Blood Notes: Software System for Promoting and Facilitating Blood Donation

Hunor Hegedüs
Babes-Bolyai University
Cluj-Napoca, Romania
hegedus.huni@gmail.com

Kata Szász
Babes-Bolyai University
Cluj-Napoca, Romania
szaszkata97@gmail.com

Károly Simon
Babes-Bolyai University
Cluj-Napoca, Romania
simon.karoly@codespring.ro

Tibor Fazakas
Codespring
Cluj-Napoca, Romania
fazakas.tibor@codespring.ro

Andor Mihály
Codespring
Cluj-Napoca, Romania
mihaly.andor@codespring.ro

Katalin Nagy
Codespring
Cluj-Napoca, Romania
nagy.katalin@codespring.ro

***Abstract*—The Blood Notes project aims to subserve and popularize blood donation, respectively to support an effective and permanently extant communication channel between the project's two target audience. One of the target audience contains persons who are willing to donate blood. For them, the system provides a mobile application for logging and scheduling their donations, for receiving notifications and for accessing useful information related to the blood donation process. The other target audience includes employees working at blood donation centers. Using a web application, they are able to manage the informative data published through the mobile application, and they are able to send notifications about the current blood necessities.**

The software system has three main components: a central server, a mobile application for donors which is running on iOS and Android platforms, and a web application for the employees of the blood donation centers.

The paper presents the functioning and the architecture of the system, listing its features together with the used technologies and development tools.

I. Introduction

Although there is a huge need for blood donors, blood donation does not enjoy sufficient popularity, and unfortunately, compared with other regions, the situation is even worse in Romania from this perspective. While in Austria 70 percent and in France 55 percent of the population donates blood, in Romania this proportion is only 1,7 percent. The health institutes and NGOs are trying to draw attention to the huge necessity, and to mobilize the population with various campaigns and events, but they can hardly obtain relevant results. The Blood Notes project presented in this paper aims to contribute to this popularization process.

The Blood Notes application creates a connection between the blood donation centers and donors, allowing the quick information exchange, and providing an easy-to-use, user-friendly platform for the users.

Before blood donation, each donor has to fill out a questionnaire at the blood donation center. Based on the answers it is decided if he/she is eligible for blood donation. Through the Blood Notes mobile application this form can be filled out online, and the answers are evaluated instantly. In this way the donors are able to test their eligibility in advance, saving time and avoiding pointless journeys to the blood centers. Furthermore, the users can log their past donations to be aware of the next time when they are allowed to give blood again, and they will receive a notification about this date. They can also receive notifications if someone needs blood urgently.

The above-mentioned notifications about urgent blood necessities are broadcasted by the blood donation centers and hospitals through the web application. The notifications will be received only by the users within that region, having the adequate blood type and being able to give blood in the given period. Due to this feature, the blood donation centers are able to develop a stable "donor-camp" around them. The employees of these centers also have the possibility to manage the informative data displayed within the mobile application: they can edit the descriptions about the blood donation process, the contact information related to blood donation centers and the content of the donor questionnaire.

The remainder of the paper is structured as follows: Section II. enumerates the features corresponding to different user roles, Section III. presents the system's architecture and Section IV. describes the used technologies and development tools. Finally, some conclusions and further development possibilities are listed.

II. User Roles and Corresponding Features

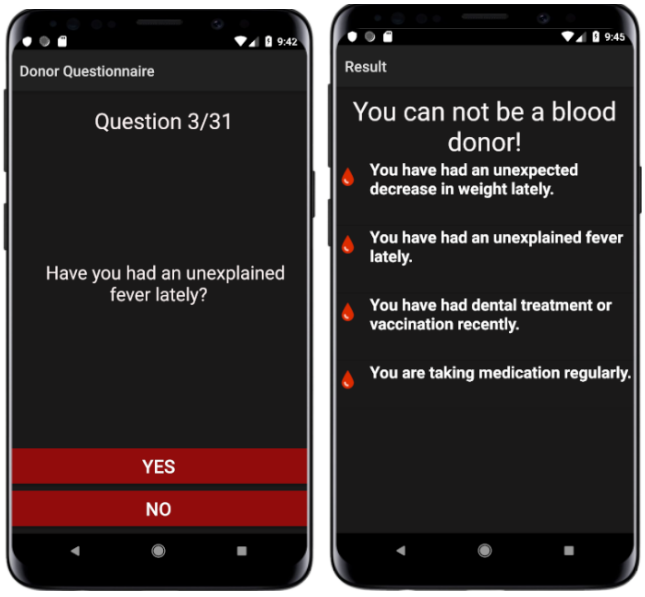
The Blood Notes system has two different user interface: a web application and a mobile application.

