ISSN 1846-6168 (Print), ISSN 1848-5588 (Online)
https://doi.org/10.31803/tg-20191219144338

Original scientific paper

Implementation of a Location Services Based Android Application and Accompanying Server Backend

Kristijan Lukaček, Matija Mikac*, Miroslav Horvatić

Abstract: This paper is focused on the usage of location services in mobile applications that were developed for the purpose of reporting different events that are based on their location. The event that is intended to be generic and universal can, as in examples used in this paper, be the reporting of some occurrence to a city's communal affairs office. Such a generic event can include both multimedia and textual data, in addition to location information obtained using mobile device running the app. The software solution that is described in this paper consists of a mobile application that was developed for the Android operating system and a web application that includes a series of PHP scripts that run on a dedicated server. The web application consists of a backend scripts that facilitate the communication of a smart phone and the server and frontend related scripts used by users and administrators to access and check the data and process the reported events.

Keywords: Android; backend; event reporting; frontend; geomap; location services; mobile application; web application

1 INTRODUCTION

The main aim of the project described in this paper was to implement a mobile application that can fetch a device location and visualize it. Later, the possibility of sending that location and other data (pictures and text) was implemented in order to extend the functionalities and redefine the purpose of the application. Initially, the goal was to define and implement a generic system that could be applied to different real-world situations. Possible real-case scenario and the purpose of the developed application is to send reports of some event occurrence to a city's communal affaires office. To implement full functionality of the system, an additional server-side web application had to be developed, including both back- and front-end subsystems. That web app consists of a series of server PHP scripts facilitating the communication and processing of data being transferred between server and the smartphone. Additional set of scripts was developed, being responsible for administration of reported events. Server data storage used for the project is based on standard MySQL or compatible (MariaDB) database management systems.

The mobile application prototype was developed as a native application for the Android operating system, which was chosen because it is currently the most wide-spread mobile operating system in the world [1]. The mobile application was developed using the Java programming language – until recently Java was preferred language for native Android development, being used in our education classes too (in May 2019, Kotlin language was announced by Google as preferred language for Android developers [2]).

Alternatively, hybrid mobile application could serve the same purpose in proposed system. Most probably, that kind of implementation will be checked in future, but that is out of scope of this paper.

The paper is organized as follows – after this introductory section, developed system overview is given. After that, the third section is used to describe the basics and the most important parts of the mobile application development process, while the fourth section describes the server-side development included in the project. Fifth section describing the project related future work is followed by the final, conclusions section.

It is important to state, that the project described in this paper started as a simple practical extension of mobile application development student class [3], being extended and finally presented as a student final thesis [4]. Local authorities were informed about the project itself and gave a positive feedback with interest of practical implementation for real-world usage in communal affaires office.

2 SYSTEM CONCEPT OVERVIEW

The basic idea of the developed system is shown in Fig 1 – starting from a simple mobile application being able to collect and process user location information, it grew to a complete generic system that can be used, with minor on-demand adaptations, in different real-world scenarios.

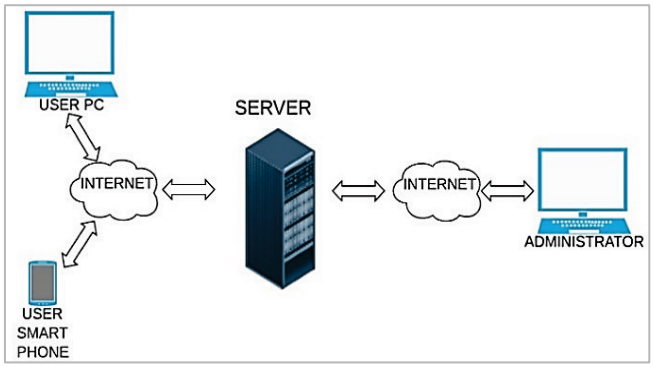

Figure 1 The developed system concept

In general, the system differentiates only two kind of users – standard users (mobile app users) and administrators. Standard users generate the systems inputs – they report certain events, including both the photo and the location information of the events. Administrators are responsible to

take actions on reported events, and, finally, provide a feedback to the users. The initial idea, implemented in the system, is that users use their smartphones to report and check the status of the events (mobile app prototype developed), while administrators use web-based interface to response.

In order to allow the anonymity of the users (of course, that could be limited in some real-case scenarios) the mobile application has the ability to be used without user authentication. Otherwise, users have to register to the system (feature included in mobile app) and authenticate prior sending the report (the feature of anonymous reports remain for the registered users, too).

