunii.

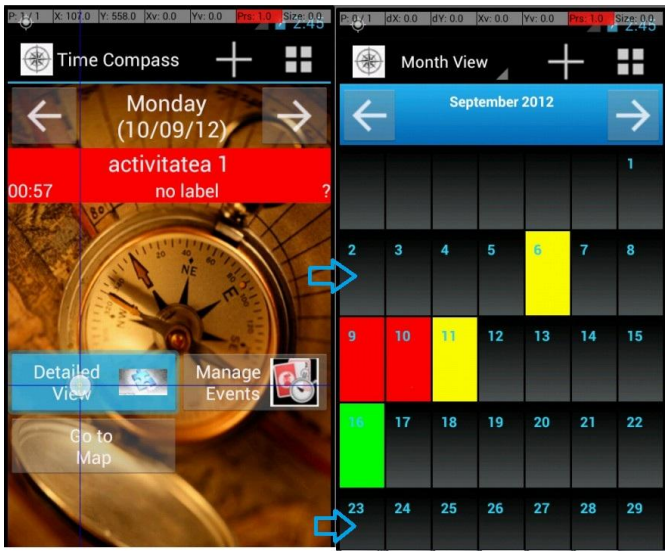


Figura 7.10 Vizualizarea zilelor din luna in care sunt planificate activitati

Vizualizarea activitatilor in functie de etichetă

Pentru a accesa acest mod de vizualizare trebuie apasat butonul *"Manage Events"* și apoi din interfața creată trebuie selectat tab-ul *Tasks.* Din interfața grafică corespunzatoare tabului Tasks se alege o eticehta din colectia de eticehte iar aplicatia va afisa toate activitatile asociate cu acea eticheta

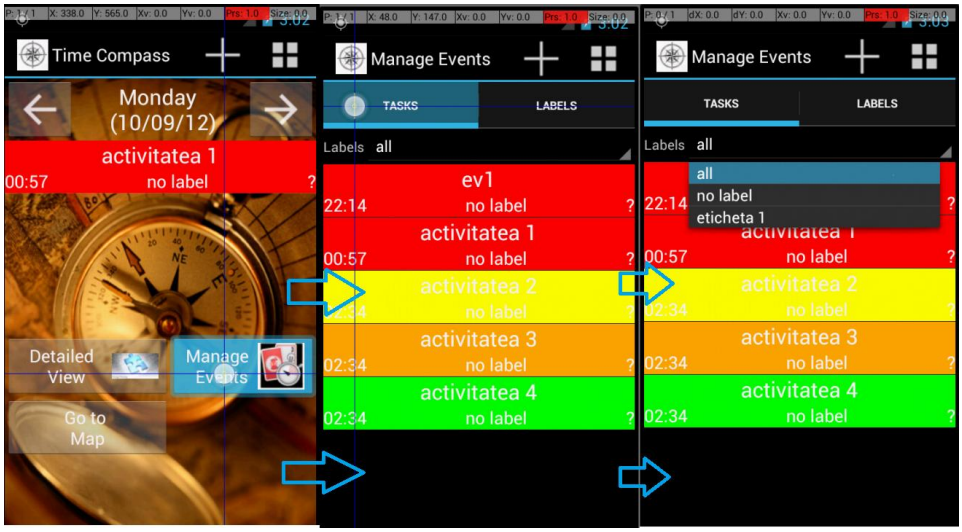


Figura 7.11 Vizualizarea activitatilor in functie de eticheta

Vizualizarea matricei Eisenhower

Pentru vizualizarea matricei Eisenhower trebuie aleasă opțiunea *View Eisenhower matrix* din meniul de opțiuni pus la dispoziție de ActionBar.

Aceatstă opțiune este disponibilă în mai multe locuri în cadrul aplicației (a se consulta figura 7.12) și are rezultate diferite în funcție de locul în care este apelată astfel că:

- Dacă este apelată în modul de vizualizare a activităților în funcție de lună atunci matricea Eisenhower se va aplica asupra activităților planificate în luna respectivă
- Dacă este apelată în modul de vizualizare a activităților în funcție de zi atunci matricea Eisenhower se va aplica asupra activităților din ziua respectivă
- Dacă este apelată în modul de vizualizare a activităților în funcție de etichetă atunci se va aplica asupra activităților asociate etichetei alese de utilizator

Odată aleasă opțiunea respectivă se genera o interfață grafică similară cu cea din figura 7.13 în care se pot vedea activitățile în funcție de gradul lor de importanță.

