Mobile Application for Daily Challenge Management

Mátyás Gagyi
Babes-Bolyai University
Cluj-Napoca, Romania
reloflex@outlook.hu

Örs-Krisztián Patakfalvi
Babes-Bolyai University
Cluj-Napoca, Romania
p.krisztian@outlook.com

Sándor Ráduly
Codespring
Cluj-Napoca, Romania
raduly.sandor@codespring.ro

Csaba Sulyok
Babes-Bolyai University
Cluj-Napoca, Romania
csaba.sulyok@gmail.com

***Abstract*—Nowadays, the various challenges in which participants have to take certain time-specific steps are growing more and more popular; examples include reading a book every week or doing specific exercises for thirty days. Although enthusiasts are eager and willing to participate, they lack a single and transparent platform.**

The Daily Challenge software project aims to provide a unified interface for users where a multitude of challenges are available for participation.

The project is divided into three components: The central server raises a scalable platform around our database and provides access options for any number of clients. Through the multi-platform native mobile application and the web-based interface, users with different roles may participate and maintain challenges.

This paper provides an in-depth view of the operation, architecture and the functionalities of the system, also fleshing out technologies and tools used.

I. Introduction

Recent years has seen a rise in challenges spreading through social media. Users take hard interest in such pursuits but usually find themselves struggling to find a structure therein. The Internet is full of training, photography and reading challenges, but the professionally made challenges tend to get lost in the mass of low quality ones. A need for a unified and transparent platform is emerging; to find challenges, track performance and read reviews or even encourage others to participate.

The Daily Challenge application is intended to serve as a challenge-based social platform in the everyday lives of its audience. Its primary goal is to provide a unified interface for its users, where all of the challenges are available for participation in one place.

The project can be divided into three main components. The central server raises a scalable platform around the database and provides access options for any number of clients. The native mobile application, which is available on iOS and Android devices, gives users the opportunity to find, follow, join and complete new challenges. The web application provides content managers insight into the user-recommended challenge ideas, allowing them to review, reject or publish challenges. Administrators may also control the activities and roles of users through the web interface.

The remainder of the document is structured as follows: Section II presents the functionalities, broken down by user rights. The general user options within the mobile application, as well as the rights of the content managers and administrators are described. In Section III, the project's architecture is explored, broken down into server, mobile and web components. Section IV contains the technologies and tools utilized during the development process. Section V explains in detail the operation of the mobile application and the web interface, providing an illustrative guide through the possibilities presented to users. Finally, Section VI contains the conclusions and plans for the future extensions and betterments of the Daily Challenge project.

II. Functionalities

The main goal of the Daily Challenge project is to globally expose a unified platform of a multitude of challenges, so the users can easily access them in a convenient and unified place. Core functionalities of the project manipulate critical and often sensitive data, such as editing the published challenges or suspending users, prompting the need for different roles, which can inherit rights from each other. The current roles include:

- The *standard user* can only log in to the mobile application and use all of its functionalities.
- The *content manager* can also log in to the web application and access part of the administration functionalities related to the challenges.
- The *administrator* has the most rights; they can access every functionalities of the project.

A. Functionalities of the standard user

After successful download of the mobile application, a standard user is given the opportunity to log in with their Facebook account. All subsequent mobile functionalities are approved upon login.

The user can view the challenges in which he/she is currently participating in a dashboard-like view, where necessary next steps attributed to these challenges are also shown. They can also view the overview of steps, where they will find information about the current step. The user can move forward or backward between steps and get accurate descriptions and instructions about the previous or following steps. Steps may be marked as completed, prompting the user to move forward towards achieving a full challenge. After the user finishes a challenge, the application congratulates them and shows

