Software Framework for REST Client Android Applications: Canvas LMS Case Study

Milan Pandurov, Srđan Milaković, Nikola Lukić, Goran Savić, Milan Segedinac, Zora Konjović
Faculty of Technical Sciences, University of Novi Sad
milanpandurov@gmail.com, srki@outlook.com, luknik94@gmail.com, {savicg, milansegedinac, ftn_zora}@uns.ac.rs

Abstract – **The paper presents a software framework for developing Android applications. The framework has been designed for mobile applications which mainly operate as a thin client for accessing functionalities provided by RESTful web services. The framework contains four layers. The first layer provides network communication with RESTful services. The second one is responsible for the storage and processing of the data at the client side, while the third layer handles interaction with a user. The fourth layer serves as a mediator between data storage and user interface. The proposed framework has been verified on the case study of developing mobile Android application for Canvas LMS. The application communicates with REST API of Canvas LMS enabling users to use common Canvas features on a mobile device.**

INTRODUCTION

Modern software applications must deal with increasing diversity of software and hardware platforms, as well as with issues related to users' mobility. This has led us to current trends in developing internet-based applications where a server provides core functionalities that are accessed from (usually "thin") client applications. A traditional approach for accessing internet-based applications included internet browser which was installed on a personal computer. Wider use of mobile devices has set a demand to access server functionalities using native mobile applications. Such applications have been specifically written for an operating system executing on a mobile device. Currently, the most popular applications of this type are those written for Android and iOS operating systems. When it comes to internet-based applications, beside platform-dependent differences, all client applications access the same core functionalities which are set on server computers. Client applications vary only in the manner in which they communicate with the server and handle user interaction. For this reason, there have been developed platform agnostic mechanisms within server-side applications that can be accessed from any client platform.

Web services [1] are the standard solution for cooperation between various client platforms. Lately, a particularly popular implementation of web services is RESTful [2], which is based on REST software architecture [3].

RESTful services provide client with a uniform access to system resources. In REST terminology, a *resource* includes data or functionalities, where each resource is identified by its uniform resource identified (URI). A client interacts with RESTful services through very limited set of operations with predefined semantics. Typically supported operations are *create* (PUT), *read* (GET), *update* (POST) and *delete* (DELETE). Operation GET gets the current state of the resource. The purpose of the POST operation is to change resource's state. Operation PUT creates a resource, while DELETE operation removes it. Resources itself are separated from their representation format. It means that the same resource may be transferred in different formats, such as HTML, XML, JSON, etc. Since REST architecture proposes a stateless protocol for communication between client and server, RESTful services are designed to use HTTP protocol.

This paper presents the architecture of a software framework for developing client Android applications that access server functionalities which are exposed as RESTful web services. Since this type of Android applications is very common currently, the paper gives a general development framework neutral to any specific domain or application functionalities.

The proposed framework has been verified on developing an Android application for Canvas LMS [4]. The application access RESTful web services of Canvas LMS enabling students to access common information about courses they are enrolled.

RELATED WORK

Our paper is focused on client applications that meet following requirements:

- They receive/send data through network from/to RESTful web services contained within the server-side application
- They have its local data storage that cache data received from server
- They implement their own UI logic

Dobjanschi in [5] proposed software patterns for communication with RESTful services from Android application. The proposed patterns send/receive data using Android Service API or Android Content Provided API. The web page [6] contains a source code of sample implementation of this pattern.

Most solutions for communication between Android application and RESTful services are based on Dobjanschi's work. His solution is primarily focused on the communication with services, not considering other components in the Android application that should process, store and display data received from the network.

When it comes to data management in a client application, besides data fetching, the application must store some of the received data locally on the mobile device. This local cache may speed up the application and optimize network traffic, given that application fetches data only when necessary. This implies data synchronization between client and server. The paper [7] proposes different synchronization patterns which can be used for this purpose. Next chapter describes synchronization methods used in the framework proposed in this paper.

