Geomedia: a framework to share comments and multimedia content based on location.

[*]Project for "Wireless Network for Mobile Application" course, Master Degree in Computer Science at University of Padua

Prof. Claudio Enrico Palazzi
Department of Mathematics
University of Padua
Padua, Italy
cpalazzi@math.unipd.it

Alberto Morini
Department of Mathematics
University of Padua
Padua, Italy
alberto.morini@studenti.unipd.it

***Abstract*—Nowadays Social Networks are the principal way to create connections among people. Platforms like Facebook, Twitter, and Instagram suggest content and profiles based not only on mutual connections or trending content but also on geographic location of the end-user.**
GeoMedia is a framework, developed into an Android mobile application, which provide a medium for users to share comments and multimedia content (such as photos, videos, and audio) based on their location. Then, users can easily see the content shared through GeoMedia moving across a geographical map.

***Index Terms*—Android, Social Networks, user location, microservices, client server, multimedia, media sharing, android location, http, three-tier, three-tier architecture, SQL, DBMS**

I. Introduction

In the last decade, mobile applications have almost replaced traditional computer programs, especially when comes to social networks. Not only because it is more convenient to use a mobile phone than a laptop, but also because mobile phones can give more functionality by using the physical sensors present in way to enrich the user experience.

The explosion of social media led media companies to invest and explore almost all functionality of mobile devices, thus to make users feel more connected and satisfied by providing more relevant content. As a result, users spend more time on these platforms, generating profit for social media companies. A perfect example is Tinder, a dating app where users can meet each other based on their geographical proximity. To reach this purpose, a variety of sensors can be used to profile the end user, often using the GPS to get the precise location, otherwise opting to the proximal location provided by Access Point / Cell antenna.

However, utilizing these features can lead to faster battery drain, higher data usage, and potential privacy concerns.
This project aims to develop a simple mobile application that allows users to share comments, photos, audio, videos, and other multimedia content on the location where they are; and then, to navigate through a map to discover others' post
The whole source code is public on GitHub reachable via this Link

II. Architecture

To ensure that the GeoMedia app functions as intended, there's the need to design an opportune architecture, even if not strictly related with the mobile application itself. In fact, the "three-tier" model adopted, will allow mobile apps to communicate to each other without a direct connection.

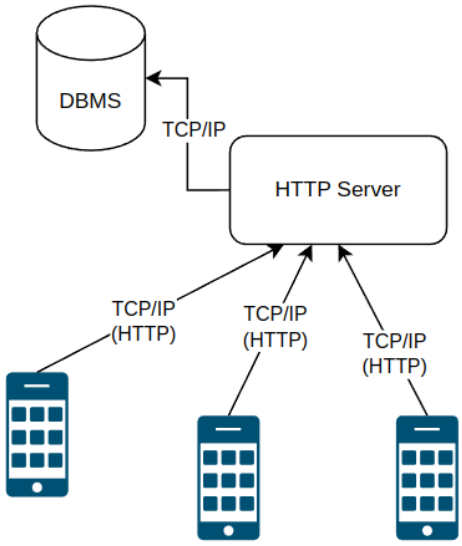

Fig. 1. Design Architecture.

A. Presentation layer

The presentation layer consists in the "GeoMedia" mobile app itself, representing the interaction point between the users and the whole architecture designed. By providing a user interface, allows users to view the published comments or to share their own content; which in the last case, the mobile app, as presentation layer, has also the duty to retrieve data

such as the user's location, username and other metadata even if not directly inserted by the user.

B. Application layer

In this project, the application layer is represented by an HTTP server, which exposes endpoints that the frontend (presentation layer) will call by making some HTTP request. Specifically, for each endpoint, server will manipulate data with opportune SQL instruction through the Data Layer.

C. Data layer

The Data Layer consists in a Database Management System (DBMS), capable of performing CRUD operations (Create, Read, Update, Delete) on data stored in defined tables.
For this project, the DBMS will also be used to create SQL stored procedures, offering a more flexible way to modify the select or insert of data queries in future extensions.