2.1 Technologies Used

Overall, as already pointed out in the introduction, the system was developed using different technologies. Most, or all, of the used technologies are completely free, being available as open-source. The mobile application was developed as a native mobile app for the Android operating system, using Java programming language and the Android Studio integrated development environment (IDE) [5]. The web application (administration interface and registered user overview interface) and server back-end scripts were developed using plain PHP on the server-side, and HTML5, CSS (with additional Bootstrap styles) and Vanilla JavaScript on the client-side. Since no advanced frameworks were used, it is expected that the system will function on all HTTP servers – Apache HTTP server [6] was used for development and testing purposes. For data storage, MySQL database system, or compatible MariaDB was used. The implementation of PHP data management was accomplished using PDO (PHP Data Objects) making it almost database independent (all PDO supported database systems could be used with minor or no-change to source code). As a development environment for server-side scripts and web pages, NetBeans [7] was used. All the development was done using Windows operating system, while server packages were installed using integrated environments such as XAMPP [8] or WAMP [9].

3 MOBILE APPLICATION DEVELOPMENT

The mobile application was developed as native application, using Java and Android Studio IDE. Currently, a new programming language, Kotlin is stated as preferred language for the Android development, but the project described in this paper started much earlier and was therefore developed in Java. Due the interoperability of Java and Kotlin in Android environment, it could be expected that some parts of the app may be rewritten to Kotlin. In addition to Java, the XML markup language was used when defining graphical user interface (GUI) and other application resources.

In contrast to the native application developed, modern mobile applications can be developed as, so-called, hybrid applications. Native applications, as the one presented as the end-user-oriented part of the proposed system, are developed for specific operating system and are considered to be more efficient than hybrid apps – the downside of native apps is that they cannot be directly ported and used on other operating systems (current duality of the mobile platforms, as seen in results in [1] suggests iOS as only valid alternative to Android). Hybrid apps, on the other hand, are usually fully portable and can be used without any significant changes and additional development efforts on literally all platforms (supporting HTML5 and JavaScript). Since the functionalities of the developed app are not so device-specific, and since the hybrid approach includes basic support for most device features required for our application, the development of the, functionally the same, hybrid application will be considered in future.

As already suggested in 2.1, most of the implementations were done as plain as possible, without usage of any additional frameworks. The same approach was used in the mobile app development – no advanced techniques or architecture patterns were used – the simplest, old-fashion architectural approach was used, without any real view and model separation. Basically, `Activity` classes were used to control the GUI and call the functions required.

Since explaining and in-depth coverage of all implemented features and related source-code is out of the scope of this paper, this section will be used to point-out the most important parts of mobile app development process – in addition to basic GUI development introduction, it will include some details about using location services, networking and location visualization using integrated maps.

3.1 User Interface

Each new version of Android operating systems tends to include some new user interface "look-and-feel", new themes and even new GUI objects offering some new features. Even though that is a great fact for end-users, delivering faster, nicer and often more functional apps, in our project we tried to keep the focus on functionalities, while using only the simplest GUI elements. Of course, when preparing the prototype app for real-world usage, that can be adapted, based on customer requirements and needs.

Therefore, the basic GUI elements implemented as native classes in Android API were used in the prototype app – standard `Button`, `TextView`, `EditText`, `ImageView`, `Menu` and `WebView` objects. Basically, the GUI consists of a "screen" (window) facing the user – in Android, the class representing the screen (usually, full-screen of the device) is *Activity*. GUI elements are positioned in the `Activity` – in Android that is called `Layout`. Currently, the preferred layout in Android Studio is `ConstraintLayout`. Our prototype app uses `ConstraintLayout` to "draw" the user interface and define the relations among the objects (all the layouts and object data are stored as XML resource files). Describing the basics of "connecting" programming source code and GUI elements is out of the scope of the paper, but shortly – in order to use certain GUI element programmatically, one has to get a reference to it – it can be done when needed or once-per-activity in a predefined `onCreate` function, for example by using `findViewById` method. That way, a programmer can

obtain a variable (object) referencing to a certain GUI element and use it when needed (each element in the application can be referenced using its unique resource identifier). Interestingly, Kotlin language overcomes that by using so-called synthetic binding, allowing the programmer to access each GUI element directly using element identifiers, with no additional coding required (while waiting for the reviews of this paper, during 2020, synthetic binding become obsolete and Google suggests another view-binding approach for accessing UI elements – however, for basic purposes `findViewById` still remains a solid solution).

In addition to mentioned basic GUI elements, in order to provide a better user experience (UX) to the end-user, some little more advanced visual elements had to be included – but, we tried to include only those really necessary, keeping in mind to stay focused on standard Android API elements – actually, only two "complex" visual elements were used – `RecyclerView` and `MapView` (or `MapFragment`). The list of the visual elements used in the application is given in Tab. 1. In case of commercialization of the developed system, it can be expected that more advanced GUI elements shall be included to additionally improve UX and modernize application looks.