UI layer of Android application must provide various visual controls for user interaction, while being at the same time flexible enough to combine these controls on the device display dynamically. A standard approach to provide this feature is by using Android Fragments [8]. A fragment [9] is a reusable UI component with UI controls and its own UI logic. It is displayed within an Android Activity element. A single fragment may contain its subfragments, while an activity may be composed of multiple fragments. A fragment can be dynamically added, removed or replaced within an activity.

Aforementioned solutions give proposals for individual requirements of REST-based Android applications. This paper's aim is to propose a comprehensive framework which covers all layers of Android applications of this type. The framework proposed in this paper implements common functionalities for each layer, while covering the connections between layers, too. The next chapter proposes the framework architecture.

Pattern Components

For many years, *de facto* standard for developing modular applications is a Model-View-Controller (MVC) software pattern. This pattern distinguishes three general application layers: data representation layer (Model), user interaction layer (View), and a layer that links data with displaying logic (Controller).

The framework proposed in this paper is also MVC-based. Figure 1 shows the general architecture of the framework and its place within REST-based Android application. An android application accesses server's REST API by executing REST methods. The server sends its response formatted as a JSON object. Android application is set on a mobile device and organized in conformance with our framework, which is MVC-based.

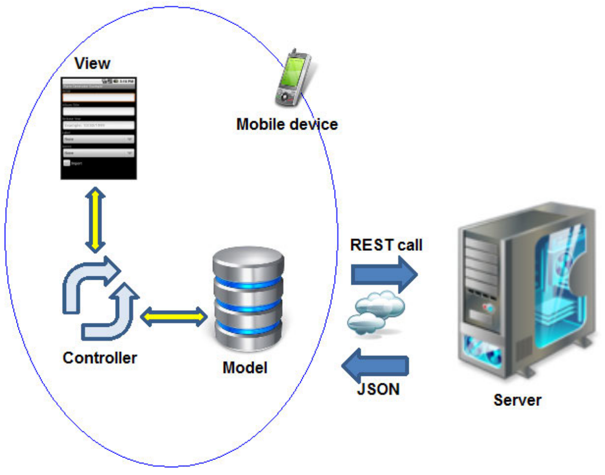


Figure 1. Framework global architecture

For each component in the framework, further text describes its functionalities and subcomponents.

As mentioned, *Model* component provides data management. Figure 2 shows its subcomponents.

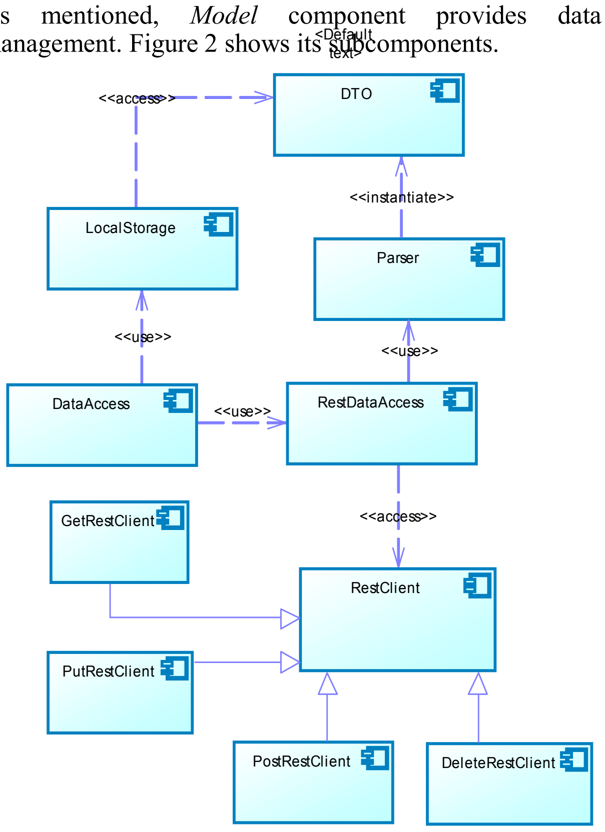


Figure 2. Model component

DataAccess is a proxy for data management. Other global components (like *Controller*) communicate with *DataAccess* to fetch/send data. Data is fetched from the application's local storage. For storing data on the mobile device, the framework uses *LocalStorage* component.