III. Mobile application

In way to create a mobile application there are several frameworks and technologies options. For this project has been selected the "Ionic Framework" [1], an open source toolkit for building cross-platform mobile, web, and desktop applications using web technologies such as HTML, CSS, and JavaScript. One of the key advantages of the Ionic Framework is its ability to integrate web frameworks like ReactJS, that offers many functionality such as the updates of data changed in the user interface, or some special functions called "hooks" triggered on particular events that make possible the customizations and the injection of personalized functions. For example, the React' hook "useEffect()" allows to call functions at the component startup or when a property change, giving to the developer the entry point to call some personalized functions; or instead the hook "useState()" allows an automatic refresh of webcomponent shown in the user interface at the change of the data inside it.

A. Functionality and behaviour

GeoMedia starts with a login component where users can choose to create an account or to sign in whenever they already have an account.
Then, the app splits into two pages: settings, where users can see their posts as form of list or to change server configuration parameters (i.e. IP Address or HTTP port); or the "Map" section that will offer a navigation through a geographical map to look for the content posted.

In the "Map" component users can also tap on a market to view the post, or tap on the floating button on the bottom right in way to create a comment.

The post is composed by some attributes:

- Title
- Comment
- Multimedia content (which is optional)
- And others metadata automatically retrieved (longitude, latitude, username, date, ...)

In GeoMedia app, has been used also the Pigeon Map [8] provided as a React Component, which renders a geographical map and allows to put some 'marker' on it.
Comments published on the platform, are retrieved by an HTTP request through the server, and each post will be rendered in the map as a marker.
There are three types of position indicators.

- Blue marker: the current user location
- Red marker: a post with just text
- Yellow marker: a post with a media attached

The content extraction is parameterized to get all posts, or to make the server compute the post closer to the physical position provided within a defined range. This strategy can be useful for scenarios like events or game-plays, where users need to be within a certain area to access exclusive data.

As follow a snippet of Map Component created

```
//various imports ...
import { useContext, useEffect, useState } from "react";
const MyPMap = () => { //declare the Map component

    const [PostList, setPostList] = useState([]) //to store the list of posts
    const ctx = useContext(mycontext)  //use the React context to retrieve data of other components

    // retrieve all post giving the actual position of user (in the server it will be computed the nearest posts)
    function getPosts() {
        if (ctx?.UserPosition != null) {
            doRequest("getPosts", {
                latitude: ctx?.UserPosition[0]
                , longitude: ctx?.UserPosition[1]
            }).then(res => {
                setPostList(res)
            })
        } else {
            doRequest("getPosts", {
                latitude: null
                , longitude: null
            }).then(res => {
                setPostList(res)
            })

        }
    }

    useEffect(() => {
        getPosts(); //to call the post
    }, []); //at the startup of component
    return( //render the graphics
        <>
        <Map defaultCenter={ctx?.UserPosition} defaultZoom={11}>
            <ZoomControl />
            <Marker width={50} anchor={ctx?.UserPosition} color={"#154c79"} />
            {
                PostList?.map(s => (
                    <Marker width={50} anchor={[s.LATITUDE, s.LONGITUDE]} color={(s?.MEDIATYPE?.length > 0) ? '#d6c531' : '#f23c3c'}
                            onClick={() => {
                                setPostSelected(s) //store the post selected to open it in the opportuned component (viewPost)
```

```
                            }}
                        />
                    ))
                }
            </Map>
            ...
            </>
    );
}
export default MyPMap;
```