Table 1 Used GUI elements and feature important non-visual API classes

Android API class	Usage
`Button`	Standard button, touch/click sensitive
`TextView`	Showing simple unformatted text on screen
`Menu`	Android menu
`EditText`	Input field allowing user input via keyboard
`ImageView`	Showing image resource/file on the screen
`WebView`	HTML viewer – part of the screen
`RecyclerView`	Optimized multiple item scrollable element
`MapView` or `MapFragment`	Visualization of Google map and locations

The `RecyclerView` class is used to optimize a display of multiple items (it offers certain optimizations in contrast to, functionally similar, `ListView`). Items visualized in `RecyclerView` can be customized and described with a complex XML defining its layout. Recycler lists are more efficient than standard lists, because they do not load and redraw the entire list at once, but only the visible list items. To properly use lists (connect the visual part and the data being actually shown), Android uses special classes called adapters. As many other Android API based "tricks", the only way to properly use it is to check the official documentation and adapt to the usage needed.

`MapView` class and elements used to work with geographic maps will be described and shown in action in 3.3, so it will be skipped here. But, there is one non-visual class important when working with user interface – the class `Intent`. Intents are an abstract description of an operation to be performed – in the application they are used to open and show other activities (screens). Another very important usage of `Intent` in our application is when obtaining photo related to the reported event – in addition to text and location information, multiple images (files available on the smartphone or direct camera photo shoots) can be attached – activating camera and obtaining a fresh photo is done using `Intent`, which connects our app with system camera and returns a result. This may be considered the simplest way to use generic, device related, camera software without additional programming – for some more complex project, when camera has to be integrated in the main application, Camera API or CameraX *JetPack* extenstion could be used.

3.2 Retrieving the Location

Device location in Android is retrieved using so called `LocationProviders`. Application presented in this paper uses the GPS `LocationProvider`. To use a location provider in the app, special permissions are necessary. Permissions required for an application to run on the device are defined in so-called application manifest file (XML document). Permissions are necessary because the location of a device is potentially sensitive data and in order to be able to access position data the user has to grant the permission prior using the app. The permission (predefined constant) `ACCESS_FINE_LOCATION` is used to allow access to precise satellite location data (GPS, GLONASS or other device supported location systems), while `ACCESS_COARSE_LOCATION` allows access to less precise location data obtained from network location sources (WiFi or mobile network data, IP geolocation or similar). The application must have permission `ACCESS_FINE_LOCATION` allowed – this kind of permission is considered "dangerous level" permission and has to be additionally processed at run-time since Android 6.

Android architecture and Android API define different kind of management classes used to communicate with certain subsystems. That applies to location services too - to access the device location a class `LocationManager` is used. It controls and connects to location services and implements programmatic events related to location services. Another important class, `LocationListener`, listens for events that are connected to a specific `LocationManager` and then triggers the predefined functions like in the case of a change of the current location. The `onLocationChanged` function is then triggered, as shown in source code in Fig 2.

```
1     LocationManager lm = (LocationManager)
          this.getSystemService(Context.LOCATION_SERVICE);

2     LocationListener li = new LocationListener()
3     {
4       public void onLocationChanged(Location loc)
5       {
6          lat = Double.valueOf(loc.getLatitude());
7          lon = Double.valueOf(loc.getLongitude());
8          vis = Double.valueOf(loc.getAltitude());
9       }
10    }

11    lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 100, 0, li);
```

Figure 2 Implementation of *onLocationChanged* method

In the code given in Fig. 2, the application connects to available location service of the device (via `LocationManager`, line 1) and creates new `LocationListener` (line 2) with implementation of `onLocationChanged` (line 4) function that will execute when current location changes – in this simple example (lines 6-8) new location will be fetched and stored in (global) variables `lat`, `lon` and `vis`. The last line of code (11) connects a manager `lm` with previously created listener `li` - the parameters given are a provider (GPS), update frequency of the listener (every 100 ms in this case), and

minimum distance required to detect new position (related to previous).

The obtained geographic location is packed using the class `Location` [10] which implements many useful functions – as in lines 6-8, function for obtaining geographical longitude, latitude and altitude/elevations.

In time of writing this paper and developing the mobile app, *LocationManager* as part of Android Location API was considered a standard approach for obtaining device location. However, lately Google suggested using a little different approach – Google services with *FusedLocationProvider* class and related API as more battery-efficient approach [11].