From the perspective of the mobile application, there are non-registered and registered users, while the web application can be accessed by users having operator or administrator roles.

In this section the main features of the system are presented based on the aforementioned roles.

A. Non-registered user

A non-registered user can only access a few features within the mobile application. After launching the application some



(a) One of the questions of the donor questionnaire.

(b) An unsuccessful test result.

Fig. 1. The donor questionnaire's structure.

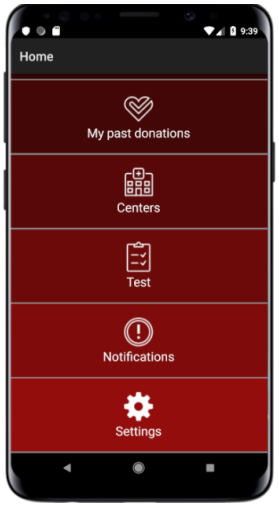


Fig. 2. Features provided by the mobile application for registered users.

general information are displayed about the blood donation process. After reading these pages, the users can access the list containing the blood donation centers and their contact information, and they have the possibility to fill out the donor questionnaire. The user can answer the questions with yes or no, the test is instantly evaluated and the results are displayed. If the user is not eligible for blood donation, then the concrete reasons will be enumerated (see Fig. 1).

B. Registered user

The registered users have all the rights of the non-registered users, and in addition they are allowed to access more features (see Fig. 2). They can log their blood donations, and they can receive two types of notifications: notification about the date when they can give blood again, and notifications from the blood donation centers about current blood necessities.

C. Operators

The operators are employees working at the blood donation centers. They have access to the web application and they are managing the data accessible within the mobile application.

They can manage the list of blood donation centers, they can edit the questionnaire and the content of the informative pages (see Fig. 3). In the case of the informative pages a preview is also available for them, which helps to check how the mobile application will look like after the updates. Lastly, the operators can send notifications to the mobile users if a patient needs blood urgently.

D. Administrators

For the administrators all the features of the web application are available. They are able to manipulate all the data mentioned in Subsection II-C., and they can also manage the operators.

III. Architecture

The software system can be divided into three main components: a *mobile application* for donors, which works on iOS and Android platforms; a *web application* developed in React.js for the employees of the blood donation centers; a .NET Core *server*, which connects the first two components, respectively serves and manages the data. All these main components have multi-layered architectures [1]. The server is communicating with a MySQL database and with the Firebase Cloud Messaging (FCM) platform. FCM is used for broadcasting notifications from the server to the mobile clients.

The communication between the server and the mobile-, respectively the web clients is realized based on the Hypertext Transfer Protocol (HTTP), corresponding to the Representational State Transfer (REST) conventions. The communication with the MySQL database is implemented using the Entity Framework.

These relationships are represented on Figure 4. and detailed in the following subsections.

A. Server

The server's main responsibility is to serve the clients with data. The requests are received by *Controller* components, and these controllers interact with the data access layer to execute the requested operations.

The classes in the *DBModels* package are representing the data stored in the database, while the classes in the *Models* package represent the data available for the users. The transformation between these two types of classes is realized by *Assembler* components, using the AutoMapper plugin. The classes stored in the *Modules* package have a role in realizing the system's functionalities.

The architecture of the server is represented on Figure 5.

