INVESTIGACIÓN & DESARROLLO, No. **15**, Vol. **1**: 19 – 31 (2015)
ISSN 2518-4431

HEYCITY: A SOCIAL-ORIENTED APPLICATION AND PLATFORM ON THE CLOUD

HEYCITY: UNA APLICACIÓN Y PLATAFORMA DE CARÁCTER SOCIAL EN LA NUBE

Maximiliano Rasguido and Alex Villazón
Centro de Investigaciones de Nuevas Tecnologías Informáticas (CINTI)
Universidad Privada Boliviana, Bolivia
avillazon@upb.edu
(Recibido el 15 de mayo de 2015, aceptado para publicación el 07 de junio 2015)

ABSTRACT

Every city has problems related to infrastructure, services or security. Unfortunately, most cities lack of an on-line service or platform where citizens can complain about those problems. Typically there is only an email address or phone number where they can call, but only during working hours, which often results in frustration and long processing time until local authorities can solve those problems. In this article, we present HeyCity, a social-oriented application and platform that addresses this problem, by making the citizen an active part of the solution, and therefore increases responsibility. HeyCity provides a technological answer where users can report problems using their smartphone and collaborate with other citizens and local authorities to solve the problems. To be able to handle a large number of users distributed in different cities, HeyCity was deployed on the Cloud. We describe the design, development, deployment and execution of HeyCity on state-of-the-art Cloud services and tools, and we describe the technical choices that we made.

RESUMEN

Toda ciudad tiene problemas relacionados con su infraestructura, servicios o seguridad. Desafortunadamente, la mayoría de las ciudades carecen de un servicio o plataforma en línea donde los ciudadanos puedan reclamar acerca de esos problemas. Normalmente sólo hay una dirección de correo electrónico o número de teléfono donde se puede llamar, pero únicamente durante horas de oficina, lo que a menudo se traduce en frustración y en tiempos espera demasiados largos hasta que las autoridades locales puedan resolver esos problemas. En este artículo se presenta HeyCity, una aplicación y plataforma de carácter social que aborda este problema haciendo del ciudadano una parte activa de la solución, aumentando así la responsabilidad social. HeyCity ofrece una respuesta tecnológica para que los usuarios puedan informar sobre problemas utilizando sus teléfonos inteligentes y así colaborar con otros ciudadanos y las autoridades locales para resolver diferentes problemas. Para poder manejar una gran cantidad de usuarios distribuidos en varias ciudades, HeyCity fue desplegada en la Nube. Describimos el diseño, desarrollo, implementación y ejecución de HeyCity utilizando servicios y herramientas con tecnología de última generación de la Nube, justificando las opciones técnicas seleccionadas.

Keywords: Social-Oriented Application and Platform, Cloud Computing, PaaS, IaaS, DBaaS.
Palabras Clave: Aplicación y Plataforma de Carácter Social, Computación en la Nube, PaaS, IaaS, DBaaS.

1. INTRODUCTION

The development of applications on the Cloud has been gaining attention in the last years, because of the advantages commonly acknowledged such as high availability, low cost and global scope [1] [2]. Cloud computing provides a shared pool of configurable resources (e.g. processing, network, software, data and storage) on demand, as a scalable and elastic service, through a networked infrastructure, on a measured (pay-per-use or subscription) basis, which needs minimal management effort and often uses virtualized resources [1]. There are several competing Cloud Platforms and Services (Amazon Web Services[1], Microsoft Azure[2], Google Cloud Services[3], Heroku[4], FIWARE[5] to mention some of them), each one with their own features. From the developer point of view, it is challenging to choose tools, middleware or programming language to build applications designed for the Cloud, so as to take advantage of such a wide infrastructure and cope to the challenge of scalability.

In this article we describe the development of HeyCity, a social-oriented application and platform on the Cloud. HeyCity is both an application and platform that allows citizens to report different infrastructure problems using their

[1] http://aws.amazon.com/
[2] http://azure.microsoft.com/
[3] http://cloud.google.com/
[4] http://heroku.com/
[5] http://www.fiware.org/

DOI: 10.23881/idupbo.015.1-3i

smartphones, and let other users and local authorities to locate hot-spots in a city that require intervention. Originally, HeyCity was tested and deployed in a single server. However, because of its wide applicability (it can be used globally for any city in the world) and its potential growth, migrating the application to the Cloud was necessary. Its simple architecture made the migration process straightforward, requiring no re-engineering for the Cloud deployment. We chose to deploy HeyCity Platform on the Cloud infrastructure provided by Heroku and to use MongoLab[6] as data storage service.

The original contributions of this article are twofold:

- We describe the architecture of the HeyCity social-oriented application and platform
- We describe the implementation details of the HeyCity prototype to enable broad access and scalability of the application on the Cloud

This article is structured as follows. Section 2 introduces the HeyCity App architecture. Section 3 gives an overview of the tools we used to develop the application. Section 4 details the implementation of HeyCity and Section 5 concludes the article and discusses Future Work.