Figura 7.12 Disponibilitatea optiunii View Eisenhower Matrix in cadrul aplicatiei

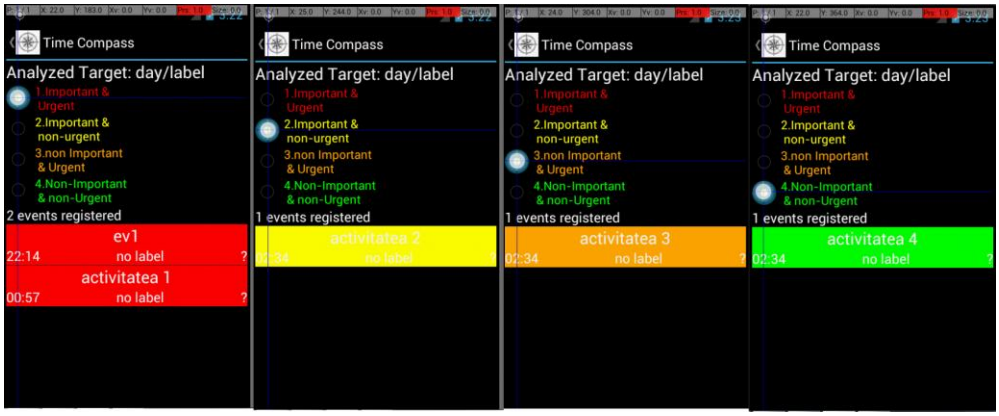


Figura 7.13 Interfata grafica corespunzatoare matricei Eisenhower

Adăugarea unui nou obiectiv pe hartă

Pentru a adăuga un nou obiectiv pe hartă care apoi poate fi folosit ca și context în definirea unei activități trebuie aleasă opțiunea *add new objective* din meniul de opțiuni pus la dispoziție de de Action Bar în cadrul modului de vizualizare generat în urma apăsării butonului *Go to Map* (a se vedea figura 7.14). În interfața grafică corespunzătoare adaugarii unui nou obiectiv sunt introduse datele ce definesc nou obiectiv, este aleasă poziția noului obiectiv și se apasă butonul save pentru pentru persistarea datelor (a se vedea figura 7.15).

