Developing an M-Learning Application for iOS

Paul POCATILU
Department of Economic Informatics and Cybernetics
Bucharest University of Economic Studies, Romania
ppaul@ase.ro

The mobile market development has a high impact on all domains including education. Smart mobile devices started to be affordable and the massive use on educational processes does not seem to be too far. Mobile learning applications are be targeted for all major mobile operating systems as native applications or Web-based. The objective of this paper is to present the implementation of the evaluation module for an m-learning application developed for iOS devices. The m-learning application is targeted to a higher education institution. The application uses Web services in order to obtain the content and to authenticate the users.
Keywords: *Mobile Learning, Mobile Application Development, iOS, Model View Controller, Web Service, Assessment*

1 Introduction

Mobile learning represents a research topic for many academics and practitioners. Around the world there are several projects ongoing or finalized that aim the development of mobile learning applications for different mobile platforms or Web based applications optimized for mobile devices, like [1], [2], [3] and [4]

Mobile learning applications could include modules for:

- Content presentation (lessons); content include text, video, audio, graphics; it could as simple as displaying only text or loading PDF or other files or very complex, using multimedia components [5];
- Short assessments (quizzes); these are available to students in order to test their knowledge; the user can take these tests whenever they want and they are not time-restricted;
- Final assessments (tests); these tests are given at a specific date and time and they have a limited duration;
- Trainer-student communication; this involves the use of well known components (e-mail or social networks) or by using a dedicated component based on a specific protocol.
- Content sharing (e-mail, social networks, cloud etc.); this will allow users to share the content or results with other registered users or anyone (if the application allows it);
- Homework and assignments; the students could load a file or fill some data fields according to requirements.

Each developer chooses to include one or more modules in their applications. The applications could be developed as standalone applications (all content is stored on the device), distributed applications (a native client and the server that provides learning contents) and Web-based applications (the client is a simple mobile Web browser [6]). It is important to design the application taking into account several quality characteristics based on the users requirements and behavior [8].

For connected mobile learning applications the client communicates with the server using standardized protocols (like HTTP) or dedicated protocols. Also, the connected applications can synchronize the mobile device with the server.

In this field, were developed several prototypes of mobile learning applications in a framework of a research project. In this respect, it was developed a SOAP-based Web service that provides methods for students' assessment. The most recent work includes the implementation of a mobile learning application for Android devices. The results were presented in [4].

As for the previous project, the main purpose

DOI: 10.12948/issn14531305/17.4.2013.07

is to keep the user interface as simple as possible and to focus on the basic functionalities. Further versions need more focus on the graphical interface and the user interaction (like using gestures etc.).
The use of Web services has the advantage that:

- client applications could be of any type (mobile, desktop);
- the client application could be developed using almost any existing language or technology;
- the client application can be targeted on almost any existing platform or operating system;

The Web service includes methods for students' assessment. Each student could login and select a test for one topic. The topic is related mobile applications development.
Figure 1 depicts the interaction between the mobile device and the Web service, highlighting some of the Web service's methods.