2. HEYCITY: A SOCIAL-ORIENTED APPLICATION AND PLATFORM

The main goal of HeyCity is to allow both, citizens and local government authorities, to collaborate in order to solve common problems in a city. Citizens can report infrastructure problems (e.g. holes on roads, broken public or traffic lights), public safety (e.g. accidents, problems related with alcohol consumption) or services (e.g. water supply, sewerage failures). Local authorities can take actions based on those reports. These problems are unfortunately very common in a lot of cities in Latin America and abroad, and citizens have almost no possibility to report them. HeyCity brings a simple yet efficient technological answer to a social problem, where the citizen is part of the solution.

HeyCity was designed to be simple to use. The citizen interaction is done through an Android App whereas local authorities (or other users) access the reports through a Web Application front-end. The HeyCity App allows users to use the smartphone camera and GPS to report problems. Users can view reports that are close to their current location within the App (in a map) and also access to the global view through the Web.
In the following we describe both the server and client sides of HeyCity, giving an overview of the used technology and the authentication process.

2.1. Server Side

The HeyCity platform follows a three-tier architecture. The front-end server handles client requests to store generated reports and deliver them on demand. The server provides a RESTful [3] Web Service Application Programming Interface (API) for these interactions. The report processing application server is implemented in JavaScript using the Express Framework[7] on top of Node.js[8]. We chose Node.js because of its simple deployment, high-performance network capabilities and simplified scalable programming model (not requiring handling concurrency explicitly) [4]. The back-end data storage is handled by a MongoDB database [5]. We chose to use MongoDB because of its flexible data modeling and scalability, compared to conventional SQL databases. Also, the use of JavaScript simplifies the interfaces between the different layers, as MongoDB uses BSON, a superset of JavaScript-based JSON serialization structure[9].

2.2. Client Side

The HeyCity platform has two different clients: An Android and a Web Client App.

- **Android App:** The Android App is used to send reports to the HeyCity platform. It uses GPS information and the camera to take pictures that are uploaded to the server. The Android App gathers a list of other user's reports that are close to the current location, and displays them on a map. The Android App only allows reporting events using the current GPS location, i.e. it does not allow uploading images or manually setting a different location. The App

[6] http://www.mongolab.com/
[7] http://www.expressjs.com/
[8] http://www.nodejs.org/
[9] MongoDB uses an extended JSON (JavaScript Object Notation) format as serialization format for documents and remote procedure calls. BSON can be seen as a "binary" representation of JSON.

however allows off-line reports by storing the GPS location and image on the device, and uploading the information once on-line, i.e. it is not mandatory to be on-line to use the HeyCity App.

- **Web App:** The Web Client App is used only to visualize reports on a map. As for the Android App, we use the Google Maps API[10]. The Web Client App uses a pull mechanism to interact with the server to gather reports according to the location that is displayed. It generates a "heat map" with the locations that have more reporting activities. The Web Client App also uses a notification mechanism to receive live report updates.

2.3. Authentication

To prevent abuses, the HeyCity Android App uses Facebook credentials obtained through the Facebook Software Development Kit (SDK) for Android[11]. This authentication is only asked upon the first report. The authentication token is stored in HeyCity server and is used to display only public information of the user, thus preserving users' privacy. By default, user's HeyCity reports are not posted on Facebook. The credentials are mainly used to prevent users to post inappropriate content in the platform, because their public profile is visible on HeyCity. The Facebook SDK itself makes the token management and renewal inside the application.

3. TOOLS OVERVIEW

Different services and their associated tools are available to build applications on the Cloud. According to NIST's widely accepted Cloud Service Model definition [1], Cloud computing can include software (Software as a Service or SaaS), hardware (Infrastructure as a Service or IaaS), or technology tools (Platform as a Service or PaaS) that are available on demand. More recently, other service models have emerged for data storage (Database as a Service or DBaaS, Storage as a Service or StaaS) and network (Network as a Service or NaaS) [6]. Depending on the application needs, one or several of these models can be combined to implement an application, i.e. applications can use different service models from different Cloud service providers. The challenge is therefore choose the correct service model and take into account the set of tools available to simplify the design, development, deployment and execution of the application.

In the following, we describe the building block tools that were used to deploy HeyCity on the Cloud. First, we discuss the tools to deploy the server. Second, we detail the tools for data storage. Finally, we give an overview of the tools for real time notifications.

3.1. Heroku

Heroku is a Cloud Platform as a Service (PaaS) that allows developers to easily deploy and scale applications to the Cloud by automating several parts of the deployment process. Heroku is able to do this, by compiling the application source code into a *slug* and running it on *dynos*.