The in-memory representation of the data is given within *DTO* (*Data Access Object*) component. This component contains an object-model of the application's data, where each domain entity should be represented with an appropriate class holding entity's data.

When the application starts there is no application's data on the mobile device. They should be obtained from the server through its REST API. *RestDataAccess* component provides this functionality. The network communication with the server is performed within *RestClient* component. For the communication with REST services, the framework uses built-in Android classes that execute REST methods through HTTP protocol. With regard to the differences between REST methods, we propose subcomponents for executing *GET*, *PUT*, *POST* and *UPDATE* REST methods, respectively.

During the communication with the server, data will be passed in JSON format. *Parser* component is responsible for data conversion from JSON format to DTO in-memory representation, which is used by application local storage.

UI layer rely on built-in Android UI components. Figure 3 shows this layer architecture.

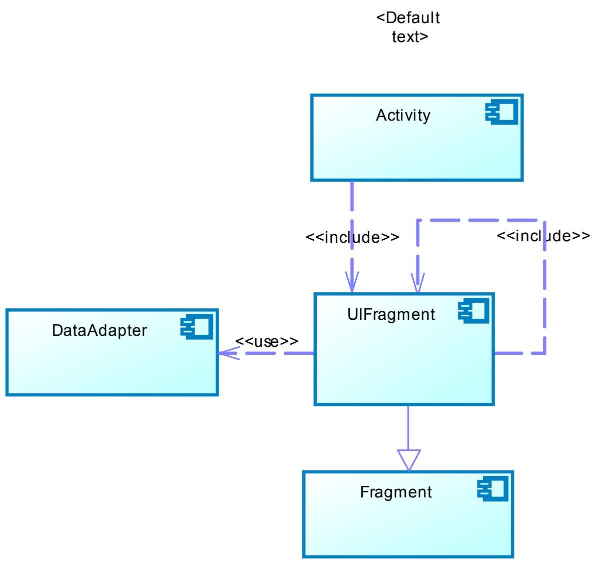


Figure 3. View component

The framework proposes just a single *Activity* element within a whole application. This *Activity* will contain different *Fragment* elements that provide interaction with a user. Fragments are represented by *UIFragment* component whose implementation relies on built-in Android *Fragment* class. *UIFragment* may contain child *UIFragment* elements. Fragment is populated with content using *DataAdapter* component. For each UI control, *DataAdapter* component will contain a separate class that contains specific logic for setting control's content. When a user navigates within the application, only fragments contained in the *Activity* element will be changed, while the Activity remains the same. Fragment management is responsibility of *Controller* layer, which is described below.

Controller layer manages UI navigation and links UI layer with data defined within *Model* layer. The architecture of *Controller* layer is shown in Figure 4.

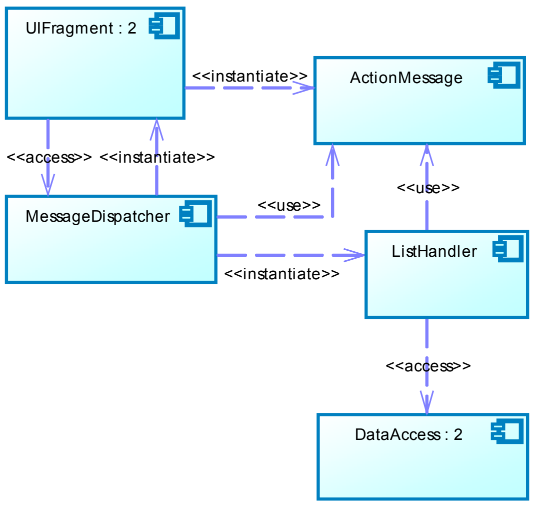


Figure 4. Controller component

UIFragment, as a part of user interface, reacts to the specific UI event (e.g. button-click) and generates a message represented by *ActionMessage* component. The message holds information about UI event. *UIFragment* passes the message to *MessageDispatcher*, which instantiates a new *UIFragment*, if necessary. This new fragment can replace current fragment in the *Activity*. *UIFragment* displays data received from *Controller* layer. *Controller* loads data from *Model* layer using *ListHandler* component which is a bridge between *Controller* and *DataAccess* component.