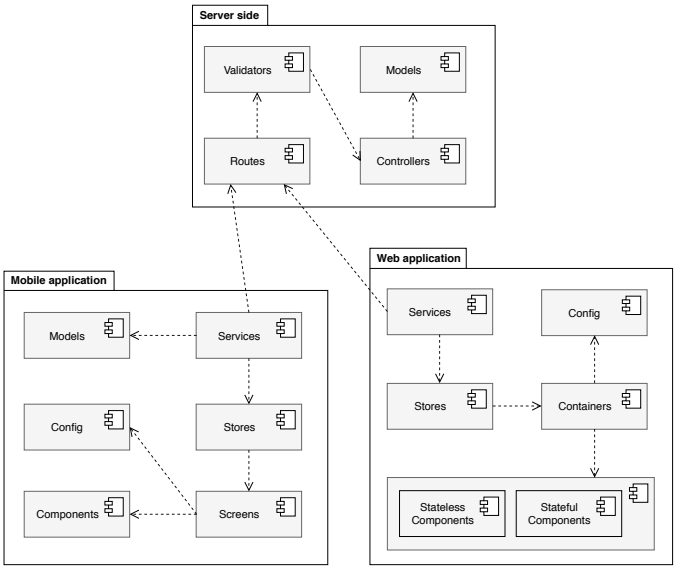

Fig. 1. The components of the system and their relationships.

progress information such as the completion time. Besides that, it provides an opportunity to share the results through other applications.

The published challenges can be browsed with the application. The challenges appears in a searchable list, with a few details like name, image and rating. The user can view the full details of a challenge, join or leave it and also mark it as favorite.

Some global settings of the application are customizable by the user. The address of the server can be changed, which besides testing benefits, ensures the correct operation of the application even after a server migration. The level of notifications can be customized as well, so the user can adjust what kind of notifications they would like to receive.

There is a way to propose challenges towards to the content managers. Users can suggest a base idea with a short description or even concrete steps and step descriptions. These submitted suggestions will appear in a list grouped by their progress. In initial states, they are still editable and deletable by the suggester.

Every user can view their own profile, the list of the completed challenges and the global leaderboard. The leaderboard contains the top 100 users ranked by their score, also injecting the same information about the current user, providing a comparative look.

After usage, the user can also log out from the application. Upon decision, they may also suspend or even delete their own account, prompting a full deletion of their data from the server/database.

B. Functionalities of the content manager

A user with content manager privileges inherits the rights of the simple authenticated user, and in addition they may also log in into the web application.

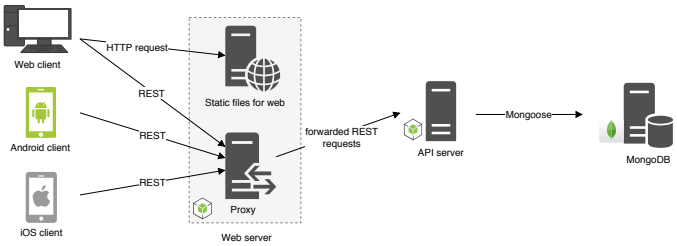

Fig. 2. Communication diagram, which presents that data flow between the different clients and components.

Here, the suggested and the published challenges can be listed, and various management operations can be carried out related to them. The content manager can view the published challenges and modify certain parts of them. The suggestions appear in another list, where the content managers can edit and supplement them. If a challenge is ready and reviewed, they can publish it, making it available globally.

C. Functionalities of the administrator

The administrators once again extend the permission base of content managers; in addition they may perform such critical operations that affects other users too. These include listing users, granting/revoking certain rights/privileges from users, etc.

III. Architecture

The system consists of three main components (see Fig. 1). The mobile application and the web interface are both served by the the central server. Each main package follows the principles of multilayer architecture [4], creating a modular and maintainable system.

The clients and components communicate with each other as presented in Fig. 2. Each client sends all types of requests to the *Web server* from which *REST* requests are forwarded to the *API server*. Static web resource requests are served by the Web server directly. To accomplish this, a *Proxy* is used, which can decide based on the pattern of the incoming URLs, whether or not it needs to be forwarded. The API server communicates with the database in the interest of serving the RESTful request.

A. The server component

In the case of the server, the requests are sent to the *routes* layer, which decodes and forwards them to the *validator* layer, where certain validations related to requests are executed. These may include authorization checks, request parameter validation etc. If the request is appropriate and feasible, the *controllers* component will perform the necessary business logic, appealing when necessary to the data schemas defined in the *models* layer.

