Teodor Traian ȘTEFĂNUȚ
Dănuț Vasile MIHON
Victor Ioan BÂCU
Dorian GORGAN

PROIECTAREA INTERFEȚELOR UTILIZATOR

Îndrumător de laborator

U.T.PRESS
Cluj-Napoca, 2015
ISBN 978-606-737-068-3

Teodor Traian ȘTEFĂNUȚ

Dănuț Vasile MIHON

Victor Ioan BÂCU

Dorian GORGAN

PROIECTAREA INTERFEȚELOR UTILIZATOR

Îndrumător de laborator

U.T. PRESS
Cluj-Napoca, 2015
ISBN 978-606-737-068-3

Editura U.T.PRESS
Str.Observatorului nr. 34
C.P.42, O.P. 2, 400775 Cluj-Napoca
Tel.:0264-401.999 / Fax: 0264 - 430.408
e-mail: utpress@biblio.utcluj.ro
www.utcluj.ro/editura

Director: Prof.dr.ing. Daniela Manea
Consilier editorial: Ing. Călin D. Câmpean

ISBN 978-606-737-068-3
Bun de tipar: 15.06.2015

Prefață

Această publicație cuprinde 12 lucrări de laborator din domeniul Proiectării Interfețelor Utilizator. Scopul principal al acestui material este de a sprijini activitatea practică de laborator a studenților din anul 4 de studiu al Departamentului de Calculatoare, din cadrul Facultății de Automatică și Calculatoare a Universității Tehnice din Cluj-Napoca. În același timp însă, publicația se adresează oricărei persoane interesată de fundamentele dezvoltării interfețelor utilizator grafice și de tehnologiile actuale utilizate în acest sens.

Lucrările de laborator incluse în acest material pot fi grupate în două categorii: *9 lucrări orientate pe prezentarea de noi cunoștințe, noțiuni sau concepte* și *3 lucrări specializate în evaluarea abilităților practice dobândite de către cursanți și a gradului de asimilare de către aceștia a cunoștințelor teoretice prezentate.*

Fiecare lucrare de laborator din prima categorie este structurată în patru secțiuni principale. Prima dintre acestea descrie pe scurt aspectele tehnologice prezentate și obiectivele lucrării, în timp ce a doua secțiune include noțiunile teoretice elementare pe care cursantul trebuie să și le însușească pentru a putea îndeplini sarcinile practice trasate. Ultimele două secțiuni ale fiecărei lucrări prezintă aspecte și recomandări practice de dezvoltare și implementare și propun exerciții specifice spre rezolvare.

Lucrările din a doua categorie sunt structurate în trei secțiuni principale. Prima secțiune descrie pe scurt scopul evaluării efectuate. A doua secțiune prezintă metodologia de testare, criteriile și modalitățile de punctare, în timp ce a treia secțiune este rezervată exemplificării tipurilor de întrebări și exerciții utilizate în evaluare.

Cluj-Napoca,
30.05.2015

Autorii

Cuprins

IMPLEMENTAREA INTERFEȚELOR UTILIZATOR ÎN TEHNOLOGII WEB

Laboratorul 1 – limbajul HTML

1 Introducere

Dezvoltarea aplicațiilor WEB profesionale presupune realizarea unor interfețe utilizator care arată și se comportă identic atunci când sunt accesate de către un utilizator, indiferent de navigatorul folosit de acesta. În realizarea unor astfel de interfețe sunt utilizate în principal tehnologiile: HTML (HyperText Markup Language), CSS (Cascading Style Sheets) și JavaScript.

1.1 Obiective

Lucrare de față își propune prezentarea noțiunilor introductive asupra limbajului HTML și a unui subset al etichetelor acestuia pentru organizarea informațiilor.

2 Considerații teoretice

HTML reprezintă limbajul standard de descriere și organizare a conținutului care urmează a fi vizualizat prin intermediul unui navigator web.

Structura unui document HTML valid cuprinde trei părți principale:

1. **prima linie din document** – specifică informațiile despre versiunea de limbaj folosită
2. **secțiunea de antet** – cuprinde informații generale despre document și este încărcată complet înainte de descărcarea restului documentului; nu poate cuprinde informații vizibile din document
3. **corpul documentului** – cuprinde conținutul efectiv

```
<!DOCTYPE html>
<HTML>
   <HEAD>
      <TITLE>My first HTML document</TITLE>
   </HEAD>
   <BODY>
      <P>Hello world!</P>
   </BODY>
</HTML>
```

2.1 Etichete HTML

Sunt utilizate pentru organizarea conținutului din cadrul documentului (în secțiunea de conținut) sau pentru atașarea de informații suplimentare (în secțiunea de antet) cum sunt: metadate, fișiere corelate externe etc. Întreaga structură a documentului poate fi reprezentată sub forma unui arbore, cu rădăcina în elementul <HTML>, denumit DOM (Document Object Model), care poate fi parcurs de către navigator și pus la dispoziția limbajelor de scripting.

Din punct de vedere al modului de reprezentare implicit, etichetele de organizare a conținutului pot fi grupate în:

- *etichete de tip bloc* – introduc o secționare a conținutului și ocupă întreg spațiul orizontal disponibil
- *etichete în linie* – sunt reprezentate de către navigator în interiorul textului, fiind incluse în regulile generale de formatare a acestuia

Cele mai uzuale etichete din cadrul unui document HTML sunt:

Etichetă HTML	**Descriere**
TITLE	Specifică titlul întregului document și este inclus în secțiunea de antet. Textul din cadrul acestei etichete este utilizat de către navigator pentru a denumi fereastra în care este prezentat documentul.
LINK	Definește o legătură către un document extern (ex. CSS, alt fișier HTML) care poate fi utilizat de către navigator în afișarea informațiilor sau în navigare. Acest element este inclus în secțiunea de antet.
SCRIPT	Permite includerea în cadrul documentului a unor instrucțiuni de programare complexe ce pot fi executate de către navigator. Aceste instrucțiuni pot fi grupate într-un document extern, ce va fi descărcat separat, sau incluse direct în documentul curent. Limbajul de programare cel mai uzual în prezent este **JavaScript**. Elementele de script pot fi incluse în oricare dintre secțiunile documentului iar execuția lor poate fi imediată (execuția are loc în momentul în care scriptul este complet încărcat, chiar dacă o parte a documentului nu a fost încă primită) sau declanșată de anumite evenimente (acțiuni utilizator, finalizarea încărcării documentului etc.).

STYLE	Permite includerea în cadrul documentului a unor instrucțiuni de formatare CSS. Conform recomandărilor de bună practică însă, este indicat ca specificațiile CSS să fie incluse într-un document extern, dedicat.
H1, H2, H3, H4, H5, H6	Sunt etichete de tip bloc, care pot fi incluse numai în corpul documentului și sunt folosite pentru a structura conținutul în secțiuni și sub-secțiuni. Recomandările de bună-practică menționează că un document ar trebui să conțină un singur element de tip H1.
DIV	Element de tip *bloc* folosit pentru gruparea conținutului într-o secțiune rectangulară.
P	Etichetă de tip *bloc* pentru separarea paragrafelor din cadrul unui text.
A	Etichetă repezentată *în linie* pentru inserarea unei legături externe sau interne documentului curent.
IMG	Etichetă cu format implicit *în linie* pentru afișarea unei imagini în cadrul documentului.
SPAN	Etichetă repezentată *în linie* pentru formatări specifice conținutului de tip text.
FORM, INPUT, LABEL	Etichete pentru descrierea formularelor în cadrul unui document HTML. • FORM are rolul de grupare a elementelor de tip INPUT și este o etichetă de tip *bloc* • INPUT descrie generic fiecare dintre elementele disponibile pentru realizarea unui formular, tipul exact fiind specificat prin atributul TYPE • LABEL specifică un titlu asociat cu un element de tip INPUT
TABLE	Permite inserarea unui tabel în cadrul unui document HTML.
THEAD	Element semantic și vizual de evidențiere a unei secțiuni dintr-un tabel sub formă de antet / secțiune de titlu.
TBODY	Element semantic și vizual de grupare a informațiilor utile din cadrul unui tabel.
TR, TD	Etichete care descriu rânduri, respectiv celule în cadrul unui tabel. Numărul de etichete <TD> inserate într-o etichetă de tip <TR> stabilește numărul de coloane ale tabelului.
OL, UL	Etichete care permit definirea de liste ordonate, respectiv neordonate.
LI	Descrie un element din cadrul unei liste ordonate / neordonate.

Etichete introduse de standardul HTML 5 cu scop de organizare semantică a conținutului:

Etichetă HTML	Descriere
HEADER	Grupează elementele care fac parte din antetul vizual al paginii (denumirea companiei, logo-ul acesteia, sloganul etc.)
SECTION	Permite organizarea conținutului din cadrul paginii în secțiuni logice
FOOTER	Desemenează zona de final a paginii, care cuprinde în general informații despre drepturile de autor, ultima actualizare, compania care a realizat situl etc.
NAV	Cuprinde elementele vizuale care descriu meniul principal al sitului și alte informații de navigare în cadrul acestuia.

Tip nou de conținut introdus nativ în cadrul HTML 5 prin definirea unor noi etichete:

Etichetă HTML	Descriere
VIDEO	Permite afișarea de conținut video direct în cadrul paginii fără interpunerea unei terțe tehnologii (ex. Flash, Silverlight)
AUDIO	Asigură redarea de sunet fără interpunerea unei terțe tehnologii (ex. Flash, Silverlight).
CANVAS	Asigură programatorilor o zonă de afișare în format bitmap pentru trasarea în timp real a imaginilor, animațiilor etc.

2.2 Atribute ale etichetelor HTML

Fiecare tip de element HTML poate conține diferite atribute care specifică anumite configurări ale instanței pentru care au fost definite. Lista completă de atribute, comune tuturor etichetelor sau specifice pentru anumite tipuri de etichete poate fi consultată în documentația oficială corespunzătoare fiecărui tip de element HTML.

În cadrul activităților de laborator, cele mai uzuale atribute ale etichetelor HTML vor fi:

Etichetă HTML	Descriere
id	Atribut comun majorității elementelor HTML. Identifică în mod unic, în întregul document, o anumită etichetă HTML
class	Atribut comun majorității elementelor HTML. Permite

	aplicarea uneia sau a mai multor clase CSS elementului curent (care conține atributul) și elementelor care sunt cuprinse în cadrul său
src	Definește calea către un fișier extern de tip script, imagine etc, în funcție de tipul etichetei pentru care a fost definit
type	Stabilește tipul de element care este specificat prin intermediul etichetei INPUT care conține acest atribut
action	Atribut specific etichetei FORM utilizat la desemnarea fișierului care primește informațiile introduse de către utilizator în formular
encoding	Stabilește modalitatea de codificare a informațiilor cuprinse în câmpurile formularului, pentru a putea fi transmise fără erori către server
onfocus	Permite specificarea unei funcții JavaScript care este apelată în momentul în care elementul primește focus-ul
onblur	Permite specificarea unei funcții JavaScript care este apelată în momentul în care elementul pierde focus-ul
onchange	Specifică funcția JavaScript care se dorește a fi apelată în momentul în care valoarea elementului se schimbă
onload	Este un eveniment care apare atunci când documentul HTML este complet încărcat și poate fi apelat din JavaScript. Se atașează în general elementului BODY
onkeydown	Specifică funcția JavaScript care este apelată la fiecare apăsare de tastă, dacă elementul curent are focus-ul
onkeyup	Specifică funcția JavaScript care este apelată la fiecare eliberare de tastă, dacă elementul curent are focus-ul
onkeypress	Specifică funcția JavaScript care este apelată la fiecare ciclu complet de apăsare + eliberare de tastă, dacă elementul curent are focus-ul

3 Practici de implementare

Codul HTML nu necesită un mediu de dezvoltare sau compilare înainte de publicare. El poate fi editat în orice editor de text (ex. Notepad), dar pentru facilități de evidențiere cod sau auto-completare recomandăm folosirea unor aplicații cum sunt Notepad++, Brackets, Sublime, Adobe Dreamweaver etc.

Vizualizarea rezultatului obținut în urma descrierii se realizează prin deschiderea fișierului cu extensia **.htm** sau **.html** în oricare dintre navigatoarele web disponibile, direct de pe HDD-ul stației pe care ați lucrat.

4 Exerciții

Exercițiul 1. Creați o pagină HTML dedicată unei agenții de turism, cu următoarele specificații minime:

- un element de tip H1 pentru titlul paginii
- o listă neordonată de legături (elemente de tip A), cu rol de meniu al sitului web, cu următoarele elemente: *Acasă*, *Croaziere*, *Plimbări în natură*, *Înot*, *Oferte speciale*
- o imagine cu dimensiunea 900 x 350 px cu rol de banner
- un element de tip H2 cu titlul *Despre noi*
- text pentru secțiunea *Despre noi* care să fie grupat în două paragrafe (etichetă de tip P)

Exercițiul 2. Adăugați la pagina creată la exercițiul 1 următoarele elemente

- o nouă secțiune, intitulată *Oferte speciale* (etichetă H2)
- 3 subsecțiuni alcătuite din: *titlu* (etichetă H3), *imagine*, *descriere*, similar exemplificării din figura de mai jos.

Barcelona

Ut elit odio, varius vel est vel, tincidunt malesuada ex. Phasellus vulputate quam placerat, mollis sapien in, laoreet mi.

Laboratorul 2 – Formatare CSS

1 Introducere

Foile de stil reprezintă o modalitate modulară de formatare vizuală a unui document HTML, care este ușor de integrat la nivel de pagină sau chiar de întreg sit web. Cele mai bune practici recomandă plasarea specificațiilor de stil într-un fișier extern, cu extensia .css, care să fie integrat în paginile HTML în care este necesar prin utilizarea etichetei LINK.

```
<LINK rel="stylesheet" href="nume_fisier.css" type="text/css">
```

1.1 Obiective

Lucrare de față își propune prezentarea noțiunilor de bază utilizate în definirea stilurilor CSS și în modalitatea de aplicare a acestora pe elemente HTML.

2 Considerații teoretice

Specificațiile de stil sunt grupate în mod obligatoriu în **clase CSS** care se aplică în mod atomic elementelor HTML la care sunt conectate. Nu este posibilă aplicarea din cadrul unei clase doar a unui subset de specificații, dar este permisă suprascrierea acestora cu ajutorul unei alte clase.

O clasă CSS are următoarea sintaxă generică în care **[nume_clasă]** poate fi oricare dintre selectorii menționați la secțiunea 2.1:

```
[nume_clasă]
{
  proprietate_1: valoare_1;
  proprietate_2: valoare_2;
  ...
  proprietate_n: valoare_n;
}
```

2.1 Selectori CSS

Legătura dintre specificațiile CSS și elementul HTML la care acestea fac referire poate fi realizată în următoarele moduri:

2.1.1 Folosind eticheta HTML

Clasele CSS care au ca și denumire etichete HTML sunt aplicate în mod automat tuturor elementelor de acel tip identificate în fișierul HTML. Spre exemplu, codul următor va aplica tuturor elementelor de tip DIV din document o bordură de 1px grosime, de culoare verde.

```
div
{
  border: 1px solid #00FF00;
}
```

Este așadar recomandat ca acest tip de etichete să fie utilizate pentru a stabili regulile generale de afișare a tipurilor de elemente HTML, urmând ca excepțiile să fie rezolvate cu ajutorul altor tipuri de selectori.

2.1.2 Folosind atributul CLASS al elementului HTML

Pentru a aplica aceleași reguli de stil unui subset heterogen din elementele HTML este necesară definirea unei clase CSS independentă, folosind sintaxa **.numeClasă**:

```
.mesajImportant
{
  color: red;
  font-style: italic;
  font-weight: bold;
}
```

Aplicarea acestei clase oricărui element HTML se realizează folosind atributul **class** al etichetei:

```
<ELEMENT class="class1 [class2 class3 ...]">
...
<ELEMENT class="mesajImportant">
```

2.1.3 Folosind ID-ul elementului HTML

Utilizarea acestei metode este recomandată atunci când specificațiile CSS se aplică numai unui singur element HTML din cadrul documentului, care este identificat prin intermediul ID-ului unic:

```
<ELEMENT id="mesajEroare">
```

În codul CSS, clasa corespunzătoare are obligatoriu denumirea **#idElement**. Pentru exemplul nostru:

```
#mesajEroare
{
  color: white;
  font-weight: bold;
  background-color: red;
}
```

2.1.4 Folosind atributul STYLE al elementului HTML

Această modalitate de specificare a codului CSS beneficiază de cea mai mare prioritate la afișare (vezi secțiunea 2.3) însă nu este recomandată în utilizarea extensivă, deoarece codul astfel introdus nu poate fi reutilizat la alte elemente (cu excepția celor imbricate în elementul formatat) sau la alte documente.

```
<ELEMENT style="proprietate_css_1:valoare_1; propriatea_css_2:valoare_2;
...">
```

2.2 Modalități de definire a claselor CSS

Clasele CSS care au reguli comune de formatare pot fi definite împreună, denumirile lor fiind separate prin virgulă:

```
[nume_clasă_1, nume_clasă_2, nume_clasă_3]
{
  proprietate_1: valoare_1;
  proprietate_2: valoare_2;
  ...
  proprietate_n: valoare_n;
}
```

Această modalitate de definire nu este influențată de tipul selectorilor utilizați, existând posibilitatea amestecării acestora după cum este necesar. Deoarece în aplicarea claselor CSS nu este posibilă selectarea unui subset din specificații, diferențierea între clase se poate realiza prin completarea / modificarea definiției acestora mai târziu în document:

```
[nume_clasă_2]
{
  proprietate_1: valoare_nouă;
  proprietate_nouă_1: valoare_nouă_1;
  proprietate_nouă_2: valoare_nouă_2;
  ...
  proprietate_nouă_n: valoare_nouă_n;
}
```

În cadrul unei specificații CSS, fie că aceasta este cuprinsă într-un singur fișier sau în mai multe fișiere, denumirea unei clase poate să apară de mai multe ori.

Formatul final al clasei se obține prin suprascrierea definițiilor în funcție de apariția pozițională a acestora și *suprascrierea specificațiilor recurente / adăugarea specificațiilor noi*. Ordinea de parcurgere a documentelor este dată de ordinea în care acestea sunt incluse în fișierul HTML.

2.3 Aplicarea specificațiilor CSS pe elementele HTML

Fiecare dintre elementele HTML are aplicate mai multe specificații de stil care provin din mai multe clase CSS. La fel ca și în cazul claselor CSS cu mai multe definiții, specificațiile de același tip sunt suprascrise în funcție de ordinea aplicării (ultimul aplicat este cel vizibil) iar specificațiile noi sunt adăugate la setările vizuale ale elementului.