3.3 Location Visualization on the Device

Locations are best visualized on a map. The maps used in the developed prototype application are Google Maps, the most common map type used in Android applications. Simply, `GoogleMap` class that is part of extended Android API (Google API for Android – Google Play Services additional libraries have to be installed and included in Android Studio project when developing that kind of applications) is used as an object for manipulating the geographic map, while `MapView` or `MapFragment` visual user-interface elements are visualizing that map and additional user elements as markers. There are multiple types of maps used – the application uses hybrid maps. These are satellite maps with elements of classic road maps. Maps are implemented as a view or as a fragment (part of the screen/activity) in the activity layout. Map data is contained on online servers, so, to successful load maps, a device must have access to the Internet (there are some ways of using offline maps, but these were not analyzed during the presented project and therefore not supported in the application).

In order to use maps, some prerequisites are necessary – the developer has to prepare the environment and the end-user has to allow the application to access the location services. Additionally, developer has to obtain so-called API key to allow his applications to access Google online services [12] and define it in application manifest file.

Maps can have added elements such as markers. Markers can be predefined system markers or custom made. These markers can be interactive. Also, an important issue regarding the map user interface is the "camera", which represents the view a user has on the map or where is it centered and to which level is it zoomed in. Since maps are network dependent elements and as such they are susceptible to delays. Therefore, it is not recommended to do operations with maps before they are completely fetched from the network. When a map is successfully loaded a predefined function called `onMapReady` is triggered [13].

The code piece in Fig. 3 gives an example of how to initialize and display a map (in a `MapFragment` visual element, which is simpler to implement, since it automatically handles lifecycle functions, while when using `MapView` the developer must handle and implement all lifecycle functions manually [14]) – usually, that is done in `onCreate` function of the activity containing the map – the example code gets the reference to a `MapFragment` with resource identification `map1` and calls the `getMapAsync` function.

```
MapFragment mapFragment = (MapFragment)
                  getFragmentManager().findFragmentById(R.id.map1);

mapFragment.getMapAsync(this);
```

Figure 3 Source showing how to get a reference to a *MapFragment*

This standard API function (`getMapAsync`) shall trigger the `onMapReady` callback function where developer should obtain reference `mMap` to `GoogleMap` object (line 3) related to the map shown in `MapFragment` or `MapView` and start manipulating it (in example shown in Fig. 4 – code line 4, defining the map type).

```
1    public void onMapReady(GoogleMap googleMap)
2    {
3      mMap = googleMap;
4      mMap.setMapType(GoogleMap.MAP_TYPE_HYBRID);
5    }
```

Figure 4 Implementation of *onMapReady* callback function

Using that reference, the variable `mMap` in our example, map can be manipulated – for example, adding markers, changing and zooming the map camera etc.

```
mMap.addMarker(new MarkerOptions()
            .position(moja_lokacija).title("TRENUTNA LOKACIJA"));

mMap.moveCamera(CameraUpdateFactory newLatLngZoom(moja_lokacija,15));
```

Figure 5 Example code for map manipulation – new marker, zoom change

The example in Fig. 5 shows how to add a standard marker to a certain location (`moja_lokacija` variable – `Location` object), and how to "center" the map to that location using `moveCamera` function – in addition to the location, `newLatLngZoom` function defines the zoom level, based on predefined constants (for this example, zoom set to 15 relates to internal street-based view, as documented in [15]).

The prototype application also includes the feature of using a so-called geocoder, which is used to retrieve an address based on location coordinates. That can be useful in real-case usage where the manual input of the address related to the reported event could be automated or at least speed up (user takes a photo for a report using the camera, the app gets the location and tries to obtain the address – which may succeed in populated areas and cities – if that succeeded, the user does not have to input address information manually). Example of usage of the geocoder trying to get an address of certain location (`try-catch` block is used to prevent possible exception in cases of geocode unavailability or other critical issues) is given in Fig. 6.

```
1   Geocoder gc;
2   gc = new Geocoder(getApplicationContext(),Locale.getDefault());
3   try
4   {
5     adrese = geocoder.getFromLocation(latitude, longitude, 1);
6     adresa = adrese.get(0).getAddressLine(0);
7   } catch (IOException e)
8   {
9     e.printStackTrace();
10  }
```

Figure 6 Source code for obtaining geocoder information

In given example, the function `getFromLocation` in line 5 returns an array of `Address` objects (to collection variable named `adrese`). In line 6, only the address line of first (most probably the most relevant one) of the obtained objects.