AUTHENTICATION

Concerning the fact that most of the REST-based server applications support user management, in addition to mention components, the framework introduces a special-purpose component providing user authentication. Since they are typically based upon the stateless HTTP protocol, the authorization in most of the REST-based server applications is achieved by using access tokens. An access token is a random server generated opaque string assigned to each authenticated user when logging in to the system. The user is required to introduce itself to the server by sending the access token along with every request. The main advantage of such an approach is the fact that clients never send their passwords (not even encrypted) to the server, but only random strings. In addition to that, it is possible to restrict the duration of an access token and to deactivate the access token manually, so to prevent possible abuses.

User authentication and access tokens generation in the proposed component is based upon OAuth2 protocol. Such an approach prevents client application to abuse the data entered when logging into the system, since entering usernames and passwords is delegated to the server, while client application manages only access tokens. OAuth2 protocol requires each client application to have its ID and secret key, obtained when registering the application to the system. Figure 5 shows the sequence of activities performed when authenticating a user in a system that uses OAuth2 protocol.

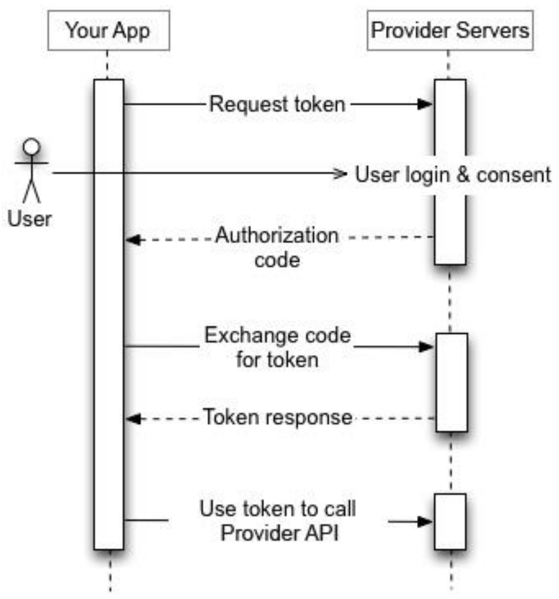


Figure 5. OAuth2 authentication

The first step in the authentication process consists of redirecting the user to the server application's page where he or she enters username and password. If the entered data are correct, the server application assigns temporary code to the user. After that, the client application sends the temporary code and the application's secret key to server application. The response to this request contains an access token that is being sent along with every other request.

This authentication mechanism is suitable when client is a Web application so that the secret key needs not be persisted at the client's local machine. In cases when client is a mobile or a desktop application there is a risk of stealing secret keys by decompiling the executable code of the application (if the secret key is hardcoded into the application or if the application keeps the secret key in working memory).

To avoid this problem, a new component is introduced: a Web application that mediates in the process of assigning the access tokens, namely Authentication server. The sequence of authentication activities in such a system is shown in figure 6.