Listing 1. Map component snippet

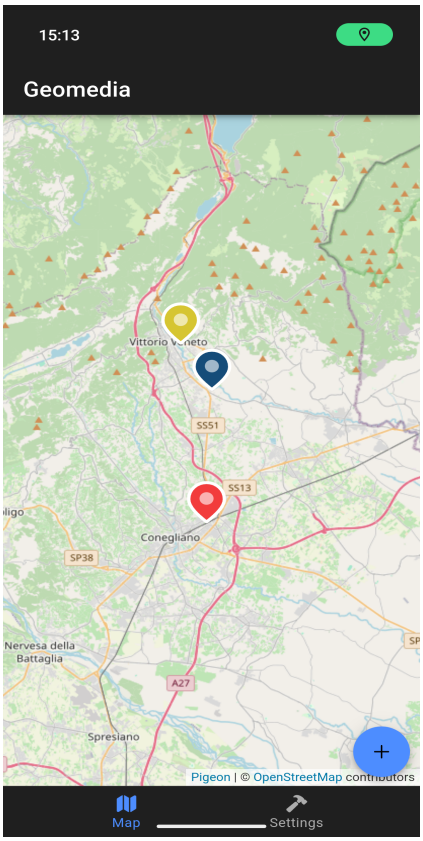

Fig. 2. Result of Map component.

Another key component is the ”ViewPost” (ViewPost.jsx), which shows the comment of the clicked market, by opening a new interface and ask the server to retrieve any associated multimedia data. This component is not limited to just viewing the comment; it also provides the functionality for downloading the attached multimedia file and store into the local storage of the mobile.

```
const ViewPost = (props) => {
     const refModalPost = useRef()
    const ctx = useContext(mycontext)
    const [DataBase64, setDataBase64] = useState(
    null) //to store the mediafile attached

    async function downloadFile(file) {
        let checkPer = await Filesystem.
    checkPermissions()
        let ask = await Filesystem.
    requestPermissions()
        try {
            let y = await Filesystem.writeFile({
                path: file.MEDIAFILENAME,
                data: DataBase64,
                directory: Directory.Documents,
            });
            if (y.uri.length > 0) {
                ctx?.showMessage("File saved into
    Documents folder", "success")
            }
        } catch (error) {
            alert(error)
        }
    }
     // get the media content in base64, different
    endpoint due to increment performance
    function getMediaPost(postid) {
        doRequest("getMediaPost", {
            POSTID: postid
        }).then(res => {
            //res[0]?.MEDIADATA
            if (res[0]?.MEDIADATA != null) {
                setDataBase64(res[0]?.MEDIADATA)
            }
        })
    }
    return(
    <>
        <IonModal>
        </IonModal>
    </>
    )
}
```

Listing 2. ViewPost

The mediafile is stored as Base64 encode into the data-layer, so the client will be also able to encode and decode the file. The owner of the comment of course is able to delete it.

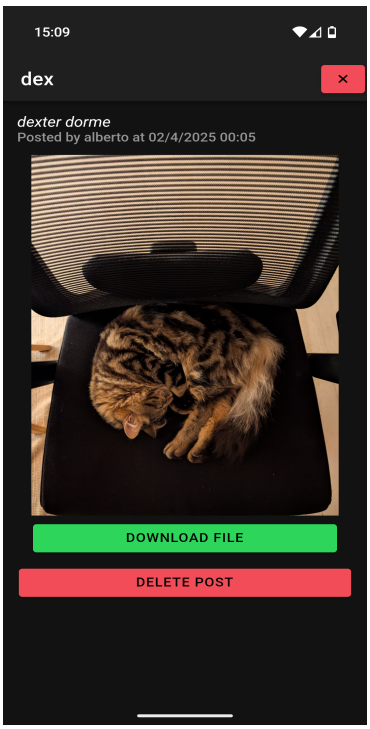

Fig. 3. Example of Post View.

B. HTTP requests

The custom function 'doRequest()' shown for example in the map component is a wrapper for an HTTP request to the server; where it's used the native JavaScript function "fetch" to make the request

```
//various import
import { Storage } from '@ionic/storage';
const store = new Storage(); store.create();
export const doRequest = (api, body = {}) => {
    return store.get("ServerConfig").then(
    ServerConfig => { //get the config from cache
        return fetch("http://" + ServerConfig.
    ipserver + ":" + ServerConfig.port + "/" + api,
    {
            method: "POST",
            mode: "cors",
            body: JSON.stringify(body)
        }).then(res => res.json()).then(res => {
            return res;
        }).catch(err => {
            return err;
        });
    })
}
```

Listing 3. Method doRequest to fetch server

Thus to allow a simple configuration of server address, has been installed the "Ionic Storage" [9] packet, which allows to store data into the application's cache installed in the mobile phones. In order to use this feature an installation is required via '$ npm install @ionic/storage'

C. Ionic Capacitor

To access physical sensors on Android, such as geolocation and file storage, Ionic Capacitor [10] can be integrated into the Ionic Framework project. Capacitor is a collection of libraries that make possible the use of native components inside web applications, then providing the API to adopt at JavaScript level.

