

Geomedia: a framework to share comments and multi-media content based on location.

*Project of "Wireless Network for Mobile Application", 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 and trending content but also on geographic location of the end-user. GeoMedia is a framework, developed into an Android 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, privacy, client server, multimedia, media sharing, android location, http, three-tier, three-tier architecture, SQL, DBMS

I. INTRODUCTION

In the last decade, mobile applications has 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 physical sensors present to enrich the user experience.

The explosion of social media led media companies to invest and explore almost all functionality of the mobile device, thus to make users feel more connected and satisfied by providing more relevant content. As 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.

For 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 mobile application that allows users to share comments, photos, audio, videos, and other multimedia content on where they are, and then, see the content through a map.

The whole source code is public on GitHub 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 allows mobile apps to communicate each other without a direct connection.

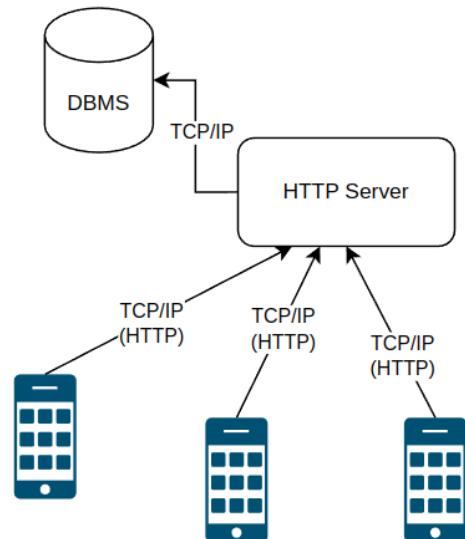


Fig. 1. Design Architecture.

A. Presentation layer

The presentation layer consists in the "GeoMedia" mobile app itself, representing the interaction point between the users and the whole architecure designed. By providing a user interface, allows users to view others users' posts or 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 others metadata even if not directly inserted by 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 via 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 of 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, thus to offer to development a more flexible way to modify the select or insert of data queries.

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 such as 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's 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 render of webcomponent shown in the user interface at the change of the data inside.

As follow an example of a React Component created

```
//various imports ...
import { useContext, useEffect, useState } from "react";
const MyPMap = () => { //declare the Map component
4
5  const [PostList, setPostList] = useState([]) // to store the list of posts
6  const ctx = useContext(mycontext) //use the React context to retrieve data of other components
7
8  // retrieve all post giving the actual position of user (in the server it will be computed the nearest posts)
9  function getPosts() {
10    if (ctx?.UserPosition != null) {
11      doRequest("getPosts", {
12        latitude: ctx?.UserPosition[0]
13        , longitude: ctx?.UserPosition[1]
14      }).then(res => {
15        setPostList(res)
16      })
17    } else {
18      doRequest("getPosts", {
19        latitude: null
```

```
        , longitude: null
21      }).then(res => {
22        setPostList(res)
23      })
24
25    }
26  }
27
28  useEffect(() => {
29    getPosts(); //to call the post
30  }, []); //at the startup of component
31  return( //render the graphics
32    <>
33      <Map defaultCenter={ctx?.UserPosition}
34      defaultZoom={11}>
35        <ZoomControl />
36        <Marker width={50} anchor={ctx?.UserPosition} color={"#154c79"} />
37        {
38          PostList?.map(s => (
39            <Marker width={50} anchor={[s.LATITUDE, s.LONGITUDE]} color={(s?.MEDIATYPE?.length > 0) ? '#d6c531' : '#f23c3c'}
40              onClick={() => {
41                setPostSelected(s) // store the post selected to open it in the opportunely component (viewPost)
42              }
43            })
44          )
45        ...
46      </Map>
47    ...
48  );
}
export default MyPMap;
```

Listing 1. Map component snippet

In the React component declared, has been used also the Pigeon Map [8] provided as a React Component, which renders a geographical map and consents to put some 'marker'. In GeoMedia, there are three types of markers.