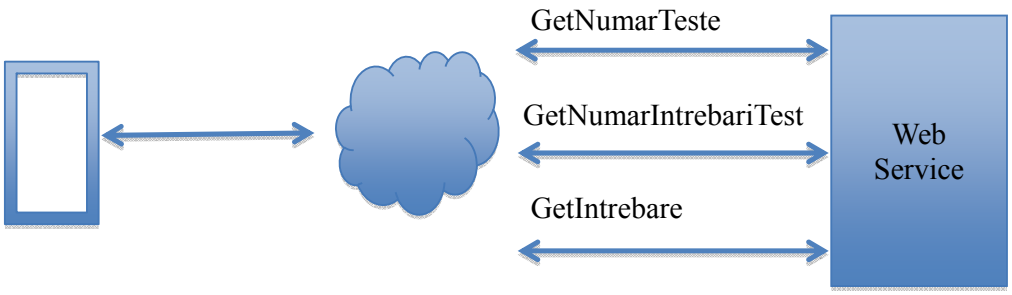

Fig. 1. Assessment module interaction using Web services

The Web service could be easily extended to support other functionalities by implementing new methods.
The paper presents the main results obtained during the implementation of a mobile learning application for iOS devices, using the same Web service. The application implements a multiple-choice test.
The paper is organized as follows:
The section entitled *iOS Applications Development* presents the main characteristics of the iOS operating system and the particularities of the iOS applications development.
Web Services Access deals with the main frameworks used to consume Web services from iOS applications.
The Mobile Learning Application section presents the mobile learning application developed for iOS.
The paper ends with conclusions and future work.

2 iOS Applications Development

iOS is the operating system developed by Apple for their mobile devices in 2007. The core operating system is similar to OS X that is based on Unix. The operating system supports processors with ARM architecture. Currently iOS is the second most popular platform after Android, counting over 12 percent in the third quarter of 2013 [8].
Figure 2 depicts the main layers of the iOS platform: Core OS, Core Services, Media and Cocoa Touch.
The *Core OS* include the kernel (based on UNIX Mach kernel), drivers, libraries and system utilities.
Core Services layer provides support for file management, threading, networking, memory management and other basic services.

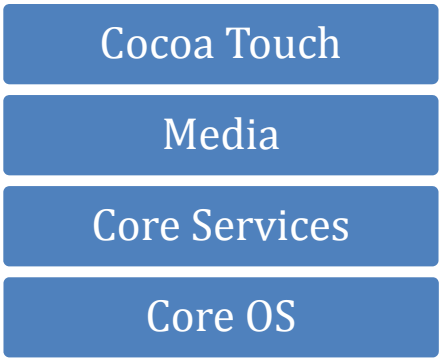

Fig. 2. iOS platform layers

DOI: 10.12948/issn14531305/17.4.2013.07

Media layer includes support for printing and graphics (OpenGL, QuickTime etc.).
Cocoa Touch is a subset of Cocoa API for Mac OS X. It handles the user interaction and GUI. Cocoa Touch includes the UIKit framework, Foundation framework and other frameworks that provide access to system resources like contacts, GPS, camera etc. [9].
On top are user and system/preinstalled applications.
iOS native applications development involves the use of Xcode IDE and Objective C as the main programming language. Objective C is an extension of the C programming language that allows the use of classes and objects.
Objective C classes are declared within an *@interface* block. Messages (methods) implementation is inside *@implementation* section. For class forward declaration *@class* keyword is used.
As a practice, class declarations are stored in header files (*.h*) and class implementation in source files (*.m*).
Instance variables are declared in @interface section, between the curly braces.
Methods (also called messages) names include the parameters' labels separated by colons. Method calls are different from C/C++ or Java method calls. For example, the class *Test* declares the method:

setRaspunsuriPentruIntrebare:cuVarianta

in this way:

```
@interface Test : NSObject
//...
-(void)       setRaspunsPentruIntrebare:
(int)idIntrebare       cuVarianta       :
(int)varianta;

@end
```

The minus sign (-) in front of a method shows that is an instance methods. The plus sign (+) in front of a methods is used for class methods (static methods).
The method is called using square brackets like in this example:

```
[self.testCurent
setRaspunsPentruIntrebare:    idIntrebare
cuVarianta: idVarianta];
```

Methods cannot be overloaded and parameter labels differentiate them.
All class-type objects have to be allocated dynamically. Object creation involves the use of two methods: *alloc* (for memory allocation and default instance member initialization) and *init* (for specific member initialization, even for superclass members).
In order to access data members, property could be added to classes. Properties are declared with *@property* keyword and, in order to internally generate setters and getters, *@synthesize* keyword is used in implementation block with the association between the property and the corresponding data member.
Cocoa classes are derived from *NSObject* class.
iOS applications are based on Model-View-Controller design pattern [10], [11], Figure 3.