Slugs are pre-compiled versions of the application ready to run in a *dyno*. To create a *slug*, Heroku receives the source code and analyzes it to determine which platform is required to build the application. Heroku supports Ruby, Node.js, Python, Java and PHP. In our case, the platform is Node.js, therefore the *Slug* compiler looks for a `package.json` file[12] to know which dependencies to fetch. With this information, the *slug* is ready for deployment.

Dynos are isolated, virtualized Linux containers, that provide the environment required to run an application [7], i.e. a *dyno* is a Linux Virtual Machine instance capable of running the code contained in the pre-compiled *slug*. *Dynos* have a limited processing capability so one *dyno* can only handle a given amount of requests at the same time. To overcome this limitation, Heroku allows scaling up an application by allowing the user to allocate several *dynos* to a single project. These *dynos* start to boot up when needed to fulfill the requests [8]. This allocation can be changed dynamically allowing us to scale up when high request traffic arrives and scale down when the service is not used very much. Each application *dynos* are spread across the "*dyno grid*", which consists of many servers running on Amazon Web Services (AWS) called *railguns*, i.e. Heroku's PaaS uses Amazon's IaaS underneath.

[10] https://developers.google.com/maps/
[11] https://developers.facebook.com/docs/android/
[12] The Node.js package management system (npm) uses a configuration file called package.json that contains metadata to handle dependencies. The description is formatted in standard JSON.

3.2. MongoLab

MongoLab is a Database as a Service (DBaaS) based on MongoDB, providing a fully-managed Cloud database service featuring highly-available MongoDB databases, automated backups, web-based tools and 24/7 monitoring [5]. It allows users to store application's data in a scalable way and providing 500 MB of free DB storage [9] for development and prototyping. Higher volumes require a pay-per-use plan. Other than MongoLab's free plan, the natural JavaScript integration and the powerful management tools were key to choose it as the DBaaS for HeyCity.

3.3. Socket.IO

Socket.IO[13] is a JavaScript library for real-time Web Applications. It has two parts: a client side library and a server-side library for Node.js. Both components have identical API and are event-driven. Socket.IO manages to do real time communication by using a series of different transports as fallback, i.e. if some transport is not working for any reason, the user can set a list of different transport[14] to be used [10]. Socket.IO allows bi-directional communication, i.e. allowing not only the client to interact with the server as in conventional AJAX, but also the server side to communicate with a web client, thus providing a robust foundation for notifications.

4. HEYCITY IMPLEMENTATION

The implementation of HeyCity follows a modular architecture, enabling easy deployment on the Cloud. Figure1 depicts the main components of HeyCity, which are structured as follows:

- A Node.js server running on the Heroku Cloud
- A MongoDB database hosted on MongoLab
- A Web Client served from the Node.js server
- An Android application connecting to the Node.js server

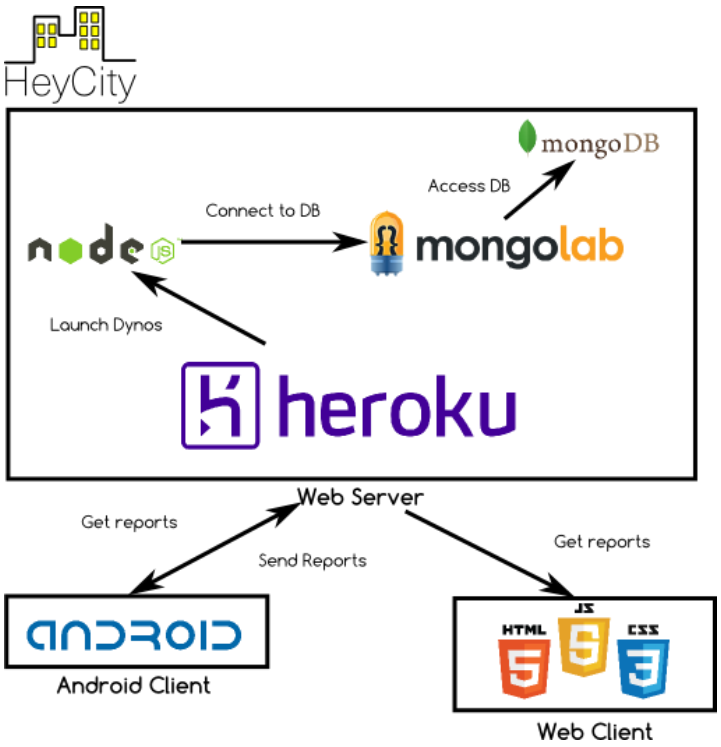

Figure 1 - HeyCity overall architecture.