The data is stored in a document-based *NoSql* database. The *models* component of the server side defines the schemas that entities must follow. There are three separate collections of *Challenges*, *Users* and *Leaderboard* as well as two additional

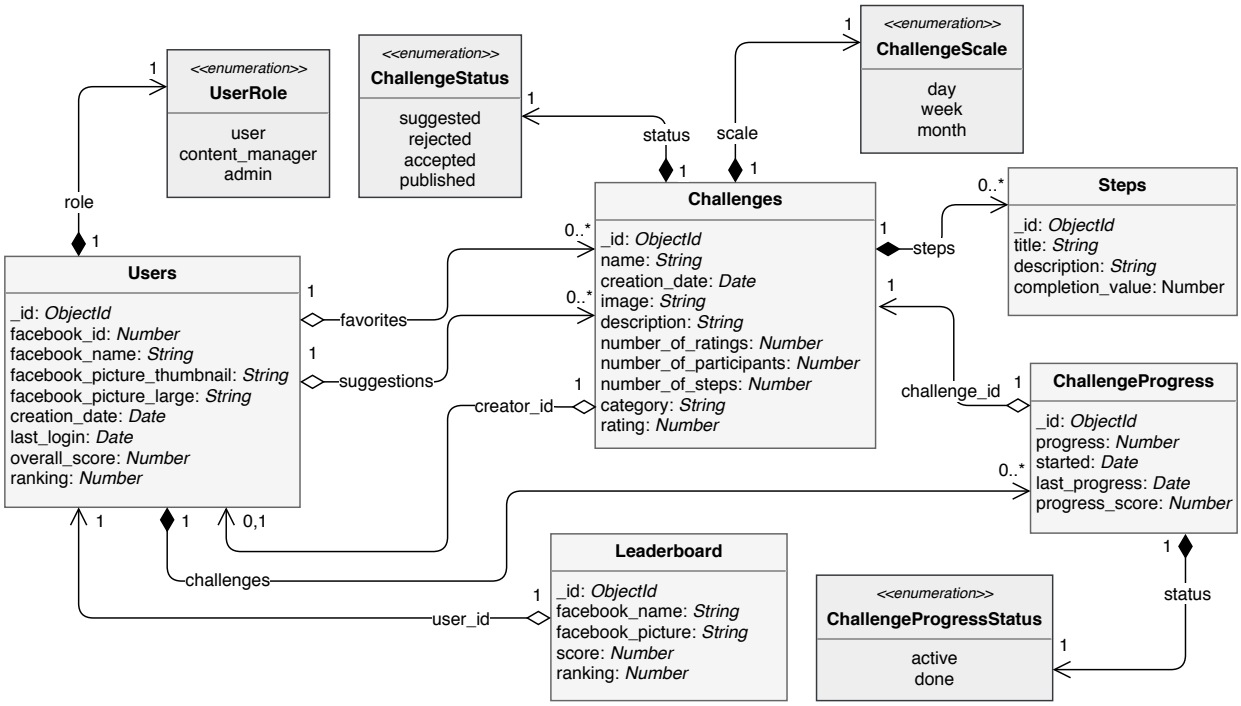

Fig. 3. The schema of the collections and nested collections in the system, and the used enum types.

schemas, *Steps* and *ChallengeProgress* embedded in other fields of the collection (see Fig. 3).

In the *Users* collection, the users who have logged in at least once with their Facebook account are stored. The data from Facebook, such as ID, name and image, are also cached for easier access. The *Challenges* collection includes the challenges accessible by everyone, and the challenge suggestions with certain statuses (*suggested*, *rejected*, *accepted* or *published*). Only the challenges with published status are available to all users. In addition, there are several things stored in the challenges: their name, description, creation time, image, number of evaluators, participants and steps, category, cumulative evaluation, identifier, steps and *scale*, which can be days, weeks, and months. The steps include an embedded block, which contains the data defined according to the *Steps* schema. Each step has a title, description, and an amount of points that users can receive for completing it. The *Leaderboard* collection caches the most important data of the users who have the highest score. The content of this collection is automatically updated at certain intervals, when the system recalculates performance, taking into account the total number of users.