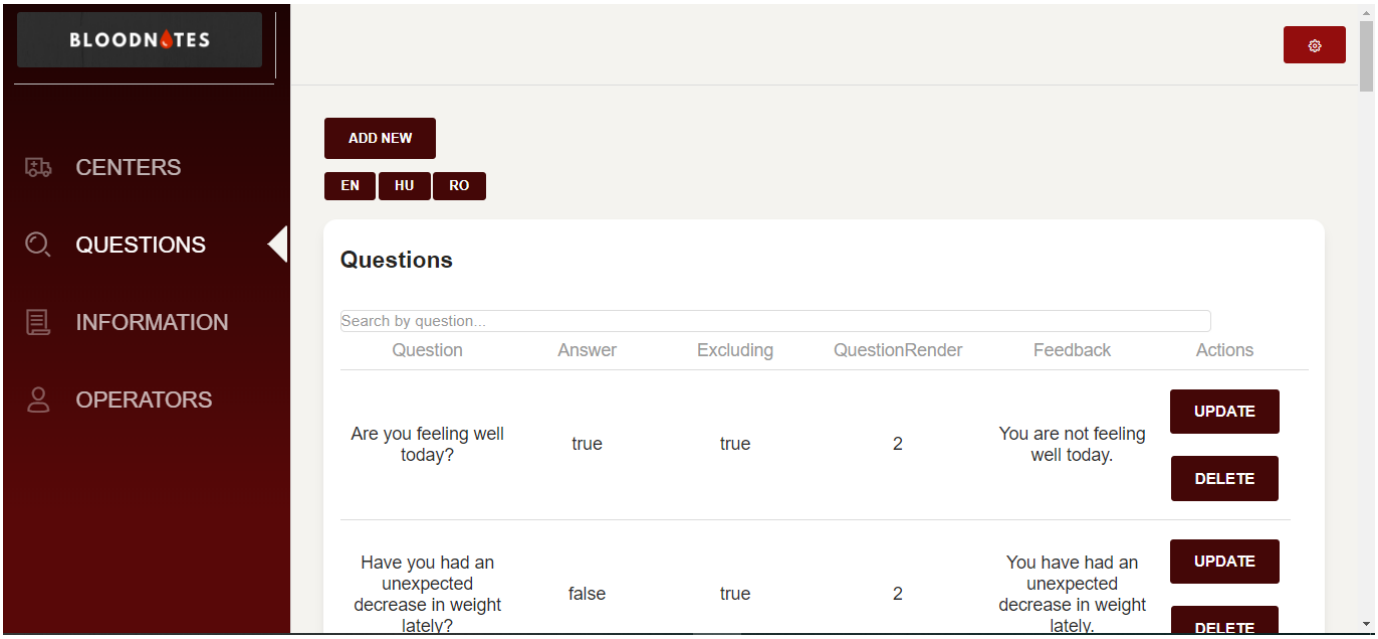


Fig. 3. The web application interface: the content of the application is available and can be managed in three languages (Hungarian, English and Romanian).

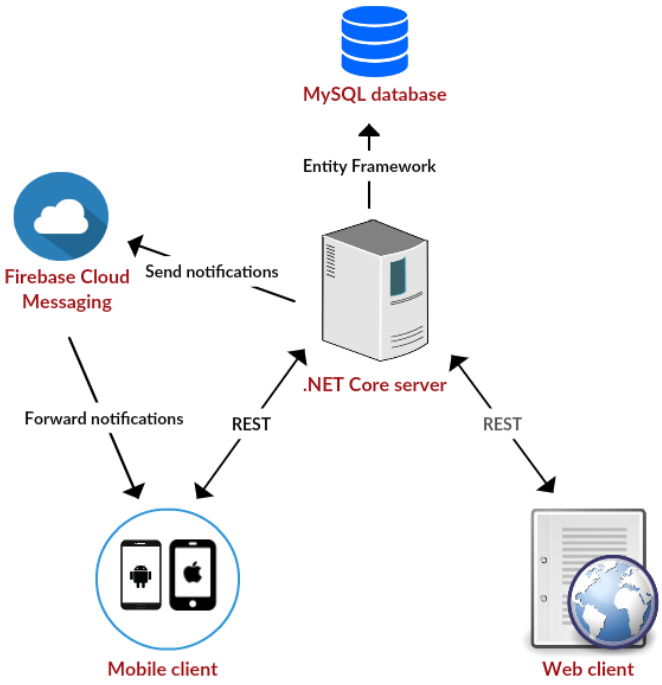


Fig. 4. The architecture of the Blood Notes software system.