[13] http://socket.io/
[14] Socket.IO unifies fallback transports by including WebSocket, Flash, AJAX long polling, AJAX multipart streaming, Iframe and JSONP Polling

4.1 Web-based Server and Client

The choice of using JavaScript on the server side (i.e. Node.js) was driven by the simplified deployment, high-performance and scalability. Furthermore, the express generator module of Node.js simplified the development, because it generates a complete Web Service skeleton code. Based on that, we developed a RESTful Web Service API (GET, POST, PUT and DELETE operations) for the HeyCity reports, comments and users, with the corresponding database connection logic. For the GET operation, the API allows to specify two coordinate points, so as to return only reports of a specific geographical area. The creation of new reports through the POST operation was enhanced with a broadcast notification mechanism using Socket.IO. That is, whenever a new report is created, the Socket.IO module on the server pushes a notification to all the connected clients transparently. The server is therefore able to serve both the Web Client and the Android App with exactly the same interfaces, making the server very lightweight. The clients are able to display the reports in both types of clients almost in real-time.

Figure 2 depicts the RESTful API to create a new report. The body for the POST operation (in JSON format) contains the report description, the location coordinates, the encoded image, and the token of the user credential from the Facebook authentication.

```
POST /reports

Parameters:
  The request body must be structured as follows

{
   Description: "A description", // Report description
   Latitude: -17.38535921951882,
   Longitude: -66.1595997056393,
   Image: "/9j/4AAQS...",        // An image encoded in Base64
   User: "731928374612"   // The id of the User who created the report
}

Returns:
  The result and id of the newly created report.
{
    result: true, // true or false whether the report was created or not
    _id: "553715fed65d320e00e5c6fa", // The recently created report id (no id if result is false)
}
```

Figure 2 - RESTful API to create new reports.

The newly created *id* and a *true* result is returned on success, no *id* and *false* otherwise. The newly created report *id* is then used to create a comment on the report as shown in Figure 3.

```
POST /reports/{id}/comments

Parameters:
   The request body must be structured as follows

{
   Comment: "Some comment", // Comment description
   User: "731928374612"      // The id of the User who commented
}

Returns:
  The result and id of the newly created report.
{
    result: true, // boolean (the report was created or not)
}
```

Figure 3 - RESTful API to create comments.

The user credential token is stored on the server and is used later on to gather the user's basic and public information from Facebook RESTful API. This information is cached in the server when any client requests a user's report or comment. PUT and DELETE operations for update and deletion of reports and comments, as well as users management API follow similar API (not shown).

Figure 4, Figure 5 and Figure 6 depict the RESTful API to gather a list of reports (based on current location), an individual report, and a list of comments associated to a given report, respectively. Gathering a list of reports uses two optional parameters: *coords* and *page*. The *coords* parameter allows specifying two location points (latitude and longitude) defining the top left and bottom right points defining the area from where we want to gather all the reports.

GET /reports

Parameters:

Location (optional): Specifies an area (two location points) from where to fetch reports.

?coords=latitude1,longitude1,latitude2,longitude2

Page (optional): The page to be returned (in case reports threshold is exceeded)

?page=N

Returns:

List of reports that follow the criteria and a pointer to the next page in case the reports are paged.
The result and id of the newly created report.

```
{
  reports: [
    {
      _id: "553715fed65d320e00e5c6fa",   // The report id
      Description: "A description",       // Report description
      Latitude: -17.38535921951882,
      Longitude: -66.1595997056393,
      Image: "/9j/4AAQS...",              // An image encoded in Base64
      User: {
            Name: "John Smith"                    // User name
            Picture: "http://www.example.com/user.jpg" // Profile picture
          }
      },
      {
       ...
      },
    ...
  ],
  next: /reports?coords=latitude1,longitude1,latitude2,longitude2&page=2
}
```

Figure 4 - RESTful API to GET reports.

GET /reports/{id}

Parameters: None

Returns:

The report with the id {id}.

```
{
  reports: [
    {
      _id: "553715fed65d320e00e5c6fa",   // The report id
      Description: "A description",       // Report description
      Latitude: -17.38535921951882,
      Longitude: -66.1595997056393,
      Image: "/9j/4AAQS...",              // An image encoded in Base64

      User: {
            Name: "John Smith"                        // Users name
            Picture: "http://www.example.com/user.jpg" // Profile picture
          }
      }
}
```

Figure 5 - RESTful API to GET reports for a given id.

To avoid returning too many reports at once, the API uses REST best practice "pagination" mechanism, by returning a next link (see Figure 4), if the number of reports exceeds a server configurable threshold (set by default to 20). The *page* parameter is used to this purpose. Similar paging mechanism is used to gather comments (see Figure 6).

GET /reports/{id}/comments

Parameters:

Page (optional): The page to be returned (in case of having too many comments)

?page=N

Returns:

The comments on the report with the id {id} and a pointer to the next page id (comments are paged if threshold is exceeded)

```
{
  comments: [
     {
      Comment: "This is a comment",  // The comment content
      User: {
            Name: "John Smith"                        // User name
            Picture: "http://www.example.com/user.jpg" // Profile picture
          }
      },
      {
       ...
      },
   ...
  ],
  next: /reports/553715fed65d320e00e5c6fa/comments?page=2
}
```

Figure 6 - RESTful API to GET comments for a given id.

