Real-time Software System for Latin Dance Festival Organizers and Participants

Nikolette-Beatrice Domokos*, Szimma Hunor*, Bertalan Vad§, Viven Bartha§, and Csaba Sulyok*
* Faculty of Mathematics and Computer Science, Babeș-Bolyai University
RO-400084 Cluj-Napoca, Romania
§ Codespring
RO-400347 Cluj-Napoca, Romania
niki.domokos@yahoo.com; szimmahunor@gmail.com; vad.bertalan@codespring.ro; bartha.vivien@codespring.ro; csaba.sulyok@ubbcluj.ro

***Abstract*—The "Wanna Dance?" project's goal is to provide an easier way to organize and manage dance festivals, while participants may keep up with its happenings. As for its features, organizers can introduce and maintain festival information, users can view this data, track the events of the festival and to maintain a connection with the other participants via open chat messages and dance requests. The system is made up of a backend server following the conventions of a microservices architecture, and a cross-platform mobile application, available both for iOS and Android devices.**

***Index Terms*—dance, festival, real-time web, microservices, timetable**

I. Introduction

When dancing comes to mind, most people think of professional dancers, sophisticated movements and well-rehearsed choreographies. Many people do not even dare to dance at an event, because they think it is embarrassing if they do not do it professionally. However, dancing is not primarily beneficial because of competition, but has many other positive effects. Dance is, not coincidentally, also called moving meditation [1]. Dancing lifts the spirit, frees the body and harmonizes the soul, a perfect way if you want to relax and have deep experiences. There are plenty of places to dance, whether at local, smaller gatherings, weddings, nightclubs or even at events and festivals revolving around dance. Caribbean-style dances such as salsa, bachata, merengue, kizomba, zouk and many different subspecies of these are becoming more and more popular at dance festivals [2].

Such Latin dances are characterized by lightness, playfulness and easy learning of the basics, so they are less stricter and tighter than traditional folk dances [3]. They do not require a pre-learned choreography in order for one to be able to dance with anyone, as the figures are invented and performed in an improvised way, meaning that the participants can freely change their partners from song to song [5]. Therefore the Latin dance congresses are characterized by the fact that you can not only have fun at the nightly social parties but also improve your dance skills in the various workshops held during the day.

Dance festivals of this kind are becoming more and more popular due to their above-mentioned characteristics [6]. Nowadays, they attract thousands of people [4] who want to be part of the experiences. In order to make it easier for the dancers, to keep up with the festival and to be informed about important information, and for the organizers to be able to set them up efficiently, a full-scale software system would be useful.

The *Wanna Dance?* project provides a solution for this issue. On the one hand, it provides a mobile application where organizers and staff can publish important information about the festival, such as main dance styles, invited artists and teachers, workshop and activities schedule and all kinds of logistical information. Furthermore they can effectively notify participants in real-time of any changes or delays that may arise in the schedule. On the other hand, the app additionally provides dancers a virtual real-time network as an extra means to socialize and establish new relationships with other participants. The app lets them ask each other for a dance, generating real-time requests that can be accepted or rejected. After acceptance, the parties have to find each other in the realm of the festival to dance with each other. After their dance they can confirm the dance they had in the system and they also can leave a public review about the other dancer. To further enhance their experience, they can use a global chat, browse the festival information and other participants or get notifications about delays in the schedule.

This paper presents the underlying software system, which has as the backbone a *microservice oriented architecture*, and as its presentational layer a *cross-platform mobile app*. The backend server plays an important role in achieving the performance needed for a real-time festival application and the use of microservices increases the scalability and fault tolerance of the application. The mobile app has a component and state-based architecture that communicates with the server through various protocols to offer a smooth user experience and real-time presence in the social network.