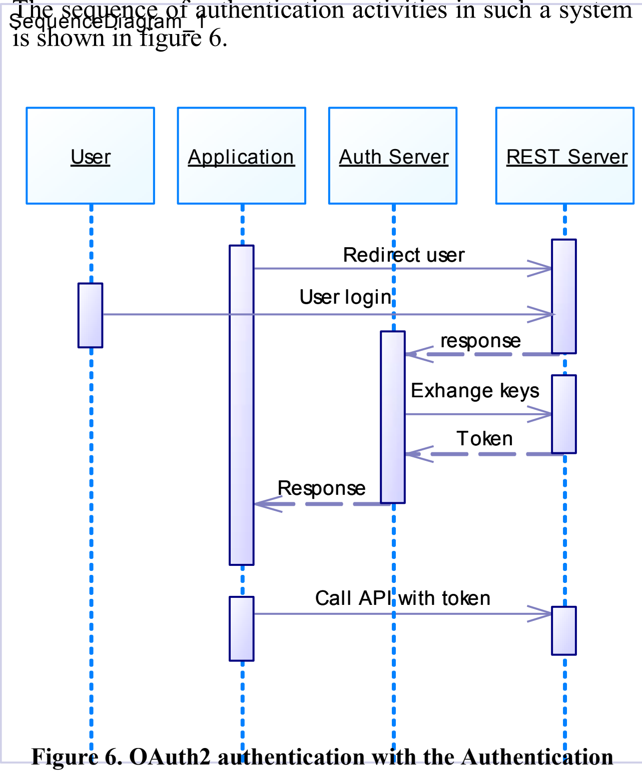


Figure 6. OAuth2 authentication with the Authentication server

In the proposed approach, instead of letting client application itself to keep the temporary code, the temporary code is being kept by the Authentication server, which holds the secret key as well. The Authentication server's task is to complete the authentication process and to send the access token to the client application. This mechanism denies client application ever to get hold of the secret key and reduces the risk to stealing the user's access token. Even if this would happen, the overall system integrity would not be threatened.

Canvas Android Client

Canvas LMS [4] is an open source cloud-native learning management system developed by Instructure. This LMS provides a wide range of e-learning functionalities based upon web 2.0 technologies, such as e-learning tools, tools that support collaborative work and system administration tools [10]. Canvas LMS itself is a Ruby on Rails application, but it can be easily extended with third party tools regardless of the particular language in which the tools have been developed, since it supports IMS LTI standard.

Instructure offers a mobile application for Canvas [11]. Even though Canvas LMS is an open source application, Canvas for Android is not open source and it can be used only in combination with the Instructure hosted instances of Canvas LMS. Therefore, this paper proposes an open source Android application for Canvas LMS (Canvas Android Client - CAC) that can be used in educational settings with self-hosted instances of Canvas LMS. CAC is based upon the framework for developing android REST clients proposed in the previous parts of this paper.

Canvas LMS has an open REST API through which it exposes some of its operations as external services. The operations that are exposed via the API include, among others, mechanisms for accessing assignments, course information, registration, roles, users and discussion topics. Full specification of Canvas API can be accessed at [12]. CAC is a thin REST client, and its entire functionality comes down to calling the services from Canvas REST API and interpreting the results. Since calling the REST services requires the client to be authenticated, the authentication mechanism is described in details in the *Canvas authentication* section that immediately follows. The set of functionalities that are supported by the current version of CAC is described in the *CAC functionalities* section.

The current version of CAC supports following functionalities:

- Browsing courses
- Browsing announcements
- Browsing assignments

All the functionalities require user to be authenticated. After the user has successfully logged in to the system, she or he can choose the action from the menu shown in figure 7.

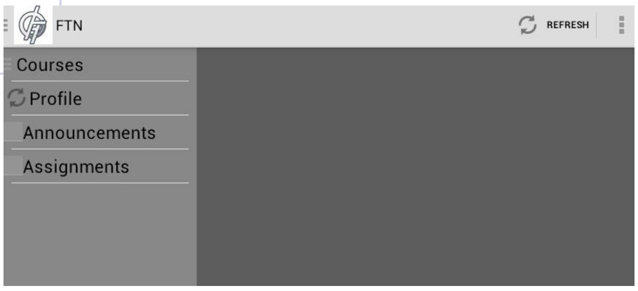
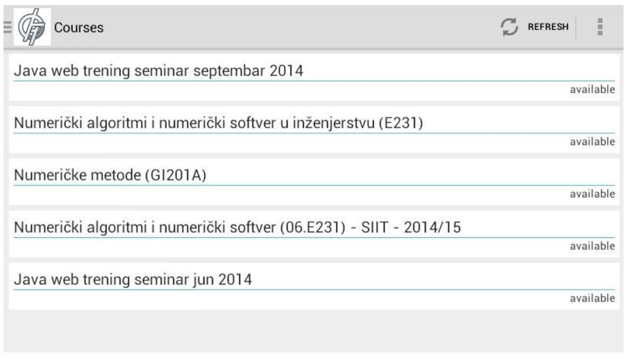


Figure 7. CAC main menu

When a user chooses the item *Courses* from the main menu, the list of all courses is being presented, as shown in Figure 8. Each item in this list has course name and the information that indicates if the course is still available.