The API server communicates with the MongoDB Satheesh, D'mello, and Krol [7] database using the Mongoose ODM [5] (Object Document Mapping) framework. Mongoose provides the opportunity to perform operations on the server with objects representative of the expected data format. Mongoose models also help manage and monitor challenges and user data. To keep the request and response message sizes within a reasonable limit and to avoid unnecessary memory usage, the *paginate* feature of Mongoose is utilized.

The login process consists of several steps performed in the background. As a first step, a request is sent to the Facebook server from the client, to request a valid access token for a specific time period. Using the acquired key, the client application retrieves user-related data (ID, name, image). Afterwards, the client sends a request to the API server with this data and the access token. Upon receipt of the request, the server will validate the Facebook ID and the access token using the appropriate authentication API endpoint. If everything is correct, the user gains entry into the application and all its relevant API endpoints. As a final step, the server sends back the ID, allowing the user to access their own data.

In case of a valid session, the server verifies whether the user has the right to perform a requested action based on their role. This is done by a proprietary role based access control (RBAC) system. For each role, associated actions its users may perform are mapped. Based on this, the RBAC system builds up a *roleMap* and performs the necessary authorization at every operation.

The API server created under the Daily Challenge project respects RESTful standards when serving any potential client. A primary access endpoint for the API is */challenges*, which addresses the challenges. The provided methods include reading type methods, such as listing challenges, searching, filtering. It also allows suggestion submission and updates.

To manage any single challenge, the */ challenges/:challengeID* endpoint format is provided. Its sub-endpoint */challenges/:challengeID/steps* allows clients to access different steps for each challenge. Just like challenges,

the */:stepID* can be used to specify the step to accomplish.

Managing users is accomplished through the */users* endpoint and its extensions. Both logging in and out are facilitated through here. By adding the appropriate userID, clients can access detailed data of other users using */users/:userID*. Challenge management assigned to certain users is also available through the */user/:userID/challenges/:challengeID* endpoint.

B. The mobile client

React Native [2] allows you to create component-based views. One component represent an elementary part of the user interface, such as a button or an input field. These can be customized, recycled and embedded. React also ensures the smooth operation of the interface, recognize which elements should be re-drawn according to data changes and updates only those parts.

In the application everything is made of React components and extensions of these. To enhance the user experience and facilitate our work we decided to use a UI toolkit, called Native Base. Thereby besides the basic React Native components (View, Text, etc.) we got many new useful element. Customizing their appearance, position, and size is done by using style sheets (StyleSheet), which are similar to CSS.

A package, called *React Navigation* is responsible for the navigation between the screens in the application. It provides several ways to implement the changing between views, using stacked, tab or switch navigators.

The presentation layer includes three components in the mobile application: *screens*, *components* and *stores* (Fig. 1). The first two are responsible for the *view* and the third are the *model* and the *controller* at the same time.

The *screens* contains the complex views that the user see in the application. These views shows the data provided by the models and interacts with the user through buttons or other elements. The *components* contain complex stateless elements which are used multiple times on the screens.

The separation of the presentation layer from the model and controller is established by using the MobX [6] state management system. With this we could associate each view with an element called *Store*. These deal with the data displayed on the screen, react to changes by updating the views, manipulate data and communicate with the central server. The store and screen elements use annotations to achieving this.

The business logic layer includes two component: the *model* where the data models are defined, and the *services*, in which the server requests are defined. The presentation layer is separated from the business logic, the stores do not contains direct requests to the server, only instances of the necessary classes from the service component.

If the response from the server is all right, then data integrity checks will be performed based on the schemas in the model. If the data structure is OK, then the information from the body will be returned in JSON format. If the response to the request is not correct, then a proper Error object will be throwed, what will be catched and handled by the upper layers.

The application communicates with the server through REST requests. For the communication we used the Fetch API provided by Mozilla, which allows sending asynchronous network requests and receiving answers for them. The answer for the request is a Promise, which makes it easy to handle and process these responses.