II. Roles and Functionalities

A. Dancer

The *dancer* role is the default user role, which is assigned to every newly registered account. As a *dancer*, the user will be granted reading access to the main information regarding

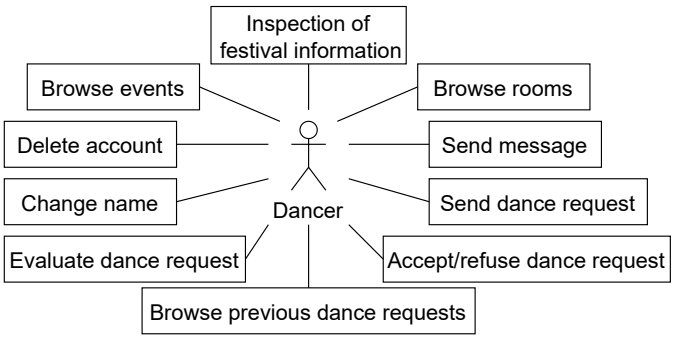

Fig. 1: Accessible functionalities with the *dancer* role

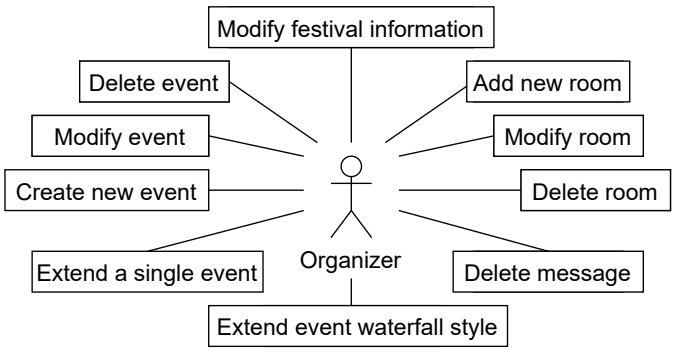

Fig. 2: Accessible functionalities with the *organizer* role

the festival, to track and filter the events taking place at the aforementioned festival, to send public messages and to address dance requests to other online users (see Fig. 1).

Information regarding the festival, such as its name, description, starting and ending date, representative dance styles and available rooms, are all available to view for the dancer. Additional data of rooms and their events are also accessible with ease.

For a better user experience, *dancers* can follow the various programs of a festival through a timetable. The user can filter the events by the day they are taking place at.

The application offers a public chat, which is accessible to every logged-in user. Through this chat, users can socialize with the other participants of the festival and be notified of sudden schedule changes. Additionally, private dance requests can be dispatched to other online users. In case both dancers deem a successful request as concluded, they get the opportunity to optionally leave an evaluation of their experiences. These reviews then become publicly available to other users, based on which they can decide whether they wish to dance with them or not.

As for now, these dance requests can be accessed only by the affiliated dancers.

B. Organizer

The *organizer* role possesses all of the functionalities that the *dancer* has, with further possibilities and responsibilities. Their main purpose is to manage the data stack of a festival. This not only takes place in the preparation stage of said festival, but during its active period as well. Currently, an organizer can only modify the basic information of a festival, such as its name, description, start and end date and representative dance styles (see Fig. 2).

Besides these, an organizer can modify the core building blocks of a festival. They can modify or delete already existing ones or even create new ones. Events and rooms are such building block entities. In the case of rooms, the user can assign primary and secondary dance styles and even a specific color, for the sake of differentiation. Upon the creation of a new event, the organizer can choose its location, name, duration, starting time and artist. A particular event can only be tied to one room and during the creation or modification of an event, in the context of a room, event overlaps are not permitted.

The organizer has two options at their disposal when it comes to the extension of an event. The first option only has a further effect on the subsequent event. This procedure is appropriate in cases where it is certain that the event following the one requiring more time, will be unavailable to utilize the integrity of the duration at its disposal.

The second option does not only affect the first upcoming event, but if needed every other event happening on that day. This provides a solution to more serious and unpredictable problems such as bad weather, power outages, or delayed arrival of an artist.

In both cases, the *organizer* and *dancer* users will be informed of these changes in real-time by the notifications sent out in the public chat mentioned before.

III. Architecture

This section discusses the architecture of the the server and the mobile application.

A. Communication

Communication between the server and the mobile application is based on three protocols: *HTTP* [7], *WebSocket* [8] and *STOMP* [9].

- User management and authentication, retrieving information about the festival and part of the process of handling dance requests are managed through *HTTP* calls. The client communicates with an API[1] [10] following the *REST*[2] conventions, through which the data is being transported in form of DTOs[3] [11].
- The real-time chat is implemented using the *WebSocket* communication protocol, which provides a permanent two-way, real-time data transfer between the server and the mobile application.
- Dance request notifications are sent out to users using *STOMP* over WebSocket, which is a message-based protocol.