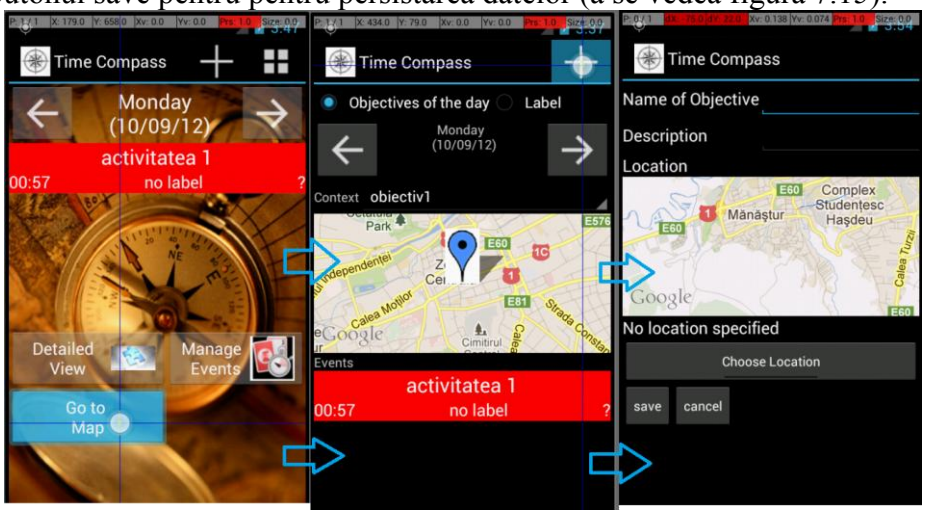


Figura 7.14 Adaugarea unui nou obiectiv

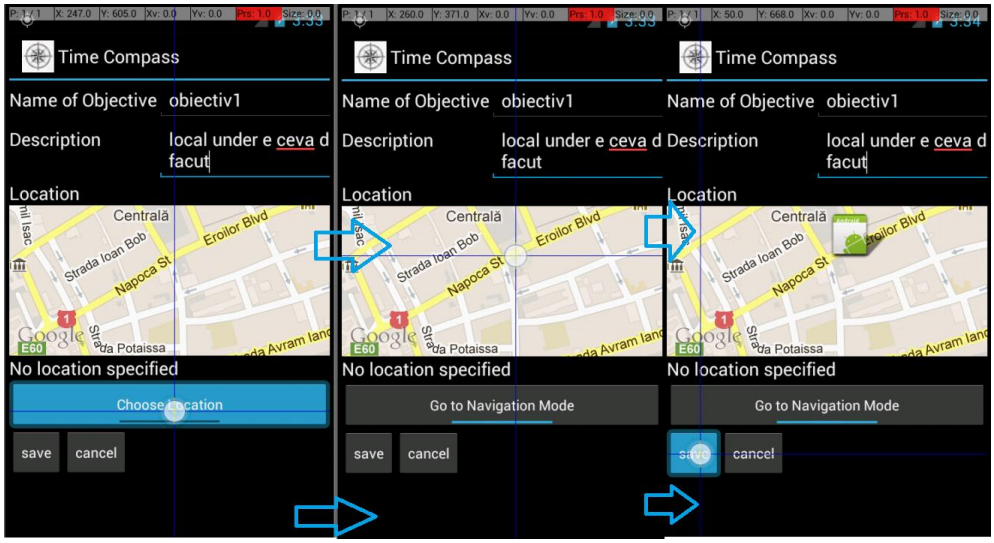


Figura 7.15 **Definirea unui nou obiectiv**

După cum s-a putut observa aplicația implementează cu success principiile metodologiei "*Getting Things Done"* prin modurile de vizualizare ce oferă liste de activități precum și elemente ce țin de "*First Things First"* cum ar fi matricea de priorități. Toate acestea fiind oferite într-o manieră ușor de folosit pentru utilizator

8 Concluzii

8.1 Realizări

Aplicația obținută reprezintă o unealtă ce vine în sprijinul utilizatorului când vine vorba de planificarea activităților și încadrarea acestora în timp. Rolul aplicației rezumându-se la a înregistrara o parte din detaliile legate de activitățile utilizatorului și de a le oferi acestuia într-o manieră utilă atunci când acesta are nevoie de ele. Astfel utilizatorul nu își mai încarcă memoria cu prea multe date legate de planificarea sarcinilor și se poate concentra mai mult pe indeplinirea acestora.

În implementarea aplicației se pot regăsi o serie de elemente de time management promovate în metodologiile "*Getting Things Done"* [1] și "*First Things First"*[2] și anume:

- **Un sistem de perspective**: Aplicația oferă o gamă variată de perspective (lună calendaristică, zi, etichetă) pe baza cărora utilizatorul obține o imagine de ansamblu asupra activităților pe care acesta dorește să le întreprindă ce îl poate ajuta în luarea deciziilor referitoare la ce acțiuni necesită atenșia lui la un moment dat și ce acțiuni pot fi amânate.
- **Corelarea activităților cu un anumit context**. Aplicația îi permite utilizatorului să definească un context în funcție de care depinde îndeplinirea activităților
- **Mecanism de grupare a activităților în categorii**. Aplicația permite gruparea diverselor activități în funcție de etichete. O etichetă poate fi văzută ca o categorie definită de utilizator și în care acesta poate încadra activitățile sale în funcție de propriile criterii. Acest aspect se dovește util în cazul în care activitățile sunt corelate între ele din punctul de vedere al utilizatorului.
- **Sistem de prioritizare a activităților**. În cadrul aplicației se poate găsi implementată matricea de priorități[2] (figura 3.1), prin intermediul căreia utilizatorul își poate grupa activitățile, fiecare activitate putând fi încadrată în unul din cele 4 cadrane prin atribuirea unui grad de prioritate și un mod de vizualizare ce permite filtrare activităților pe baza gradului de importanță.
- **Sistem de notificare**. Aplicația are implementată un sistem responsabil de notificarea utilizatorului în cazul în care acesta se află în apropierea unui punct de interes relevant pentru îndeplinirea unei anumite activități.