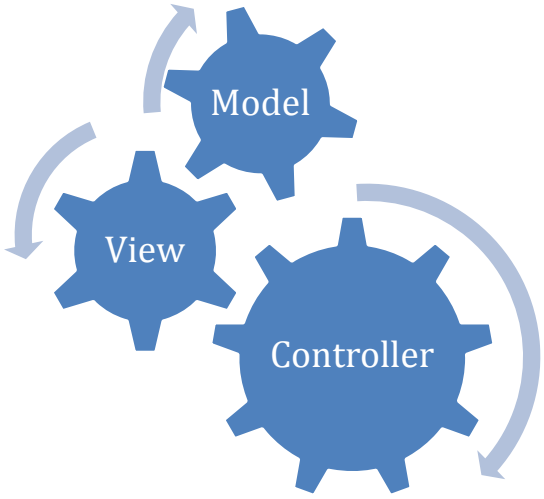

Fig. 3. Model-View-Controller design pattern

The model refers to data and data management. Here is the business logic of the applications. For the mobile learning application a representative class of the model is *Test* class. It manages the current test, knows the current question and controls the navigation between the questions of the current test.
The view deals with the user interface and user interaction. The mobile learning

DOI: 10.12948/issn14531305/17.4.2013.07

application includes three views: one with all available tests (*TestViewController* class) one for current question and the possible answers (*IntrebareViewController* class) and the last one display the list of users' responses (*RaspunsuriViewController* class).

The controller assures the link between the view and the model. It handles the users' actions on the interface, changes the model state, and displays the requested views. Usually for each view there is a controller. The controllers of the mobile learning application:

- handles users' selections (tests and answers);
- fills the lists and user controls with data (questions, tests, answers etc.);
- shows other views when it is required by users' actions or by other normal or exceptional situations (like displaying error messages).

The user interface was implemented using storyboards and Interface Builder. A storyboard is a collection of scenes (views). The scenes are connected by segues. Figure 4 depicts the scenes designed in Xcode for the mobile learning application.