C. The web client

When the user first opens the web application, the browser loads the JavaScript bundle built by Webpack allowing the page to be displayed. The architecture of the web page is almost the same as the mobile application's architecture presented before. Hereinafter, only the most important differences are highlighted.

All the necessary code, such as JavaScript, HTML or CSS, are uploaded to the user's browser, when the page first appears, and the data is dynamically loaded only when needed.

Using the Bootstrap features, the display of our web application is optimized for each screen type. When viewed on a large display, an on-screen side menu is located on the left side of the webpage. In addition, a full-length header ensures that the user can always see on which main page is he watching. With a reduced screen size, the overview of the challenges and users changes, showing fewer columns depending on the size. If the size is lowered below the mobile look, the side menu turns into a collapsible menu on the left.

The interface accessible for the users consists of *stateless* and *stateful* components. The *stateless* components do not contain dynamical data and do not have status. The *stateful* components, on the other hand, are connected to a *MobX* state manager class, so they are directly related to the business logic. Through these components, the website responds to internal changes and user activity.

The web application can be divided into two parts, symbolized by two containers. The first contains the *Home* page, which is a public static page. The other one is the *Admin* page, where a user can enter only with appropriate rights.

IV. Technologies

The project is developed using appropriate current technologies. In this chapter, the main technologies and tools were used during development are presented.

For the back-end server, Node.js [1] is used. The main reason for the decision is JavaScript acting as a shared language between front-end and back-end. Express.js [1] handles the HTTP and API requests. Because of the varied structure of the data, the persistence is realized with the MongoDB [7] NoSQL database. The Mongoose ODM [5] provides a flexible but powerful bridge between the server and the database.

The user interface is built using React [3] for the web, and React Native [2] for the mobile client. The choice for the React family lies in the reusable JSX and HTML components. React Native compiles into native code and therefore provides native look and feel on different mobile operating systems. Webpack is used as a static bundle maker, for assembling all scripts, styles, assets and dependencies for the web application

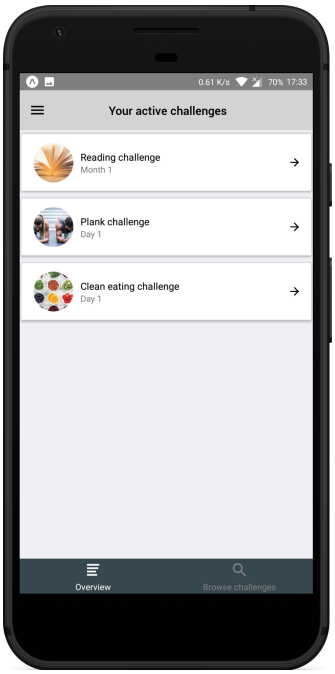

Fig. 4. View of the active challenges

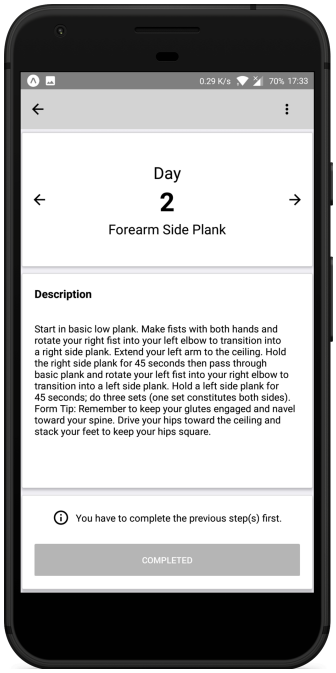

Fig. 5. View of the active challenge's steps and their descriptions.

into one static distributable asset. MobX [6] is used as a state management system on both clients, chosen because of its simplicity and efficiency. As a user interface toolkit, Bootstrap is used for the web client and Native Base in used for the mobile application.

Several auxiliary tools also played significant roles in the development process. For example, Expo is a toolchain built for React Native, which helps to build a native iOS or Android project and provides a rich SDK with useful tools. GitLab is utilized for version control, project management and continuous integration. Automatic deployment to a staging server are also integrated into a full continuous deployment pipeline. Quality assurance is achieved using ESLint for static code analysis, as well as Mocha, Chai and Jest for testing.