Există o integrare a serviciului GoogleMaps în cadrul implementării aplicației, acest serviciu este folosit în definirea contextelor de care depind îndeplinirea activităților. Astfel contextul unei activități poate fi vazut ca un punct de interes pe hartă.

Stocarea datelor se face local pe dispozitiv scutindu-l astfel pe utilizator de realizarea vreunui cont pentru stoarea datelor.

O parte din componentele aplicației au fost realizate folosindu-se elemente furnizate de Android 4 ce au permis un design modular a interfeței grafice (prin folosirea componentelor de tip Fragment și ActionBar) și un mecanism de execuție a operațiilor pe fire de execuție diferite fără a necesita mecanisme suplimentare de sincronizare oferind astfel un plus de performanță aplicației (prin folosirea componentelor de tip Loader). De asemenea, deoarece aplicația a fost

proiecatata pentru Android 4 este posibilă adaptarea ei pentru a rula pe tablet cu Android 4, această operatie necesitând puține modificări.

8.2 Comparație cu alte produse

Comparativ cu Google Calendar[9], Time Compass se aseamănă prin faptul că oferă moduri de vizualizare comune (vizualizarea activităților în funcție de zi și în funcție de lună) însă aplicația de fata oferă în plus și un set de perspective de vizualizare a conținutlui ce îl pot ajuta pe utilizator în luarea deciziilor. De asemenea aplicația îi permite utilizatorului să definească mai mult detalii despre activitäti spre deosebire de Google Calendar. Un alt element pe care îl are în plus aplicația de fața este componenta de Google Maps cu ajutorul căreia pot fi definite puncte de interes ce pot fi apoi asociate ca și context diverselor activități.

Dezavantajul pe care îl are TimeCompass față de Google Maps este că datele sunt disponibile doar pe dispozitivul clientului pe când Google Calendar le face disponibile prin intermediul unui browser web și un cont de Google.

Comparativ cu MyLifeOrganized[8], TimeCompass prezintă o interfață grafică mai prietenoasă fiind intuitivă și oferă perspective de vizualizare precum vizualizarea activităților în funcție de luna și în funcție de zi. MyLifeOragnized are în plus o funcție de căutare a activităților ce poate fi apelată din meniul de opțiuni indifferent de poziția utilizatorului în aplicație. De asemenea MyLifeOragnized permite definirea unor entități tip proiect ce pot avea în componența lor diverse activități, un rezultat similar îl prezintă și TimeCompass prin sistemul de etichete însa nu la fel de elaborat. În rest majoritatea functiilor oferite de cele 2 aplicații sunt similare

Comparativ cu ToDoMatrix[3] TimeCompass prezintă în plus componenta de Google Maps, folosită pentru definirea contextelor activităților și sistemul de notificare bazat pe poziția utilizatorului în rest majoritatea functionalităților acestei aplicații se regăsesc și în ToDoMatrix alături de altele.

8.3 Dezvoltari ulterioare

Aplicația de față poate fi extinsă cu un sistem de recomandare care ar interpreta textul introdus de utilizator în momentul când acesta definește titlul sau descrierea unei activități și pe baza unor cuvinte cheie și expresii aplicația poate returna o serie de locuri pe harta ce se pot dovedi un context relevant pentru îndeplinirea activității. De exemplu, în cazul în care utilizatorul își definește o activitate ce conține cuvintele "*revizie tehnică la mașină* "fie în titlu fie în descriere aplicația ar putea returna o lista cu service-uri din apropierea utilizatorului și din aceasta lista utilizatorul își poate alege un service context activității respective.

De asemenea aplicația poate fi extinsă prin definirea unui element de tip *proiect* în care pot fi grupate activitățile și care să prezinte un indicator de timp și stadiu pe baza datelor furnizate de activitățile componente. De asemenea se poate implementa o funcționalitate de gantt chart care să afișeze desfășurarea în timp a proiectelor.

Proiectul mai poate fi extins prin definirea unei componente server și mai multe componente client pentru diverse platforme alături de un mecanism de sincronizare a conținutului între acestea. Similar sistemului oferit de Google Play[x]

De asemenea se mai poate introduce o funcționalitate de căutare cu ajutorul căreia se pot caută diverse elemente (activități etichete, context) în funcție de diferite criterii.