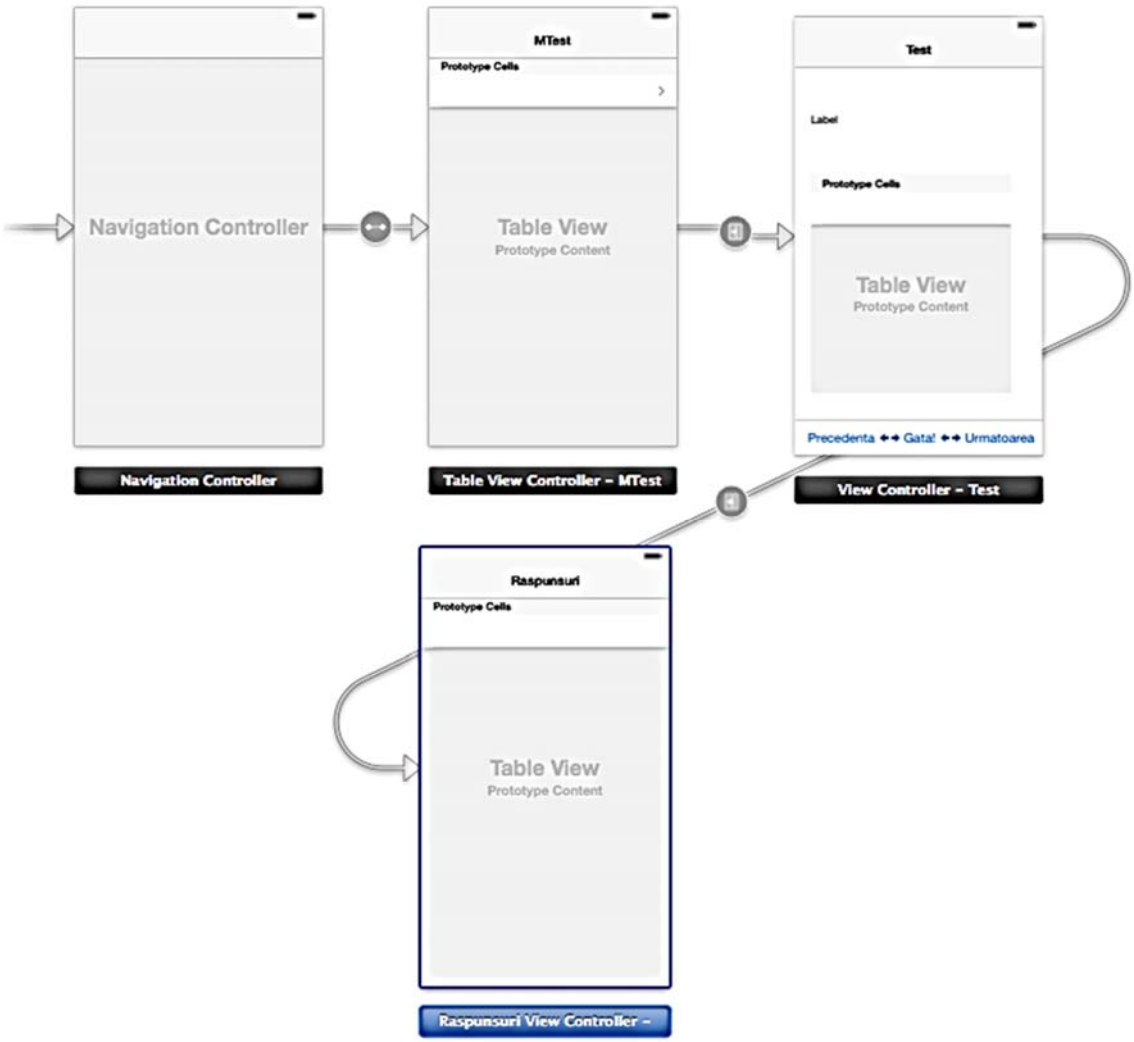

Fig. 4. Mobile learning application scenes within the storyboard file

In order to test and iOS application on a real device the developers has to register to Apple to iOS Developer Program [12]. A regular registration requires an annual fee.

The mobile learning application was developed using Xcode 5 with iOS 7 SDK on

DOI: 10.12948/issn14531305/17.4.2013.07

OS X Mavericks.
The application was tested using the iOS Simulator for an iPhone Retina (4-inch). The current version does not make use of sensors or camera and, for this stage, the simulator is sufficient.

3 Web Services Access

Web services could be implemented using SOAP and WSDL protocols or using REST architectural style. REST services requires the use of URLs and, if HTTP is used, HTTP commands (GET, POST, DELETE etc.).
For the current application, the Web service is based on SOAP and WSDL and it is implemented using .NET technologies. The Web service includes methods for obtaining the number of tests, for obtaining the number of questions for a test and for obtaining a specific question based on its number and the test number etc. The Web service is provided with login possibilities for its users.
For iOS there are several approaches to consume Web services:

- third-party, open source implementations for consuming Web services like *RESTKit* and *Spaghetti* or advanced networking frameworks like *AFNetworking*;
- developers own implementation of libraries, classes or methods for Web service communication using standard networking classes like *NSURLRequest* and *NSURLConnection*;
- third-party tools for proxy generation using Web services URL, based on WSDL, like *wsdl2objc* [13] and *SudzC* [14].

The proposed solution uses a SOAP based Web service and for current implementation the *wsdl2objc* was used. Based on the Web service address, using WSDL, the tool generated all client files required to access the Web service. The name of the Web service is *Service1* and the tool will generate several files:

- some files required for network access and result processing
- two files associated to the Web service: a header file (Service1.h) and an implementation file (Service1.m).

Table 1 presents several classes generated by the *wsdl2objc* tool from the Web service and theirs role.
As it can be seen, for each method of the Web service are generated two classes: one for method call and one for method's result.