Since the report image is stored in the HeyCity server, the encoded image is returned for each report. In contrast, for the user's public Facebook profile image, a URL is returned instead. The client is responsible to retrieve this image. The HeyCity server uses the stored authentication token to gather the user's name and picture URL without gathering the actual image. Note that the *_id* and the *Timestamp* are MongoDB identifiers and timestamps that are used to display comments in the correct ordering of creation and to show the time of creation.

On the Web Client implementation, the use of JavaScript was natural. This enables the use of Google Maps API, display markers with the information retrieved from the HeyCity RESTful API and implement interactions to add comments and receive notifications. Figure 7 shows the Web Client App displaying reports in a given location, whereas Figure 8 shows a report view.

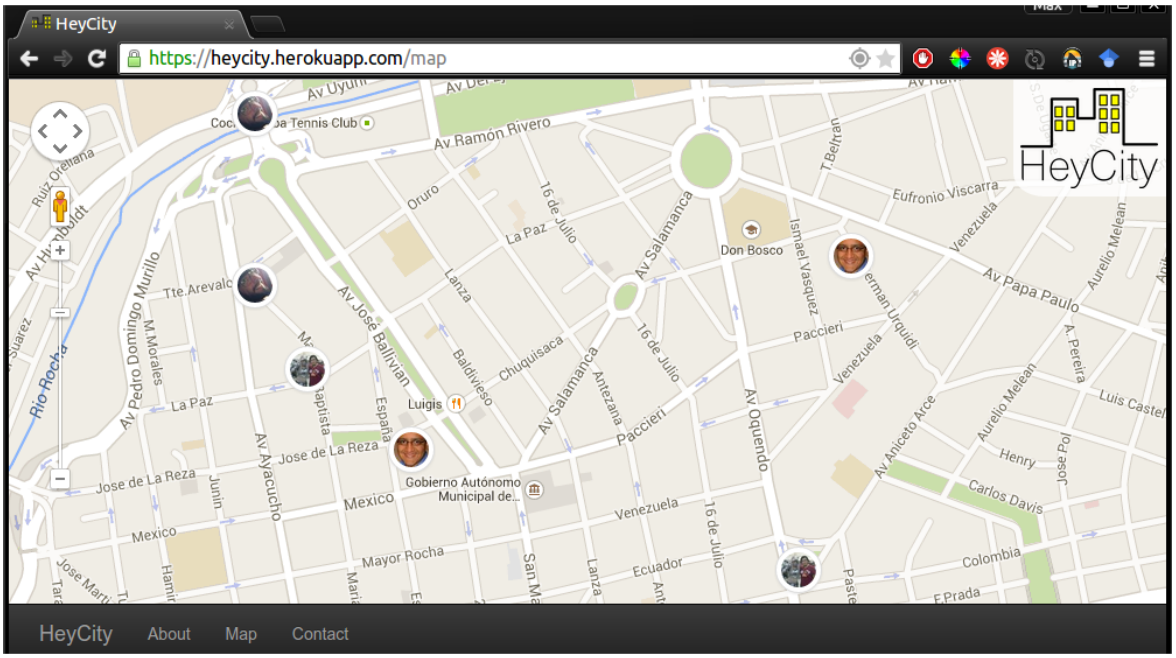

Figure 7 - Web Client displaying reports.