When user chooses one item from the list, a view with the details about the selected course is being shown, as in the figure 9. In addition to the general information on the course, this view contains a list of announcements and a list of assignments from the selected course.


Courses REFRESH

Java web trening seminar septembar 2014
available

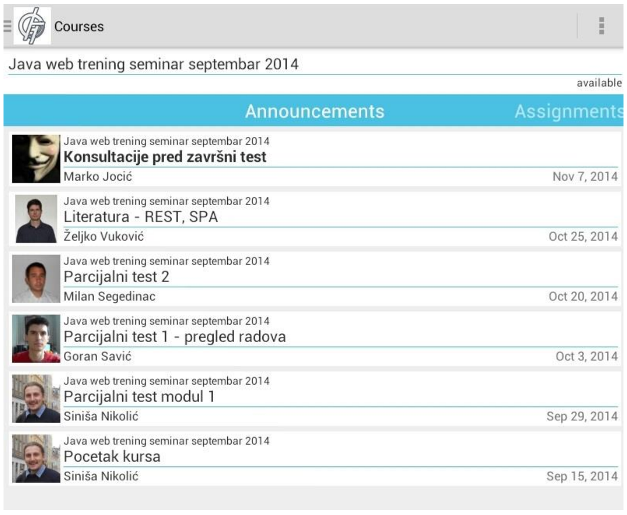
Numerički algoritmi i numerički softver u inženjerstvu (E231)
available

Numeričke metode (GI201A)
available

Numerički algoritmi i numerički softver (06.E231) - SIIT - 2014/15
available

Java web trening seminar jun 2014
available

Figure 8. Courses


Courses

Java web trening seminar septembar 2014
available

Announcements Assignments

Java web trening seminar septembar 2014
Konsultacije pred završni test
Marko Jocić Nov 7, 2014

Java web trening seminar septembar 2014
Literatura - REST, SPA
Željko Vuković Oct 25, 2014

Java web trening seminar septembar 2014
Parcijalni test 2
Milan Segedinac Oct 20, 2014

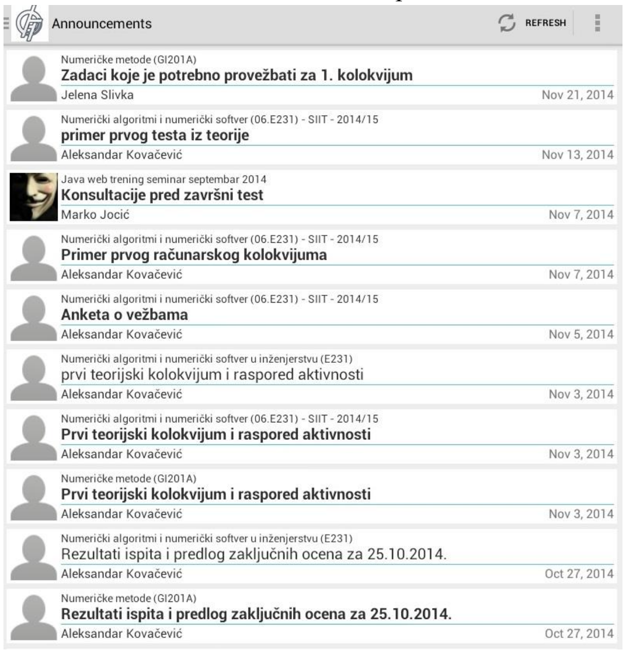
Java web trening seminar septembar 2014
Parcijalni test 1 - pregled radova
Goran Savić Oct 3, 2014

Java web trening seminar septembar 2014
Parcijalni test modul 1
Siniša Nikolić Sep 29, 2014

Java web trening seminar septembar 2014
Pocetak kursa
Siniša Nikolić Sep 15, 2014

Figure 9. Course details

When the user chooses the item Announcements from the main menu (figure 7), the list of all announcements from all the courses that the user is enrolled is being presented, as shown in figure 10. Each item in this list contains the avatar of the user who has posted the announcement, course name, as well as the announce title and date the announcement has been posted.