Table 1. Example of Web service generated classes using *wsdl2objc*

Class	**Role**
Service1SoapBinding	Defines the binding for the Web service
Service1SoapBindingResponse	Manage the responses
Service1_GetNumarTeste	Initiate the corresponding Web service's method *GetNumarTeste*
Service1_GetNumarTesteResponse	Associated to the result of *GetNumarTeste* method

The classes are used to initialize the connection, to call the Web service's methods and to obtain the results.
Listing 1 represents a function used to call a method of the Web service. The function is implemented at a class level (static) and calls the method *GetNumarTeste* provided by the Web service, obtaining the number of tests available on the assessment platform.

Listing 1. Example of a Web service access function

```
+(int) getNumarTeste :(int)idUser
{
    Service1SoapBinding* binding = [Service1 Service1SoapBinding];
    Service1SoapBindingResponse* response;

    //request initialization
    Service1_GetNumarTeste* request = [[Service1_GetNumarTeste alloc]init];
```

DOI: 10.12948/issn14531305/17.4.2013.07

```
    //pass the parameter
    request.idUser = [NSNumber numberWithInt:-1];

    response = [binding GetNumarTesteUsingParameters:request];
    NSArray *responseBodyParts = response.bodyParts;
    id raspuns;

    @try
    {
        //get the response
        raspuns = [responseBodyParts objectAtIndex:0];
    }
    @catch (NSException* exception)
    {
        NSLog(@"getNumarTeste Exceptie: %@", exception.reason );

        return -1;
    }

    if ([raspuns isKindOfClass:[SOAPFault class]])
    {
        NSString* errorMesg = ((SOAPFault *)raspuns).simpleFaultString;
        NSLog(@"getNumarTeste Eroare: %@", errorMesg);

        return -1;
    }
    else
         if([raspuns isKindOfClass:[Service1_GetNumarTesteResponse class]])
         {
              Service1_GetNumarTesteResponse* nrIntrebResponse = raspuns;
              return nrIntrebResponse.GetNumarTesteResult.intValue;
         }
    //error
    return -1;
}
```

The method provided in Listing 1 is called with the user id parameter:

```
nrTeste = [AccessSW getNumarTeste: uid];
```

AccessSW is the class that implements the methods for Web service access.

The Web service calls are made asynchronous in order to allow a continuous interaction with the user interface, even for time-consuming transactions.

Compared with the same implementation for Android platform, that uses kSOAP, *wsdl2objc* tool generates all the required classes to use the Web service. This includes the classes for complex results like questions and their possible answers. The prototype of the application for Android required the implementation of these complex classes in order to process the responses correctly.

4 The Mobile Learning Application

The application connects to a Web service in order to obtain the questions and the variants of responses. The request and responses are made using plain XML content.

The application start screen is presented in Figure 5. The available tests are displayed like a list in a table control (from *UITableView* class) and the user can choose any test from the list. The associated controller is derived from *UITableViewController* and implements the *UITableViewDataSource* (for data source) and *UITableViewDelegate* (for handle user interaction) protocols that are required when working with tables.

UITableViewDataSource protocol requires implementation of at least two methods called when it needs to display table's data. The methods provide information about the number of section, the number of rows within a section, header and footer titles etc.

UITableViewDelegate controls the way the cells are displayed (cell height, custom

DOI: 10.12948/issn14531305/17.4.2013.07

header and footer) and the user actions (click in cell, cell content editing etc.).
The tests presented in Figure 5 are available for one topic (mobile applications development), but future versions of the application will be extended for more topics.