3.4 Network Operations

Since the proposed system requires that the report data is sent to a central location, it is clear that the prototype smartphone app must be able to send requests and get responses from the server. Android API provides common data communication features based on `HttpsURLConnection` class (and `HttpURLConnection` if using unsecure connections). There are some additional libraries included in API and *JetPack* framework, but for the prototype app only the basic class was used. As already explained in 2.1, server-side is based on HTTP (web) server, so that the only communication protocol required to contact and exchange data with the server is HTTP, meaning that `HttpsURLConnection` class should be sufficient for our implementation.

Theoretically, network operations may be time consuming and, as with all time-sensitive operations, it is required to execute those tasks in working (background) thread in order to prevent well-known "Application not responding" (ANR) exceptions due to the blockage of main UI thread! Network operations, in general, include sending HTTP request to the server and waiting for response. In Android applications written in Java, the "waiting part" is mostly done using the `AsyncTask` Android API class[16]. All the sensitive and time-consuming operations in the `AsyncTask` are done in the `doInBackground` function, while the user-interface related functionalities are implemented in other predefined class functions such as `onProgressUpdate`, `onPreExecute`, `onPostExecute` function. To access the main UI thread from the `doInBackground` function the `runOnUiThread` function can be used, representing an interrupt - interrupts are put in a waiting line and are executed from first to last.

When using HTTP, requests sent to the server are most often standard POST or GET requests. Since the proposed system itself includes quite simple functionalities (at least for now), all the server scripts were implemented as simple POST or GET requests processing scripts – no advanced REST API or similar approaches were used on server-side. Most calls to the prototype system are POST based – client-side requests in the mobile application are simply sent to the server and returning results are obtained using the function containing the code similar to one given in Fig. 7.

Unique resource locator (the addresss, URL) (line 1) and data transferred (line 8) are defined as the parameters `urlString` and `dataString`, while the results obtained from the server (line 15) are stored in variable `res` (in case of implementation as a function, function could return the string value of `res`). The response from the server is, as usual with HTTP, formatted as a string. Our server-side implementation uses a JavaScript Object Notation format (JSON) as response format. That kind of response results shall be processed after receiving it in variable `res`, and UI thread should update the GUI accordingly.

```
1   url = new URL(urlString);
2   veza = (HttpURLConnection)url.openConnection();
3   veza.setRequestMethod("POST");
4   veza.setDoOutput(true);
5   veza.setDoInput(true);
6   izS = veza.getOutputStream();
7   bfW = new BufferedWriter(new OutputStreamWriter(izS,"UTF-8"));
8   String post_data = URLEncoder.encode(dataString,"UTF-8");
9   bfW.write(post_data);
10  bfW.flush();
11  bfW.close();
12  izS.close();
13  ulS = veza.getInputStream();
14  bfR = new BufferedReader(new InputStreamReader(ulS,"UTF-8"));
15  res = bfR.readLine();  // Single line or loop to get multiline result
16  bfR.close();
17  ulS.close();
18  veza.disconnect();
```

Figure 7 Implementation of standard HTTP request and response operation

3.5 User Interface and Implemented Functionalities

The prototype mobile application uses the simplest possible, generic, user interface. It is completely customizable for different real-case usage scenarios. This section will depict only the most used activities (screens) and comment the functionalities implemented. Some activities (e.g. settings/options activity) are not shown due the limited paper length.

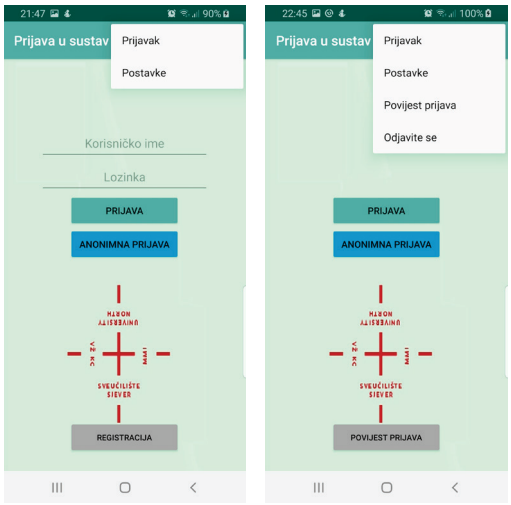

Figure 8 Adapted appearance of the main activity

The main app activity (with additional ability to include initial splash screen when customizing for real-world usage) is used for user authentication and starting a procedure of sending the report. The user interface is adapted to current app and user status – Fig. 8 depicts two varieties, the first on the left shows "logged out" screen, where the user can authenticate by entering username and password, register to the system, or directly send anonymous report without being authorized. The application menu is also adapted, based on current user and app status. The second appearance of main activity, on the right of Fig. 8 shows the activity when user is properly authenticated – standard and anonymous report is enabled, including the overview of reports history.