Aplicarea claselor asupra unei etichete se realizează pe baza unui număr care indică **specificitatea clasei** CSS. Acest număr se calculează astfel (vezi Figura 1.1):

- dacă elementul are specificații CSS în interiorul atributului **style**, acestea vor fi prioritare (1,0,0,0 puncte)
- pentru fiecare **ID** din specificarea selectorului, sunt calculate 0,1,0,0 puncte
- pentru fiecare **selector CSS generic** (.numeClasă) sunt calculate 0,0,1,0 puncte
- pentru fiecare **element HTML** din specificarea selectorului sunt calculate 0,0,0,1 puncte

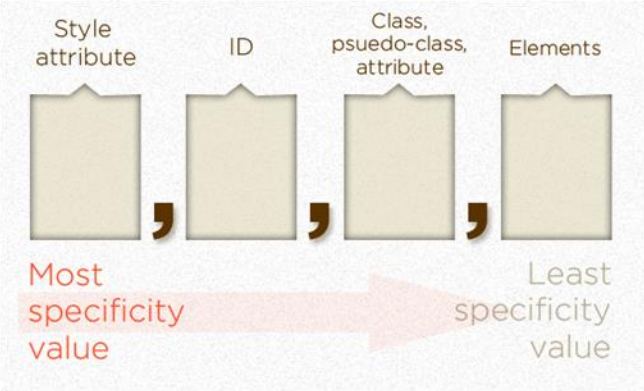


Figura 1.1: Ilustrare a calculului de specificitate pentru clasele CSS. Preluare din [1]

Este important de menționat aici că unele proprietăți sunt implicit moștenite de către etichetele imbricate în eticheta formatată (ex. dimensiunea textului, culoarea, tipul de font etc...). Aceste proprietăți au însă gradul de specificitate 0,0,0,1 ceea ce înseamnă că oricare formatare specifică etichetei imbricate va suprascrie valoarea moștenită.

3 Practici de implementare

Codul CSS nu necesită un mediu de dezvoltare sau compilare înainte de publicare. El poate fi editat în orice editor de text (ex. Notepad), dar pentru facilități de evidențiere cod sau auto-completare recomandăm folosirea unor aplicații cum sunt Notepad++, Brackets, Sublime, Adobe Dreamweaver etc.

Vizualizarea rezultatului obținut poate fi realizată numai prin utilizarea unui fișier HTML în care clasele definite să fie aplicate elementelor din cadrul acestuia. Când legătura dintre fișierul HTML și cel CSS este realizată corect, deschiderea primului în oricare dintre navigatoarele web disponibile va declanșa automat afișarea corectă a etichetelor specificate folosind regulile CSS atașate.

Figura 2.2: Exemplu de unelte pentru inspectarea modului de aplicare a stilurilor și de suprascriere a proprietăților CSS (Firefox)

Inspectarea modului de aplicare a stilurilor și de suprascriere a diferitelor proprietăți poate fi realizată, în majoritatea navigatoarelor, folosind uneltele specifice de dezvoltare (accesibile fie din meniul aplicației fie prin apăsarea tastei F12), similar cu exemplul din Figura 2.2.

4 Exerciții

Exercițiul 1. Implementați designul din figura de mai jos folosind HTML și CSS

Specificații de implementare:

- **titlul** – font: Arial 24px îngroșat, culoare: #800000; este și legătură către pagina principală (Acasă)
- **meniul** – culoare fundal: #DEB887, colțuri rontunjite cu raza de 10px
- **elementele de meniu** – elemente de tip A, font: Arial 12 px, culoare implicită: #800000, culoare interacțiune: #FFFFFF
- **titlu secțiune** – font: Arial 18px îngroșat, culoare: #000000, margine: 10px sus și jos
- **text pagină** – font: Arial 12px, culoare: #666666, aliniere: justified; spațiere între paragrafe: 10px

- pot fi folosite doar următoarele elemente HTML: *html*, *head*, *body*, *title*, *header*, *section*, *footer*, *h1*, *h2*, *h3*, *div*, *p*, *span*, *ul*, *li*, *a*, *img*, *strong*, *em*
- este permisă numai utilizarea selectorilor CSS bazați pe etichete HTML
    - nu este permisă utilizarea atributului *style*
    - nu este permisă introducerea de cod CSS în documentul HTML
    - nu este permisă definirea atributelor ID sau CLASS pentru elementele HTML

Exercițiul 2. În continuarea documentului de la exercițiul 1 implementați designul din figura de mai jos utilizând HTML și CSS:

Specificații de implementare:

- **titlu secțiune** – font: Arial 18px îngroșat, culoare: #000000, margine: 10px sus și jos
- **titlu subsecțiune** – font: Arial 14px îngroșat, culoare: #0000FF, culoare fundal: #7FFFD4, spațiu interior: 5px
- **subsecțiune** – dimensiune: 33% din lățimea de afișare, spațiere: 10px față de celelalte secțiuni
- **imagine** – lățime: 100% din lățimea subsecțiunii, înălțime fixă: 300px
- **text** – font: Arial 12px, culoare: #666666, aliniere: justified, culoare fundal: #DEB887
- **paragraf** – margine: 10px sus și jos, spațiu interior: 10px

- pot fi folosite doar următoarele elemente HTML: *html*, *head*, *body*, *title*, *header*, *section*, *footer*, *h1*, *h2*, *h3*, *div*, *p*, *span*, *ul*, *li*, *a*, *img*, *strong*, *em*
- este permisă numai utilizarea selectorilor CSS bazați pe etichete HTML
    - nu este permisă utilizarea atributului *style*
    - nu este permisă introducerea de cod CSS în documentul HTML
    - nu este permisă definirea atributelor ID sau CLASS pentru elementele HTML

Laboratorul 3 – limbajul JavaScript

1 Introducere

Deoarece HTML și CSS sunt doar limbaje descriptive, care pot interacționa cu utilizatorul numai prin intermediul navigatorului și se pot adapta interacțiunilor cu acesta doar într-o măsură foarte redusă, pentru realizarea procesărilor mai complexe pe client (realizare de animații, actualizare conținut în mod asincron etc.) este necesară scrierea de instrucțiuni care pot fi interpretate și executate de către navigator în mod dinamic. Limbajul utilizat se numește JavaScript.

1.1 Obiective

Lucrarea de față își propune prezentarea noțiunilor de bază în modelarea și aplicarea interacțiunilor utilizator folosind limbajul JavaScript.

2 Considerații teoretice

Prin intermediul codului JavaScript programatorii au acces la structura DOM a documentului și pot modifica diferitele proprietăți ale elementelor HTML. Este important de menționat că structura DOM nu este disponibilă de la momentul 0 al accesării paginii, ci numai după ce toate elementele din eticheta BODY au fost descărcate și prelucrate de către Navigator. La acel moment este generat evenimentul **window.onload** care poate fi folosit ca un inițiator de lansare a codului JavaScript ce trebuie să ruleze automat la încărcarea paginii.

Codul JavaScript poate fi conectat la un document HTML folosind etichetele SCRIPT în două modalități:

2.1 Includere cod direct în fișierul HTML

Modalitate de specificare utilă pentru situațiile în care codul nu este reutilizat în alte pagini HTML.

```
<SCRIPT type="text/javascript">
    // cod JavaScript
```

```
</SCRIPT>
```

În general acest mod de inserare a codului JavaScript se realizează în cadrul etichetelor HEAD sau BODY ale documentului, ca fiu direct al acestora.

2.2 Includere cod JavaScript dintr-un fișier extern

Este modalitatea recomandată de organizare a codului, pentru un grad mai ridicat de flexibilitate (același cod poate fi ușor refolosit în mai multe documente HTML).

```
<SCRIPT src="nume_fisier.js" type="text/javascript"></SCRIPT>
```

În marea majoritate a cazurilor, utilizarea sub această formă a etichetei SCRIPT se întâlnește în cadrul secțiunii HEAD a documentului HTML. În cadrul aceluiași document HTML pot fi incluse oricâte fișiere externe de cod JavaScript.

2.3 Funcții JavaScript

În cadrul activității de laborator vor fi utilizate următoarele elemente ale limbajului JavaScript:

window	Cuvânt cheie care referențiază fereastra navigatorului în care rulează scriptul curent
window.onload	Eveniment generat de către navigator în momentul în care DOM-ul este finalizat și accesibil
document	Cuvânt cheie care referențiază rădăcina arborelui DOM și este echivalent cu nivelul etichetei BODY
document.getElementById("id_element")	Returnează o referință către elementul HTML care are atributul id = "id_element"
document.getElementsByTagName("etichetă")	Funcție care returnează șirul elementelor de tip ETICHETĂ din documentul HTML

this	Cuvânt cheie care referențiază instanța HTML în a cărui membru este apelat. Ex:
	`elem.onblur = function()`
	`{`
	`  formInputBlur(this);`

	`};` În exemplul de mai sus, this = elem la momentul execuției funcției anonime.
FORM.onsubmit	Eveniment generat de către navigator în momentul în care un formular este pregătit pentru trimitere către server. Pentru anulare, funcția atașată trebuie să returneze **false**.
ELEMENT.onblur	Eveniment generat de către navigator atunci când ELEMENT pierde focalizarea
ELEMENT.onclick	Eveniment generat de către navigator atunci când asupra ELEMENT se efectuează un click de mouse
ELEMENT.getAttribute("nume_atribut")	Returnează valoarea atributului "nume_atribut" al obiectului ELEMENT
ELEMENT.setAttribute("nume_atribut", "valoare")	Setează valoarea atributului "nume_atribut" la "valoare" în cadrul obiectului ELEMENT
ELEMENT.innerHTML	Returnează conținutul etichetei ELEMENT sub formă de cod HTML. Dacă i se atribuie o valoare, conținutul ELEMENT va fi înlocuit.
VARIABILĂ.length	Returnează numărul de elemente din VARIABILĂ: • numărul de caractere dacă variabilă este de tip STRING • numărul de elemente dacă VARIABILĂ este de tip ARRAY
STRING.indexOf(string)	Returnează poziția (numărată de la 0) la care se regăsește **string** în STRING sau -1 în caz contrar.
Date.parse(string)	Returnează numărul de milisecunde între data definită prin **string** și 1 Ianuarie 1970, sau NaN dacă **string** nu poate fi convertit la o dată validă.

isNaN(object)	Returnează **true** dacă **object** are valoarea NaN.

Pentru utilizarea expresiilor regulate în cadrul limbajului JavaScript este necesară definirea expresiei de forma **/expresie/** și apoi utilizarea funcției **test** asupra elementului dorit pentru verificare.

```
var re = /^(([^<>()[\]\\.,;:\s@\"]+(\.[^<>()[\]\\.,;:\s/;
re.test(string); // returnează true dacă string respectă expresia
```

3 Practici de implementare

La fel ca și în cazul limbajelor HTML și CSS, codul JavaScript nu necesită un mediu specific de dezvoltare sau compilare înainte de publicare. El poate fi editat în orice editor de text (ex. Notepad), dar pentru facilități de evidențiere cod sau auto-completare recomandăm folosirea unor aplicații cum sunt Notepad++, Brackets, Sublime, Adobe Dreamweaver etc.

Testarea și depanarea codului scris poate fi realizată numai în cadrul unei aplicații de navigare pe internet și doar dacă fișierele cu codul JavaScript au fost corect integrate cu un fișier HTML. După deschiderea acestuia din urmă în navigator, uneltele de dezvoltare oferite de cele mai multe dintre versiunile noi ale aplicațiilor de navigare vor permite selectarea fișierului de cod dorit și stabilirea de puncte de întrerupere a execuției (vezi Figura 3.1).

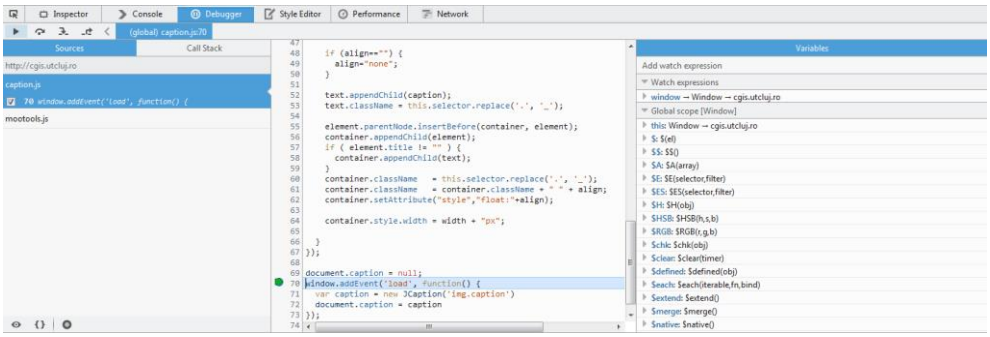


Figura 3.1: Exemplu de unelte pentru depanarea codului JavaScript (Firefox)

Relativ la specificul de execuție al codului JavaScript este important de subliniat faptul că la fiecare reîncărcare a paginii execuția este reluată de la zero. Niciunele dintre variabilele setate la o execuție anterioară nu își păstrează starea. Singurele

modalități de transmitere a informațiilor folosind numai JavaScript între două pagini HTML succesive sau între reafișări succesive sunt prin intermediul cookie-urilor sau prin transmiterea de parametri GET.

4 Exerciții

Exercițiul 1. Implementați un formular de contact cu următoarele câmpuri: nume, prenume, adresă, data nașterii, telefon, email, culoarea favorită. Adăugați două butoane intitulate **Reset** și **Trimite**. Implementați în JavaScript următoarea funcționalitate:

- la apăsarea butonului **Reset**, câmpurile formularului sunt aduse la starea inițială (conținut gol sau valoare implicită)
- la apăsarea butonului **Trimite**, formularul va fi ascuns iar informațiile din acesta vor fi afișate în următoarea formă:

  Nume: [valoarea_câmpului_nume]
  Prenume: [valoarea_câmpului_prenume]
  Adresă: [valoarea_câmpului_adresă]
  Data nașterii: [valoarea_câmpului_dataNașterii]
  Telefon: [valoarea_câmpului_telefon]
  Email: legătură către adresa de email introdusă
  Culoarea favorită: dreptunghi colorat conform valorii selectate

- funcționalitatea de la punctele anterioare va fi realizată fără reîncărcarea paginii HTML

Exercițiul 2. Folosind formularul de mai sus, implementați în JavaScript funcționalitatea necesară astfel încât la apăsarea butonului Trimite să se realizeze următoarele validări:

- **nume** trebuie să conțină cel puțin trei litere
- **prenume** trebuie să conțină cel puțin trei litere
- **adresa** trebuie să conțină cel puțin trei caractere și cel puțin o cifră. Nu sunt acceptate caracterele: @#$%^&*
- **data nasterii** trebuie să conțină o dată validă în format dd/mm/yyyy

- cîmpul **telefon** poate conține numai cifre și un caracter -, în formatul nnn-nnnnnnnnn (3 - 9)
- câmpul email trebuie să accepte numai o adresă de email corectă din punct de vedere sintactic

La fiecare apăsare a butonului **Trimite** erorile identificate, indiferent de numărul acestora, vor fi anunțate utilizatorului prin intermediul unei singure ferestre de dialog de tip **Alert**.

Laboratorul 4 – verificarea cunoștințelor în Tehnologii Web

1 Introducere

În cadrul acestui laborator va fi evaluat nivelul înțelegerii noțiunilor din domeniul Tehnologiilor Web prezentate în cadrul laboratoarelor anterioare. Fiecare dintre studenți va primi o notă ce va reflecta atât nivelul cunoștințelor teoretice însușite cât și nivelul abilităților practice dobândite.

1.1 Obiective

Lucrarea de față își propune stabilirea măsurii în care noțiunile fundamentale din domeniul Tehnologiilor Web au fost însușite de către cursanți.

2 Metodologia de evaluare

Evaluarea cunoștințelor dobândite în timpul lucrărilor practice de laborator din domeniul Tehnologiilor Web se va realiza pe două niveluri:

2.1 Evaluarea teoretică

Presupune testarea înțelegerii noțiunilor teoretice de bază, care permite atât profesorului cât și studentului identificarea eventualelor puncte neclare în însușirea cunoștințelor elementare.

Această evaluare se va desfășura prin intermediul unui chestionar cu aproximativ 10 - 15 întrebări care au multiple variante de răspuns, nota finală obținută fiind accesibilă imediat după completarea chestionarului, prin corectare automată.

Fiecare dintre participanții la laborator vor avea acces la forma electronică a chestionarului pentru un timp limitat, contorizat automat, și numai de pe contul propriu din cadrul platformei de gestiune a materialelor de laborator. Întrebările pentru fiecare chestionar vor fi selecționate aleator din cadrul unei baze de date iar răspunsurile fiecărei întrebări vor fi de asemenea ordonate aleator.

2.2 Evaluarea practică

Are ca și scop verificarea abilității studenților de a aplica noțiunile învățate în vederea dezvoltării unor interfețe utilizator Web. Fiecăruia dintre participanți îi va fi asignat un exercițiu, având la dispoziție spre consultare în timpul rezolvării atât îndrumătorul de laborator cât și documentațiile oficiale ale tehnologiilor folosite.

Implementarea prezentată va fi evaluată conform următoarelor criterii generale:

- **structură** – modalitatea de utilizare a etichetelor HTML trebuie să corespundă bunelor practici descrise în cadrul lucrării de laborator și să asigure corectitudinea semantică a datelor prezentate
- **aspect** – gradul în care formatarea vizuală utilizând clase CSS respectă cerințele exercițiului
- **interacțiune** – în ce măsură implementarea realizată în JavaScript corespunde cerințelor de:
    - verificarea a validității datelor
    - prevenirii erorilor
    - sprijinire a utilizatorului în revenirea dintr-o situație de eroare
    - etc.
- **respectarea recomandărilor de bune practici**
    - evitarea utilizării atributului **style**
    - separarea structurii documentului (HTML) de formatarea vizuală (CSS) și implementarea interacțiunilor (JavaScript)
    - etc.

La finalul timpului de lucru, fiecare dintre studenți prezintă cadrului didactic implementarea efectuată și fundamentele teoretice care au stat la baza deciziilor de implementare.