9 Bibliografie

[1] Allen David, *Getting Things Done: The Art of Stress-Free Productivity*, USA: Penguin Books, 2002.

[2] Stephen R. Covey, A. Roger Merrill, Rebecca R. Merrill, *First Things First*, Detroid, USA: Free Press, 1994.

[3] *Task Management Best Practices White Paper*, REXwireless Inc**.** [Online] Available at: http://www.rexwireless.com/bestpractices/todomatrix_best_practices1f.pdf.

[4] Laurence Goasduff , Christy Pettey , *Gartner Says Worldwide Smartphone Sales Soared in Fourth Quarter of 2011 With 47 Percent*, Gartner Inc Egham, UK, Februarie 15, 2012 [Online] Available : http://www.gartner.com/it/page.jsp?id=1924314.

[5] *Android Developer*, [Online] Available :http://developer.android.com/about/index.html.

[6] Gargenta Marko, *Learning Android*, USA:O'REILLY, 2011.

[7] Komatineni Satya, Maclean Dave, Hashimi Sayed, *Pro Android 3*, New York USA:Apress, 2011.

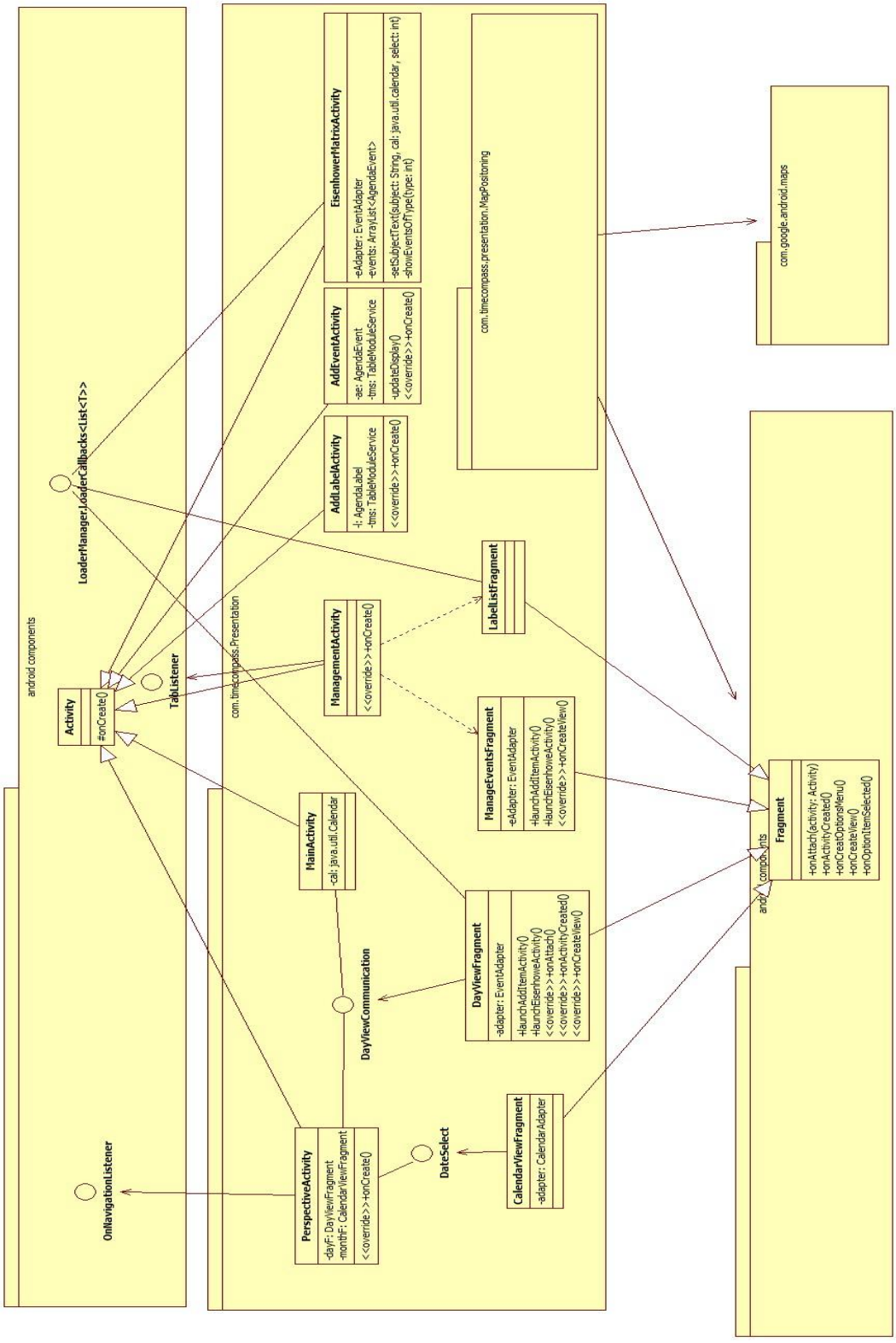
[8] *MyLifeOrganized* [Online] Available: http://www.mylifeorganized.net/.