The most important activity, by its functionality, is the activity representing the report submission form, as shown in Fig. 9. It is a scrollable activity offering all required report features – selecting multiple existing photos from device

gallery or taking new photos with integrated system camera (implemented as `RecyclerView`), custom text input for describing the reported event, automatic location processing and geocoder address lookup. The address data will contain the street address, postal code, municipality and region of the location. In case when a geocoder cannot fetch the address, the location address data is left empty.

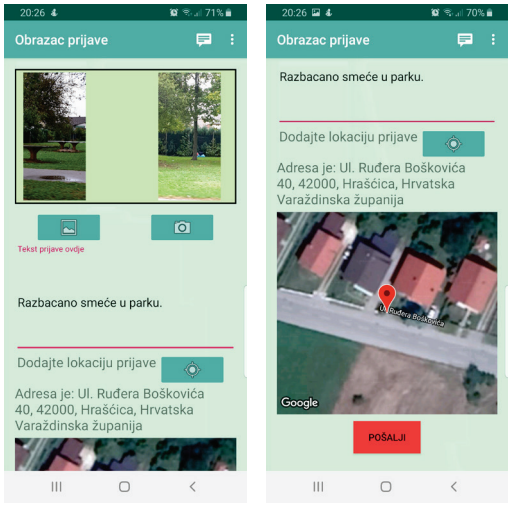

Figure 9 Activity with the report submission form

At the bottom of the report submission activity a map view which is moved to the reported location is included, with a location spot being marked using a standard marker. After manually (button) sending the report to the server, the end-user is informed about the progress of the reporting operation via `Toast` or `Snackbar` GUI messages and finally sent back to the main activity.

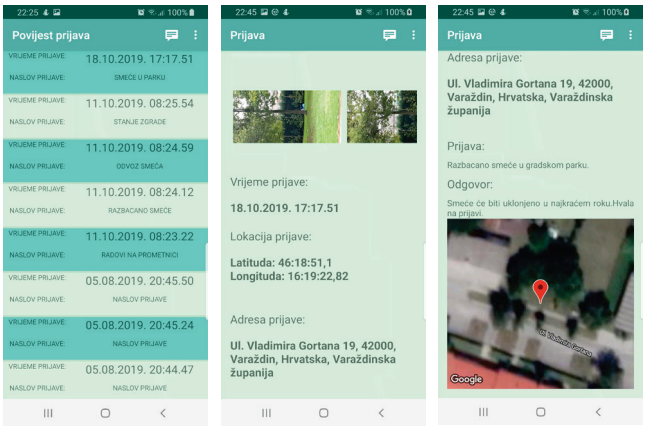

Figure 10 Reports history and overview activities

The prototype application does not include any special notification services that would periodically check the status of the reported events – instead, it offers simple reports overview (Fig. 10) allowing the end-user to check if the system administrator responded to the reported issues. Independent from the smartphone application, registered users sending authenticated report events could be informed of status changes via e-mail (server-side implementation).

The list of reported events is sorted by decreasing report date and time. Each report can be overviewed, showing the current data from server including administrator response.

4 SERVER-SIDE DEVELOPMENT

Two-way communication is expected in the proposed system – user reports shall be accepted and processed, if necessary, by the system administrator(s). Additionally, tracking, analysis and different kinds of back-office reports may be required. That is where the server-side development comes into the focus. Server-side subsystem requires a HTTP server and database server – the proposed concept uses server-side to upload reported events from mobile users and implements processing of the scripts using the PHP programming language. Also, it includes the web application for the administration (responding to reported events) and alternative authenticated user reports overview.

Plain PHP, with no usage of any frameworks was used for server-side scripts. Front-end (user interface for the web application) uses HTML5, CSS and plain JavaScript – additionally, some stylesheets and GUI widgets from Bootstrap [17] were used.

4.1 Mobile Application Accompanying Server Scripts

As explained in section 3, the mobile application uses the device (smartphone) to collect and prepare data related to a certain event and send a report and all prepared data to the server. Data handling and processing is managed on server-side using backend scripts described in this subsection. Additional server scripts were developed as part of web application and frontend (4.2).

Main functions implemented as PHP scripts, required by the mobile application are listed in Tab. 2. Please note that for the prototype and test purposes, only the plain .php files/scripts were used, with no advanced API or any server-side routing enabled – this can be setup easily when adapting the system for real-world usage, allowing more "fancy" and user-oriented API and addressing.