[1]API - Application Programming Interface
[2]REST - Representational State Transfer
[3]DTO - Data Transfer Object

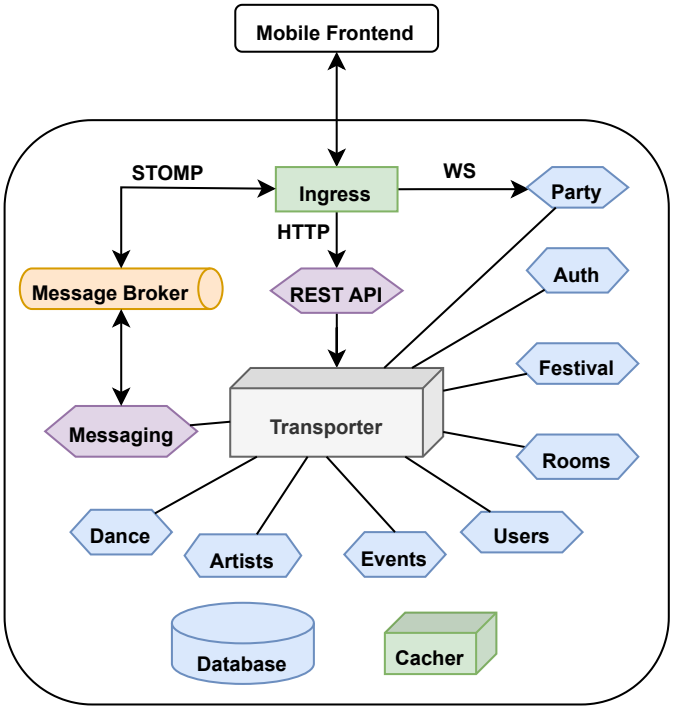

Fig. 3: Architecture of the server application

B. Server

The core of the *Wanna Dance?* software system is a server application based on a microservice architecture [12]. Designing such an architecture offers many advantages for a festival application, such as *scalability, resilience, improved user experience*.

In a microservice architecture, the integrated services are independent from one another. This allows them to be deployed and developed separately, making them easier to manage. If one of these services malfunctions or shuts down, the rest would remain unaffected and continue working without issues, eliminating the possibility of a software level shutdown. Furthermore, their segregation on such a level allows a better scalability of the software, as individual services can be horizontally scaled up, thus making this architecture more resource and cost efficient. Based on load tests, in case of the Festival service, five instances can satisfy up to a thousand of simultaneous users with minimal timeout occurrences.

In order for the application to support festivals, after a deep domain analysis, the following entities were defined: festival, room and event. Currently, only one festival can be organized at once, but the server-side implementation also supports the management of multiple festivals.

The *Moleculer.js* framework was used to build the server application. The framework provides a database-specific module through which the database can be accessed, and also provides basic CRUD operations, such as **C**reate, **R**ead, **U**pdate and **D**elete. This module is wrapped in a *Mixin* that follows the [13] JavaScript pattern. This pattern allows functionality to be reused between classes, multiple functionality sets can be picked up from different mixins. With regards to security concerns, the system performs business logic checks. A dancer entity is made up of a username, a name, a hashed password, a dancer role, a privilege level and an activity status.

To make the system truly fault-tolerant, these services run in a separate environments and the built-in *transporter* module is responsible for the interservice communication.

C. Mobile

The presentation layer of the software system is the mobile application, which communicates with the server through the previously presented communication protocols. The application is written in *React Native* [16]. Similarly to React, it assembles the application from building block-like components written in *JavaScript* [17].

Data related to the events and users are managed in global stores within the application. For this, the *unstated-next* library is used. Each larger entity has a corresponding state container, in which, in addition to data, data manipulation methods are also present. These containers are located at the top levels of the DOM[4] used by React Native, so that data can be easily accessed in the child components.

The aforementioned methods included in these entities' states mostly make API calls to the server. When using these methods, the system first attempts to perform modification operations on the server side, then the same corresponding actions will occur in the local state as well.