3 Exerciții

3.1 Exemple de întrebări pentru verificarea cunoștințelor teoretice

1. **Selectați grupul de etichete minim necesare pentru a defini structura de bază a unui document HTML?**
    a. <html> <script> <table>

b. <html> <head> <body> <doctype>
c. <html> <body>
d. <html> <head>

2. **Care dintre următoarele etichete permite afișarea textului ”PIU” la o dimensiune cât mai mare?**
   a. <h2>
   b. <h6>
   c. <heading>
   d. <h1>

3. **Care este instrucțiunea pentru a deschide un link într-o fereastră nouă din navigator?**
   a. <a href=”www.google.com” open_new>
   b. <a href=”www.google.com” target=”_blank”>
   c. <a href=”www.google.com” target=”new” >
   d. <a link=”www.google.com” tab=”_new”>

4. **Specificați tipurile de elemente HTML care sunt folosite pentru definirea unui tabel:**
   a. <table> <tr> <td>
   b. <table> <head> <td>
   c. <table> <row> <column>
   d. <table> <tc> <tr>

5. **Alegeți răspunsurile corecte:**
   a. Atributul ”id” identifică în mod unic o etichetă în cadrul unui document HTML
   b. Atributul ”class” nu poate fi folosit pentru etichete care conțin deja un ”id”
   c. Codul HTML trebuie compilat de către utilizator, la fel ca și orice aplicație C++
   d. Pentru numerotarea unei liste de elemente se folosește eticheta

6. **Pentru a include un fișier extern în formatul CSS, care dintre următoarele instrucțiuni sunt adevărate?**
   a. <style src=”fisier.css”>
   b. <style>fisier.css</style>

c. <link rel=”stylesheet” type=”text/css” href=”fisier.css”>
d. <link rel=”stylesheet” type=”text/css”>fisier.css</link>

7. Stilul CSS

```
table
{
   border: 1px solid #FF0000;
}
```

are rolul de a seta culoarea roșie și dimeniunea de 1 pixel pentru conturul primului tabel din documentul HTML în care a fost inclus?

a. Adevărat
b. Fals

8. Legătura dintre elementele HTML și stilurile CSS se face prin:

a. Selectori HTML
b. Selectori CSS
c. Fișier de configurare extern în care se specifică aceste relații sub forma <etichetă_html, stil_css>
d. Nu este nevoie să se specifice această legătură, întrucât navigatoarele (browsere-le) moderne preiau automat această informație din fișierul HTML, fără a mai fi nevoie de fișiere CSS. Fișierele CSS sunt folosite doar de către navigatoarele Web învechite

9. Alegeți varianta corectă de stil CSS care poate fi folosit pentru a selecta toate etichetele care au clasa ”selectat”:

a. div.class=”selectat”
b. div selectat
c. div.selectat
d. div#selectat

10. Prin care stil CSS se poate realiza selectarea tuturor elementelor <p> din interiorul etichetelor ?

a. div p
b. div.p
c. p div
d. select p from div

11. **Specificați care dintre următoarele afirmații sunt adevărate în ceea ce privește limbajul JavaScript (JS):**
    a. Codul JS trebuie compilat de către utilizator, la fel ca și orice aplicație C++
    b. Codul JS este delimitat de codul HTML prin folosirea etichetei <script>
    c. Codul JS asociat unui document HTML trebuie neaparat să fie plasat în același fișier. Nu este permisă încărcarea unor resurse externe JavaScript
    d. Limbajul JavaScript permite accesarea obiectelor din structura DOM

12. **Alegeți variantă corectă pentru afirmația: accesarea structurii DOM este permisă înainte de generarea unui eveniment de tipul ”window.onload”?**
    a. Adevărat
    b. Fals

13. **Considerând faptul că documentul HTML conține instrucțiunile**

```
<body>
  <div>
    <p id=”p-elem-id” class=”p-elem-class” />
  </div>

  <div id=”div-elem-id” />
</body>
```

    care dintre următoarele secvențe JS pot fi folosite pentru a selecta elementul <p>?
    a. document.getElementById(“p-elem-id”)
    b. document.getElementsByTagName(“p”)
    c. document.getElementByClass(“p-elem-class”)
    d. document.getElement(“p”)

14. **Specificați valoarea de adevăr a următoarei afirmații: acțiunile uzuale (ex.: click, focalizare etc.) asupra elementelor HTML pot fi recepționate prin evenimente JavaScript.**
    a. Adevărat
    b. Fals

15. **Ce se va afișa în urma execuției secvenței de cod JavaScript de mai jos?**

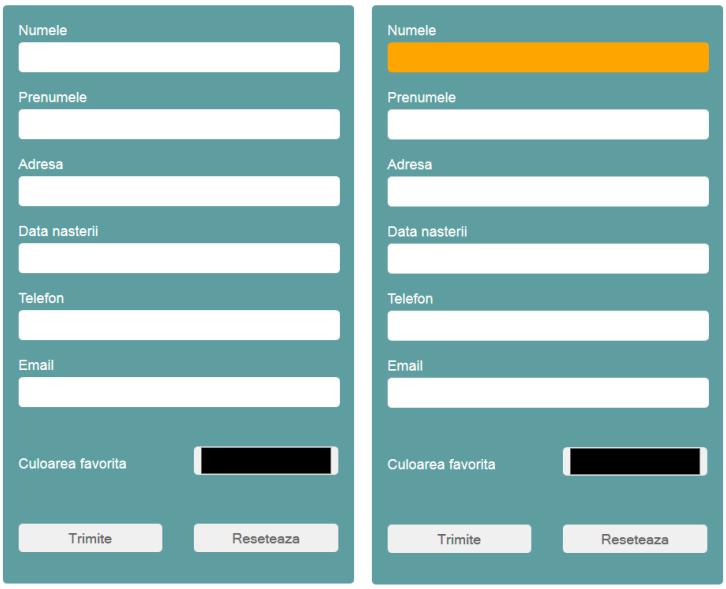
```
var x = 10;

alert(x);
x = “ab”;
alert(x);
```

a. Se va genera o eroare la execuție, pentru că variablia x este de tipul int și nu este permisă asignarea de valori șir de caractere (x = ”ab”)
b. Se va afișa: 10 și ab
c. Se va afișa: 10 și ab, după care un mesaj de eroare identic cu cel de la punctul a)
d. Nu se va afișa nimic, întrucât funcția alert nu are rolul de afișare de mesaje în limbajul JavaScript

3.2 Exemple de probleme practice

Exercițiul 1. Folosind tehnologiile HTML, CSS și JavaScript, realizați un formular de contact similar celui prezentat în imaginea de mai jos.



La părăsirea unui câmp din formular, (evenimentul onblur) verificați validitatea informațiilor introduse în acel câmp, conform condițiilor de mai jos

- **nume** trebuie să conțină cel puțin trei litere și să nu conțină cifre

- **prenume** trebuie să conțină cel puțin trei litere și să nu conțină cifre
- **adresa** trebuie să conțină cel puțin trei caractere și cel puțin o cifră. Nu sunt acceptate caracterele: @#$%^&*
- **data nasterii** trebuie să conțină o dată validă în format dd/mm/yyyy
- cîmpul **telefon** poate conține numai cifre, un caracter + și unul sau mai multe caractere - separate prin alte cifre
- câmpul email trebuie să accepte numai o adresă de email corectă din punct de vedere sintactic

În cazul în care valoarea introdusă într-un anumit câmp nu este validă, culoarea de fundal a cîmpului respectiv va fi schimbată la #FFA500. Altfel, ea va fi setată la #FFFFFF.

Exercițiul 2. Implementați funcționalități de restricționare a valorilor introduse de utilizator în câmpurile formularului de la exercițiul anterior, după cum urmează:

- în câmpurile **Nume** și **Prenume** nu pot fi introduse cifre de la tastatură
- în cîmpul **Data nașterii** sunt acceptate numai cifre și caracterul **/**
- în câmpul telefon pot fi introduse numai cifre și caracterele **+** și **–**
- lângă cîmpul **Parola** adăugați un buton denumit **Vezi parola**: cât timp acest buton este menținut apăsat cu mouse-ul, parola va fi afișată în clar

Exercițiul 3. Implementați o mini-galerie de imagini cu următoarele funcționalități:

- imaginile afișate au dimensiunea de 300 x 300 px, prima imagine fiind afișată la încărcarea paginii
- buton NEXT – încarcă imaginea următoare din galerie, în locul celei curent afișate
- buton PREV – încarcă imaginea anterioară din galerie, în locul celei curent afișate
- se va afișa de asemenea progresul în interiorul galeriei, sub forma: imagine curentă / total imagini

Exercițiul 4. Simulați o aplicație care necesită autentificare. Numele de utilizator (sub forma unei adrese de email) și parola acceptate de sistem sunt salvate în limbajul JavaScript. Implementați următoarele caracteristici:

- la încărcare, în centrul paginii vor fi afișate doar:

- textul: **Introduceți numele de utilizator și parola**
- un câmp text cu eticheta **Nume utilizator**
- un câmp de tip password cu eticheta **Parola**
- un buton intitulat **Autentificare**

- la apăsarea butonului Autentificare
  - verificați valorile introduse în câmpurile **Nume utilizator** și **Parola**
    - afișați mesaj de eroare dacă nu au fost introduse valori
    - colorați eticheta câmpului gol în culoarea #FF0000
  - afișați mesaj de eroare dacă valorile introduse nu sunt corecte
  - dacă valori introduse corespund cu cele predefinite, ascundeți formularul de autentificare și afișați în locul său mesajul „**Autentificare reușită!**” cu font: Helvetica 20px îngroșat, culoare: #00FF00

IMPLEMENTAREA INTERFEŢELOR UTILIZATOR ÎN TEHNOLOGII WINDOWS MOBILE

Laboratorul 5 – Noțiuni introductive

1 Introducere

În prezent există o varietate de tipuri de telefoane inteligente (smartphones) care oferă funcţionalităţi de înaltă performanţă în domeniile de comunicaţie. După ani de încercări, Microsoft revine pe piaţa dispozitivelor mobile cu sistemul de operare Windows Phone şi cu un nou stil de a descrie interacţiunea cu utilizatorul prin intermediul dispozitivelor mobile.

Spre deosebire de sit-urile Web, dezvoltarea interfeţelor utilizator pentru dispozitivele mobile presupune condensarea informaţiilor într-un spaţiu restrâns ca dimensiune (ecranul telefonului) şi prezentarea acestora într-un mod cât mai inteligibil pentru utilizatori.

1.1 Obiective

Lucrarea de față își propune prezentarea conceptuală a tehnologiei Windows Phone 7 cu aplicabilitate în dezvoltarea unei aplicaţii practice.

2 Considerații teoretice

Deşi sunt multe de discutat în ceea ce priveşte funcţionalităţile oferite de sistemele de operare Windows Phone, iOS, Android şi BlackBerry, această secţiune prezintă principalele caracteristici care diferenţiază Windows Phone de celelalte platforme existente.

2.1 Particularităţi hardware

Există o serie de producători de telefoane mobile (LG, HTC, Acer, etc) care folosesc acest sistem de operare, dar el poate fi întâlnit în special pe dispozitivele Nokia. Spre deosebire de alte sisteme de operare, Windows Phone 7 necesită existenţa a şase butoane hardware:

- Înapoi (Back): deschide vederea (view) anterioară;

- Start: deschide vederea iniţială (Start screen) afişată pe ecranul telefonului;
- Căutare (Search): operaţie bazată pe unealta de căutare Bing;
- Pornire/Oprire: are o dublă funcţionalitate de a deschide/închide ecranul telefonului prin apăsarea uşoară a butonului, şi de oprire/pornire a telefonului la acţiunea de apăsare îndelungată;
- Control de volum;
- Control cameră: pune în funcţiune camera harware a telefonului, acest buton fiind folosit şi pentru a face poze.

2.2 Interfaţa grafică Metro

Interfaţa grafică Metro reprezintă una dintre cele mai interesante caracteristici ale sistemului de operare Windows Phone 7. Aceasta foloseşte conceptul de Dale Active (Live Tiles) în locul tradiţionalelor pictograme, fiind inspirate din interfaţa Zune HD. Avantajul acestei reprezentări îl constituie vizualizarea în timp real a stării diferitelor aplicaţii, funcţii, resurse media (reprezentate de aceste dale), întrucât conţinutul acestora este actualizat în mod dinamic. Spre exemplu, apelurile pierdute sunt listate în dala Phone. Astfel, aplicaţiile Windows Phone nu necesită secţiuni speciale de notificare (ca şi în cazul iOS şi Android) deoarece informaţiile actualizate sunt prezentate în dala corespunzătoare aplicaţiei.

Aceste dale active pot fi poziţionate în diferite locaţii pe ecran, pot fi redimensionate în funcţie de importanţa informaţiilor afişate pentru utilizator, iar navigarea se face prin gestul de swipe care constă într-o secvenţă de două operaţii: tap, urmat de mişcarea stânga-dreapta a degetului.

Pentru o mai uşoară identificare a aplicaţiilor instalate, Windows Phone 7 afişează toate aceste resurse ordonate alfabetic. Prin funcţionalităţile pe care le oferă interfaţa Metro, se doreşte o creştere a utilizabilităţii sistemului de operare, fără a afecta performanţa de execuţie a aplicaţiilor.

3 Practici de implementare

Practicile de implementare descrise în această secţiune recomandă următoarele pentru dezvoltarea aplicaţiei practice Windows Phone 7:

3.1 Configurare proiect de tipul Windows Phone

1. Instalare unealtă Visual Studio 2010 cu Service Pack 1.

   Deşi există o serie de unelte de dezvoltare a aplicaţiilor Windows, în general se recomandă folosirea platformei Visual Studio care oferă un mediu compact şi profesional de implementare, compilare şi testare a acestor aplicaţii.
2. Instalare Windows Phone SDK 7.1 pentru dezvoltarea aplicaţiilor Windows Phone 7.
3. Creare proiect de tipul *Visual C# -> Silverlight for Windows Phone -> Windows Phone Application*.
4. Creare directoare *View*, *Model* şi *ViewModel*. Structura finală a proiectului trebuie să fie similară cu cea din Figura 5.1.

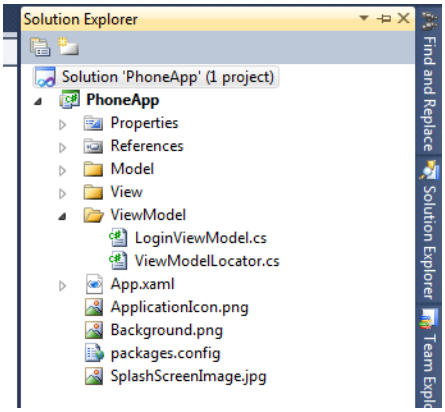


Figura 5.1: Structura proiectului

3.2 Elemente XAML

Limbajul XAML este folosit pentru declararea componentelor vizuale din cadrul tehnologiilor Microsoft. Este similar limbajului HTML, cu menţiunea că pune la dispoziţia programatorilor o gamă mult mai variată de componente grafice: butoane, etichete, tabele, module de organizare pe verticală şi orizontală etc.

Realizarea aplicaţiei practice presupune folosirea unor elemente XAML de organizare (layout), care permit aranjarea în pagină a căsuţelor text, a etichetelor şi a butoanelor. Pentru aceasta se pot folosi clasele Grid şi StackPanel. Prima

dintre acestea împarte întreg spaţiul în linii şi coloane a căror dimensiune poate fi specificată programatic, folosind proprietatea RowDefinition. Clasa StackPanel se foloseşte pentru a ordona vertical sau orizontal modulele componente.

Etichetele şi căsuţele text pot fi descrise tot prin intermediul elementelor XAML şi anume: TextBlock, respectiv TextBox. În cazul în care se doreşte codificarea parolei prin caractere de tipul *, se poate folosi clasa PasswordBox. Similar cu limbajul HTML, butoanele se pot defini ca şi elemente de tipul Button.

Există de asemenea şi posibilitatea de aplicare a unui set de atribute vizuale (stiluri) asupra unui grup de elemente XAML. Pentru aceasta se pot folosi clasele Style şi Setter. Există trei nivele de aplicabilitate a stilurilor:

- Pentru elementele din clasa în care au fost definite: spre exemplu, pentru stiluri declarate în cadrul clasei Grid, acestea au efect doar asupra elementelor din această clasă.
- Pentru elementele din întreaga pagină XAML.
- Pentru elementele din întregul proiect: dacă stilurile sunt definite în fişierul *App.xaml*.

3.3 Elemente de logică C#. Modelul MVVM (Model View ViewModel)

Bunele practici de dezvoltare a aplicaţiilor Window Phone recomandă o separare a elemetelor grafice de modelul de date, în care sunt păstrare valorile acestora. Astfel, în modelul de dezvoltare MVVM, legătura dintre componentele grafice din View şi obiectele din ViewModel se realizează prin intermediul conceptului de Binding. Spre exemplu dacă în clasa din View avem definiţia unei căsuţe text

```
<TextBox Text="{Binding Username}" />
```

iar în ViewModel avem proprietatea Username

```
private string _username;
public string Username
{
   get { return _username; }
   set { _username = value; }
}
```

atunci maparea se face in clasa View prin construcţia `Binding Username`. Operația de mapare presupune preluarea valorii *Username* și afișarea acesteia în căsuța text *TextBox*, în mod automat.

Deoarece *Username* este o proprietate a unei clase din categoria ViewModel, este necesară definirea explicită a conexiunii dintre instanța de tip View și cea de tip ViewModel pentru ca platforma WindowsPhone să poate identifica în mod corect proprietatea la care se face referire. În acest sens, este necesar ca în constructorul clasei de tip View să declarăm o instanță a clasei de tip ViewModel și să o conectăm la proprietatea DataContext:

```
class View
{
   private ViewModel _vm;
   public View
   {
      IntitializeComponent();

      _vm = new ViewModel();
      DataContext = _vm;
   }
}
```

Este important de menționat faptul că în modelul MVVM actualizarea interfeței se realizează de cele mai multe ori printr-o implementare a design-ului Observer. Astfel, orice modificare survenită în *ViewModel* trebuie notificată explicit interfeței grafice prin generarea unui eveniment la care aceasta din urmă este conectată. Pentru aceasta, clasa din ViewModel trebuie să implementeze interfața INotifyPropertyChanged, similar exemplului de mai jos:

```
class ViewModel : INotifyPropertyChanged
{
   public event PropertyChangedEventHandler PropertyChanged;
   private void NotifyPropertyChanged(String propertyName)
   {
      PropertyChangedEventHandler handler = PropertyChanged;
      if (null != handler)
      {
         handler(this, new PropertyChangedEventArgs(propertyName));
      }
   }
}
```

Până în acest moment, exemplele de mai sus definesc numai o legătură uni-direcţională, de la elementul din View la proprietatea *Username*. În cazul în care

conținutul elementului din *View* este modificat, proprietarea Username nu va fi actualizată. Pentru a corecta acest neajuns, vom modifica linia din fișierul XAML astfel:

```
<TextBox Text="{Binding Username, Mode=TwoWay}" />
```

În cazul în care se doreşte folosirea unei variabile de tipul X pentru a seta o proprietate a unui element XAML care acceptă numai valori de tipul Y se recomandă folosirea unui convertor care implementează interfața IValueConverter. Convertorul poate fi folosit în pagina XAML în interiorul etichetei *<phone:PhoneApplicationPage.Resources >* şi apelat ca şi o resursă statică. Implementarea convertorului se face în limbajul C#. Exemplul următor descrie modul în care se poate face o conversie a variabilei *VisibleText* de tipul Visibility la tipul String (aşteptat de clasa *TextBlock*), prin implementarea convertorului *VisibilityToString*.

```
<TextBlock Text="{Binding VisibleText,
                  Converter={StaticResource VisibilityToString}}" />
```

Pentru a simplifica arhitectura aplicaţiei, se recomandă folosirea unui obiect care să memoreze credenţiale statice de autentificare (spre exemplu Username = “user”, Password = “password”). Iniţializarea acestui obiect se poate face în constructorul clasei *App.xaml.cs*. Astfel, la fiecare acţiune de login se verifică valorile introduse de utilizator cu cele deja existente în acest obiect.