Geomedia require two specific modules of Capacitor, in specific:

- GeoLocation [2]: In way to get latitude and longitude coordinate of user logged into the app
- FileSystem [3]: In order to store file into mobile filesystem whenever users want to download the media uploaded

These plugins can be installed via a shell command, respectively:

- '$ npm install @capacitor/geolocation'
- '$ npm install @capacitor/filesystem'

The geolocation package provide the method 'getCurrentPosition()' which returns an Object with properties like: latitude, longitude, altitude and the accuracy for each type of coordinate.
The altitude even if not relevant for this project, could be integrated in future development thus to create more peculiar scenario or use cases.

D. Building for Android

A key strength of the Ionic Framework is its ability to build web applications into native mobile apps; in fact the framework acts as a wrapper/interpreter for the webpage, essentially embedding a custom browser inside the app.
In order to create an Android Package (APK) for installing the app, some shell commands are required as reported in the official documentation, such as:

- 'npm install @capacitor/android' to add the capacitor
- 'npx cap add android' to denote the android platform, since IonicFramework can also build for "iOS" operating system
- 'npx cap build android' to export the application in production mode and so encapsulate the app into a bundle that later will be compiled by Android Studio [11] tool and the relative SDK

At this point, the Android Manifest needs to be modified, by adding the opportune permissions which the users will grant when requested on startup/installation of the app.
The only permission requested is the access to location, if denied the application will not work; a potential solution to avoid this scenario is to use the proximal location provided by antennas of the IPS, but could results in a very different position than the real one.

Another customization is to allow clear traffic (HTTP), because since from Android 8 only HTTPS packets are allowed by default. In fact, HTTP protocol is an insecure format where data is not encrypted and it's intended to be used just in the development phase, so the explicit declaration in the manifest is required in way to make the Android OS create an exception.

```
<application
    android:allowBackup="true"
    android:icon="@mipmap/mymap"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/mymap_round"
    android:supportsRtl="true"
    android:requestLegacyExternalStorage="true"
    android:largeHeap="true"
    android:usesCleartextTraffic="true"
>
...
</application>
...
<uses-permission android:name="android.permission.
    ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.
    ACCESS_FINE_LOCATION" />
<uses-feature android:name="android.hardware.
    location.gps" />
```

Listing 4. Snippet of Android Manifest

IV. HTTP Server

In the application layer the HTTP server takes place, which represent the bridge between data stored and the mobile application explained before. The GeoMedia HTTP server use NodeJS [4] as engine to start a listening process on a designated network port; even more, the server exposes several path called endpoints, each corrisponde to a specific function within the application. In fact, the server map the request to

the execution command of an SQL stored procedure, which handles the respective queries on the opportune tables.

The packages handled in this platform has a JSON (JavaScript Object Notation) content-type thus to allow a rapid extendability of data included and providing an easily human-readable format. Only the ”POST” method is implemented, since every request/response include data and parameters.