The festival's entire data stack is requested sequentially from the server. First, the festival identifier and then its general information are loaded in. Following that, with the festival's identifier its rooms and related events are also retrieved.

Certain graphical elements only become visible and functional to users with the *organizer* role, for data safety reasons. Most of these components have a navigational scope, as they direct the user to views through which data manipulating functions are accessible.

Another important global store to mention is the *i18n*[5] state container. For this purpose, the *React Intl* package is used, which loads in the texts which correspond to the specified locale and then formats them in the appropriate manner.

IV. Microservices

A. Applied patterns

The backbone of the system is built on a microservice oriented architecture (MSOA) and the following patterns [14] are applied in the system design:

- The *API Gateway* pattern assures that the system's REST Services have a joint entry point, through which they share the *JWT* [15] based authorization and authentication logic. Also, this *Gateway service* contains all the

[4]DOM - Document Object Model
[5]i18n - abbreviated form of internalization

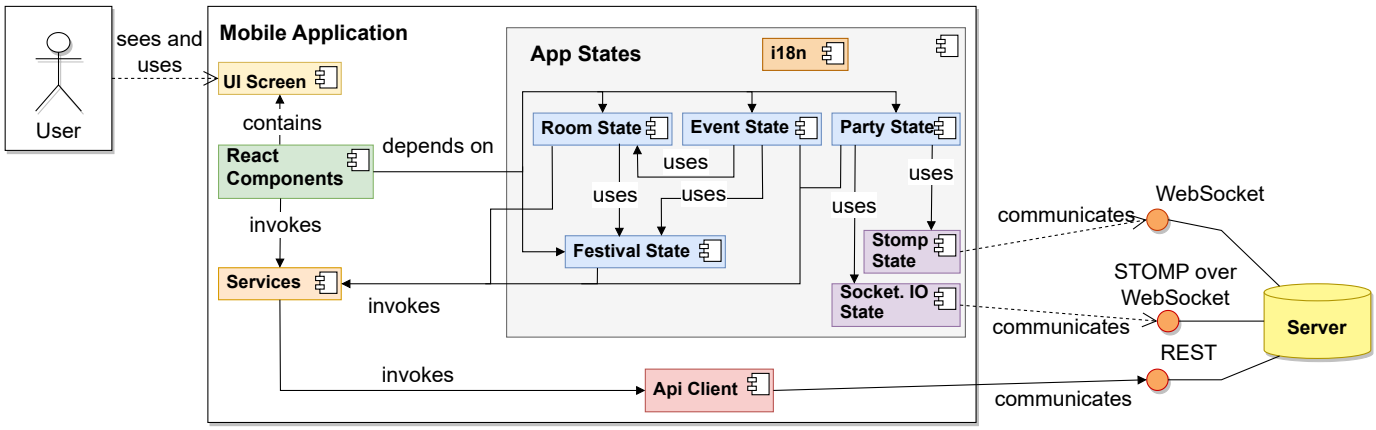

Fig. 4: Architecture of the mobile application

important web server settings, such as routing, CORS policies, pre- and post-processors, etc.

- According to the *Database per service* pattern [14], each service is responsible for maintaining the data of a single entity, which is important in creating well-isolated services. Relationships between entities are also implemented at the level of services, which also further serves the purpose of demarcation.
- The request-response based data exchange between services complies with the *Remote Process Invocation* pattern, further underlining the loosely coupled trait of the system.
- There are use cases that require fire and forget type of communication in the form of events, according to the *Asynchronous messaging*. It applies when the notifier service is not interested in the response of the notified services.

V. Real-time virtual party

In the system, the *Party service* is the one that ensures the real-time processing of the participants' activities both on and outside of the dance floor. It keeps count of the online participants, who are also seeing each other's availability and are able to chat and ask each other for a dance. The dance requests are processed by the *Dance service*, which also verifies a certain set of adherent business rules regarding of their status. These state changes can be best defined by a final state machine (see Fig. 5). The final states of a dance request are *rejected*, *revoked* and *confirmed*. The dance requests in the first two states have to be archived by a cron[6] job after a certain time, but the ones in the confirmed state are kept since the parties most likely will want to remember that dance.