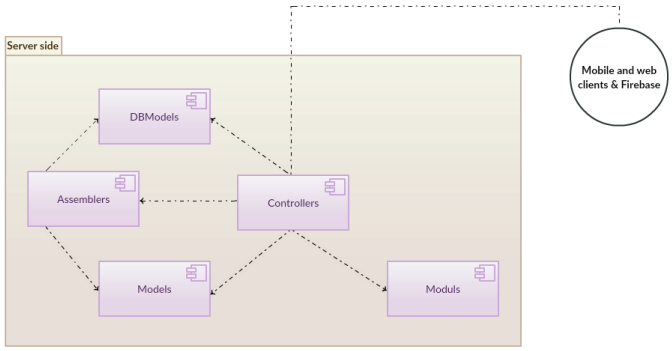


Fig. 5. The component diagram of the server.

1) Data model: The data is stored in a MySQL relational database. Each table from the database is represented by a corresponding model class. For example, the table, which contains information about the blood donation centers is represented by the *BloodDonationCenters* class, having the corresponding attributes, like center name, phone number, address, etc. Similarly, there is a model class for the introductive information, for the questionnaire and for the operators.

In the case of the operators the system stores their name, email and role, together with their encrypted password and firebase token. An operator has a firebase token only if he/she is subscribed for notifications from the web application.

There is no need for using SQL commands to execute CRUD (Create, Read, Update, Delete) operations, because the Entity Framework provides simple methods for data manipulation. The ORM (Object Relational Mapper) system provides two pattern for managing the database: the Code-First and the DataBase-First patterns. The Blood Notes follows the Code-First pattern, which means that the first time the application starts, the server checks if the database exists or not, and if not, then creates the relational schema based on the model classes. The newly created tables are filled using predefined SQL scripts.

2) Security: The HTTP requests received from the mobile and the web application are served by an API server, which

operates as a RESTful service. The communication is based on the HTTP protocol using JSON as transfer format.

There are different permissions available for different roles (detailed in Section II.). The users can access only the contents available for their role.

After a successful login operation, the server generates a JSON Web Token (JWT) for the user, which includes the username and the token's expiration-date in the form of key-value pairs, using an encryption method. For each request the generated token will be included into the HTTP *header*. In the case of the web application the tokens are saved in the *sessionStorage* on the client side.

B. Mobile

The mobile application operates on iOS and Android platform. It is developed using the Xamarin Cross Platform technology, and in this way the two versions have a common C# codebase. Most of the *business logic* is written in this shared library, but the *user interfaces* have to be implemented separately. The architecture of the mobile application is represented on Figure 6.

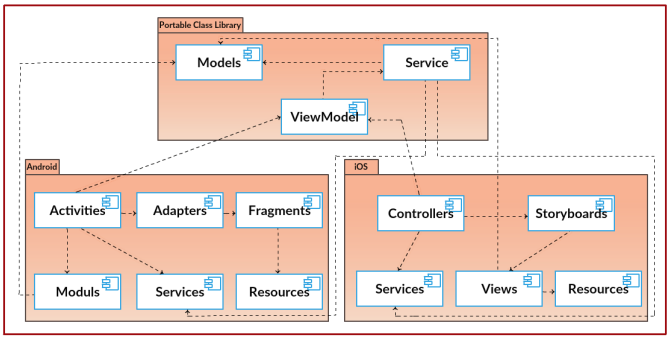


Fig. 6. The components of the Blood Notes mobile application

1) Communication with the server: The mobile application communicates with the server through a RESTful API. The REST requests are implemented in the shared library containing the common codebase for the Android and iOS applications.