4 Exerciții

Exercițiul 1. Creați un nou proiect Windows Phone 7.0 și descrieți în XAML un ecran de autentificare cu următoarele componente:

- titlul "Autentificare" (font: 20px, culoare: #0000FF)
- textul "Nume utilizator" (font: 14px, culoare: #FFFFFF, aliniere la stânga)
- un câmp de intrare de tip **text**
- textul "Numele de utilizator nu poate fi vid. " (aliniat la stânga, font: 14px, culoare: #FF0000)
- textul "Parolă" (font: 14px, culoare: #FFFFFF, aliniat la stânga)
- un câmp de intrare de tip **password**

- textul "Parola nu poate fi vidă. " (aliniat la stânga, font: 14px, culoare: #FF0000)
- textul "Nume de utilizator / parolă incorecte. " (centrat orizontal, font: 14px, culoare: #FF0000)
- un buton cu titlul "Autentificare", centrat orizontal
- spațiere:
    - sub titlu: 60px
    - deasupra textului "Parolă": 40px
    - deasupra textului "Nume de utilizator / parolă incorecte": 20px
    - deasupra butonului **Autentificare**: 20px

Exercițiul 2. Adăugați următoarea funcționalitate aplicației de la punctul 1:

- la încărcarea aplicației, textul de culoare #FF0000 de pe ecran va fi ascuns
- la apăsarea butonului **Autentificare**:
    - dacă textul introdus în câmpul **Nume utilizator** este vid:
        - culoarea textului "Nume utilizator" este schimbată la #FF0000
        - textul "Numele de utilizator nu poate fi vid. " este afișat
    - dacă textul introdus în câmpul **Parolă** este vid:
        - culoarea textului "Parolă" este schimbată la #FF0000
        - textul "Parola nu poate fi vidă. " este afișat
    - dacă cele două câmpuri conțin valori
        - dacă valorile introduse sunt incorecte, textul "Nume de utilizator / parolă incorecte" este afișat
        - dacă valorile introduse sunt corecte, culoarea textului "Nume utilizator" și "Parolă" este schimbată la #00FF00

Restricții de implementare:

- în implementarea aplicației se va folosi modelul MVVM descris:
    - numele de utilizator și parola acceptate de aplicație vor fi descrise într-o clasă Model
    - caracteristicile de culoare și vizibilitate a textului vor fi conectate prin metoda **Binding** de proprietăți ale componentei ViewModel a aplicației

- verificările de validitate a datelor introduse se vor realiza în componenta ViewModel

Laboratorul 6 – utilizarea controalelor de tip listă

1 Introducere

În cadrul acestui laborator se urmăreşte adăugarea unui nou ecran care permite afişarea cărţilor din bibliotecă, sub formă de listă. Acest ecran devine vizibil numai în cazul autentificării cu succes a utilizatorului.

1.1 Obiective

Lucrarea de față își propune folosirea şi personalizarea listelor în aplicaţii de tipul Windows Phone 7.

2 Considerații teoretice

Practicile de implementare descrise în această secțiune recomandă următoarele pentru dezvoltarea aplicaţiei practice Windows Phone 7.

2.1 Organizarea elementelor vizuale

Deoarece dispozitivele care rulează Windows Phone dispun de mai multe rezoluții (variind de la 480x800 px până la 1080x1920 px, cu rapoarte de aspect de 15:9 sau 16:9), nu este recomandată plasarea și dimensionarea în coordonate absolute ale elementelor de interfață.

Pentru organizarea vizuală a elementelor, SDK-ul Windows Phone pune la dispoziția dezvoltatorilor mai multe structuri de organizare implicite, dintre care cele mai uzuale sunt:

2.1.1 Grid

Permite organizarea sub formă de grilă / tabel a elementelor vizuale. În cadrul fiecărei celule, plasarea elementelor se face în coordonate absolute (cu ajutorul proprietății *Margin*), ceea ce permite suprapunerea acestora. După definirea generică a structurii (număr și dimensiune coloane / rânduri) fiecare dintre elementele plasate în grilă își specifică celula în care va fi afișat, folosind atributele *Grid.Row* și *Grid.Column*.

În exemplul de mai jos se definește o grilă cu două rânduri și 3 coloane:

```
<Grid x:Name="LayoutRoot" Background="Transparent">
   <Grid.RowDefinitions>
      <RowDefinition Height="Auto"/>
      <RowDefinition Height="*"/>
   </Grid.RowDefinitions>

   <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="*" />
   </Grid.ColumnDefinitions>

   ...
   <TextBlock Grid.Row="1" Grid.Column="2">Celula 2</TextBlock>
   <TextBlock Grid.Row="2" Grid.Column="2">Celula 5</TextBlock>
</Grid>
```

Pentru a ocupa tot spațiul disponibil, înălțimea rândurilor și lățimea coloanelor nu este specificată în valori absolute. Simbolurile utilizate au următoarea semnificație:

Auto Înălțimea rândului sau lățimea coloanei este calculată în funcție de conținutul afișat. Astfel, toate elementele din interiorul containerului vor fi vizibile iar spațiul ocupat este minimul necesar pentru afișare.

***** Înălțimea rândului sau lățimea coloanei este calculată în funcție de spațiul rămas disponibil în containerul părinte, după afișarea elementelor cu dimensiuni absolute sau Auto. Spațiul rămas este împărțit în mod egal între toate elementele de același tip care folosesc acest simbol pentru stabilirea înălțimii sau lățimii.

Pentru specificarea proporțională a înălțimii / lățimii elementelor, simbolul ***** poate fi utilizat cu un multiplicator. În exemplul de mai jos, rândul al doilea va fi de două ori mai înalt decât primul rând:

```
<Grid.RowDefinitions>
   <RowDefinition Height="*"/>
   <RowDefinition Height="2*"/>
</Grid.RowDefinitions>
```

Mai multe informații despre organizarea dinamică a elementelor pot fi găsite în documentația oficială iar exemple de dimensionare pot fi consultate în articolul Understanding XAML Layout and Events.

2.1.2 StackPanel

Elementele vizuale incluse sunt plasate relativ unele la celelalte, fără a fi permise suprapunerile. Orientarea de afișare în cadrul containerului poate fi verticală sau orizontală (elementele sunt plasate unele sub celelalte sau unele după celelalte) în timp ce spațierile definite pentru elemente (ex. margini) sunt aplicate relativ la elementul anterior și nu la marginile containerului. Înălțimea elementelor de tip StackPanel este implicit setată la valoarea Auto, ea fiind actualizată în funcție de elementele vizuale plasate în interior. Lățimea elementelor de tip StackPanel este implicit setată la lățimea pusă la dispoziție de containerul părinte.

2.1.3 ScrollViewer

Este un container virtual care permite afișarea unor elemente mai mari decât ecranul prin implementarea implicită a interacțiunii de defilare.

2.2 Elemente XAML

Una dintre metodele de reprezentare a listelor poate fi realizată prin folosirea clasei ListBox. Aceasta permite definirea de elemente personalizate (similare cerinţei acestui laborator) prin intermediul componentelor ListBox.ItemTemplate și DataTemplate. Clasa DataTemplate permite organizarea elementelor de interfaţă folosind containere vizuale dintre cele descrise în secțiunea 2.1.

```
<ListBox x:Name="ListName" ItemsSource="{Binding ElementsList}">
   <ListBox.ItemTemplate >
      <DataTemplate >
         // elemente vizuale care reprezintă formatul unui element din
            listă
      </DataTemplate>
   </ListBox.ItemTemplate>
</ListBox>
```

De cele mai multe ori afişarea datelor în interfaţă presupune o etapă de colectare a acestora din diferite surse (de exemplu: baze de date, fişiere XML etc.). Atunci când există o întârziere între iniţierea cererii şi afişarea datelor, se recomandă folosirea unei bare de progres care să ofere utilizatorului o mai bună înţelegere a stării aplicaţiei. În Windows Phone 7 componenta corespunzătoare este ProgressBar.

2.3 Stiluri

Pentru a eficientiza volumul de muncă se pot folosi stiluri, care pot fi aplicate unui set de elemente vizuale, asigurând astfel și un aspect unitar al aplicaţiei. Pentru aceasta se recomandă folosirea claselor Style şi Setter, în care fiecare atribut este specificat prin cuvântul rezervat *Property*, iar valoarea acestuia prin *Value*.

```
<StackPanel Orientation="Horizontal">
   <StackPanel.Resources>
      <Style x:Name="MyStyle" TargetType="TextBlock" >
         <Setter Property="Margin" Value="0, 10, 0, 0" />
         <Setter Property="FontSize" Value="20" />
         <Setter Property="FontWeight" Value="Bold" />
      </Style>
   </StackPanel.Resources>

   <TextBlock Text="Formatted using styles"
              Style="{StaticResource MyStyle}"/>
   <TextBlock Text="Formatted using same style"
              Style="{StaticResource MyStyle}"/>
</StackPanel>
```

3 Practici de implementare

Pentru a respecta standardul MVVM se recomandă implementarea a trei componente și structurarea codului astfel:

Componenta	Detalii
View	Gestionează toate elementele vizuale ale interfeței (fișierul .xaml) și comportamentul acestora (fișierul .xaml.cs). Cele două fișiere ar trebui plasate în directorul *View* al proiectului.
ViewModel	Componenta care formatează informațiile din model pentru a fi vizualizate în interfața utilizator. Fișierul .cs va fi plasat în directorul *ViewModel*.
Model	Clasă sau grup de clase care simulează o bază de date. Valorile vor fi introduse manual în cod, pentru fiecare dintre elementele de date necesare.

3.1 Navigarea în aplicațiile Windows Phone

Pentru a încărca un nou ecran într-o aplicație Windows Phone, din cadrul unei componente View se apelează componenta NavigationService, astfel:

```
NavigationService.Navigate(new Uri("/TargetView.xaml",
                                   UriKind.Relative));
```

Se poate astfel concluziona că partea de navigare în cadrul unei aplicații Windows Phone este o componentă vizuală. În MVVM însă, deciziile care determină navigarea sunt cel mai adesea luate în ViewModel, deoarece aceasta este componenta care prelucrează informațiile și controlează comportamentul aplicației.

Pentru a evita referințele dinspre ViewModel înspre View există mai multe abordări implementate în biblioteci de unelte cum sunt MVVM Light Toolkit, nRoute, MicroModels, Composite WPF/Prism și altele. Pentru simplitate, în cazul nostru vom utiliza o clasă personalizată, conform modelului de mai jos:

```
using System;
using Microsoft.Phone.Controls;

namespace PhoneApp.Infrastructure
{
   public class MyNavigationService
   {
      private PhoneApplicationFrame frame;

      public MyNavigationService(PhoneApplicationFrame phoneAppFrame)
      {
         frame = phoneAppFrame;
      }

      public void Navigate(String pageName)
      {
         frame.Navigate(new Uri(pageName, UriKind.Relative));
      }
   }
}
```

Pentru a folosi această clasă în cadrul aplicaţiei, se recomandă implementarea unui model de bază care sa fie extins de toate celelalte componente *ViewModel*:

```
public class BaseViewModel
{
   public static NavigationService NavService;

   public BaseViewModel()
   {
      NavService = new NavigationService(
                                   ((App)Application.Current).RootFrame);
   }
```

```
}
```

Pentru a naviga la un nou ecran, în cadrul clasei ViewModel care extinde clasa de bază BaseViewModel vom apela:

```
NavService.Navigate("/TargetView.xaml");
```

4 Exerciții

Exercițiul 1. Realizați o aplicație pe tema „Biblioteca virtuală” în care cărțile disponibile sunt afișate sub forma:

Restricții de implementare:

- la încărcarea ecranului
  - va fi vizibilă numai o bară de progres cu comportament nedefinit
  - după două secunde, lista de cărți va deveni vizibilă iar indicatorul de progres va dispărea
- fiecare carte va fi reprezentată de un element în cadrul unui control de tip listă
- informațiile cărților afișate vor fi descrise în cadrul componentei Model a aplicației

Exercițiul 2. La selectarea unui element din lista descrisă la Exercițiul 1:

- schimbați culoarea de fundal în #00FF00, transparență de 30%
- schimbați culoarea textului pentru descrierea cărții în #000000
- schimbați culoarea textului pentru numele autorului în #00FF00

Laboratorul 7 – elemente avansate de interfață și interacțiune utilizator

1 Introducere

În cadrul acestui laborator se urmăreşte extinderea aplicației de gestiune a cărților dintr-o bibliotecă (începută în laboratoarele anterioare), prin introducerea secțiunii de detalii despre o carte, a posibilității de adăugare/ștergere de elemente din listă etc.

1.1 Obiective

Utilizarea conceptelor de meniu contextual, bară de aplicaţie, precum şi modificarea funcției implicite a evenimentelor din aplicaţiile de tipul Windows Phone 7.

2 Considerații teoretice

Practicile de implementare descrise în această secțiune recomandă următoarele concepte și soluții pentru dezvoltarea aplicaţiei practice Windows Phone 7.

2.1 Adăugare unei vederi noi

Pagina de detalii despre o carte presupune crearea a două clase în directorul *View* şi *ViewModel*, iar organizarea elementelor grafice se poate realiza prin intermediul claselor Grid şi StackPanel. În acest pas este necesară stabilirea conexiunii dintre *View* şi *ViewModel*, aceasta putând fi realizată în constructorul clasei *xaml.cs* prin setarea unui DataContext:

```
this.DataContext =
            ViewModelsFactory.GetViewModel("nume_clasa_ViewModel");
```

2.2 Utilizarea meniului contextual

Meniul contextul este un alt concept introdus în Windows Phone. Acesta devine vizibil la acţiunea de apăsare lungă (long press) pe elementele grafice dintr-un

ecran. Pentru aceasta trebuie inslatat pachetul *toolkit* folosind NuGet (acest proces este descris în primul laborator de Windows Phone şi în Figura 7.1).

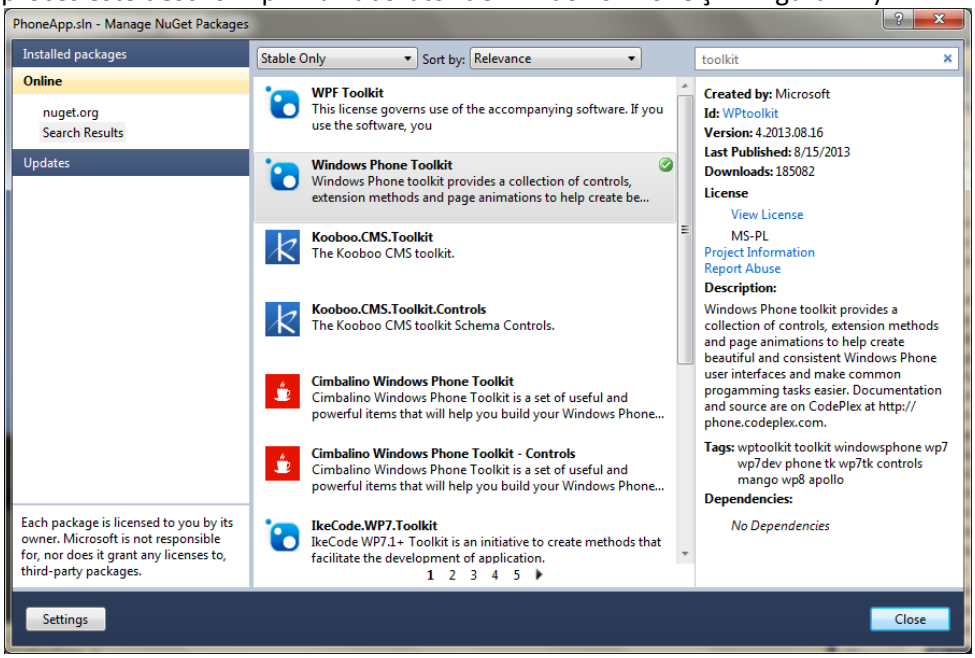


Figura 7.1: Instalarea pachetului *toolkit*

Pasul următor în includerea unui astfel de meniu contextual este adăugarea unei referinţe spre pachetul nou adăugat în cadrul vederii:

```
xmlns:toolkit="clr-namespace:Microsoft.Phone.Controls;
               assembly=Microsoft.Phone.Controls.Toolkit"
```

urmată de specificarea opţiunilor disponibile:

```
<toolkit:ContextMenuService.ContextMenu>
   <toolkit:ContextMenu>
      <toolkit:MenuItem Header="Option1" />
      <toolkit:MenuItem Header="Option2" />
   </toolkit:ContextMenu>
</toolkit:ContextMenuService.ContextMenu>
```

Evenimentele (clic, de exemplu) generate de aceste opţiuni pot fi gestionate în clasa *xaml.cs*, corepsunzătoare clasei *View*. Cu toate aceste bunele practici recomandă folosirea unei clase speciale care implementează interfaţa *ICommand* şi care permite această gestionare din *ViewModel*.

În cazul în care se implementează operaţii de adăugare sau ştergere a elementelor dintr-o listă se recomandă definirea acestei liste ca şi ObservableCollection în loc de List:

```
ObservableCollection<Item> ItemsList
```

Avantajul utilizării clasei *ObservableCollection* constă în faptul că elementele din *View* sunt notificate automat în cazul unor acţiuni de adăugare/ștergere.