VI. Tools and Technologies

The used framework, *Moleculer.js*, is a modern, fast, microservices-based framework for *Node.js*. It enables

[6]Cron Jobs allows to automate specific commands or scripts on a server to complete repetitive tasks automatically.

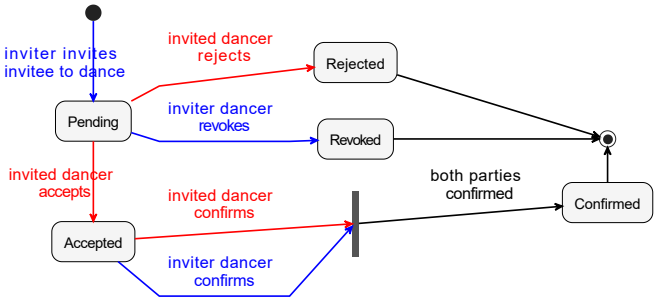

Fig. 5: A final state machine modeling the dance request status changes

developers to create efficient, scalable and fault-tolerant distributed systems. The framework supports multiple communication protocols and provides built-in features for load balancing, fault tolerance, and service registry[7]. Moleculer.js enables developers to build microservice-based applications easily and efficiently.

As for data management, each service has its own separate collection, emphasizing its segregation. For this *MongoDB* is used, which is a fast, document-based, NoSQL database, which is easily scalable.

For caching purposes *Redis*, a high-performance database based on key-value pair is integrated into the software, allowing for quick data access and updates.

The mobile application is based on a platform-independent framework called *React Native*. It offers components and APIs through which JavaScript code gets transformed into native platform-specific code, meaning the same code base can be used for the development for both iOS and Android devices.

Additional external libraries were also used, such as *React Navigation*, which is responsible for the navigation within the application, *React Native Elements*, which provides uniform components and other community libraries, through which

[7]allowing services to dynamically discover each other

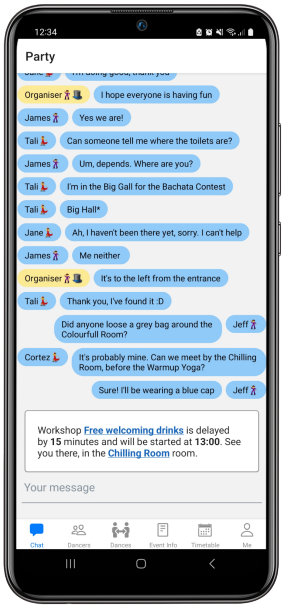

Fig. 6: Public chat

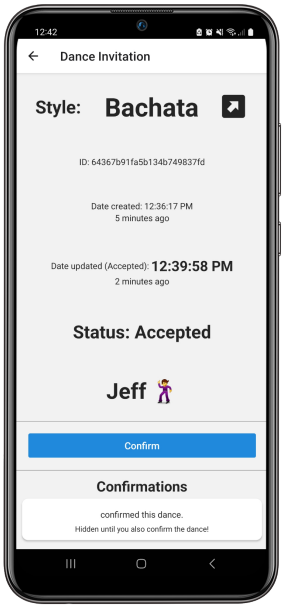

Fig. 7: Incoming dance request

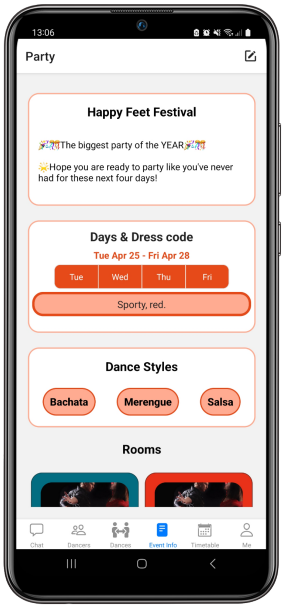

Fig. 8: *Festival Info* screen

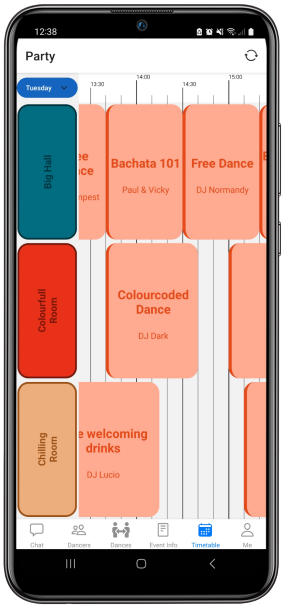

Fig. 9: Incoming dance request

general-purpose components can be accessed (date selectors, toast-type messages, etc.).