The server requires just one external dependency not included natively, which is ’tedious’ [5] (installable via ‘$ npm install tedious‘) a module to establish a connection to Microsoft SQL Database (DBMS) then to execute query commands and transactions.

```
const http = require("http");
const port = 9911

/**
 * Invia una risposta HTTP
 * @param {Object} res res of http
 * @param {int} status of response
 * @param {Object} body
 * @param {String} mime
 */
function sendResponse(res, status, body = null, mime = "application/json") {
    res.writeHead(status, { 'Content-Type': mime, "Access-Control-Allow-Origin": "*" });
    try {
        if (mime == "application/json") {
            res.write(JSON.stringify(body));
        } else {
            res.write(body)
        }
    } catch (err) {
        res.write(err);
    }
    res.end();
}
http.createServer((req, res) => {
    let body = "";
    req.on("data", (chunk) => {
        body += chunk
    });

    req.on("end", () => {
        try {
            body = JSON.parse(body)
        } catch (error) {
            //nthg
        }
         if(req.url=="/checkConnection"){ // uitility, client on startup send this request, to make sure its configuratin is correct. If server responds the configuration is right
            sendResponse(res,200,{"HELLO":"From server!"})
        }

        try {
            switch (req.url) {
                case "/doLogin": //LOGIN OR REGISTER
                    dispatcher.doLogin(body.username, body.password, body.newuser).then(resQuery => {
                        sendResponse(res, 200, resQuery)
                    }).catch(err => {
                        sendResponse(res, 500, err)
                    })
                    break;
                case "/newPost":
                    dispatcher.newPost(body.author, body.postcontent).then(resQuery=>{
                        sendResponse(res,200, resQuery)
                    }).catch(err=>{
                        sendResponse(res,500,err)
                    })
                    break
                case "/getPosts":
                    dispatcher.getPosts(body?.LATITUDE, body?.LONGITUDE,body?.USERNAME).then(resQuery=>{
                        sendResponse(res,200, resQuery)
                    }).catch(err=>{
                        sendResponse(res,500,err)
                    })
                    break
                ...
                default:
                    sendResponse(res, 404, { 'msg': "Unknown path:" + req.url })
                    break;
            }
        } catch (error) {
            fs.appendFileSync('./log.txt', 'ERROR ,' + Date.now() + ',' + error)
        }

    })
}).listen(port);

console.log("Server started on port: "+port);
```

Listing 5. Snippet of GeoMedia HTTP server

The dispatcher class is just a helper to wrap the execution of stored procedure, for example

```
const SQL_MANAGER = require("./SQL_MANAGER"); // wrapper for tedious package

function getPosts(latitude=null,longitude=null, username=null){
    return SQL_MANAGER.selectQuery(SQL_MANAGER.loadConfig(),"EXEC GETPOSTS @LATITUDE= "+latitude+", @LONGITUDE="+longitude+",@USERNAME='"+username+"'")
}
function getMediaPost(postid){
    return SQL_MANAGER.selectQuery(SQL_MANAGER.loadConfig(),"EXEC GETMEDIAPOST @POSTID="+postid)
}
...
module.exports = {
    ...
    , getPosts
    , getMediaPost
}
```

Listing 6. snippet example of dispatcher.js class

A. Microservices

The server represent a single access point, which is a fragile solution in a real-world scenario, as it represents a ”single point of failure” where the failure of a this node makes unavailable the whole system.
There are many alternatives to mitigate this event. One approach is to replicate the HTTP server across different machines, introducing the challenge of synchronizing multiple

main servers. A more sophisticated solution, is to change the whole design of this solution, creating a distributed architecture by adopting microservices [6]; where the server is fragmented into multiple smaller process that acts one for each specific function (as shown in Fig. 4). For example, one microservice could handle the retrieving process of the posts, another could manage the accounts, another the post creation, and so on. This approach not only imrpoves reliability but also helps to distribute traffic loads thust to prevening the system from an overwhelmed; a common problem with social media platforms when a mass requests of a viral content happens, causing the ”break the internet” effect.

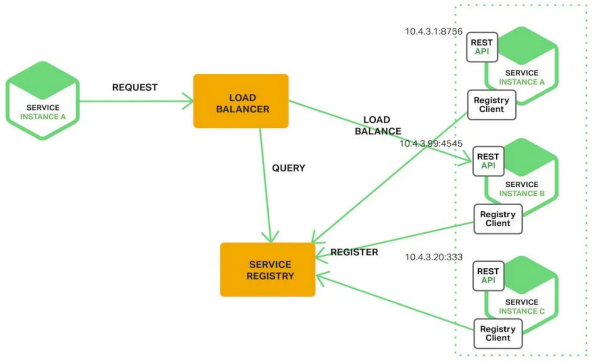

Fig. 4. Example of microservices architecture.