2.3 Bara de aplicaţie (AppBar)

Este un concept specific platformei Windows Phone, care permite plasarea de imagini (în partea de jos a ecranului) care asigură accesul rapid la cele mai utilizate operaţii referitoare la resursele dintr-o anumită pagină. Adăugarea acestui control într-o pagină xaml se poate realiza astfel:

```
<phone:PhoneApplicationPage.ApplicationBar>
   <shell:ApplicationBar IsVisible="True" IsMenuEnabled="True">
      <shell:ApplicationBarIconButton IconUri="image_path"
                                      Text="Option1 " />
      <shell:ApplicationBarIconButton IconUri="image_ path"
                                      Text="Option2" />
   </shell:ApplicationBar>
</phone:PhoneApplicationPage.ApplicationBar>
```

De obicei imaginile utilizate trebuie să îndeplinească anumite condiţii de aspect şi dimeniune. Un set restrâns de astfel de imagini sunt stocate în locaţia:

```
C:\Program Files (x86)\Microsoft SDKs\Windows Phone\v7.1\Icons\
```

Înainte de utilizarea acestor imagini trebuie să ne asigurăm de faptul că acestea au fost importate în proiectul din Visual Studio şi că proprietatea *Build Action* (accesibilă din Visual Studio la clic dreapta pe imagine) are valoarea *Content*.

Gestionarea evenimentului de clic pe o opţiune din AppBar se poate realiza în clasa *xaml.cs* corespunzătoare vedereii care conţine acest modul:

```
private void ApplicationBarIconButton_Click(object sender,EventArgs e)
{

}
```

2.4 Lifecycle pentru aplicaţiile Windows Phone

Sistemul de operare Windows Phone asigură faptul că la un moment dat o singură aplicaţie este activă, în timp ce restul sunt în starea de aşteptare. Figura 7.2 evidenţiează stările unei aplicaţii Windows Phone:

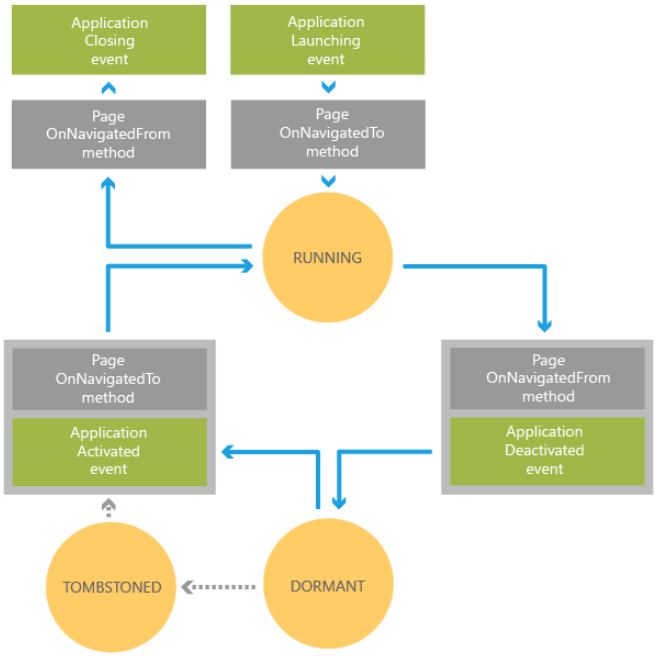


Figura 7.2: Stările unei aplicaţii Windows Phone [2]

- **Launching event**: deschiderea aplicaţiei de către utilizator.
- **Running**: starea de funcţionare rămâne până în momentul în care utilizatorul navighează cu butonul de Înapoi (Back) până înainte de prima vedere din aplicaţie.
- **Metoda OnNavigatedFrom**: această metodă se generează atunci când utilizatorul navighează din accea pagină.
- **Starea Deactivated**: acest eveniment este generat atunci când se apasă butonul de Start şi aplicaţia devine inactivă sau când se deschide o altă aplicaţie.

- **Dormant**: apare după starea *Deactivated* şi păstrează conţinutul memoriei corespunzătoare aplicaţiei.
- **Tombstoned**: sistemul de operare Windows Phone permite existenţa a cel mult 8 aplicaţii în starea *Dormant*. Atunci când apare o a noua aplicaţie, prima dintre acestea trece în starea *Tombstoned*, caz în care datele generate de aceastea sunt eliminate din memorie.
- **Metoda OnNavigatedTo**: această metodă se generează atunci când utilizatorul navighează spre accea pagină.
- **Evenimentul Close**: utilizatorul navighează cu butonul de Înapoi (Back) până înainte de prima vedere din aplicaţie.

3 Practici de implementare

Pentru a crea secţiunea de detalii a cărții selectate este nevoie de transmiterea informaţiilor aferente acesteia din vederea care conţine colecţia de cărţi în vederea cu detalii. Există două abordări în acest sens:

3.1.1 Navigarea prin intermediul clasei View (xaml.cs)

În clasa *xaml.cs* corespunzătoare vederii care conţine colecţia de cărţi este necesară implementarea evenimentului de schimbare a selecţiei (*SelectionChanged*). Navigarea spre vederea de detalii se face prin intermediul mecanismul intern Windows Phone, iar parametrii se pot transmite folosind noţiunea de query string, care conţine o listă de perechi cheie-valoare, precedată de caracterul “?”:

```
private void SelectionChanged (object sender,
                               SelectionChangedEventArgs e)
{
   this.NavigationService.Navigate(
            new Uri("/View/DetailsPage.xaml?param=selected_item",
                    UriKind.Relative)
   );
}
```

În vederea cu detalii despre cartea selectată se suprascrie metoda *OnNavigatedTo* în care se gestionează setul de parametrii trimiși anterior:

```
protected override void
OnNavigatedTo(System.Windows.Navigation.NavigationEventArgs e)
{
   base.OnNavigatedTo(e);
```

```
    string parameter = NavigationContext.QueryString["param"];
}
```

Similar se pot suprascrie și metodele care gestionează evenimentele de *OnNavigatedFrom* și *OnBackKeyPress*.

3.1.2 Navigarea prin intermediul clasei din ViewModel

Pentru aceasta se implementează o nouă clasă *NavigationService* în care sunt suprascrise evenimentele de Navigating, Navigated şi BackKeyPress. Această clasă conţine şi metoda *Navigate* care permite trecerea dintr-o vedere în alta, permiţând inclusiv transmiterea de parametrii. Definiţia completă a acestei clase este:

```
public class NavigationService
{
   private PhoneApplicationFrame frame;
   private object _param;

   public NavigationService(PhoneApplicationFrame phoneAppFrame)
   {
      frame = phoneAppFrame;
      frame.Navigated +=
               new System.Windows.Navigation.NavigatedEventHandler
                                                      (frame_Navigated);

      frame.Navigating +=
               new System.Windows.Navigation
                                .NavigatingCancelEventHandler
                                                       (frame_Navigating);
      frame.BackKeyPress +=
               new EventHandler<System.ComponentModel.CancelEventArgs>
                                                     (frame_BackKeyPress);
   }

   void frame_BackKeyPress(object sender,
                           System.ComponentModel.CancelEventArgs e)
   {

   }

   void frame_Navigating(object sender,
                         NavigatingCancelEventArgs e)
   {

   }

   void frame_Navigated(object sender, NavigationEventArgs e)
   {
```

```
    }

    public void Navigate(String pageName)
    {
        frame.Navigate(new Uri(pageName, UriKind.Relative));
    }

    public void Navigate(String pageName, object parameter)
    {
        _param = parameter;
        frame.Navigate(new Uri(pageName, UriKind.Relative));
    }
}
```

Instanţierea acestei clase se face o singură dată în *BaseViewModel*:

```
public static NavigationService NavService =
            new NavigationService(((App)Application.Current).RootFrame);
```

Se mai foloseşte o clasă Factory pentru a avea acces la instanţele *ViewModel* din fiecare vedere. Aceasta implementează şablonul Singletone, astfel încât se evită crearea mai multor instanţe de acelaşi tip:

```
public class ViewModelsFactory
{
    private static DetailsViewModel _detailsVM;
    public static DetailsViewModel DetailsVM
    {
        get
        {
            if (_detailsVM == null)
            {
                _ detailsVM = new DetailsViewModel();
            }
            return _ detailsVM;
        }
    }

    public static Object GetViewModel(string type)
    {
        switch (type)
        {
            case "option1":
                return DetailsVM;

            case "option2":
                return …;

            default:
                return DetailsVM;
        }
```

```
    }
}
```

În cazul nostru se doreşte transmiterea unui parametru (cartea selectată) din vederea care conţine colecţia de cărţi în vederea de detalii despre cartea respectivă. Pentru aceasta, în prima vedere se foloseşte clasa *NavigationService* astfel:

```
NavService.Navigate("/View/DetailsPage.xaml", selectedBook);
```

după care se modifică metoda *frame_Navigating* din *NavigationService*

```
void frame_Navigating(object sender, NavigatingCancelEventArgs e)
{
   ViewModelsFactory.GetViewModel(e.uri.toString())
                    .OnNavigatedTo(_param);
}
```

în timp ce în *ViewModel*-ul din a doua vedere se gestionează parametrul primit:

```
public void OnNavigatedTo(object param)
{

}
```

4 Exerciții

Exercițiul 1. Utilizând aplicația de la laboratorul precedent, creați un nou ecran, denumit "Detalii carte". Când o carte este selectată din lista de cărți afișată (conform instrucțiunilor din laboratoarele anterioare), detaliile despre ea vor fi prezentate în noul ecran.

Exercițiul 2. Numărați de câte ori este accesat ecranul de detalii, indiferent pentru care dintre cărți. Afișați acest număr într-o fereastră de dialog la fiecare deschidere a ecranului.

Laboratorul 8 – verificarea cunoștințelor în Tehnologii Windows Mobile

1 Introducere

În cadrul acestui laborator va fi evaluat nivelul înțelegerii noțiunilor din domeniul Windows Mobile prezentate în cadrul laboratoarelor anterioare. Fiecare dintre studenți va primi o notă ce va reflecta atât nivelul cunoștințelor teoretice însușite cât și nivelul abilităților practice dobândite.

1.1 Obiective

Stabilirea măsurii în care noțiunile fundamentale din domeniul Windows Mobile au fost însușite de către cursanți.

2 Metodologia de evaluare

Evaluarea cunoștințelor dobândite în timpul lucrărilor practice de laborator din domeniul Windows Mobile se va realiza pe două niveluri:

2.1 Evaluarea teoretică

Presupune testarea înțelegerii noțiunilor teoretice de bază, care permite atât profesorului cât și studentului identificarea eventualelor puncte neclare în însușirea cunoștințelor elementare.

Această evaluare se va desfășura prin intermediul unui chestionar cu aproximativ 10 - 15 întrebări care au multiple variante de răspuns, nota finală obținută fiind accesibilă imediat după completarea chestionarului, prin corectare automată.

Fiecare dintre participanții la laborator vor avea acces la forma electronică a chestionarului pentru un timp limitat, contorizat automat, și numai de pe contul propriu din cadrul platformei de gestiune a materialelor de laborator. Întrebările

pentru fiecare chestionar vor fi selecționate aleator din cadrul unei baze de date iar răspunsurile fiecărei întrebări vor fi de asemenea ordonate aleator.

2.2 Evaluarea practică

Are ca și scop verificarea abilității studenților de a aplica noțiunile învățate în vederea dezvoltării unor interfețe utilizator în tehnologia Windows Mobile. Fiecăruia dintre participanți îi va fi asignat un exercițiu, având o oră și jumătate timp de rezolvare.

Implementarea prezentată va fi evaluată conform următoarelor criterii generale:

- **structura aplicației** – implementarea soluției trebuie să respecte design-pattern-ul arhitectural Model-View-ViewModel
- **interfața utilizator**
    - gradul în care formatarea vizuală a elementelor de interfață respectă cerințele exercițiului
    - fluiditatea designului (adaptare la orientarea dispozitivului)
    - respectarea recomandărilor de design a interfețelor utilizator și a interacțiunilor cu acesta
- **interacțiune utilizator** – în ce măsură implementarea realizată îndeplinește cerințele de:
    - verificare a validității datelor
    - prevenirea erorilor și sprijinirea utilizatorului în revenirea dintr-o situație de eroare
    - gestiune corectă a butoanelor hardware
    - etc.
- **respectarea recomandărilor de bune practici**
    - definirea și utilizarea stilurilor pentru formatarea elementelor vizuale
    - separarea corectă a componentelor de View, Model și ViewModel
    - etc.

La finalul timpului de lucru, fiecare dintre studenți prezintă cadrului didactic implementarea efectuată și fundamentele teoretice care au stat la baza deciziilor de implementare.

3 Exerciții

3.1 Exemple de întrebări pentru verificarea cunoștințelor teoretice

1. **Alegeți afirmațiile corecte despre conceptul de Dale active (Live tiles):**
   a. Permit afișarea diferitelor mesaje care reflectă starea aplicației
   b. Identifică pictograma aplicației
   c. Pot fi redimensionate la dimensiuni prestabilite
   d. Nu fac parte din interfața Metro

2. **Care dintre următoarele limbaje sunt folosite pentru dezvoltarea aplicațiilor Windows Phone?**
   a. Java
   b. C#
   c. MXML
   d. XAML

3. **Specificați nivelul de aplicabilitate a stilurilor definite în fișierul App.xaml:**
   a. Numai în cadrul fișierului xaml care este cel mai apropiat (în structura de directoare) de clasa App.xaml
   b. La niveul întregului proiect
   c. Stilurile sunt valabile doar pentru containerele care permit gruparea de elemente grafice, cum ar fi: Grid, StackPanel etc.
   d. Au efect doar asupra elementelor care nu au deja definite stiluri locale

4. **Ce se poate spune despre următoare secvență de cod?**

```
<StackPanel>
  <StackPanel.Resources>
    <Style Target=”TextBlock”>
      <Setter Property=”Foreground” Value=”Red” />
    </Style>
  </StackPanel.Resources>

  <TextBlock Text=”PIU1” />
  <TextBlock Foreground=”Green”>PIU2</TextBlock>
  <TextBlock>PIU3</TextBlock>
```

```
</StackPanel>
```

a. Culoarea textului va fi roșie pentru fiecare TextBlock
b. Se va afișa PIU1 și PIU3 cu roșu, iar PIU2 cu verde
c. Stilul definit în secțiunea de resurse nu are efect dacă nu e adăugat explicit pentru un TextBlock
d. Se va genera o eroare de compilare pentru că stilul definit la resurse nu are setată proprietatea x:Key

5. **Alegeți dintre opțiunile de mai jos pe cele care reprezintă moduri valide de mapare (binding):**
   a. OneWay
   b. ManyToMany
   c. TwoWay
   d. OneToMany

6. **Care dintre următoarele secvențe de cod are ca și rezultat afișarea textului ”PIU” pe un buton?**
   a. <Button Content=”PIU” />
   b. <Button x:Name=”PIU” />
   c. <Button>PIU</Button>
   d.
```
<Button>
    <TextBlock>PIU</TextBlock>
</Button>
```

7. **Specificați lățimea în pixeli a fiecărei coloane definite în următoarea** secvență XAML:

```
<Grid Width=”500”>
  <Grid.ColumnDefinitions>
      <ColumnDefinition Width="*" />
      <ColumnDefinition Width="Auto" />
      <ColumnDefinition Width="2*" />
  </Grid.ColumnDefinitions>

  <StackPanel Grid.Column=”1” Width=”200” />
</Grid>
```

   a. 0, 200, 0
   b. 100, 200, 200
   c. 200
   d. 500

8. **Care dintre următoarele variante de răspuns sunt adevărate?**
   a. StackPanel permite poziționarea relativă a componentelor interioare
   b. Este permisă definirea mai multor coloane în container-ul StackPanel
   c. Plasarea elementelor într-un container Grid se face în coordonate absolute
   d. Atât StackPanel cât și Grid permit accesarea elementelor interioare, folosind proprietatea Children

9. **Care dintre interfețe permite actualizarea în timp real a claselor View, atunci când au loc modificări în ViewModel?**
   a. IRaisePropertyChanged
   b. IBindingChanged
   c. INotifyPropertyChanged
   d. IObservableObjectChanged

10. **Un convertor are rolul de a schimba valoarea afișată în interfața grafică pe baza unei proprietăți sau a unui parametru:**
    a. Adevărat
    b. Fals

11. **Care dintre stările de mai jos indică funcționarea activă a aplicațiilor Windows?**
    a. Launching
    b. Dormant
    c. Running
    d. OnNavigatedFrom

12. **Selectați denumirea clasei care se folosește pentru afișarea unei liste:**
    a. ListBox
    b. List
    c. HorizontalList
    d. ListItem

13. **Unde este poziționată bara de opțiuni într-o aplicație Windows Phone?**
    a. În partea de sus a ecranului
    b. În josul ecranului
    c. În dreapta

d. În stânga

14. **Legătura dintre ViewModel și View, în cadrul aplicațiilor Windows Phone se realizează prin:**
    a. Setare DataContext în pagina dorită
    b. Referirea în ViewModel a View-ului
    c. Maparea (binding) la proprietățile din ViewModel
    d. Folosirea variabilelor statice în clasa ViewModel

15. **Ce se poate afirma cu privire la secvențele de cod:**

```
View:
<TextBlock Text="{Binding MyInput}" />
<TextBox Text="{Binding MyInput, Mode=TwoWay}" />

ViewModel:
private string _myInput;
public string MyInput
{
   get {return _myInput;}
   set {
      if (string.IsNullOrEmpty(value))
         _myInput = "Introduceti o valoare";
     else
         _myInput = value;
     NotifyPropertyChanged("MyInput");
   }
}
```

    a. Valoarea inițială a elementului TextBlock este " Introduceti o valoare"
    b. Modificarea proprietății MyInput din ViewModel nu se reflectă în View
    c. Valorile introduse în TextBox vor fi vizibile automat și în elementul TextBlock
    d. Valorile introduse în TextBox NU vor fi vizibile automat și în elementul TextBlock

3.2 Exemple de probleme practice

Exercițiul 1. Creați o aplicație cu tema "Biblioteca virtuală", cu următoarele caracteristici:

- utilizatorul trebuie să se autentifice înainte de a putea vizualiza lista de cărți, folosind o adresă de email și o parolă; datele de autentificare corecte vor fi salvate în componenta Model a aplicației
- lista de cărți disponibile va fi afișată conform designului din Laboratorul 6
- la selectarea unei cărți din listă, va fi deschis un nou ecran de detalii cu informații despre cartea aleasă

Exercițiul 2. Creați un **"meniu de context"** activ pentru elementele listei de cărți, cu următoarele opțiuni:

- **Adaugă** - la apăsare se afișează un nou ecran care permite adăugarea unei noi intrări în lista de cărți; imaginea utilizată va fi una generică
- **Șterge** - la apăsare, elementul selectat din listă este șters

Exercițiul 3. Creați un **"meniu aplicație"**, vizibil numai pe pagina de detalii, cu două opțiuni:

- **Adaugă** - la apăsare cartea curent afișată este adăugată la o listă de Preferințe și este afișat un mesaj de confirmare
- **Șterge** - la apăsare
    - dacă în lista de preferințe există cartea curent afișată, aceasta este ștearsă
    - dacă în lista de preferințe nu poate fi găsită cartea curent afișată, un mesaj de eroare este prezentat utilizatorului

Exercițiul 4. Modificați comportamentul butonului hardware **Back** astfel încât, la apăsarea pe ecranul **"Listă cărți"** utilizatorul este solicitat să confirme acțiunea **Log out** prin intermediul unei ferestre de dialog sistem:

- dacă utilizatorul apasă **Cancel/No** acțiunea de **Back** este anulată
- dacă utilizatorul apasă **Yes**, este prezentat ecranul de **Autentificare**.