As it was stated previously in Section III-C, data management on the mobile end is realized through the usage of global states, created with the *unstated-next* library, which is based on *React Context*.

For establishing the necessary channels for aforementioned communication protocols the following dedicated libraries were used: *Axios*, for HTTP calls, *Socket.io* for real-time, two-way data transfer and communication and *Stomp.js* paired with *RabbitMQ* for dance request notifications.

VII. Usage of the Application

Upon the launch of the application, the user is greeted by a login page. When creating a new account, the user must select whether they are a leader or a follower in the context of couple dances.

After successfully logging in, the *Chat* screen (see Fig. 6) will appear, where the last fifty messages and notifications are visible. By long touching on a selected message, the user can choose to copy its content or in case of an *organizer*, can delete it.

Using the *navigation bar* visible at the bottom of the screen, users can navigate to the following screens: *Dancers*, *Dances*, *Event Info*, *Timetable*, *Me*.

The *Dancers* screen contains a list of the other, participating and logged in users. Through the elements of this list, users can easily navigate to another user's profile page, where they can browse through the reviews left by other users and address them a private dance request.

A dancer can view dance requests related to them on the *Dances* screen, where they are classified into three categories: Incoming, Outgoing, and Accepted. After accepting such a request, both dancers can then confirm the conclusion of their dance and optionally leave an evaluation based on their experiences (see Fig. 7). In case both parties decide to leave a review, those will become visible to every other user through their profile pages.

Information regarding the festival can be found on the *Festival Info* screen, where these are segregated into two groups: one grouping the festival's general information and the other depicting the available rooms of the festival (see Fig. 8). Tapping one of these room cards will navigate the user to the room's detailing screen.

As an *organizer*, on the two aforementioned screens, the *Festival Info* and the room detailing *Room Info* screens, a small pencil icon is visible on the top right corner, by which they can navigate to the corresponding modification screens, where they can alter existing information. Upon certain changes additional data may also be modified indirectly, to which the application draws the organizer's attention.

During the modifications on the entities' fields, continuous validation is taking place. Partial entity update will only take place if the new data is corresponding to the predetermined constraints.

The events belonging to the previously mentioned rooms can be viewed both on the room's detailing screen and on the *Timetable* screen (see Fig. 9). From a drop-down element, right above the column of room cards, users can choose which day's events to be displayed. On the upper right hand corner, a refresh icon can be found, which will reload the application's data stack, as described in Section III-C.

A vertical timetable style is also available for the users, which can be accessed by tapping one of the room cards on

the *Timetable* screen.

Similarly to room cards, touching an event card navigates the user to a detailing *Event Info* screen.

Akin to the previous screens, organizers have the option to navigate to an event modification screen. In the case of modifications that affect the duration of the event, the application will notify the user of possible overlaps that may occur. Creating an event itself works similarly. The screen required for this can be accessed through the floating button on the *Timetable* screen, which is only displayed in function of the logged-in user's role.

As an organizer, by long-pressing an event card, an overlay appears, on which two options show up. One is used to delete, while the other to extends the selected event. The extension is performed through a floating panel. The user defines the required additional duration using a slider, and then decides the type of extension that should take place via a checkbox.

The default extensions prolong the duration of the selected event and then moves the subsequent events, which could overlap another one, to a later start time in a waterfall fashion, until there are no more conflicts between the events assigned to the room. The second option following the extension of the selected event, will shift the start time of the following event if an overlap occurs between them. This time, the end date will remain unchanged, thus resulting in the reduction of its duration.

The system notifies users of these event extensions in the form of warning messages on the *Chat* screen (see Fig. 6). Such a message provides a detailed description of the affected event and the changes made to it.

Last but not least, on the *Me* screen, a user can view their own profile, which is similar to a dancer's profile, but contains additional options. Here the user can change their name, log out or, as a last resort, delete their account. Past dances can be accessed here as well. Tapping on the gear icon on the upper right hand corner, the *Settings* screen can be accessed, where the user can select the application's language and change the server's access URL. The latter is not available to real users.