V. The client applications

This chapter presents the usage of the mobile application and the web interface, as well as the use of more relevant features, with a brief description and screenshots from different situations.

A. Mobile application

After opening the application the user is greeted by the login screen. There a series of images provide insights into some of the functionality of the application. Below is a button to sign in with Facebook. After a successful login, a view of active challenges is displayed. At first login, the user does not have any challenges, so the application recommends joining one.

There are two tabs at the bottom of the screen, the first one covers the overview of the user's active challenges (Fig. 4), the other leads to the list of available challenges. The second tab shows the public challenges available on the server. The user can search for specific challenges by typing in the search bar at the top of the screen. After entering some characters, search is started and only those challenges that have the requested text in the title or description are displayed. By clicking on any item the user can view the details of the selected challenge on a new screen (Fig. 6). There's a more detailed description to read, the steps can be seen with other useful information about the challenge.

If the user has joined at least one challenge, they will be displayed in the above-mentioned active challenges view. Clicking on any of the items will provide a detailed description and instructions about the step in progress (Fig. 5). The information on the *Completed* button shows the selected step's status. If he is looking at the right step, he'll also get information about the possible time of the completion. If the user completes the last step of the challenge, he'll see a congratulatory view with data about his activity. There is also the ability to share performance through social networks or other services.

Within the application, the sidebar can be used for navigating between the different major units. The views and functionalities described in the paragraphs above can be accessed through the *Home* menu item. There are also three additional menu items and a *Logout* button that can be used to log out of the application. The *Settings* item collects available application settings in a list. The following menu item is named *Suggestions*, here the user can submit a new challenge idea for the content managers. Opening the menu, the user can see his ideas, that have been suggested before. Ideas are grouped according to their status, which can be one of four types: *Suggested*, *Accepted*, *Rejected* or *Published*.

The user can click on the ideas in the list to view the details and see the options for editing or deleting them. In addition to this, challenges appearing in the published state will also have a button, which will immediately lead to the details of the published challenge, where the user can join them immediately.

In the sidebar, the last menu is named *Profile* and it navigates to the user's profile where the image, name, and total score of the user can be seen (Fig. 7). Below we can find three buttons which navigate to the *Leaderboard* view, access completed challenges and show achievements added in the future.

B. Web application

The main features of our web application can only be accessed by users with roles of Content Manager or Administrator, but anyone can view the main page where a brief description of the app, the contact of a developer team, and the mobile application's download options are displayed. From here, the user can log in and proceed with Facebook. If the user has a Content Manager or Administrator permission, the Control Panel will be displayed after login, if not he will be returned to the Home page.