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

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

```
//various import
import { Storage } from '@ionic/storage';
const store = new Storage();
store.create();
export const doRequest = (api, body = {}) => {
6  return store.get("ServerConfig").then(
    ServerConfig => { //get the configuration stored in the cache of the mobile phone
7    return fetch("http://" + ServerConfig.ipserver + ":" + ServerConfig.port + "/" + api,
{
8      method: "POST",
9      mode: "cors",
10      body: JSON.stringify(body)
11    }).then(res => res.json()).then(res => {
12      return res;
13    }).catch(err => {
14      return err;
15    });
}
```

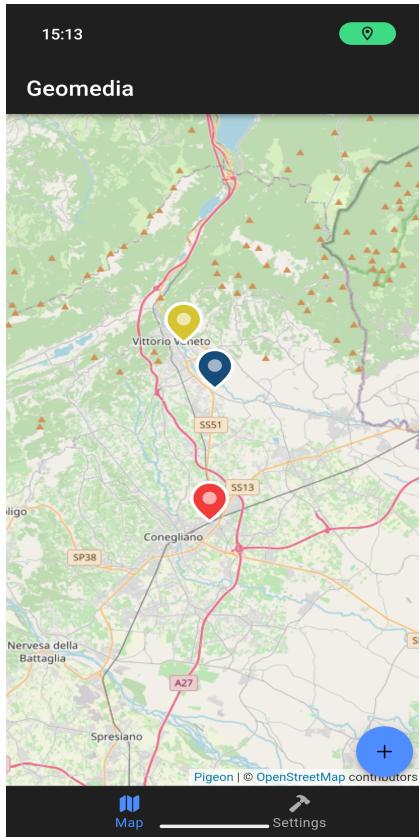


Fig. 2. Example of Map component.

```
16    })
}7
```

Thus to allow a simple configuration of server address, has been installed the "Ionic Storage" [?] 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'

At the startup of the component (triggered after the user login), an HTTP request is made to the server to retrieve the posts previously uploaded in GeoMedia, then each post will be rendered as marker on the map.

The method is parameterized to get all posts, or to make the server compute the post closer to the location 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.

Another key component is 'ViewPost.jsx', which shows the comment of the clicked market, by opening a new interface and fetching 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) => {
2  const refModalPost = useRef()
3  const ctx = useContext(mycontext)
```