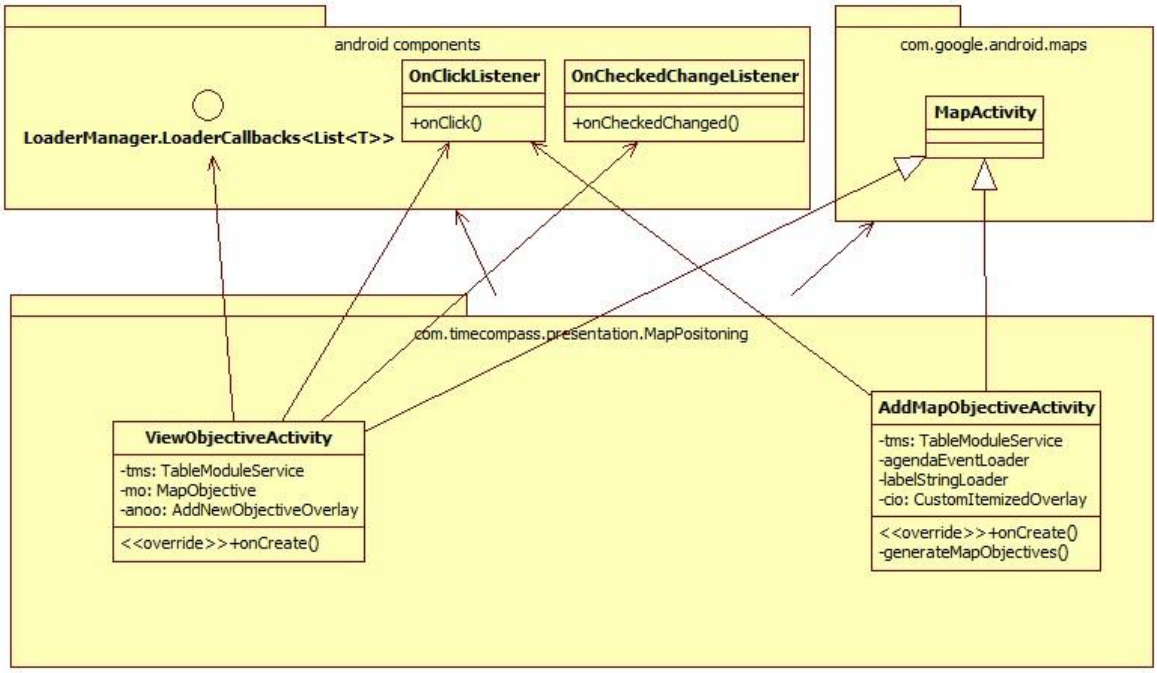
[9] About *Google Calendar*, Google Inc, [Online] Available: http://support.google.com/calendar/bin/answer.py?hl=en&answer=2465776.

[10] *About SQlite* Hipp, Wyrick & Company, Inc. North Carolina, USA **[**Online] Available: http://www.sqlite.org/about.html