The requests are sent by the *HttpClient* class from the *.NET System.Net.Http* namespace, using the *SendAsync()* method. This method has only one parameter, a *HttpRequestMessage*, which specifies the path and the type of the request (GET, POST, PUT, DELETE). The response is created by the *Newtonsoft.Json* framework, the data is transferred in JSON format, and the data processing continues separately on the two platforms.

2) Business logic layer: On mobile side the business logic is implemented in the *Portable Class Library*. The model classes are stored in the *Models* sub-package. These classes are used by the business logic components from the *Services* sub-package to implement the required operations. The services are invoked from the *ViewModel*, which communicates with the view components from the iOS and Android libraries.

3) User interface: While the business logic is implemented in the shared library, in the case of the user interface different components are used for the two supported platforms. The Android user interface is a hierarchy of *Views*, where a *view* is a given screen from the application. *Activities* instantiated from the *Activity* class are responsible for controlling the *views*.

A declarative approach is considered for creating the *views*, using AXML files. Within these files the screens are assembled using *tags, layouts and UI widgets*. Each of these elements has a corresponding class in the codebase. The views defined in the AXML file can be easily created by the *activities*, usually within the *onCreate()* method.

The *Resources* package contains the media and style files, together with the resources required for localization and internationalization.

The iOS user interface is developed using *.xib* and *.storyboard* files provided by the *Interface Builder*. A *.xib* file defines a *view* and a *.storyboard* file contains a set of views with a continuous transitions between them. The views are located in the *Content View Hierarchy* and *ViewControllers* are used for handling these views. Each view has its specific view controller, like *SignInViewController, LandingPageViewController, etc.*. All these controllers are instantiated from the *iOSViewController*, which is responsible for updating the views, for responding to user interactions, for handling the constraints of the views and layouts, respectively for communicating with the other components.

C. Web

The web application provides a user interface for data manipulation and for broadcasting notifications to the mobile clients. It has a component based architecture implemented in *React.js*. A component is a given page from the web application, stored in the *components* package. The communication with the server is managed by specific *services* stored in the *services* library. The UI components are mapped to URL paths using a *routing* process, which is implemented in the *config* package. Figure 7. presents the architecture of the web application.

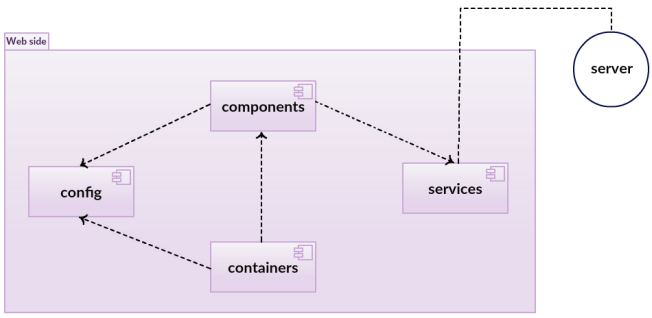


Fig. 7. The architecture of the Blood Notes web application.

1) Communication with the server: The communication between the web application and the server is also based on a RESTful API. The requests and responses are handled by the

Fetch API, which serves a global *fetch()* method, respectively a *Request* and a *Response* object. The response arrives in *JSON* format and it is processed by the *Response.json()* method.

2) Components: The components representing the pages of the web application are implemented in *.jsx* files. Within these files the *JavaScript* code can be completed with simple, HTML-like elements (*tags* with different attributes). These elements are provided by external libraries, like *reactstrap, reactjs-popup, react-confirm-alert*, etc. Each component extends the *React.Component* class, implementing the *render()* method, which adds the component to the Document Object Model (DOM). Within the components *state* objects are used for storing the data received from the server. Each time when the state is changed due to an event (i.e. a button click), the component will be automatically updated.

3) Routing: By the *routing* mechanism the components are assigned to URL paths. *Path-component* key-value pairs are defined in a file and these values are used by the *containers* to display the *components*. For example, if the user would like to reach the page, which contains the blood donation centers, a request has to be sent to the */centers* path and as an effect the *centers* component will appear in the browser.