```
4  const [DataBase64, set DataBase64] = useState(null) //to store the mediafile attached
5
6  async function downloadFile(file) {
7    let checkPer = await Filesystem.checkPermissions()
8    let ask = await Filesystem.requestPermissions()
9    try {
10      let y = await Filesystem.writeFile({
11        path: file.MEDIALFILENAME,
12        data: DataBase64,
13        directory: Directory.Documents,
14      });
15      if (y.uri.length > 0) {
16        ctx?.showMessage("File saved into
Documents folder", "success")
17      }
18    } catch (error) {
19      alert(error)
20    }
21  }
22  // get the media content in base64, different
endpoint due to increment performance
23  function getMediaPost(postid) {
24    doRequest("getMediaPost", {
25      POSTID: postid
26    }).then(res => {
27      //res[0]?._MEDIADATA
28      if (res[0]?._MEDIADATA != null) {
29        set DataBase64(res[0]?._MEDIADATA)
30      }
31    })
32  }
33  return (
34    <>
35      <IonModal>
36        </IonModal>
37    </>
38  )
39}
```

Listing 2. ViewPost

A. Ionic Capacitor

In order to access physical sensors on Android, such as geolocation and file storage, Ionic Capacitor [?] can be integrated into the Ionic Framework. Capacitor is a collection of libraries that make possible the integration of web applications with native components, providing the API to adopt at Javascript level.

Geomedia require two specific modules provided by Ionic 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 parameters like: latitude, longitude, altitude and the accuracy for each type of coordinate. The altitude even if not relevant for this project,

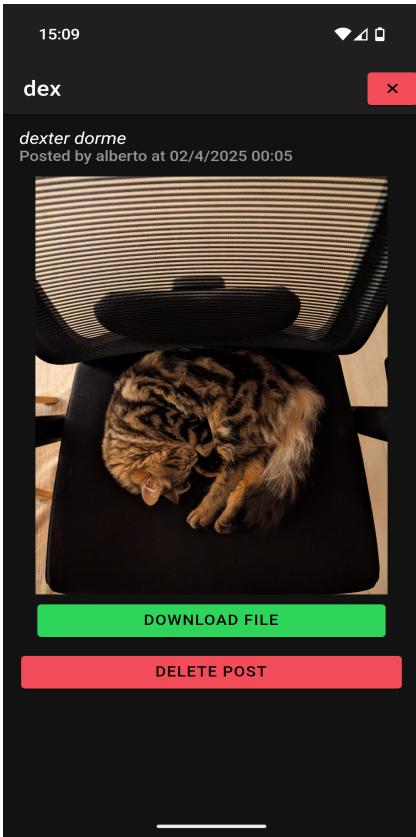


Fig. 3. Example of Post view.

could be integrated in future development thus to create more peculiar scenario or use cases.

B. 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 [?] tool and 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. Specifically, the only permission requested is the access to location, if denied the application will not work; a potential solution to this scenario is to use the proximal location provided by antennas

of the IPS, but could result in a very different location 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
    2     android:allowBackup="true"
    3     android:icon="@mipmap/mymap"
    4     android:label="@string/app_name"
    5     android:roundIcon="@mipmap/mymap_round"
    6     android:supportsRtl="true"
    7     android:requestLegacyExternalStorage="true"
    8     android:largeHeap="true"
    9     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 3. 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 a SQL stored procedure, which handles the respective queries on the respectively tables.

The package handled in this platform has a content-type of JSON (JavaScript Object Notation) 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
...
/***
 5* Invia una risposta HTTP
 6* @param {Object} res res of http
 7* @param {int} status of response
 8* @param {Object} body
 9* @param {String} mime
10*/
function sendResponse(res, status, body = null, mime =
    "application/json") {
12    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) {
        //nths
    }
    if(req.url=="/checkConnection"){ //utility, 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);  
74  
console.log("Server started on port: "+port);
```

Listing 4. Snippet of GeoMedia HTTP server

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 improves reliability but also helps to distribute traffic loads thus preventing 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.

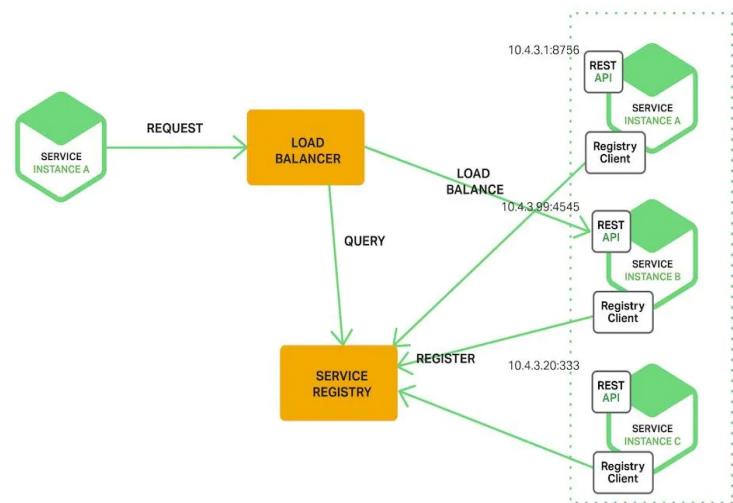


Fig. 4. Example of Map component.

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,
);

```

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;

```

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 acting as 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 devices [Link](#)
- [3] Ionic Capacitor - FileSystem package to store data on mobile devices [Link](#)
- [4] NodeJS as server engine [Link](#)
- [5] Tediou - 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](#)
- [b9] Ionic Storage to store the configuration of server [Link](#)
- [b10] Ionic Capacitor official website as documentation for the modules installed [Link](#)
- [b11] Android Studio SDK official website, as a tool for building the Android APK [Link](#)