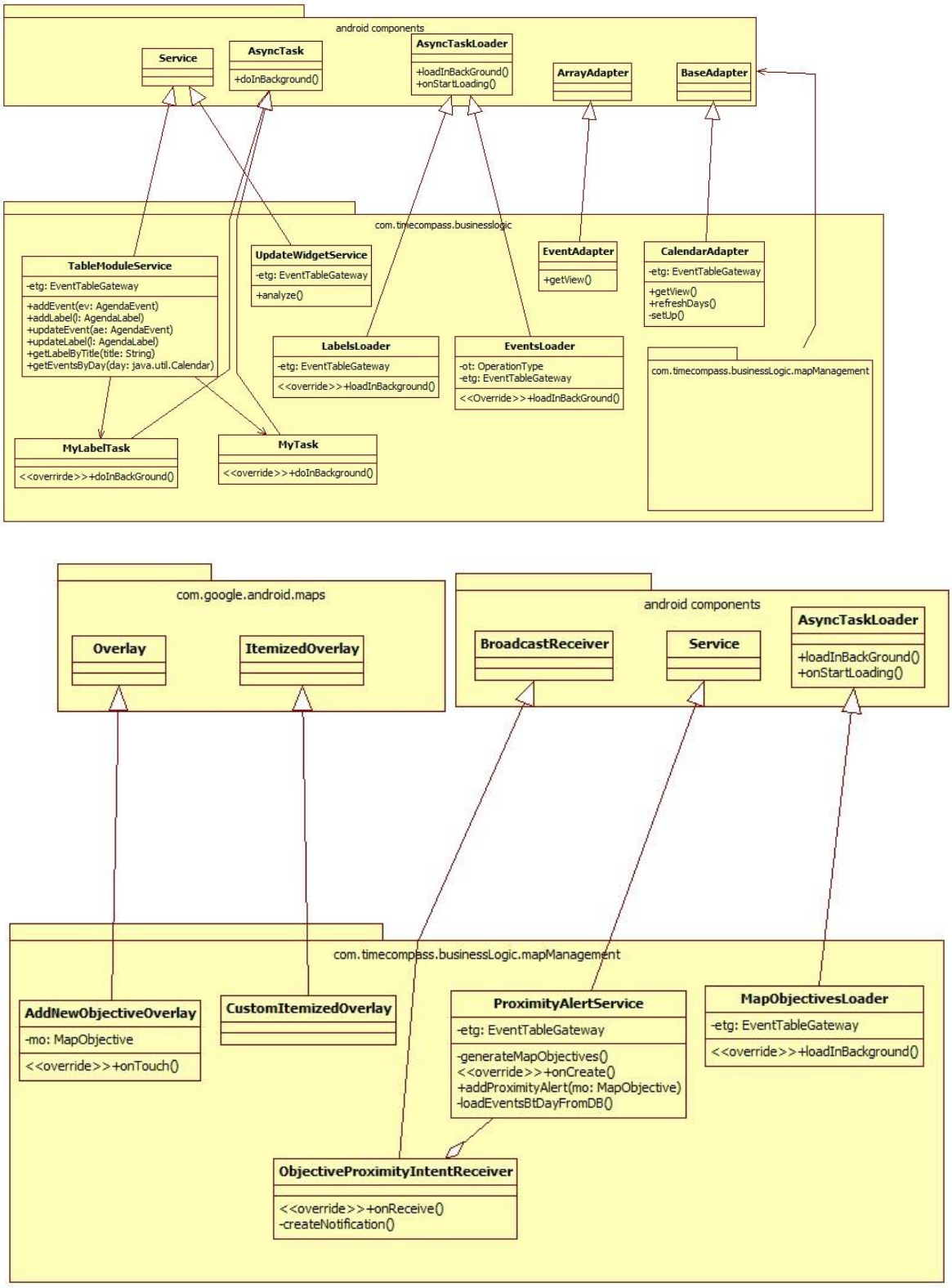
[11] *About Google Play*, Google Inc., [Online] Available : https://play.google.com/about/

Anexa 1. Digrama de clase pentru componenta de prezentare




android components
OnClickListener
+onClick()
OnCheckedChangeListener
+onCheckedChanged()
LoaderManager.LoaderCallbacks<List<T>>
com.google.android.maps
MapActivity
com.timecompass.presentation.MapPositoning
ViewObjectiveActivity
-tms: TableModuleService
-mo: MapObjective
-anoo: AddNewObjectiveOverlay
<<override>>+onCreate()
AddMapObjectiveActivity
-tms: TableModuleService
-agendaEventLoader
-labelStringLoader
-cio: CustomItemizedOverlay
<<override>>+onCreate()
-generateMapObjectives()