Table 2 Server scripts required by the mobile application prototype

Script name	Functionality
`login.php`	App user authentication check
`register.php`	New user registration
`update.php`	Multipurpose script for disabling or changing user account details
`report.php`	Script for receiving reported event details
`photos.php`	Script for managing and receiving images related to certain reported event
`fetch.php`	Multifunctional script for fetching list of reported events or details and complete data for a single event

The scripts respond with JSON formatted messages that are parsed and processed by the mobile app. To process user authentication the `login.php` script is used. It receives a POST request from the mobile device and returns a success or failure message depending on user status. The `register.php` script receives data from the mobile applications registration activity and proceeds to register a

new user. After a successful registration it returns a success message to the mobile device. The `update.php` script receives a request containing an operation type and some operation specific data, making it multipurpose – it implements few functions - the user deletion or deactivation and user data modifications. To receive a report on server, the `report.php` script is used. It receives the entry data excluding multimedia files (photos or pictures, mainly, as tested in prototype system). Each reported event receives new unique ID number which is sent back with script response. If pictures (most usual case) are related to the report and prepared on the device, the device will send the pictures to the `photos.php` script – it will store the pictures to a file on the server, not directly in the database. Generated picture ID number, picture filename and additional data associated with the picture are, of course, saved in the database. Since the pictures are sent as part of a HTTP request, *base64* encoding is used. Prior to storing the picture as a file on the server, decoding has to be performed, using simple PHP code, as depicted in Fig. 11.

```
1    $dekodPic = base64_decode($picStr);
2    $loc = 'photos/'.$name;
3    $file = fopen($loc, 'wb');
4    $res = fwrite($file, $dekodPic);
5    fclose($file);
```

Figure 11 PHP code for decoding and storing photo received from user device

To get the list and details of previously sent reports, the `fetch.php` script is used. It will send a structured list of JSON formatted data, based on the operation required – it can be used to obtain the list of all events reported by the authenticated user of the app, or to get all details of a certain reported event. Also, the same script is used to get report related images from the server.

4.2 Web Application and Frontend

The web application developed as part of prototype system is used primarily for administration purposes (the system administrator has to respond to end-user reports and give a feedback), but it was decided to extend it for a simple show-case, allowing authenticated users to check their reported events via a web-based interface (the same functionality is included with the mobile application). As already mentioned when describing backend scripts, on the server-side plain PHP with no frameworks or special routing API was used, while frontend uses HTML5 and plain JavaScript. The main web page is implemented in `index.php` and it is used to provide user login interface (Fig. 12). Authentication status is remembered using super global `$_SESSION` variables (additional improvements could be implemented prior to public release of prototype app).

If the login succeeds, it redirects the user to the `list.php` using simple PHP `header("Location:list.php")` directive.

The `list.php` script shows the list of all user entries. If the user has administration rights all the entries of all the users will be shown (Fig. 13).

Each entry in the list can be opened and all the details can be shown – location, description, uploaded images and all related data are visualized using `show.php` script, as depicted in Fig. 14. If an authenticated user is the administrator, response field will be available, with corresponding button to save the response to the database – the `admin.php` script is used to make an update.