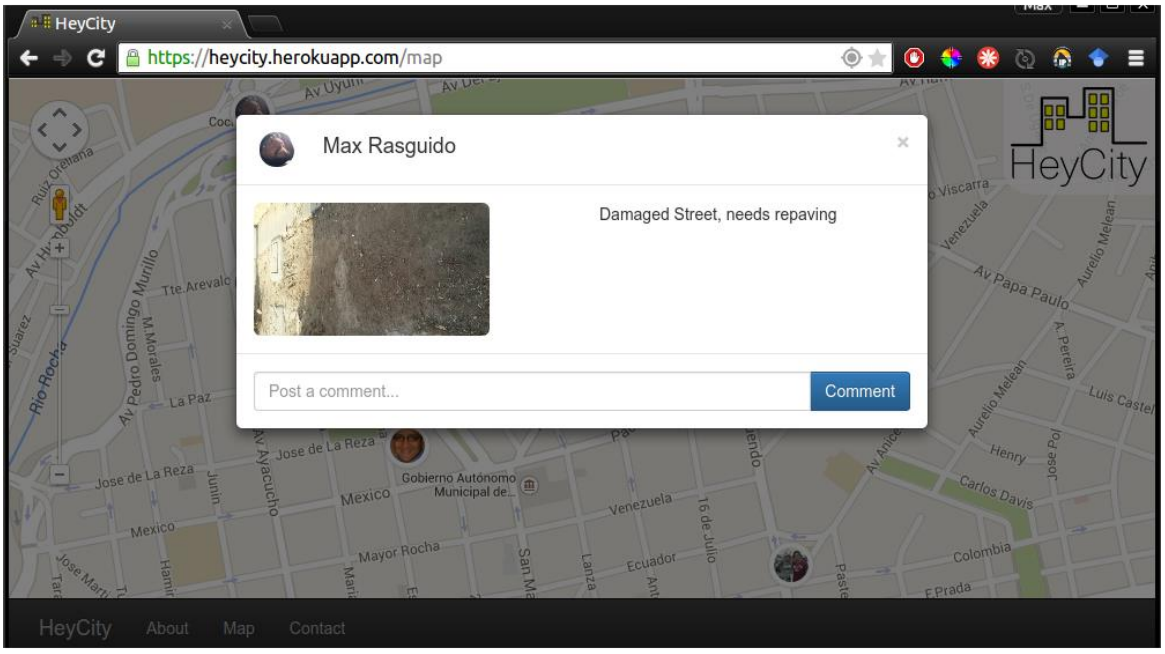

Figure 8 - Web report view.

4.1. Android-based Client

The Android client is the most important piece of HeyCity. It allows users to report problems by snapping a picture, adding a description and sending it to the server. Similar to the Web client, it displays reports in a map with markers. Figure 9 depicts the complete reporting workflow.

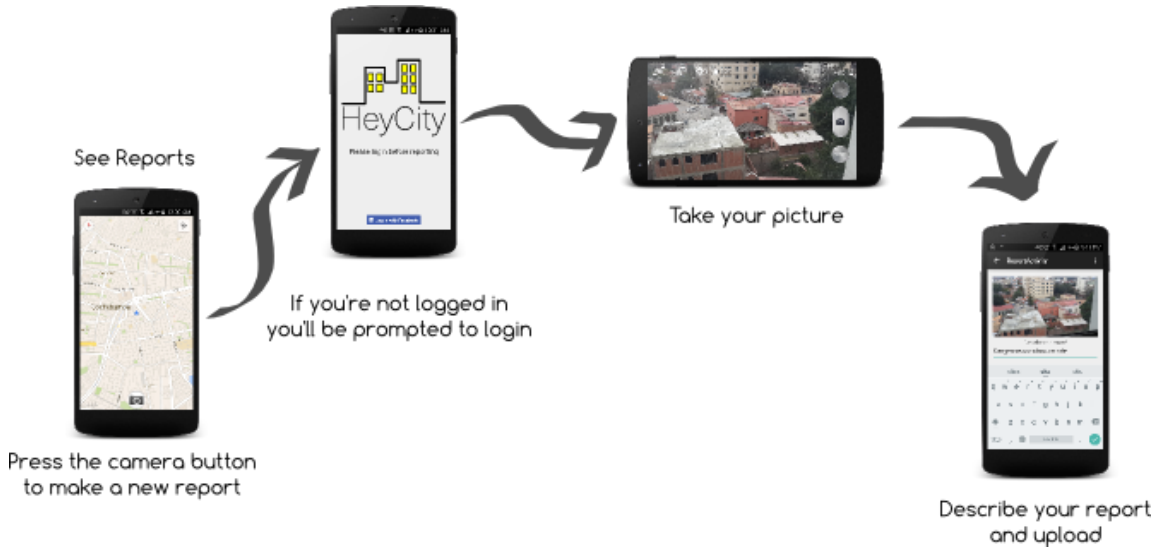

Figure 9 – Reporting workflow.

The Android client has four UI interfaces (called Activities in Android's programming model[15]) to interact with the user:

- **Map Activity.** This Activity is the first one presented to the user. As shown in Figure 10, it displays a map around his current location with markers representing the places where other users have posted reports. Tapping on the markers opens the Report Activity. The Map Activity is unauthenticated, i.e. any user that installs the app is able to see reports without login in. The Map Activity also includes the native Socket.IO for Android[16] module. This module allows receiving real-time notifications of reports from the server. The client then verifies if the currently displayed map is near the received report, and displays, caches or discards the received notifications.

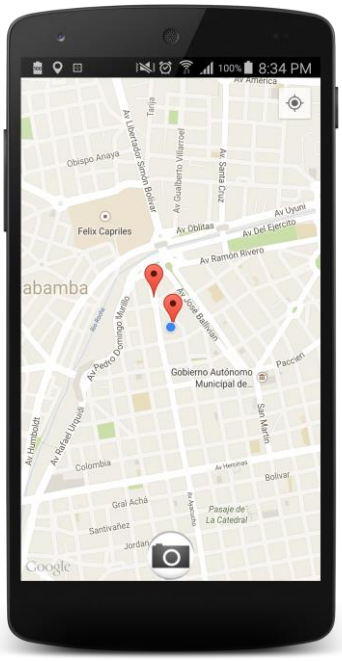

Figure 10 - Map Activity.