Announcements REFRESH

Numeričke metode (GI201A)
Zadaci koje je potrebno provežbati za 1. kolokvijum
Jelena Slivka Nov 21, 2014

Numerički algoritmi i numerički softver (06.E231) - SIIT - 2014/15
primer prvog testa iz teorije
Aleksandar Kovačević Nov 13, 2014

Java web trening seminar septembar 2014
Konsultacije pred završni test
Marko Jocić Nov 7, 2014

Numerički algoritmi i numerički softver (06.E231) - SIIT - 2014/15
Primer prvog računarskog kolokvijuma
Aleksandar Kovačević Nov 7, 2014

Numerički algoritmi i numerički softver (06.E231) - SIIT - 2014/15
Anketa o vežbama
Aleksandar Kovačević Nov 5, 2014

Numerički algoritmi i numerički softver u inženjerstvu (E231)
prvi teorijski kolokvijum i raspored aktivnosti
Aleksandar Kovačević Nov 3, 2014

Numerički algoritmi i numerički softver (06.E231) - SIIT - 2014/15
Prvi teorijski kolokvijum i raspored aktivnosti
Aleksandar Kovačević Nov 3, 2014

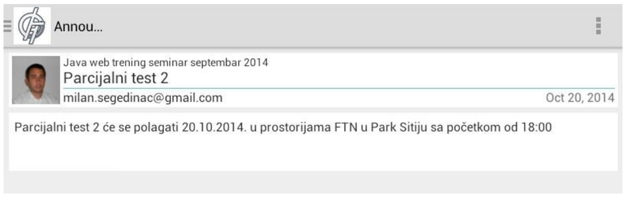
Numeričke metode (GI201A)
Prvi teorijski kolokvijum i raspored aktivnosti
Aleksandar Kovačević Nov 3, 2014

Numerički algoritmi i numerički softver u inženjerstvu (E231)
Rezultati ispita i predlog zaključnih ocena za 25.10.2014.
Aleksandar Kovačević Oct 27, 2014

Numeričke metode (GI201A)
Rezultati ispita i predlog zaključnih ocena za 25.10.2014.
Aleksandar Kovačević Oct 27, 2014

Figure 10. Announcements

When an item in the list is selected, the general information along with the content of the announcement is shown, as presented in figure 11.

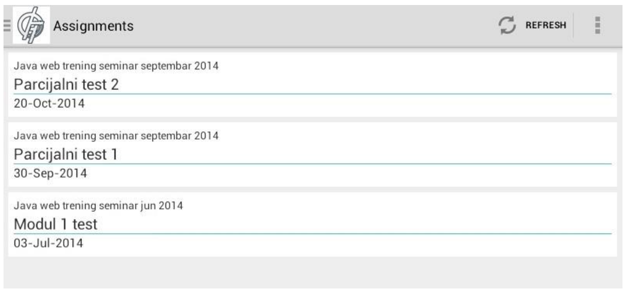

Annou...

Java web trening seminar septembar 2014
Parcijalni test 2
milan.segedinac@gmail.com Oct 20, 2014

Parcijalni test 2 će se polagati 20.10.2014. u prostorijama FTN u Park Sitiju sa početkom od 18:00

Figure 11. Announcement details

When a user chooses the item assignments from the main menu (figure 7), the list of all current assignments from all the courses that the user is enrolled part in is being shown, as presented in figure 12. Each item in this list contains course title, assignment name as well as the date upon which the assignment is available.


Assignments REFRESH

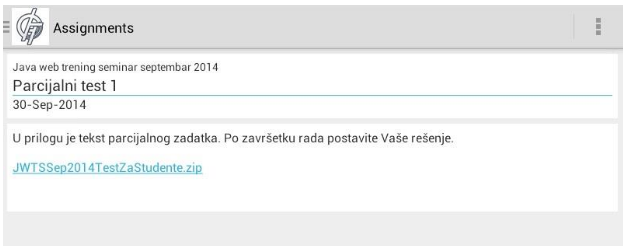
Java web trening seminar septembar 2014
Parcijalni test 2
20-Oct-2014

Java web trening seminar septembar 2014
Parcijalni test 1
30-Sep-2014

Java web trening seminar jun 2014
Modul 1 test
03-Jul-2014

Figure 12. Assignments

When one item from the assignments list is being selected, a view with the assignment details containing assignment general information along with the content of the assignment is being presented, as shown in figure 13.