Anexa 2. Diagrama de clasa a componentei de businesslogic



Anexa 3. Listă de figure și tabele

Capitol	Figuri si tabele
Capitolul 1- Introducere	-
Capitolul 2- Obiectivele Proiectului	-
Capitolul 3- Studiu bibliografic	Figură 3.1 Matricea de priorități [3] Figura 3.2 Interfață grafică MyLifeOrganized Figura 3.3 Intefață grafică Google Calendar pentru Android
Capitolul 4- Analiza si fundamentare teoretica	Figură 3.1 Matricea de priorități [3] Figura 3.2 Interfață grafică MyLifeOrganized Figura 3.3 Intefață grafică Google Calendar pentru Android Figura 3.4 Interfață grafică ToDoMatrix (iPhone) Figură 4.1 Grafic distribuție versiuni Android (preluat de pe http://developer.android.com) Figura 4.2 Structura Android Figură 4.3 Structura ierarhică a interfețelor grafica în Android (preluată de pe http://developer.android.com) Figura 4.4 Diagrama cazurilor de utilizare Figura 4.5 Diagrama use case pentru adaugarea unuei noi activitati Figura 4.6 Diagrama use-case pentru afisarea evenimentelor in functie de zi Tabel 4.1 Structura pluging-urile Eclipse RPC Tabel 4.2 Componente din Android SDK Table 4.3 Cerințe funcționale Tabel 4.4 Caracteristici non-funcționale Tabel 4.5 Speficații despozitiv Tabel 4.6 Caracteristici non-functionale ce tin de factorul uman
Capitolul 5 Proiectare de detaliu și implementare	Figura 5.1 Diagrama conceptuală Figura 5.2 Structura modurilor de vizualizare Figura 5.3 Diagrama Bazei de date Figura 5.4 Diagramă de secvențiere pentru adaugarea unei noi activități Figura 5.5 Diagrama de secventiere pentru modificarea unei activitati Figura 5.6 Diagrama de secventiere pentru adaugarea unei noi etichete Figura 5.7 diagrama de secventiere pentru actualizarea unei etichete Figura 5.8 Diagrama de secventiere pentru vizualizarea activitatilor in functie de eticheta Figura 5.9 Diagrama de secventiere pentru vizualizarea activitatilor sub forma unei matrici

	eisenhower Figura 5.10 Diagrama de secventiere pentru vizualizarea activitatilor in functie de zi Figrua 5.11 Diagrama de secventiere pentru vizualizarea zilelor din luna in care sunt planificate activitati
Capitolul 6 Testare si validare	Figura 6.1 Locul indicat pe harta de c[tre obiectul de test de tip MapObjective Figura 6.2 Notificarea transmisă de aplicatie Figura 6.3 aparitia notificarii in bara de notificari Figura 6.5 Interfata grafica corespunzatoare Figura 6.4 Vizibilitatea Etichetelor create Figura 6.7 Afisarea Activitatilor in functie de eticheta Figura 6.6 Vizualizarea activitatilor in functie de luna Figura 6.8 Returnarea activitatilor in functie de zi
Capitolul 7- Manual de instalare si utilizare	Figură 7.2 Icoana aplicației în ecranul principal Figură 7.1 Fereastra principală a aplicației Figură 7.3 Disponibilitatea optiunii Add Event Figura 7.4 Tranziția către interfața grafică pentru adaugarea unei noi activități Figură 7.5 Adaugarea cu succes a unei noi activitati Figură 7.6 Adaugarea unei noi etichete Figura 7.7 Salvarea unei noi etichete Figura 7.8 Actualizarea unei activitati deja existente Figura 7.9 Vizualizarea activitatilor in functie de zi Figura 7.10 Vizualizarea zilelor din luna in care sunt planificate activitati Figura 7.11 Vizualizarea activitatilor in functie de eticheta Figura 7.12 Disponibilitatea optiunii View Eisenhower Matrix in cadrul aplicatiei Figura 7.13 Interfata grafica corespunzatoare matricei Eisenhower Figura 7.14 Adaugarea unui nou obiectiv Figura 7.15 Definirea unui nou obiectiv

Capitolul 8- Concluzii	-

Anexa 4. Acronime

SDK	- Software Development Kit
AVD	- Android Virtual Device
ADT	-Android Development Tool
XML	-Extensive Markup Language
GPS	-Global Positioning System
GTG	-Getting Things Done
FTF	-First Things First