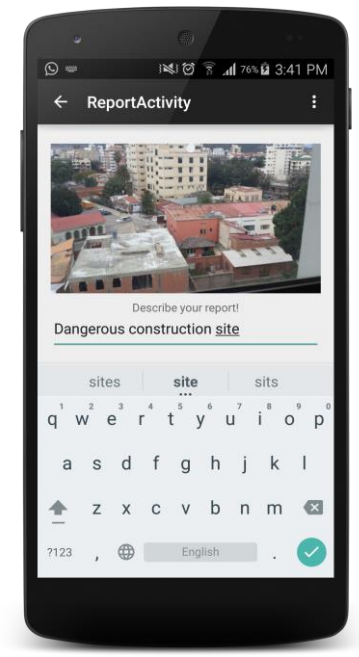

Figure 11 - Report Upload Activity.

[15] http://developer.android.com/guide/components/activities.html
[16] http://socket.io/blog/native-socket-io-and-android/

- **Login Activity.** This Activity is shown when an action requires the user to login. For example, if the report button is pressed, it first triggers the user authentication verification. If the user is not authenticated, the Login Activity screen is prompted requesting for Facebook credentials. Once authenticated, the reporting can proceed.
- **Report Upload Activity.** This Activity appears to the user after a picture is taken. It provides a field to describe what the reporting is about. This is an authenticated Activity, as it requires the user to be logged in, in order to post a report. Figure 11 shows the Report Upload Activity.
- **Report Activity.** This Activity provides a detailed view of a report that has been tapped on the map. It contains the report image, the report description and also the list of comments that have been made in the report. It also allows the user to add comments to the report (if the user wants to comment and hasn't logged in yet, the Login Activity will be prompted).

In general, the images captured by current smartphones are too big to be sent and stored efficiently on the server. Thus, the image is first compressed and then converted to Base64 as a convenient way to send the report images to the server and store them in the MongoDB database. Figure 12 depicts Java code for the image compressing and encoding in the Android App. Since the image is stored on the local storage of the device, the compression is done by reading the image from the local storage, rather than in memory only (see the file argument of the `encodeImage(...)` method). This ensures that the App has always a backup and cache image. The image reference, the credentials and location are locally stored in a SQLite[17] embedded database. The Android App also handles off-line reporting by running a background Service Activity on a separate thread to check the Internet connectivity and upload information when available.

```
// Encoding an image  in Base64 format
public static String encodeImage(String file){
    Bitmap bm = compressFile(file);
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    bm.compress(Bitmap.CompressFormat.JPEG,100,baos);
     byte[] b = baos.toByteArray();
     return Base64.encodeToString(b, Base64.NO_WRAP);
}
```

Figure 12 - Converting an image to Base64 in Android.

For authentication, we use the Facebook Android SDK[18] by launching a Login Activity waiting for a positive authentication result. The login functions are fully managed by the Facebook, i.e. HeyCity does not handle its own login functionality. Upon a positive Facebook authentication event, we store the generated token and inform the main HeyCity Activity. Facebook authentication requires a developer account credentials, Facebook App ID (i.e HeyCity registration), a generated Android Hash Key added to the Facebook App and a Facebook Activity declared in the HeyCity's Android application manifest file. In order to manage the login attempts properly, the Login Activity (see Figure 13) uses the `FacebookManager` class to handle notifications of user data upon token changes. When the Main Activity or the Map Activity is started or resumed, we check if the user is logged in (see Figure 14) and refresh user's data accordingly. If the user is not logged in, we do not ask user to login, as HeyCity App must be usable also just to see reports. However, when the user tries to make a report by pressing the camera button, then we check if the user has logged in, and start the Login Activity accordingly (see Figure 15).

- **Service deployment**

The deployment of the service uses Heroku Toolbelt[19], which is a set of command line tools that allows sending the application server to Heroku repositories, where it can be built, deployed and executed in Heroku. Internally, Heroku Toolbelt uses *git*, the widely used distributed version control system[20], for service deployment. The deployment step is made in a `git push` command to the Heroku master repository, assigned to the HeyCity application by the Toolbelt. This copies all the required source code of the HeyCity platform and allows simplified service deployment, i.e. it does not require any additional configuration or settings to deploy all the required components.