VIII. Conclusions and Further Development

The main objectives of the *Wanna Dance?* project have been partially realized, as its social aid role and festival organization and monitoring functions are functional.

The system provides the organizer with the opportunity to provide general information about the festival, its rooms, and its events, along with the possibility of efficiently managing those. Furthermore, through the message board and private dance requests new social relationships are easier to initiate for the users.

Although the organization of only one festival is supported currently, the project aims to extend this to managing several festivals in parallel and to serve their participants. Due to the architecture of the server, the extension of the more frequently used services is easily manageable, thus providing steady connectivity.

In addition to these, a website is also planned for DJs, through which they can interact with their crowd. By connecting the two systems, dancers would be able to provide live feedback on the played music or even vote on what they would like to dance to next.

As already mentioned, dancers can be informed in real-time about the postponement of events through the message board, these are however only internal notifications. With the introduction of push notifications, it would be ensured that the users are notified of these changes in time.

Currently, only text-based descriptions are used to describe the festival, halls and events. These could be embellished with pictures, thereby providing a more direct insight for the dancers and even enhancing their mood. These pictures would be stored in a cloud. With this additional storage, the application's security would also be extended to accommodate the users' privacy.

To ensure that only those users can send dance requests and access the messaging boards that are physically present at the ongoing festival, various checks could be integrated into the application, such as verification through QR codes or location tracking.

References

[1] Christensen, J. F., & Chang, D. S. (2021). "Dancing is the Best Medicine: The Science of How Moving to a Beat is Good for Body, Brain, and Soul". Greystone Books Ltd.

[2] Video interview - Cotiso, founder of the Bailarte "empire": what are the next steps of salsa, during the pandemic

[3] Senecal, S., Nijdam, N. A., Aristidou, A., & Magnenat-Thalmann, N. (2020). "Salsa dance learning evaluation and motion analysis in gamified virtual reality environment". Multimedia Tools and Applications, 79, 24621-24643.

[4] Fonseca, M., Gonçalves, G., Leal, R., Biro, D., Silva, L., & Sá, R. (2007). "Dancing for sustainability: Steps towards the sustainable design of a dance festival".

[5] Franks, A. (2021). "Social dance: a short history". Routledge.

[6] Romano, G., Schneider, J., & Drachsler, H. (2019). "Dancing Salsa with Machines—Filling the Gap of Dancing Learning Solutions". Sensors, 19(17), 3661.

[7] Wong, Clinton. "HTTP Pocket Reference: Hypertext Transfer Protocol.United States: O'Reilly Media", 2009.

[8] Lombardi, A. (2015). "WebSocket: Lightweight Client-Server Communications". United States: O'Reilly Media.

[9] Mesnil, J. (2014). "Mobile and Web Messaging: Messaging Protocols for Web and Mobile Devices". United States: O'Reilly Media.

[10] Amundsen, M., Richardson, L., Ruby, S. (2013). "RESTful Web APIs: Services for a Changing World". United States: O'Reilly Media.

[11] Ezzio, D. (2003). "Using and Understanding Java Data Objects". Germany: Apress.

[12] McLarty, M., Amundsen, M., Nadareishvili, I., Mitra, R. (2016). "Microservice Architecture: Aligning Principles, Practices, and Culture". United States: O'Reilly Media.

[13] Osmani, A. (2012). "Learning JavaScript Design Patterns: A JavaScript and JQuery Developer's Guide". United States: O'Reilly Media.

[14] Richardson, C. (2018). "Microservices Patterns: With Examples in Java". United States: Manning.

[15] Rasyada, Naufal. "SHA-512 Algorithm on JSON Web Token for RESTful Web Service-Based Authentication." Journal of Applied Data Sciences 3.1 (2022): 33-43.

[16] Masiello, E., & Friedmann, J. (2017). Mastering React Native. Packt Publishing Ltd.

[17] Horváth, G., & Menyhárt, L. (2014). Teaching introductory programming with JavaScript in higher education. In Proceedings of the 9th International Conference on Applied Informatics (Vol. 1, pp. 339-350).