V. DBMS

All posts and user profile present on GeoMedia are stored in a Database Management System (DBMS), where Microsoft SQL Server [7] serving as the engine.

The DBMS stores data in two primary tables:

- ‘USERS‘ for users’ profiles
- ‘POSTS‘ for posts

```
CREATE TABLE USERS(
    ID INT IDENTITY,
    USERNAME VARCHAR(50) PRIMARY KEY,
    PASSWORD NVARCHAR(256) NOT NULL, --stored the
    hash of MD5 function
);
CREATE TABLE POSTS(
  ID INT IDENTITY PRIMARY KEY,
  AUTHOR VARCHAR(50) FOREIGN KEY REFERENCES USERS(
    USERNAME),
  TITLE VARCHAR(50),
  COMMENT VARCHAR(MAX),
  MEDIATYPE VARCHAR(100), -- IMAGE/JPG, AUDIO/M4A
  MEDIADATA NVARCHAR(MAX), --BASE64
  POSTDATETIME DATETIME,
  COORD_X FLOAT,
  COORD_Y FLOAT,
);
```

Listing 7. Tables creation

Data is managed through stored procedures, which allows more efficient and accurate query modifications. For example, when storing a post, the author (user) of the post and all associated metadata (as the whole JSON data received by server) are passed as parameters, then manipulated and formatted into SQL. Giving also the isolation and independence from the application layer.

```
CREATE PROCEDURE NEWPOST
@AUTHOR VARCHAR(50), @POSTCONTENT NVARCHAR(MAX)
AS
BEGIN
  INSERT INTO POSTS(AUTHOR,TITLE,COMMENT,MEDIATYPE,
    MEDIADATA,POSTDATETIME, LATITUDE, LONGITUDE)
  SELECT @AUTHOR, JJ.title, JJ.comment, JJ.mediatype
    , JJ.media_b64,GETDATE(), JJ.latitude ,JJ.
    longitude
  FROM OPENJSON(@POSTCONTENT,'$') WITH(
    title  VARCHAR(50),
    comment  VARCHAR(MAX),
    media_b64  NVARCHAR(MAX),
    mediatype  VARCHAR(100),
    latitude FLOAT,
    longitude FLOAT
  ) JJ
END;
```

Listing 8. Stored Procedure of post creation

VI. Future extensions

In conclusion, GeoMedia offers a simple platform for sharing content on a geographic map. As previously mentioned, there are some key features that could be expanded.

Some ideas include:

A. Live comments

A potential feature to temporize posts to be available only at specific times or within a defined geographic area. This could be used to create interactive experiences, such as ”treasure hunts” or ”hide and seek” games.

B. Time capsule

Extending the concept of limiting the content to a specific area, another idea could involve storing posts in small sensors distributed randomly in places such as forests, mountains, or cities. These sensors would be reachable only with short-range medium like Bluetooth or Wi-Fi, or even NFC.
This idea gives life to several use cases, such as a waypoints to mark hiking routes in the mountains, or actings time capsules at panoramic points.

References

[1] Ionic Framework is cross-platform framework to create web-application and export them to mobile platform. Link
[2] Ionic Capacitor - Geolocation package to access to GPS location on mobile devicesLink
[3] Ionic Capactor - FileSystem package to store data on mobile devicesLink
[4] NodeJS as server engine Link
[5] Tedious - Node Package able to handle SQL Server communication Link
[6] Microservices panoramic and image in figure Link
[7] Microsoft SQL Server 2022 as DBMSQ for data layer Link
[8] Pigeon Map as React Component to render a geographical map Link
[9] Ionic Storage to store the configuration of serverLink
[10] Ionic Capacitor official website as documentation for the modules installed Link
[11] Android Studio SDK official website, as a tool for building the Android APK Link
[12] React JS as chosen front-end framework Link