4) User interface: The browsers are not able to interpret the *.jsx* files directly. The *babel-plugin-transform-react-jsx* plugin is required from the *Babel* library to compile the *.jsx* into *JavaScript* code. From this compiled code React creates a virtual DOM, from which the browser constructs its own DOM and displays the content for the user based on this DOM. The component style of the user interface is defined in *CSS* and *SCSS* files.

5) Security: After a successful login operation, the user's username, token and role are saved in the *sessionStorage*, until the browser closes. When a container is created, the system checks the session's content and the components will be displayed only if the user has the proper permissions. For example, the operators are not able to see the *Operators* item in the menu, because they have no rights for managing other operators.

IV. Technologies and tools

The central server is developed using **.NET Core** [2], which provides platform independence and a rapid development process, by the help of this technology a reliable API server is realised. The C# code is written in Microsoft Visual Studio.

The data is stored in a **MySQL Database** and the server communicates with this database using the **Entity Framework**. Data belonging to the project has a strict, deliberate structure, each of data records, stored in the same table, are build alike. Hence, data are relational, so the MySQL Database had been chosen for storing, because this provides relational schema.

The mobile application is also implemented in Visual Studio using **Xamarin** [3]. The cross-platform technology allows the parallel development of the iOS and Android applications. There is a Shared Library, which contains the business logic layer and this code is used invariably by both the Android and the iOS platform, only the UI layer is implemented separately.

The source code fall into line with the **MVVM design pattern**, because this is one of the best way to write a maintainable, testable and extensible code. Because this pattern can sometimes be hard to implement, the **MVVMLight toolkit** was used for accelerating its creation.

For notification broadcasting the **Firebase Cloud Messaging** Google service is used. The web application sends the notification's content to the server, the server forwards it to the Firebase service, and the messages are broadcasted to the subscribed mobile users.

The Blood Notes data administration interface is implemented using **React.js** [4], a JavaScript library for building web applications. The biggest advantage of React is the possibility to update the content of the view without reloading the whole page. The web application is developed in Microsoft Visual Studio Code and the Node Package Manager (npm) is used for managing the external dependencies. The responsive user interface is created using the Bootstrap CSS framework.

Docker [5] is a container-based virtualization tool, which makes possible to run a software in a virtual environment. It is working with containers providing the required environment for running the application. In the case of the Blood Notes project, there is a container for the application's central server, and a second one for the MySQL database server. These two containers are connected to each other by the Docker Compose tool.

V. Conclusions and further development

Blood Notes is a software system for simplifying and promoting the blood donation process. It provides a mobile application for donors to log their blood donations, to access useful information related to the donation process and to receive notifications. It also provides a web application for the employees of the blood donation centers for data management and notification broadcasting. In its current state the project is a prototype, but it was already successfully presented at several conferences in Seklerland, like the *Digitlis Szkelyfld Konferencia* and the *Erdlyi Tudomnyos Dikkri Konferencia (ETDK)*, where the Blood Notes obtains the II. place in the *Innovative computing products and applications* section.

Gamification is the main priority from the perspective of further development. A ranking system could be integrated into the application, allowing the users to compete with each other. For example, users could gain rewards and achievements by logging their blood donations, by sharing notifications on social platforms, or by inviting new users, etc.

A scheduler module and a calendar could be included into the system, allowing the users to easily reserve free dates at the blood donation centers. Popularization campaigns and promotion events could be also supported by the application.

References

[1] M. Fowler, *Patterns of enterprise application architecture*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[2] M. J. Price, *C#6 and .NET Core 1.0*. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK.: Packt Publishing Ltd., 2016.
[3] c. B. Jonathan Peppers, George Taskos, *Xamarin: Cross-Platform Mobile Application Development*. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK.: Packt Publishing Ltd., 2016.
[4] A. Fedosejev, *React.js Essentials*. Livery Place, 35 Livery Street, Birmingham B3 2PB, UK.: Packt Publishing Ltd., 2015.
[5] J. Turnbull, *The Docker Book: Containerization is the New Virtualization*. James Turnbull, 18092 edition, 2014.