Exercițiul 5. Creați un ecran intitulat **"Listă de preferințe"**, accesibil prin intermediul **"meniului aplicație"** definit pe ecranul **"Listă cărți".** În cadrul acestui ecran vor fi afișate Titlul și Autorul cărților care au fost adăugate la lista de preferințe. Prin intermediul unui **"meniu contextual"** utilizatorul poate elimina oricare dintre cărțile din listă.

IMPLEMENTAREA INTERFEŢELOR UTILIZATOR ÎN TEHNOLOGII ANDROID

Laboratorul 9 – Noțiuni introductive

1 Introducere

Dezvoltarea de aplicații în tehnologia Android presupune modelarea interfeței utilizator în limbajul XML specific platformei și scrierea codului aplicației în limbajul Java. Principalele unelte de dezvoltare folosite sunt:

- Android SDK – pentru lucrările de laborator vom utiliza versiunea 4.4.2 (API 19)
- Android Development Tools (ADT Plugin) – mediul de dezvoltare creat pentru Android pe baza aplicației Eclipse
- Android Development Studio – noul mediu oficial de dezvoltare a aplicațiilor Android

1.1 Obiective

Prezentarea noțiunilor introductive în dezvoltarea unei aplicații Android și realizarea în cursul orelor de laborator a unei prime aplicații.

2 Considerații teoretice

Una dintre caracteristicile principale ale dispozitivelor pe care rulează sistemele de operare Android este **diversitatea**. Aceasta se manifestă în:

- configurațiile hardware (procesor, memorie, spațiu de stocare, senzori disponibili etc.)
- rezoluțiile disponibile
- numărul și funcționalitatea butoanelor hardware disponibile

O aplicație de succes trebuie să se adapteze cât mai bine condițiilor asigurate de fiecare dispozitiv și să ofere întotdeauna un maximum de funcționalitate.

2.1 Configurațiile hardware

Sistemul de operare Android este utilizat pe o gamă foarte largă și variată de dispozitive: televizoare, telefoane mobile, tablete, ceasuri etc. O listă completă a dispozitivelor compatibile cu Google Play, actualizată la data de 02.12.2014 poate fi consultată la adresa: https://storage.googleapis.com/support-kms-prod/35E61236E9B2475D8E9C9F58AD1002A7CECE

2.2 Dezvoltarea de aplicații Android pentru mai multe rezoluții

- **dimensiunea ecranului** – indică dimensiunea fizică a acestuia, măsurată pe diagonală și exprimată cel mai adesea în *inch*. Pentru simplitate, Android grupează ecranele în *mici*, *normale*, *mari* și *foarte mari*.
- **densitatea ecranului** – indică numărul de pixeli localizați într-o unitate de suprafață, exprimată cel mai adesea în *dots-per-inch (dpi).* Pentru simplitate, Android grupează densitățile în *scăzute*, *normale* și *mari*.
- **orientare** – indică orientarea dispozitivului în raport cu utilizatorul.
- **rezoluție** – reprezintă numărul total de pixeli ai afișajului.
- **pixel independent de densitate (density-independent pixel)** – reprezintă un pixel virtual utilizat în descrierea interfeței utilizator pentru a asigura independența față de caracteristicile fizice ale afișajelor.
    - 1 dp = 1 px pentru un ecran cu densitatea de 160 dpi
    - `px = dp * (dpi / 160)`

2.2.1 Tipuri de afișaje suportate de Android

Pentru simplificarea dezvoltării aplicațiilor, Android utilizează următoarea clasificare a afișajelor:

- ldpi (low) ~120dpi
- mdpi (medium) ~160dpi
- hdpi (high) ~240dpi
- xhdpi (extra-high) ~320dpi
- xxhdpi (extra-extra-high) ~480dpi
- xxxhdpi (extra-extra-extra-high) ~640dpi

3 Practici de implementare

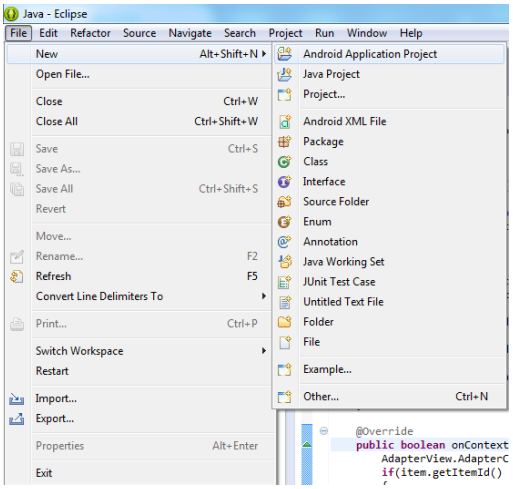
În lucrările de laborator pentru tehnologia Android vom utiliza ca și mediu de dezvoltare Android Development Tools.

3.1 Crearea unei noi aplicații Android

Pentru crearea unei noi aplicații, urmați pașii de mai jos:

3.1.1 Deschiderea aplicației Eclipse cu plugin-ul ADT instalat

3.1.2 Crearea unui nou proiect, de tipul Android Application Project



3.1.3 Introduceți informațiile specifice aplicației

- ***Application Name*** – reprezintă numele cu care APP-ul va fi afișat pe telefon
- ***Project Name*** – numele proiectului în Eclipse. Acest câmp este completat automat cu valoarea din câmpul *Application Name* fără spații
- ***Package Name*** – la fel ca și la alte aplicații Java, clasele sunt organizate într-un pachet cu denumire unică, specific organizației și aplicației. În cazul nostru, organizația este cea implicită: *example*
- ***Minimum Required SDK*** – reprezintă versiunea minimă de Android pe care va rula APP-ul. În selectarea acestei opțiuni trebuie luat în considerare faptul că, fără a avea în vedere dezvoltarea de cod specific

anumitor versiuni de Android, funcțiile disponibile aplicației sunt cele din SDK-ul minim

- ***Target SDK*** – reprezintă versiunea de Android pentru care această aplicație este optimizată
- ***Compile With*** – versiunea de SDK utilizată pentru compilarea aplicației
- ***Theme*** – tema implicită pentru care este optimizată aplicația

NOTĂ: pentru dezvoltarea aplicațiilor de laborator noi vom utiliza Android 4.4 (KitKat), API 19.

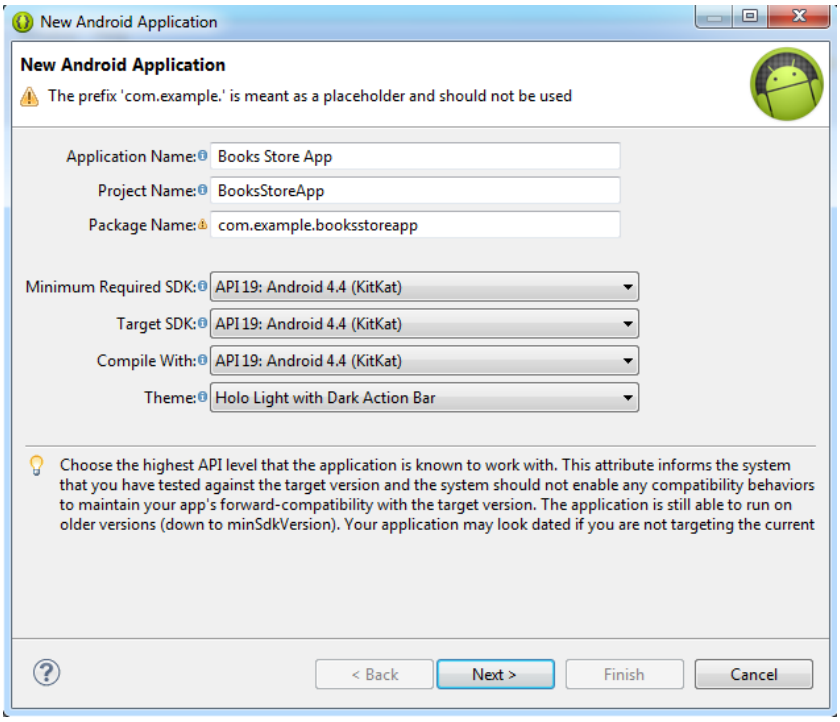
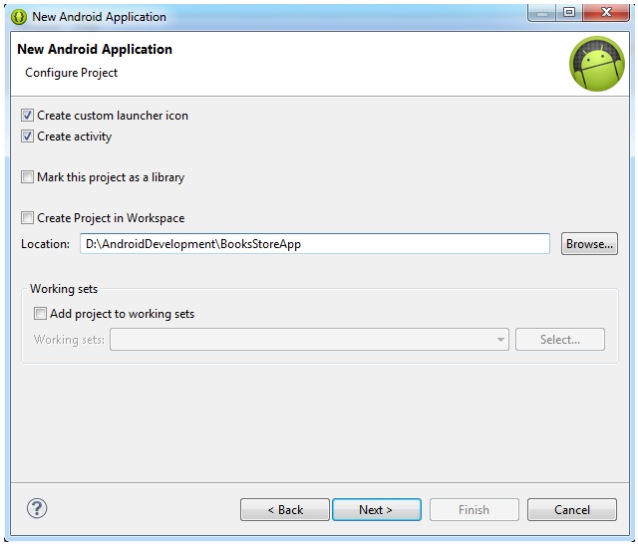
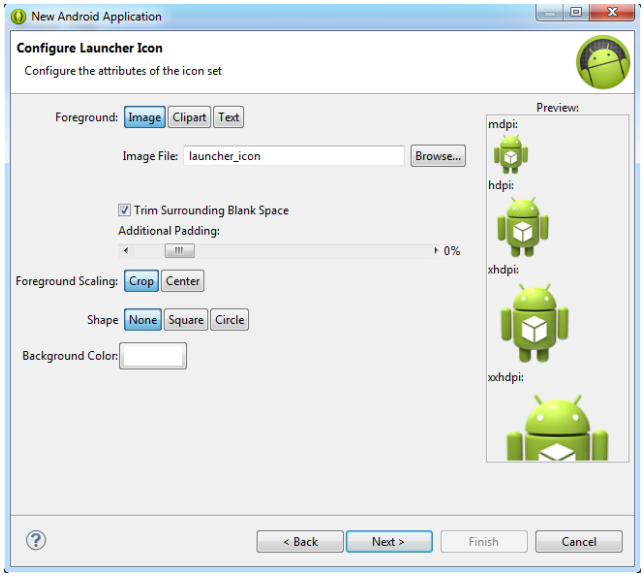


Figura 9.2: Stabilirea numelui aplicației și a versiunilor de Android pe care aceasta va rula

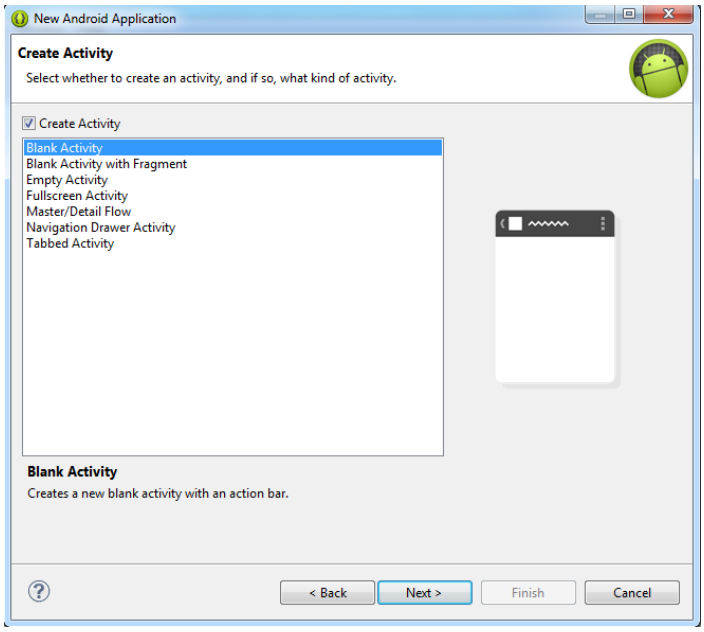
3.1.4 Definirea directorului în care este salvată aplicația și optarea pentru crearea unei pictograme a aplicației și a unei activități în cadrul etapelor următoare din tutorialul curent



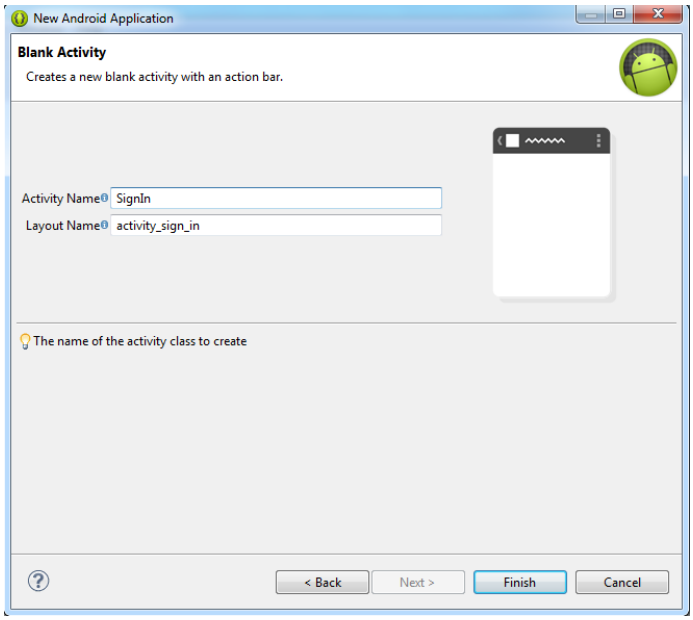
3.1.5 Personalizarea pictogramei aplicației și definirea fișierelor imagine optimizate pentru diferitele rezoluții ale dispozitivelor



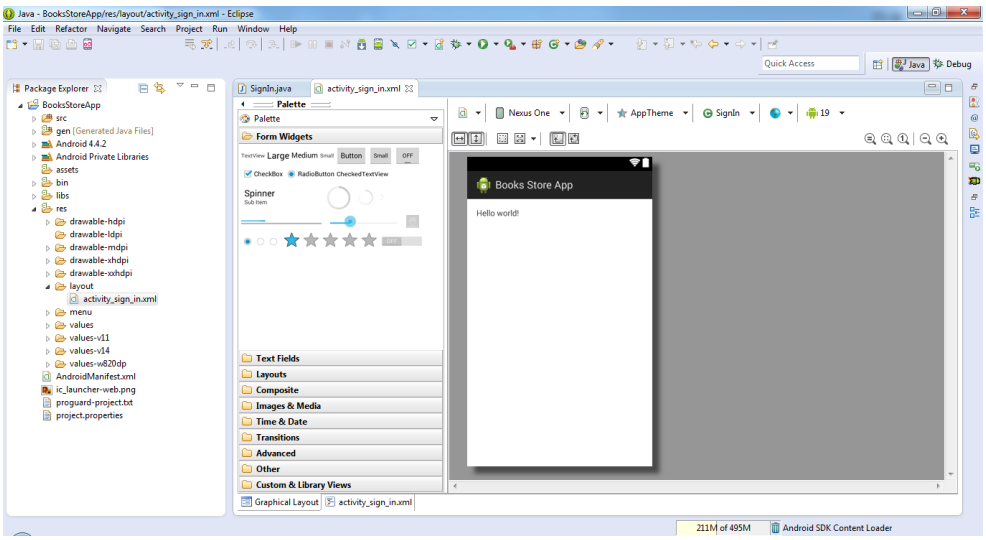
3.1.6 Selectarea tipului de activitate pe care îl dorim creat de către tutorial



3.1.7 Introducerea numelui activității și a denumirii fișierului care va descrie interfața utilizator



3.1.8 Rezultatul obținut



3.2 Structura generică a unei aplicații Android

După cum se poate observa din Figura 9.1, secțiunea **A** din structura de directoare a aplicației Android este similară cu a altor aplicații Java, cuprinzând:

- directorul **src** în care sursele aplicației sunt organizate pe pachete
- referințele către librării externe (în cazul nostru Android 4.4.2 și altele)
- directorul **bin** cu versiunea executabilă a proiectului

Secțiunea **B** conține un set de directoare specifice aplicațiilor Android, cu denumiri prestabilite (care nu pot fi modificate) și cu următoarea semnificație:

- **res** – directorul rădăcină al secțiunii; în acest director pot fi adăugate și alte tipuri de resurse.