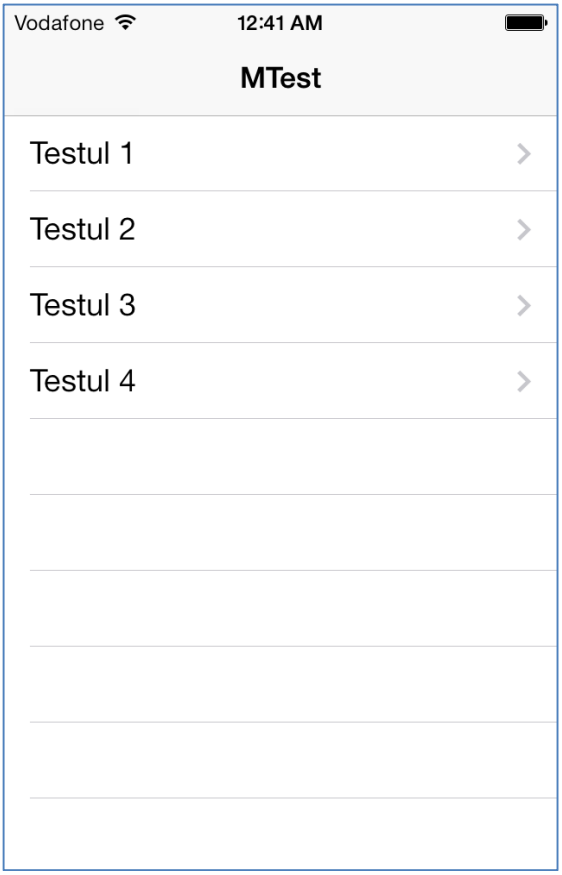

Fig. 5. Initial application screen

The table has one section and the number of rows is taken from the number of tests available for the current topic in database. These values are returned by the two methods of data sources protocol used for these controls: *numberOfSectionsInTableView* and *numberOfRowsInSection*.
The method *cellForRowAtIndexPath*, provided by the same protocol, will initialize the label of each row with test name. The current implementation contains four tests identified by corresponding numbers and the label are initialized accordingly.
The list is initialized on *viewDidLoad* message. In order to assure that all data coming from the server will be available before the list is displayed, Web service methods calls are made using blocks as can be seen from the code excerpt in Listing 2. This will send the *reloadData* message to the table so the user will see all required items within the list.
The link between the cells and the next screen is created in Interface Builder and the transition is made without writing any line of code.
After the user selects a test, the next screen will display the first question from selected test.

Listing 2. Excerpt for data loading in the *UITableView* control

```
dispatch_async(dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0), ^{
      if (_nrTeste > 0)
      {
            //initialize data model

            //reload table data
            dispatch_sync(dispatch_get_main_queue(), ^{
                  [self.tableView reloadData];
             });
});
```

The objects are sent between views in *prepareForSeque* message, as can be seen from Listing 3.

Listing 3. Parameter passing between views

```
- (void)prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender
{
    ViewController *detailController = [segue destinationViewController];
```

DOI: 10.12948/issn14531305/17.4.2013.07

```
    //selected cell
    UITableViewCell *cell = (UITableViewCell*)sender;
    //path to selected cell
    NSIndexPath *indexPath = [self.tableView indexPathForCell:cell];
    //questions controller initialization
    detailController.testCurent =  [[Test alloc] init];
    //model members initialization
    detailController.testCurent.nrTest = indexPath.row + 1;
    [detailController.testCurent initNrIntrebariTest:
       [AccessSW getNumarIntrebariTest : detailController.testCurent.nrTest]];
}
```

The navigation between the screens is implemented in a standard way using the *UINavigationController*. This is the containing controller for the *TesteViewController*.

The questions screen is presented in Figure 6. The possible answers are displayed in a control derived from *UITableView* class. The selected answer is highlighted by setting the cell's property *accessoryType* to *UITableViewCellAccessoryCheckMark*. A check mark is shown on the selected answer, as it can be seen from Figure 6.

The toolbar buttons controls the navigation between the questions. There are three buttons:

- *Precedenta* – goes to the previous question from the current test;
- *Urmatoarea* – goes to the next question;
- *Gata* – ends the test and display the given answers.

The navigation between questions could be implemented using gestures like swipe. This could from left to right (for the previous question) and from right to left (for the next question). To return to the test list a back button is provided by the standard navigation controller.

The number of current question and the total number of questions are displayed on the view's title.

The selected answers are stored in a dedicated array in order to be checked against the correct ones. The answer could be sent to server at the end of the test or after each response of the candidate.