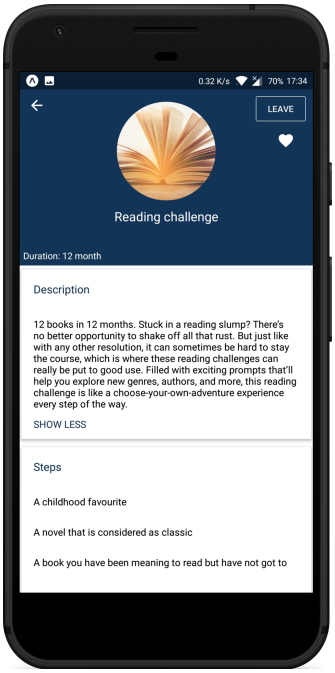

Fig. 6. The details of a selected challenge

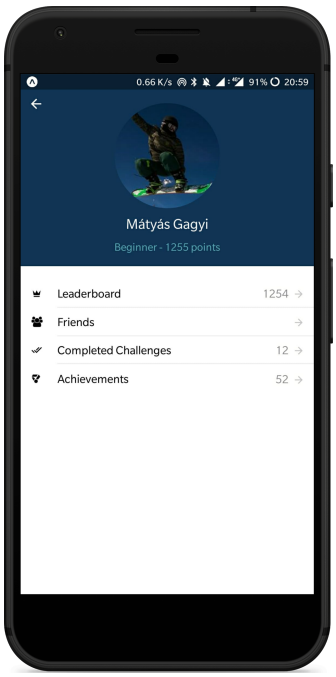

Fig. 7. Te profile of the user

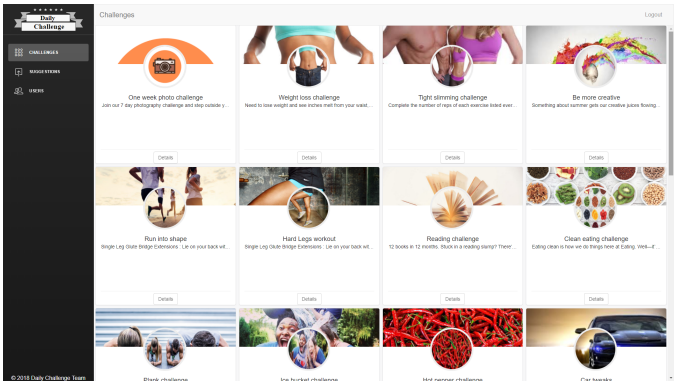

Fig. 8. Overview of the available challenges

On the Control Panel, users have the option to view and manage the challenges (Fig. 8). Selecting the *Suggestions* tab in the side menu allows to review submitted suggestions. By clicking on a proposal, its data is displayed on an editorial interface where changes can be made to the recommendation, or even be published as a full-featured challenge.

Users with *Administrator* role have a *Users* menu in the sidebar. Clicking on this menu item will display a user management page where users can be searched, listed and their roles can be managed.

VI. Conclusion & Future Work

This paper has presented the architecture of the successfully implemented Daily Challenge mobile system, the related Web interface, as well as the API server. Furthermore, details have been provided related to the functionalities accessible for the different roles of users, and also the technologies used. In accordance with predefined goals, the application provides the user multiple challenges on a single interface, and the web-based application ensures the tools needed for those in the background to maintain and supervise the content of the application.

The main priorities of the further development of the application and the web interface include the connection and communication between users. Providing users with the ability to comment on challenges is a planned feature, evaluating them for each step and when finished, being able to write reviews and provide ratings. The application could also use the friend list of Facebook, so everyone will be able to follow the performance and interests of their friends, even inviting them to participate. The users could be able to chat, whether directly or in the context of a challenge, to promote inter-user communication. An improved ranking list is also planned, which should encourage users to participate in many challenges and strengthen the gamification aspect of the application. With available achievements, challenges should become more fulfilling, rewarding for the persistent users.

Also included in further development plans is a new role to moderate the behavior of the users by way of automated filters. Additionally, administrators should also be allowed to suspend users or revoke various functionalities from them in case of negative behavior.

The creation and recommendation of challenges could be improved with a pre-defined package of usable step types. The user could assemble challenges through steps utilizing the capabilities of the smartphone sensors, such as location, gyroscope, camera, and microphone.

From feedback in early usage, it has also been concluded that an offline mode could help users progress with their challenges even without a working Internet connection, being able to synchronize later when the connection is restored.

References

[1] Mike Cantelon et al. *Node. js in Action*. Manning Publications, 2017.

[2] Bonnie Eisenman. *Learning React Native: Building Native Mobile Apps with JavaScript*. " O'Reilly Media, Inc.", 2015.

[3] Artemij Fedosejev. *React. js Essentials*. Packt Publishing Ltd, 2015.

[4] Martin Fowler. *Patterns of enterprise application architecture*. Addison-Wesley Longman Publishing Co., Inc., 2002.

[5] Simon Holmes. *Mongoose for Application Development*. Packt Publishing Ltd, 2013.

[6] Luca Mezzalira. "MobX: Simple State Management". In: *Front-End Reactive Architectures*. Springer, 2018, pp. 129–158.

[7] Mithun Satheesh, Bruno Joseph D'mello, and Jason Krol. *Web Development with MongoDB and NodeJS*. Packt Publishing Ltd, 2015.