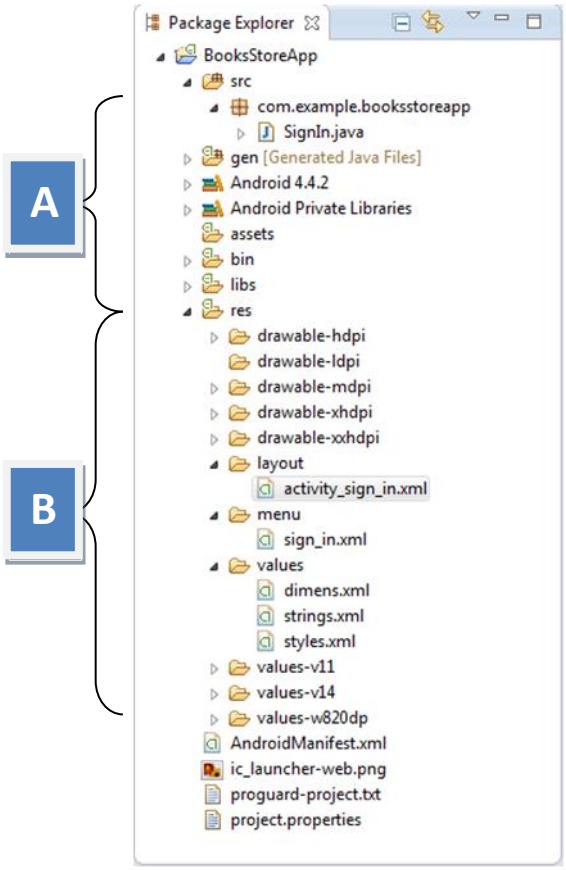


Figure 9.1: Structura generică de fișiere a unei aplicații Android

- **drawable**-**** - directoare care conțin elemente afișabile pe ecranul dispozitivului (imagini, setări vizuale etc.), cu versiuni optimizate pentru diferite rezoluții de afișare (vezi secțiunea...)
- **layout** – este directorul care cuprinde elementele de organizare a interfeței utilizator, descrise în limbajul propriu al sistemului Android, limbaj bazat pe XML
- **menu** – cuprinde toate fișierele XML care descriu structura unui meniu
- **values-v****** - cuprind fișiere XML în care sunt declarate elemente statice ale aplicației, cum sunt: dimensiuni, stiluri, text, constante etc. Modul de utilizare a informațiilor este următorul: fiecare versiune Android va căuta informațiile statice referite în directorul cu valoarea mai mică cea mai apropiată de versiunea sa. Cu alte cuvinte, în cazul nostru:

    - **values-v14** – este primul director verificat de către versiunile de Android cu API >= 14. Dacă valorile căutate nu sunt identificate aici, sistemul va căuta, descrescător, în restul directoarelor
    - **values-v11** – este primul director verificat de către versiunile de Android cu API < 14
    - **values** – este ultimul director verificat, în cazul în care valorile căutate nu au fost identificate în celelalte directoare
- **values-w820dp** – este fișierul de valori dedicat dispozitivelor cu rezoluție de afișare pe lățime >= cu 820dp

Fișierul **AndroidManifes.xml** cuprinde toate elementele descriptive la nivel de aplicație care sunt necesare dispozitivului pentru instalare, determinarea compatibilităților și acordarea de diferite drepturi de acces. Detalii despre structura și informațiile cuprinse în acest fișier pot fi studiate la adresa: http://developer.android.com/guide/topics/manifest/manifest-intro.html

3.3 Descrierea interfeței utilizator

Definirea interfeței utilizator se realizează în fișierele XML din directorul **/src/layout/** și include elemente vizuale din directoarele **drawable,** meniuri descrise în directorul **menu** sau constante și stiluri specificate în directoarele **values.** Pentru organizarea vizuală a elementelor se utilizează containere invizibile, care definesc anumite reguli implicite de organizare a componentelor pe care le conțin și permit atașarea de atribute XML specifice pentru entitățile copil (ex: layout_alignLeft, layout_height).

În cadrul lucrării de laborator vom utiliza cu precădere două astfel de containere (vezi Figura 9.2):

3.3.1 Linear Layout

Ordonează elementele vizuale pe care le conține unele după celelalte, fără a permite suprapuneri, fie în direcție verticală fie orizontală. Atributele XML disponibile entităților copil pot fi consultate la adresa: http://developer.android.com/guide/topics/ui/layout/linear.html

3.3.2 Relative Layout

Permite ordonarea elementelor vizuale prin poziționare relativă unele față de celelalte. Atributele XML disponibile entităților copil pot fi consultate la adresa: http://developer.android.com/guide/topics/ui/layout/relative.html

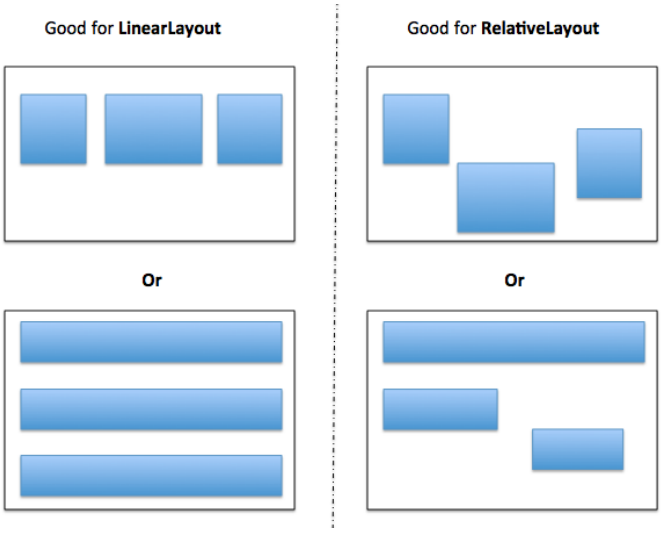


Figura 9.2: Exemplificarea ordonării elementelor vizuale folosing Linearlayout sau RelativeLayout

3.4 Funcționalitatea interfeței utilizator

Pentru a realiza legătura dintre modelul de date al aplicației și afișarea informațiilor este necesară scrierea de cod Java. Este esențial de reținut faptul că întreaga structură a directorului **/res/** este mapată de către SDK-ul Android într-o structură Java denumită generic **R**, sub forma unor resurse de tip **int**. Organizarea diferitelor elemente este realizată pe categorii, astfel:

- fiecare dintre ID-urile definite în cadrul fișierelor XML de layout sub forma *android:id="@+id/valoareID"* sunt accesibile prin *R.id.valoareID*
- fiecare dintre resursele grafice este accesibilă prin *R.drawable.nume_fisier_fara_extensie*
- fiecare dintre fișierele de interfață este accesibil prin *R.layout.nume_fisier_fara_extensie*
- etc.

Pentru a avea acces la resursele incluse în SDK-ul folosit pentru aplicație, se va utiliza instrucțiunea

```
import android.R
```

iar pentru a accesa resursele aplicației se va utiliza instrucțiunea

```
import com.example.booksstoreapp.R;
```

3.4.1 Accesarea elementelor vizuale

Pentru a obține o referință din codul Java la un element vizual este necesară în primul rând parsarea fișierului XML și transformarea structurii acestuia în obiecte Java. Acest lucru se realizează în cadrul funcției onCreate, la linia:

```
setContentView(R.layout.activity_sign_in);
```

După executarea cu succes a comenzii de mai sus, orice element vizual poate fi accesat prin comanda:

```
TipElement variabilă = (TipElement) findViewById(R.id.TVErrorPassword);
```

3.4.2 Atașarea de funcții la interacțiunile utilizator

Pentru a atașa o funcție la un eveniment onClick, este necesar ca în fișierul XML să definim atributul *android:onClick* pentru elementul vizual dorit.

```
<Button
   android:id="@+id/button1"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:layout_marginTop="100dp"
   android:minWidth="300dip"
   android:onClick="SignInClick"
   android:text="Sign in" />
```

În clasele Java care vor extinde acest layout este necesară definirea funcției SignInClick, astfel:

```
public void SignInClick(View view)
{

}
```

4 Exerciții

Exercițiul 1. Creați un nou proiect Android și descrieți un ecran de autentificare cu următoarele caracteristici:

- titlul ecranului - font: 20px, culoare: #0000FF, aliniere centrată orizontal
- textul din cadrul ecranului:

    - etichetele câmpurilor de introducere a datelor: font: 14px, culoare: #FFFFFF, aliniere la stânga
    - mesajele de eroare: font: 14px, culoare: #FF0000, aliniere la stânga
- câmpurile pentru introducerea datelor
    - un câmp de tip **text** pentru introducerea numelui de utilizator în format email
    - un câmp de tip **password** pentru introducerea parolei
- butonul de **Autentificare**, dimensiune 400px, centrat orizontal
- spațiere:
    - sub titlu: 60px
    - deasupra etichetelor pentru câmpurile de intrare: 40px
    - deasupra butonului **Autentificare**: 20px
- afișare la rotirea dispozitivului:
    - titlul trebuie să rămână centrat vertical
    - câmpurile de intrare trebuie să se redimensioneze pentru a ocupa tot spațiul orizontal disponibil

Exercițiul 2. La apăsarea butonului **Autentificare,** adăugați următoarea funcționalitate pentru aplicația de la punctul 1:

- valorile introduse vor fi verificate după cum urmează
    - **numele de utilizator** și **parola** nu pot fi vide
    - **numele de utilizator** trebuie să fie o adresă de email corectă sintactic
    - **parola** trebuie să
        - conțină cel puțin o literă mare și o cifră
        - să aibă cel puțin 5 caractere
- dacă valorile introduse sunt valide
    - dacă sunt corecte: se va afișa textul „Autentificare reușită” cu culoarea #00FF00
    - dacă sunt incorecte: se va afișa textul „Autentificare eșuată. Nume utilizator sau parolă invalide!” cu culoarea #FF0000

Restricții de implementare:

- pentru fiecare dintre erorile de mai sus, va fi afișat un mesaj corespunzător imediat sub câmpul de intrare verificat
    - sub câmpul **Nume utilizator** se vor afișa, după caz, mesajele
        - Numele de utilizator nu poate fi vid
        - Introduceți o adresă de email validă
    - sub câmpul **Parolă** se vor afișa, după caz, mesajele:
        - Parola nu poate fi vidă
        - Parola introdusă este prea scurtă
        - Parola trebuie să conțină cel puțin o literă mare și cel puțin o cifră
    - mesajul pentru autentificare reușită / eșuată va fi afișat deasupra butonului **Autentificare**

Laboratorul 10 – utilizarea controalelor de tip listă

1 Introducere

Ca și în cazul altor tehnologii orientate spre interfețe mobile, Android utilizează frecvent pentru afișarea informațiilor controale bazate pe liste de elemente. Pentru interacționarea cu anumite elemente dintr-o listă există posibilitatea definirii unui meniu contextual, care apare la gestul de apăsare prelungită a unui element.

1.1 Obiective

Folosirea şi personalizarea listelor în aplicaţiile Android. Definirea meniurilor contextuale.

2 Considerații teoretice

2.1 Crearea listelor personalizate

Pentru realizarea unei liste cu elemente personalizate în tehnologia Android este necesară îndeplinirea următorilor pași:

2.1.1 Descrierea structurii vizuale a unui element din listă

Fiecare element din listă este descris prin intermediul unui fișier XML plasat în directorul **res/layout/**.

2.1.2 Realizarea unui adaptor pentru listă

Legătura dintre modelul de date și structura vizuală a fiecărui element din listă este realizată prin intermediul unei clase Adapter. După adăugarea / eliminarea unui element din listă este necesară actualizarea adaptorului și apoi notificarea Listei conectate la acesta.

Pentru definirea unei clase Adapter personalizate este necesară implementarea unei interfețe specifice, definită ca și o clasă virtuală în SDK-ul Android. Astfel, este necesară moștenirea din clasa BaseAdapter (sau una din subclasele acesteia) și implementarea funcțiilor:

getCount returnează numărul de elemente din colecție și implicit numărul de elemente din listă

getItem returnează elementul din modelul de date care se găsește la o anumită poziție în listă

getItemId returnează ID-ul elementului din modelul de date care este afișat la o anumită poziție în listă

getView creează și returnează reprezentarea vizuală a unui element din listă, completată cu datele din model. În cazul în care lista conține mai multe elemente decât pot fi afișate deodată pe ecran, unul din parametrii acestei funcții va conține o referință către elemente de listă create anterior și care pot fi refolosite (pentru o gestiune mai eficientă a memoriei).

2.2 Crearea și interacționarea cu un meniu contextual

Pentru crearea unui meniu contextual care este afișat la apăsarea prelungită a unui element vizual, este necesară îndeplinirea următorilor pași:

2.2.1 Descrierea elementelor meniului

Elementele meniului contextual dorit se descriu într-un fișier XML din directorul **res/menu/**. Pentru fiecare dintre elemente se adaugă o etichetă **item:**

```
<item
   android:id="@+id/menu_item_id"
   android:title="@string/menu_item_title"/>
```

2.2.2 Înregistrarea elementelor vizuale pentru meniul contextual

Elementele vizuale pentru care se dorește afișarea meniului contextual la gestul de apăsare prelungită trebuie să fie înregistrate în prealabil utilizând comanda **registerForContextMenu**.

2.2.3 Afișarea și personalizarea meniului contextual

În clasa controller a ecranului în care este afișat meniul contextual este necesară suprascrierea metodei onCreateContextMenu, care este apelată la afișarea meniului. În cadrul acesteia se stabilește care meniu contextual va fi afișat (putem afișa meniuri diferite în funcție de tipul elementului vizual declanșator) și se face încărcarea structurii vizuale definite în fișierul XML.

2.2.4 Reacția la selectarea unei intrări din meniul contextual

La momentul selectării unui element al unui meniu contextual, în cadrul clasei controller va fi apelată funcția onContextItemSelected.

3 Practici de implementare

Vom considera în cele ce urmează că structurile vizuale pentru elementele din listă și meniul contextual au fost descrise în fișierele **res/layout/activity_books_list.xml** și **res/menu/books_list_context.xml**.

3.1 Navigarea la o altă activitate

Navigarea către o altă activitate (un alt ecran) în Android se realizează folosind codul următor:

```
Intent intent = new Intent(this, new_activity_class);
startActivity(intent);
```

3.2 Crearea clasei Adapter

```
public class MyAdapter extends BaseAdapter
{
   public List<Object> items;
   private Context context;

   public MyAdapter(Context context)
   {
      this.context = context;
   }
}
```

3.2.1 Implementarea funcției getView

```
public View getView(int position, View convertView, ViewGroup parent)
{
   // referință spre serviciul LayoutInflater
   LayoutInflater inflater = (LayoutInflater)
               context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);

   // posibilitatea de refolosire a unui rând definit anterior și care
      nu mai este vizibil
   View myRow = (convertView == null) ?
   inflater.inflate(R.layout.layout_name, parent, false) : convertView;

   // referință spre elementele vizuale. Actualizarea acestora cu
      informații din model

   return myRow;
}
```

3.3 Conectarea adaptorului cu lista

Conectarea instanței Adapter cu lista se realizează de obicei în funcția **onCreate** a controller-ului, după următorul algoritm:

```
listAdapter = nouă instanță a adaptorului;
listAdapter.items = conector la model;
referințăListă.setAdapter(listAdapter)
```

3.3.1 Actualizarea listei la schimbarea informațiilor din model

Actualizarea listei se face prin notificarea sa, utilizând clasa Adapter, după ce noile informații sunt deja disponibile adaptorului:

```
listAdapter.notifyDataSetChanged();
```

3.4 Implementarea meniului contextual

Pentru a putea afișa un meniu contextual, elementele vizuale trebuie să se înregistreze în acest sens. De obicei, înregistrarea se efectuează în metoda onCreate a controller-ului, prin apelarea metodei:

```
registerForContextMenu(view_reference);
```

3.4.1 Controlul afișării meniului contextual

Înainte de a afișa meniul contextual, sistemul de operare apelează funcția onCreateContextMenu. Vom exemplifica în cele ce urmează personalizarea acestei funcții prin atașarea la meniu a unui titlu diferit în funcție de elementul vizat dintr-o listă. Pentru a identifica mai multe informații despre elementul vizual la care este atașat meniul se poate utiliza parametrul de tipul ContextMenuInfo, conform exemplului de mai jos.

```
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo)
{
   super.onCreateContextMenu(menu, v, menuInfo);
   // verificăm dacă meniul este creat pentru lista vizată

   if (v.getId()==R.id.id_lista_vizata)
   {
      // identificăm elementul selectat din listă
      AdapterView.AdapterContextMenuInfo info =
                          (AdapterView.AdapterContextMenuInfo)menuInfo;

      menu.setHeaderTitle(String valoare_specifica_elementului);
      // încărcăm structura vizuală a meniului

      getMenuInflater().inflate(R.menu.meniu_contextual, menu);
   }
}
```

3.4.2 Prelucrarea elementului selectat din meniul contextual

```
public boolean onContextItemSelected (MenuItem item)
{
   // accesarea informației atașate meniului contextual
   AdapterView.AdapterContextMenuInfo info =
                (AdapterView.AdapterContextMenuInfo)item.getMenuInfo();

   // identificarea elementului selectat din meniu, folosind ID-urile
      predefinite
   if(item.getItemId() == R.id.id_item_1)
   {

   }
   else if(item.getItemId() == R.id.id_item_2)
   {

   }

   return super.onContextItemSelected(item);
}
```

4 Exerciții

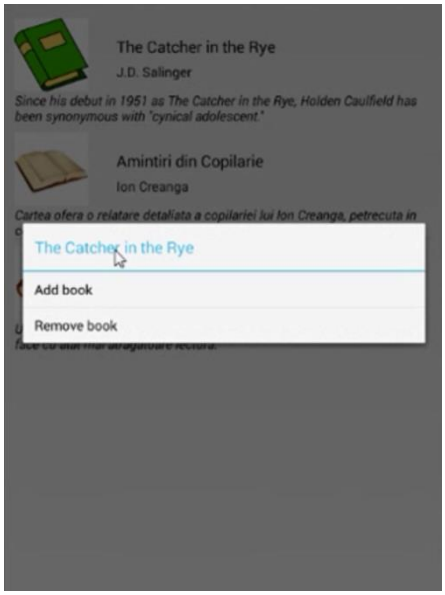
Exercițiul 1. Extindeți aplicația din laboratorul anterior pentru a simula accesul la o bibliotecă virtuală. După autentificarea cu succes a utilizatorului, acestuia îi va fi afișat un nou ecran cu lista de titluri disponibilă în bibliotecă, conform imaginii de mai jos:

Restricții de implementare

- informațiile despre fiecare carte vor fi modelate în cadrul unei componente cu rol de model în aplicație (clasă Java dedicată)

- afișarea informațiilor se va realiza cu un control de tip listă, care va încărca în mod dinamic datele despre fiecare carte
- fiecărei cărți afișate îi corespunde un element din listă

Exercițiul 2. La apăsarea prelungită a unui element din listă va fi afișat un meniu contextual conform imaginii de mai jos:



Specificații de implementare

- pentru a identifica elementul selectat, meniul contextual va afișa în partea de sus titlul cărții
- meniul afișat va avea două elemente: **Add book** și **Remove book**

Laboratorul 11 – elemente avansate de interfață și interacțiune utilizator

1 Introducere

Aplicațiile pentru Android permit afișarea în cadrul unei bare de opțiuni, plasată în partea de sus a ecranului, comenzile generale pentru activitatea curentă. Aceasta trebuie să reflecte opțiunile disponibile utilizatorului în fiecare moment. De asemenea, pentru situațiile în care anumite operațiuni durează un timp mai îndelungat, este necesară introducerea barei de progress care să avertizeze utilizatorul.

1.1 Obiective

Folosirea barei de opțiuni, a barei de progres și a butonului Back.

2 Consideraţii teoretice

2.1 Bara de opțiuni

Include în mod implicit un buton de revenire la activitatea anterioară (denumită și *activitate părinte*) și un text care afișează titlul activității curente. Ambele elemente sunt aliniate la stânga.

În restul spațiului disponibil pot fi afișate diferite opțiuni disponibile utilizatorului pentru activitatea curentă. Definirea acestor elemente se realizează în cadrul unei resurse de tip meniu, fiind inclusă în activitate în funcția **onCreateOptionsMenu**, apelată automat de către sistemul de operare. Modalitatea de reprezentare a comenzilor în cadrul barei de opțiuni poate fi controlată din fișierul XML, prin setarea parametrului **android:showAsAction**.