The first option will send only the final responses to the server. There is only one request to the server related to candidate's answer.

The last option has the advantage of keeping the responses updated, but has to deals with user changes and requires frequent network access.

The responses are stored in a persistent data structure in order to assure that data will be recoverable in case that application crashes or other unexpected situation occurs.

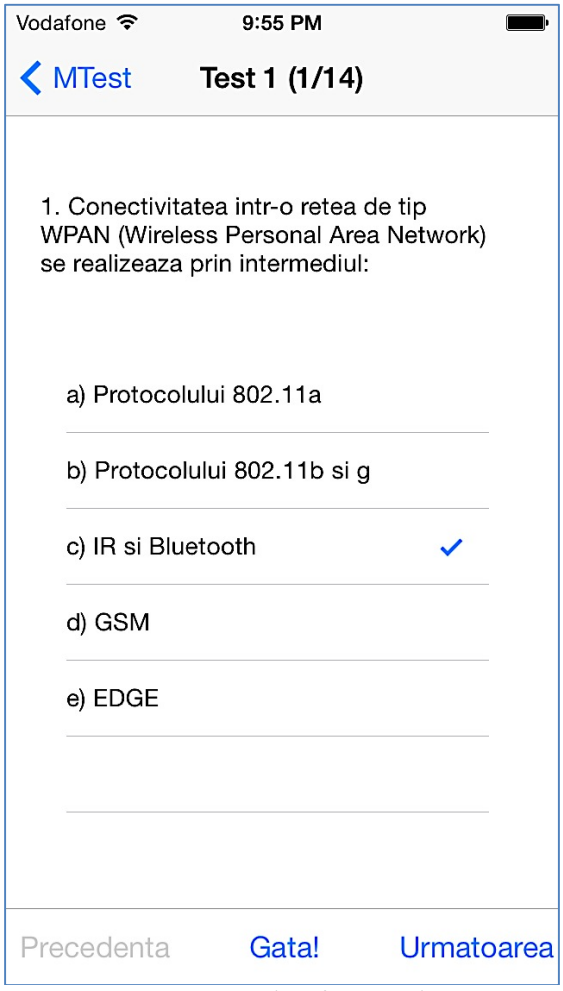

Fig. 6. Example of a question

After the candidate chooses to finish the test,

DOI: 10.12948/issn14531305/17.4.2013.07

all the answers will be displayed on the screen, Figure 7. Possible future actions will be to allow the users to save the answers and to send the answers by e-mail or to share them using social networks. This requires additional buttons that can be added on a redesigned toolbar and/or on navigation toolbar in a standardized way.

Fig. 7. The selected responses screen

The application does not provide user with the correct answers, this feature is available only for quizzes. The user will find the final mark and will get only a statistic about his or her answers (number of correct answers, number of unanswered questions, number of incorrect answers, percentage of answers per each topic/subtopic etc.).

5 Conclusion and Future Work

Compared with de development of the similar application for Android devices, the first conclusion is that also there is no direct support for Web services client and third party tools have to be used.

Next application developments will focus in design improvement. Also, the application will be integrated with the components that provide educational content to the students. Also, the implemented Web services will be extended to support JSON and/or XML responses that are very popular and can be easily processed.

Another important objective is to optimize the amount of data sent between the client and server. All questions and possible answers can be transferred in one transaction, this having impact on duration but resulting in a smaller overhead. Also, data can be compressed by the server and decompressed on the device. This will increase the duration on some method calls.

References

[1] M. H. Ferrer, J. Hodges and N. Bonnardel, "The MoLE project: an international experiment about mobile learning environment," *Proc. of the 31st European Conference on Cognitive Ergonomics (ECCE '13)*, New York: ACM, 2013

[2] D. G. de la Iglesia and D. Weyns, "Guaranteeing robustness in a mobile learning application using formally verified MAPE loops," *Proc. of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS '13)*, Piscataway: IEEE Press, 2013, pp. 83-92.