Assignments

Java web trening seminar septembar 2014
Parcijalni test 1
30-Sep-2014

U prilogu je tekst parcijalnog zadatka. Po završetku rada postavite Vaše rešenje.

JWTSSep2014TestZaStudente.zip

Figure 13. Assignment details

CONCLUSION

A new software framework for developing REST-based Android applications has been proposed. The framework can be used for any thin-client Android application that access functionalities exposed as REST web services. For this type of Android applications, the framework proposes application architecture, as well as particular implementation of common functionalities. The framework generally follows MVC pattern, organizing programming code into three layers where each layer provides a single point for communication with other layers. Concerning the fact that most of the REST-based server applications support user management, in addition to mention components, the framework introduces a special-purpose component providing user authentication.

The framework has been evaluated on the Android application for Canvas LMS. In contrast to the official Canvas for Android application developed by Instructure, which can be used only in combination with Instructure-hosted Canvas instances, the application proposed in this paper is open-source and can be used with self-hosted Canvas instances. Current version of the application provides access to common Canvas features within mobile environment.

The future plans for the proposed application include:

- Start using the application at the Faculty of Technical Sciences at University of Novi Sad
- Extending current set of the application’s functionalities with new features
- Integrating the application with other educational services at University of Novi Sad

ACKNOWLEDGMENT

Results presented in this paper are part of the research conducted within the Grant No. III-47003, Ministry of Education, Science and Technological Development of the Republic of Serbia.

REFERENCES

[1] W3C (2004), Web Services Glossary". Retrieved 24.11.2014

[2] L. Richardson, S. Ruby (2007), RESTful web service, O'Reilly Media, ISBN 978-0-596-52926-0

[3] R. Fielding, R. Taylor (2002), Principled Design of the Modern Web Architecture (PDF), *ACM Transactions on Internet Technology* (*TOIT*) (New York: Association for Computing Machinery) 2 (2): 115–150, doi:10.1145/514183.514185, ISSN 1533-5399

[4] Instructure Inc. (2012), Canvas LMS, http://www.instructure.com, Retrieved: 24.11.2014.

[5] V. Dobjanschi (2010), Developing Android REST Client Applications, https://dl.google.com/googleio/2010/android-developing-RESTful-android-apps.pdf, Retrieved 24.11.2014.

[6] CodeProject (2012), Sample Implementation of Virgil Dobjanschi's Rest pattern, http://www.codeproject.com/Articles/429997/Sample-Implementation-of-Virgil-Dobjanschis-Rest-p. Retrieved 24.11.2014.

[7] Z. McCormick and D. C. Schmidt (2012), Data Synchronization Patterns in Mobile Application Design, *Proceedings of the Pattern Languages of Programs* (*PLoP*) *2012 conference*, Tucson, Arizona.

[8] J. Wilson (2013), Creating Dynamic UI with Android Fragments, *Packt publishing*, ISBN: 9781783283095

[9] Android Developers (2014), Fragments, http://developer.android.com/ guide/components/fragments.html. Retrieved 24.11.2014.

[10] N. Nikolić, G. Savić, M. Segedinac and Z. Konjović (2014), Migration from Sakai to Canvas, P*roceedings of the 4th International Conference on Information Society and Technology (ICIST 2014)*, Kopaonik, Serbia, 366 – 370, ISBN: 978-86-85525-14-8

[11] Instructure Inc (2014), Canvas for Android, https://play.google.com/store/apps/details?id=com.instructure.candroid, Retrieved: 26.11.2014.

[12] Instructure Inc. (2012), Canvas LMS API Documentation, https://canvas.instructure.com/doc/api/, Retrieved: 26.11.2014.