2.2 Bara de progres

Pentru a nu „îngheța” interfața utilizator pe durata realizării anumitor operații mai costisitoare este necesară includerea unei bare de progres pentru notificarea

utilizatorului. În sistemul Android, executarea periodică a unui set de instrucțiuni poate fi realizată folosind un obiect de tipul **CountDownTimer**, care inserează automat instrucțiunile în firul principal de execuție.

2.3 Butonul BACK

Majoritatea dispozitivelor Android din prezent oferă utilizatorului un set de butoane fizice, dintre care unul implementează funcționalitate de „pas anterior". Există însă situații specifice în care comportamentul acestui buton trebuie modificat pentru a corespunde cerințelor aplicației. În acest sens este necesară suprascrierea funcției **onBackPressed** din cadrul activității.

2.4 Ferestre de dialog

Ferestrele de dialog au rolul de a solicita confirmarea anumitor acțiuni care pentru utilizator nu sunt neapărat evidente sau au rezultat destructiv. Afișarea unei ferestre de dialog se realizează cu ajutorul clasei **AlertDialog.Builder**.

2.5 Mesaje scurte, denumite TOAST

Pentru afișarea de mesaje informative scurte către utilizator, care nu necesită confirmare de vizualizare din partea acestuia, poate fi utilizată clasa **Toast**. Mesajele astfel afișate sunt vizibile pe ecran un anumit timp stabilit, fiind apoi automat ascunse de către sistemul de operare.

3 Practici de implementare

3.1 Definirea unui timer

Navigarea către o altă activitate (un alt ecran) în Android se realizează folosind codul următor:

```
_progressTimer = new CountDownTimer(interval_de_apelare,
                                    timp_maxim_de_rulare)
{
   @Override
   public void onTick(long millisUntilFinished)
   {
      // cod ce va fi executat repetitiv, după fiecare
         interval_de_apelare
   }

   @Override
   public void onFinish()
   {
```

```
      // cod ce va fi executat o singură dată, după timp_maxim_de_rulare
   }
}.start();
```

3.2 Definirea listei de opțiuni

Lista de opțiuni se definește în cadrul unui fișier XML din directorul de resurse **menu**. Descrierea generală a unui element este similară cu codul de mai jos:

```
<item
   android:id="@+id/id_actiune_1"  // id-ul de identificare a acțiunii
   android:showAsAction="always"   // opțiunile pentru afișare
   android:title="Acțiune 1"/>     // textul afișat
```

3.3 Popularea barei de opțiuni

```
public boolean onCreateOptionsMenu(Menu menu)
{
   // adăugare de elemente la bara de opțiuni.
   getMenuInflater().inflate(R.menu.fisier_xml_meniu, menu);
   return true;
}
```

3.4 Identificarea opțiunii selectate

```
public boolean onOptionsItemSelected(MenuItem item)
{
   // Handle action bar item clicks here. The action bar will
   // automatically handle clicks on the Home/Up button, so long
   // as you specify a parent activity in AndroidManifest.xml.
   int id = item.getItemId();

   if (id == R.id.id_actiune_1)
   {
      // cod pentru Opțiunea1
      return true;
   }
   else if(id == R.id.id_actiune_2)
   {
      // cod pentru Opțiunea2
      return true;
   }
   else if ...

   return super.onOptionsItemSelected(item);
}
```

3.5 Afișarea unui Toast

```
Toast.makeText(this, "textul_pentru_afișare", durata_afișării_milisec)
     .show();
```

3.6 Afișarea unei ferestre de dialog

Pentru captarea și prelucrarea evenimentelor generate de butoanele unei ferestre de dialog este necesară implementarea interfeței **OnClickListener**. Astfel, pentru a capta aceste evenimente în activitatea curentă, la lista interfețelor implementate trebuie adăugată și cea menționată anterior.

```
AlertDialog.Builder myDialog = new AlertDialog.Builder(this);
logoutConfirmation
      .setTitle("titlu_dialog")
      .setMessage("mesaj_dialog")
      .setPositiveButton("titlu_buton",
                         referință_implem_interf_ OnClickListener)
      .setNegativeButton("titlu_buton",
                         referință_implem_interf_ OnClickListener)
      .show();
```

3.7 Prelucrarea butonului Back

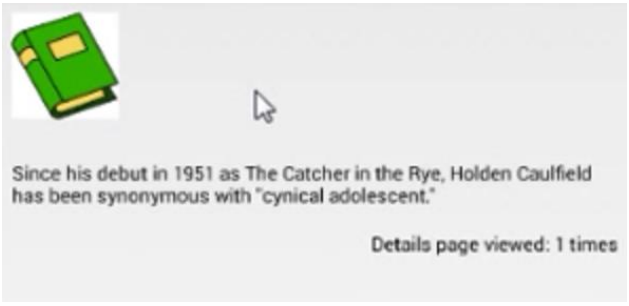
La apăsarea butonului fizic Back, în cadrul activității curente este apelată funcția:

```
public void onBackPressed()
{
   // cod executat la apăsarea butonului Back
}
```

Pentru comportamentul implicit, este necesară apelarea funcției din clasa părinte: **super.onBackPressed()**.

4 Exerciții

Exercițiul 1. Extindeți aplicația din laboratorul anterior prin adăugarea unui ecran de detalii, conform imaginii de mai jos:



Specificații de implementare:

- pagina pentru detaliile unei cărți este afișată la gestul Tap
- contorizați și afișați de câte ori pagina de detalii este afișată pentru fiecare carte în parte

Exercițiul 2. Adăugați aplicației următoarele funcționalități:

- la apăsarea butonului fizic **Back**, atunci când ecranul cu lista de cărți este afișat, solicitați utilizatorului confirmarea încheierii sesiunii de lucru prin intermediul unei ferestre de dialog a sistemului Android
    - dacă utilizatorul confirmă, afișați ecranul de Autentificare
    - dacă utilizatorul anulează, rămâneți la ecranul curent
- definiți un meniu de opțiuni care permite inițializarea încheierii sesiunii de lucru prin intermediul opțiunii **Sign out**
    - implementați scenariul de interacțiune de la punctul anterior

Laboratorul 12 – verificarea cunoștințelor în Tehnologii Android

1 Introducere

În cadrul acestui laborator va fi evaluat nivelul înțelegerii noțiunilor din domeniul Tehnologiilor Android prezentate în cadrul laboratoarelor anterioare. Fiecare dintre studenți va primi o notă ce va reflecta atât nivelul cunoștințelor teoretice însușite cât și nivelul abilităților practice dobândite.

1.1 Obiective

Stabilirea măsurii în care noțiunile fundamentale din domeniul Tehnologiilor Android au fost însușite de către cursanți.

2 Metodologia de evaluare

Evaluarea cunoștințelor dobândite în timpul lucrărilor practice de laborator din domeniul Tehnologiilor Android se va realiza pe două niveluri:

2.1 Evaluarea teoretică

Presupune testarea înțelegerii noțiunilor teoretice de bază, care permite atât profesorului cât și studentului identificarea eventualelor puncte neclare în însușirea cunoștințelor elementare.

Această evaluare se va desfășura prin intermediul unui chestionar cu aproximativ 10 - 15 întrebări care au multiple variante de răspuns, nota finală obținută fiind accesibilă imediat după completarea chestionarului, prin corectare automată.

Fiecare dintre participanții la laborator vor avea acces la forma electronică a chestionarului pentru un timp limitat, contorizat automat, și numai de pe contul propriu din cadrul platformei de gestiune a materialelor de laborator. Întrebările

pentru fiecare chestionar vor fi selecționate aleator din cadrul unei baze de date iar răspunsurile fiecărei întrebări vor fi de asemenea ordonate aleator.

2.2 Evaluarea practică

Are ca și scop verificarea abilității studenților de a aplica noțiunile învățate în vederea dezvoltării unor interfețe utilizator în tehnologia Android. Fiecăruia dintre participanți îi va fi asignat un exercițiu, având o oră și jumătate timp de rezolvare.

Implementarea prezentată va fi evaluată conform următoarelor criterii generale:

- **structura aplicației** – implementarea soluției trebuie să respecte design-pattern-ul arhitectural Model-View-Controller
- **interfața utilizator**
    - gradul în care formatarea vizuală a elementelor de interfață respectă cerințele exercițiului
    - fluiditatea designului (adaptare la orientarea dispozitivului)
    - respectarea recomandărilor de design a interfețelor utilizator și a interacțiunii cu acesta
- **interacțiune utilizator** – în ce măsură implementarea realizată îndeplinește cerințele de:
    - verificare a validității datelor
    - prevenire a erorilor
    - sprijinire a utilizatorului în revenirea dintr-o situație de eroare
    - evitarea blocării elementelor de interfață în timpul efectuării unor operații mai complexe
    - etc.
- **respectarea recomandărilor de bune practici**
    - definirea și utilizarea stilurilor
    - separarea componentelor de vizualizare, model de date și control
    - procesarea în fire de execuție secundare a operațiilor costisitoare
    - etc.

La finalul timpului de lucru, fiecare dintre studenți prezintă cadrului didactic implementarea efectuată și fundamentele teoretice care au stat la baza deciziilor de implementare.

3 Exerciții

3.1 Exemple de întrebări pentru verificarea cunoștințelor teoretice

1. **Specificați valoarea de adevăr a următoarei afirmații: aplicațiile Android pot fi testate pe dispozitive fizice, precum și în simulatoare software oferite de anumite medii de dezvoltare cum ar fi Android Development Tools:**
    a. Adevărat
    b. Fals

2. **Aplicațiile native Android sunt compatibile cu:**
    a. Telefoanele și tabletele care rulează sistemul de operare Android
    b. Smart TV bazate pe sistemul de operare Android
    c. Pot funcționa corect pe orice dispozitiv, indiferent de sistemul de operare instalat
    d. Toate opțiunile de mai sus

3. **Este corectă afirmația conform căreia aplicațiile Android folosesc formatul XML pentru a defini elementele grafice ale unui ecran?**
    a. Adevărat
    b. Fals

4. **Ce se poate afirma despre container-ul Relative Layout?**
    a. Permite ordonarea de noi elemente grafice în funcție de poziția componentelor existente
    b. Se poate specifica faptul că elementul A să fie aliniat deasupra unui alt element B
    c. Permite plasarea verticală a componentelor grafice
    d. Permite plasarea orizontală a componentelor grafice

5. **Datorită numărului mare de dispozitve care rulează sistemul de operare Android, a fost realizată o convenție de clasificare a afișajelor. Selectați tipurile de afișaje care fac parte din această convenție:**
    a. ldpi
    b. smalldpi
    c. xsdpi

d. xhdpi

6. **Care dintre următoare afirmații, referitoare la AndroidManifest.xml sunt adevărate:**
    a. Cuprinde toate elementele descriptive necesare instalării pe dispozitive
    b. Definește nivelul minim pentru versiunea de Android SDK
    c. Restricționează accesul la aplicațiile din magazinul virtual Google Play, pentru anumite dispozitive
    d. Deservește la identificarea aplicației pe baza unui nume unic

7. **Cum se numește clasa care realizează legătura dintre modelul de date și componentele vizuale în aplicațiile Android?**
    a. Link
    b. Adapter
    c. MVC
    d. LiniarLayout

8. **Care este numele clasei care se folosește pentru a crea elementele grafice ale unui ecran (screen)?**
    a. Activity
    b. ViewGroup
    c. View
    d. Adapter

9. **Identificați metoda Android care permite navigarea între activități:**
    a. startActivity
    b. gotoActivity
    c. open
    d. create

10. **Care este metoda folosită pentru notificarea clasei Adapter despre modificările din clasa model?**
    a. adapterNotify
    b. notifyDataSetChanged
    c. updateAdapter
    d. notification

11. **Unde este poziționată bara de opțiuni într-o aplicație Android?**
    a. În partea de sus a ecranului
    b. În josul ecranului
    c. În dreapta
    d. În stânga

12. **Specificați numele directorului (din structura implicită a unui proiect Android) care conține clasele XML folosite pentru definirea interfeței grafice:**
    a. src
    b. res/layout
    c. android
    d. res/view

13. **Care metodă este folosită pentru actualizarea la fiecare 3 secunde a unei liste de elemente, cu informații preluate dintr-o bază de date?**
    a. CountDownTimer(3000, 100)
    b. CountDownTimer(3, 100)
    c. Count()
    d. CountDownTimer(3000, 1000)

14. **Specificați numele metodei care identifică elementul selectat din bara de opțiuni:**
    a. selected
    b. optionSelect
    c. onOptionsItemSelected
    d. selectedItem
    e. onItemSelect

15. **Selectați clasa prin intermediul căreia se afișează o fereastră de dialog în aplicațiile Android:**
    a. Alert
    b. Echo
    c. AlertDialog.Builder
    d. Display.Builder

3.2 Exemple de probleme practice

Exercițiul 1. Implementați o aplicație care gestionează o listă de cumpărături, conform următoarelor specificații:

- la pornirea aplicației este afișată lista curentă de produse
- pentru fiecare produs se afișează
    - denumirea, numărul de unități, prețul estimat
- meniul **Opțiuni** conține următoarele comenzi:
    - **Șterge toate elementele** - șterge toate elementele din listă
    - **Adaugă un element** – afișează un nou ecran prin intermediul căruia poate fi adăugat un nou element în listă
- la apăsarea prelungită a unui element se va afișa un meniu contextual cu următoarele caracteristici
    - titlul meniului corespunde denumirii elementului
    - **Șterge din listă –** opțiune care elimină elementul din listă
    - **Editează** – opțiune care deschide un nou ecran ce va permite utilizatorului editarea informațiilor despre elementul selectat

Exercițiul 2. Implementați o aplicație care permite vizualizarea unor anunțuri de vânzare autoturisme, conform următoarelor specificații:

- la deschiderea aplicației va fi afișată automat lista de autoturisme, cu următoarele informații:
    - poză autoturism, model, denumire proprietar, anul fabricației, număr de kilometri, tip combustibil, descriere
    - poza autoturismului va fi plasată în partea din stânga sus a fiecărui element din listă
- la apăsarea prelungită a unui element va fi afișat un meniu contextual cu următoarele opțiuni:
    - **Ascunde anunț** - va ascunde elementul selectat din listă dar nu îl va elimina din modelul de date
    - **Marchează anunț** - va modifica afișarea elementului selectat prin plasarea imaginii în partea din dreapta sus
    - **Șterge anunț** - va elimina anunțul din listă și din modelul de date
- implementați bara de opțiuni a aplicației cu funcționalitatea

- **Afișează anunțurile ascunse** – afișează din nou în listă toate anunțurile ascunse
- **Demarchează toate anunțurile** – va readuce toate anunțurile marcate la stadiul de anunțuri nemarcate

Exercițiul 3. Implementați o aplicație de gestiune a profilului utilizatorilor unei rețele sociale, conform următoarelor specificații:

- profilul utilizatorului conține informațiile:
  - poză, nume, prenume, hobby
- la gestul Tap pe poză, se va deschide un nou ecran cu o listă de poze predefinită, formatată astfel:
  - fiecare element din listă afișează în partea din stânga poza iar în dreptul acesteia denumirea pozei
- la selectarea unei poze din listă aplicația revine la ecranul cu contul utilizatorului, actualizând poza acestuia
- în mod similar, utilizatorul poate selecta dintr-o listă de hobby-uri pe cel pe care îl dorește afișat în profil

Exercițiul 4. Implementați o aplicație similară unei agende de telefon, conform următoarelor specificații:

- pentru fiecare contact se poate reține
  - nume, prenume, telefon, email
- meniul de opțiuni al aplicației permite următoarele acțiuni
  - **Adaugă contact** – deschide un nou ecran în care pot fi introduse datele contactului
  - **Caută contact** – afișează un câmp de text în care poate fi introdus cuvântul cheie utilizat pentru filtrarea contactelor
- meniul contextual afișat la apăsarea lungă a unui element din lista de contacte permite efectuarea următoarelor acțiuni
  - **Șterge contact** - șterge un contact din listă
  - **Sună contact** - efectuarea unui apel telefonic
  - **Trimite email** - inițierea trimiterii unui mesaj email
- la fiecare dintre acțiunile menționate, verificați dacă informațiile necesare sunt disponibile (ex. dacă pentru contactul selectat există specificată o adresă de email), iar în caz contrar afișați un mesaj de eroare

Exercițiul 5. Implementați o aplicație de mesagerie instantanee, conform următoarelor specificații:

- ecranul principal al aplicației prezintă lista de mesaje primite, afișând pentru fiecare mesaj următoarele informații:
    - mesaj, autor, data și ora primirii mesajului
- sub lista de mesaje primite, introduceți o căsuță de text și un buton denumit **Trimite**
    - la introducerea unui mesaj în căsuța de text și apăsarea butonului, mesajul va fi adăugat în lista de mesaje
- meniul contextul afișat la apăsarea îndelungată a unui mesaj permite utilizatorului să executarea următoarelor acțiuni
    - **Șterge mesaj** - permite eliminarea unui mesaj primit din listă
    - **Răspunde la mesaj** – va introduce automat în căsuța de răspuns textul: **@[autor_mesaj_selectat]**
- la fiecare 5 secunde, generați automat un mesaj către utilizatorul curent, simulând o scurtă conversație

Bibliografie

[1] Chris Coyier, *Specifics on CSS Specificity*, 10 Mai 2010, http://css-tricks.com/specifics-on-css-specificity/ [ONLINE, 30 mai 2015]

[2] *App activation and deactivation for Windows Phone 8*. http://msdn.microsoft .com/en-us/library/windows/apps/ff817008%28v =vs.105%29.aspx [ONLINE, 30 mai 2015]

[3] Clifton, I.G. *Android User Interface Design: Turning Ideas and Sketches into Beautifully Designed Apps*. Addison-Wesley Professional, 2013

[4] Krug, S. *Don't Make Me Think, Revisited: A Common Sense Approach to Web Usability (3rd Edition) (Voices That Matter)*. New Riders, 2014

[5] Lewis, C., Rieman, J. *Task-Centered User Interface Design. Electronic publishing*, 1993. http://hcibib.org/tcuid/ (2015)

[6] Matteo, P. Windows Phone 8 Development Succinctly. Electronic publishing, Syncfusion, 2014. http://www.syncfusion.com/resources/techportal/ ebooks/windowsphone8

[7]McCracken D.D., Wolfe R.J. *User-Centered Website Development. A Human-Computer Interaction Approach*. Pearson Prentice Hall, 2004

[8] Neil, T. *Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps (Second Edition)*. O'Reilly Media, 2014

[9] Nudelman, G. *Android Design Patterns: Interaction Design Solutions for Developers*. Wiley, 2013