[3] D. Furió, S. González-Gancedo, M. C. Juan, I. Seguí and N. Rando. "Evaluation of learning outcomes using an educational iPhone game vs. traditional game.", *Computer & Education* Vol. 64, May 2013, pp. 1-23.

[4] P. Pocatilu, Developing Mobile Learning Applications for Android using Web Services, *Informatica Economica*, vol. 14, no. 3, 2010

[5] A. Reveiu, I Smeureanu and M. Dardala,

DOI: 10.12948/issn14531305/17.4.2013.07

"Generating Multimedia Components for M-Learning," *Informatica Economică*, vol. 13, no. 3/2009, pp. 88-95

[6] A. Butoi, N. Tomai, D. Mican and G. C. Silaghi, "Designing Effective Web-Based M-Learning Systems," *Proc. of the IE 2013 International Conference*, Bucharest, 2013, pp. 126-130

[7] C. Boja and L Batagan, "Analysis of M-Learning Applications Quality," *WSEAS Transactions on Computers*, Issue 4, Vol. 8, May 2009, pp. 767-777

[8] Gartner Says Smartphone Sales Accounted for 55 Percent of Overall Mobile Phone Sales in Third Quarter of 2013, [Online]. Available at: http://www.gartner.com/newsroom/id/2623415 (November 2013)

[9] P. Pocatilu, *Programarea dispozitivelor mobile*, Bucharest: ASE Publishing House, 2012

[10] M. Neuburg, *iOS 7 Programming Fundamentals*, O'Reilly Media, 2013

[11] S. G. Kochan, *Programming in Objective-C, Fourth Edition*, Pearson Education, Inc., 2012

[12] Apple Developer, [Online]. Available at: https://developer.apple.com/ (October 2013)

[13] wsdl2objc - Generates Objective-C (Cocoa) code from a WSDL for calling SOAP services [Online]. Available at: https://code.google.com/p/wsdl2objc/ (October 2013)

[14] SudzC | clean source code from your web services [Online]. Available at: http://sudzc.com/ (October 2013)

Paul POCATILU graduated the Faculty of Cybernetics, Statistics and Economic Informatics in 1998. He achieved the PhD in Economics in 2003 with thesis on Software Testing Cost Assessment Models. He has published as author and co-author over 45 articles in journals and over 40 articles on national and international conferences. He is author and co-author of 10 books, (Mobile Devices Programming and Software Testing Costs are two of them). He is professor at the Department of Economic Informatics and Cybernetics within the Bucharest University of Economic Studies, Bucharest. He teaches courses, seminars and laboratories on Mobile Devices Programming, Economic Informatics, Computer Programming and Project Management to graduate and postgraduate students. His current research areas are software testing, software quality, project management, and mobile application development.

DOI: 10.12948/issn14531305/17.4.2013.07

Start Developing iOS Apps Today: This is Apple's official starting guide. It walks you through setting up Xcode, structuring your app, implementing everything, and submitting it to the App Store. Introducing Swift: Apple's new programming language, Swift, is made specifically for iOS and Macs. It's supposedly much easier to work with and use, so if you're totally new to iOS development it's a good place to start. It works with and is similar to Objective-C (which you can also use if you prefer).Â Stanford's iOS Development Classes: Stanford has a set of free classes to learn iOS development. It's still only available for iOS 7, but most things you learn should transfer over nicely to iOS 8. Chances are they'll have an updated class for iOS 8 in the near future. Advertisement. An online development course that teaches you how to build iPhone applications using Objective-C. A learn by doing course. Created by. Mark Petherbridge.Â At the end of each section I would also like to develop an online quiz were the learner can type code directly into the browser and is able to recap and thus better retain the information taught in the course. Why am I building this? Three years ago I started iOS-Blog (http://bit.ly/1hnkkRz) with the hope of helping the iPhone application development community. At the time I was bewildered by Objective-C, desperately wanting to learn but finding it hard to find informative, concise and valid information to learn from.