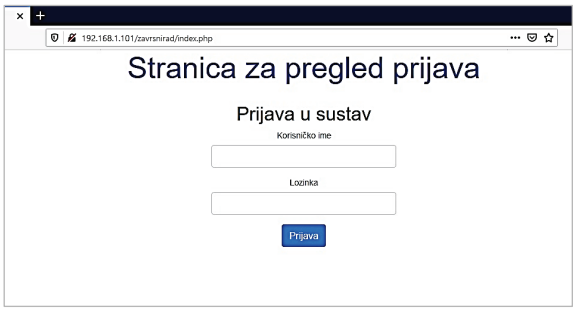

Figure 12 Login screen of the web application

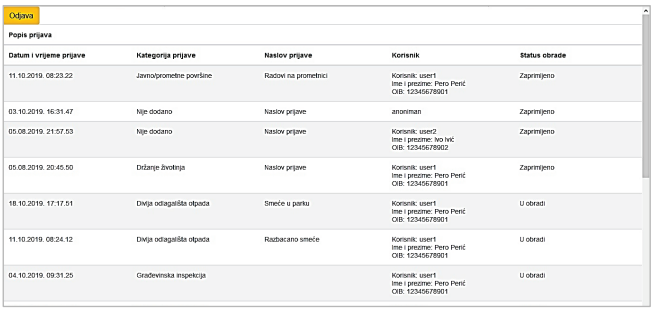

Figure 13 List of reported events

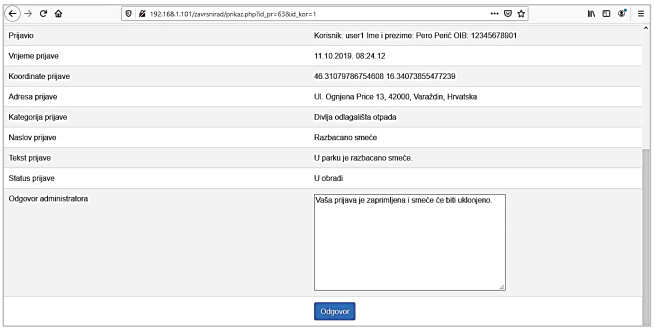

Figure 14 Reported event overview and administrator response

```
1    var xhttp = new XMLHttpRequest();
1    xhttp.onreadystatechange = function() {
2       if (this.readyState == 4 && this.status == 200) {
3          if(this.responseText=='testOk') {
4             alert('Data updated successfully!');
5          }
6       }
7    }
8    var addr =
9    xhttp.open('GET','http://server/admin.php?id_rpt='+id),true);
10   xhttp.send();
```

Figure 15 Vanilla JavaScript AJAX implementation

As usual, in order to make the web application more responsive, asynchronous JavaScript (AJAX) requests are used – standard `XMLHttpRequest` is implemented in plain JavaScript, as shown in Fig. 15.

The `logoff.php` script is activated when the user sends as request to logoff from the web application – it will basically destroy all the session data and reload `index.php` to allow the next user to login.

5 FUTURE WORK

From the beginning of the project, the system described in the paper grew and finally ended up with the development of both the mobile application and the server backend. The project started as a student project, non-commercial and educative – therefore, the deliverables actually consist of generic applications which could be extended with additional features. Through tests and analysis of possible real-case scenarios, the authors collected numerous ideas related to functional and user-oriented improvements of the implemented apps.

Even so, the mobile application already provides some enhancements over the functionally similar apps being used by some public institutions (e.g. multiple photos can be sent with single reported event).

Of course, additional tests have to be done, especially considering scalability of the system. The described initial steps and implementation can be considered a fundament of the future system or product.

6 CONCLUSION

Modern mobile platforms allowed the developers to use available location services and other platform subsystems in their applications. This paper describes the implementation of a system that uses location services in mobile application developed for the purpose of reporting different location related events to the central servers. The event that is intended to be generic and universal can be applied in different real-world scenarios.

The prototype system proposed in this paper was fully implemented, both as end-user mobile application and administrative web application. The complete system was developed from scratch, using only the basic and standardized technologies, most of them being free and open-source, making initial costs of the system stay low. Generic applications are quite adaptable and their practical usage can be considered quite wide. One real-case scenario that was already negotiated was a usage of the system to send reports to a city's communal affairs office.

The applications (both native mobile app for Android platform and web application) are fully functional, with ability to be expanded in the future, based on authors ideas or demands from potential customers and users. The design and user-interfaces of the applications are also completely adaptable, but, for now, that was out of the scope of the project.

7 REFERENCES

[1] https://gs.statcounter.com/os-market-share/mobile/worldwide
[2] https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/
[3] Mikac, M. (2020). Introductory mobile application development course for undergraduate students experiences, ICERI2020 *Proceedings*, 5003-5011. https://doi.org/10.21125/iceri.2020.1084
[4] Lukaček, K. (2019). Implementation of Android location services based application. (Bachelor Final Thesis) https://urn.nsk.hr/urn:nbn:hr:122:213590
[5] Android Studio IDE, see https://developer.android.com/studio
[6] Apache HTTP Server, see https://httpd.apache.org
[7] NetBeans IDE, see https://netbeans.org
[8] XAMPP environment, see https://www.apachefriends.org
[9] WAMP environment, see http://www.wampserver.com
[10] Android API Location class, https://developer.android.com/reference/android/ location/Location
[11] Fused Location Provider API, https://developers.google.com/location-context/fused-location-provider
[12] Google API developer key, https://developers.google.com/maps/documentation/javascript/get-api-key
[13] https://developers.google.com/maps/documentation/android-sdk/intro
[14] https://developers.google.com/android/reference/com/google/android/gms/maps/MapView
[15] https://developers.google.com/maps/documentation/android-sdk/views
[16] AsyncTask in Android, https://developers.google.com/j2objc/javadoc/android/reference/android/os/AsyncTask
[17] Bootstrap library, https://getbootstrap.com/

Authors' contacts:

Kristijan Lukaček, B.E.E.
University North,
Jurja Krizanica 31b, 42000 Varaždin, Croatia
krlukacek@unin.hr

Matija Mikac, M.Sc.E.E.
(Corresponding author)
University North,
Jurja Krizanica 31b, 42000 Varaždin, Croatia
matija.mikac@unin.hr

Miroslav Horvatić, M.E.E.
University North,
Jurja Krizanica 31b, 42000 Varaždin, Croatia
miroslav.horvatic@unin.hr