[17] http://www.sqlite.org/
[18] https://developers.facebook.com/docs/android
[19] https://toolbelt.heroku.com/
[20] https://git-scm.com/

Figure 13 - Login Activity.

```
public class FacebookManager {
   // gathers token and update user credentials information
   public AccessTokenTracker getAccessTokenTracker(final Activity parent){
      return new AccessTokenTracker() {
         @Override
         protected void onCurrentAccessTokenChanged(
               AccessToken oldAccessToken,
               AccessToken currentAccessToken) {
                  AccessToken.setCurrentAccessToken(currentAccessToken);
                  refreshUser(currentAccessToken, parent);
               }
      };
   }
   // refresh user token
   public static void refreshUser(AccessToken accessToken, Activity parent){
      SharedPreferences sharedPreferences = parent.getSharedPreferences("edu.upb.heycity",
                                                Context.MODE_PRIVATE);
      GraphRequest request = GraphRequest.newMeRequest(
            accessToken,
            new GraphRequest.GraphJSONObjectCallback() {
               @Override // generates anonymous callback entry
               public void onCompleted(
                     JSONObject object,
                     GraphResponse response) {

                  JSONObject user = object;
                  sharedPreferences.putString("user",object.toString())
               }
            });
      Bundle parameters = new Bundle();
      parameters.putString("fields", "id,name,email,picture");
      request.setParameters(parameters);
      request.executeAsync();
   }
}
```

Figure 14 - Facebook Manager class.

```
FacebookSdk.sdkInitialize(getApplicationContext());
AppEventsLogger.activateApp(this);
loggedIn = AccessToken.getCurrentAccessToken()!= null; // checks current token
if (loggedIn){
   // updates user credentials
   FacebookManager.refreshUser(AccessToken.getCurrentAccessToken(), this);
}
```

Figure 15 - Checking if the user is logged in.

4.3 Evaluation

We evaluated HeyCity running on a single *dyno* on Heroku, varying the number of requests, with requests = 2^n (n = 0...10) concurrent requests. We used the benchmark-pages[21] module for Node.js to measure the response time for the three main pages of HeyCity: main, map and report. These queries triggers also requests to the MongoDB database in order to retrieve reports and comments. Figure 16 shows the evolution of the requests for each of the pages for 1 to 16 requests. Here we can see that the most time consuming requests are related to the access to the reports information, by gathering information from the corresponding MongoDB collection. After 8 requests the response time grows linearly because the single dyno instance has to handle all the requests. The response time per request from 1 to 1024 concurrent requests vary between 1577 ms. and 2262 ms.

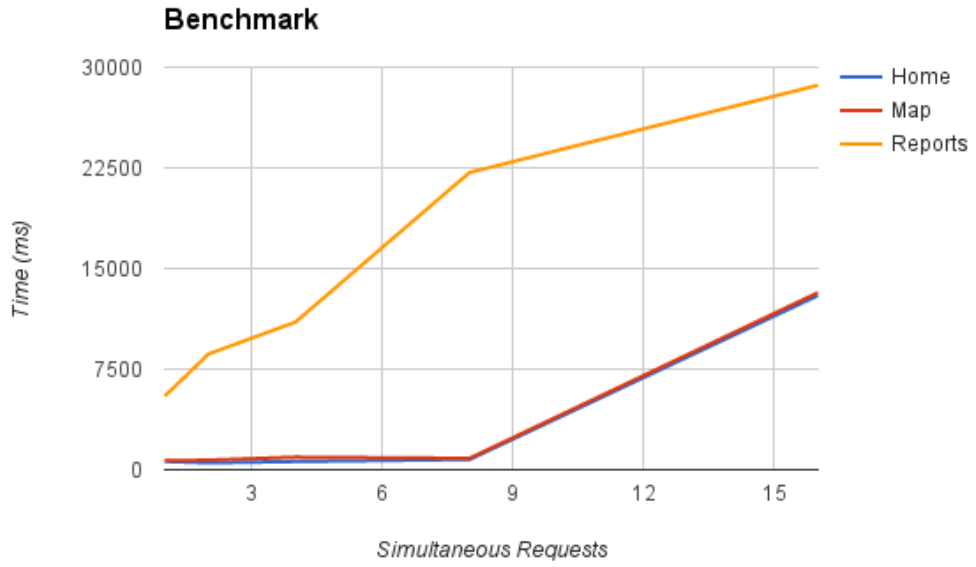

Figure 16 - Benchmark measuring total response time (in ms.) for HeyCity running a single dyno instance on Heroku.

5. CONCLUSION AND FUTURE WORK

In this article we described HeyCity, a Social-Oriented Application and Platform on the Cloud. We detailed the architecture and implementation based on state-of-the-art Cloud services and tools. Our implementation uses a RESTful Web Service API to simplify the interactions between the server and two types of clients (Android and Web) and leverages a powerful notification mechanism. Since HeyCity was designed to let normal citizens to report different problems in their cities, the HeyCity prototype and the scalability provided by the Cloud infrastructure can be the base for a massive social service to help citizens all over the world. Future work includes enhanced reporting service for

[21] https://www.npmjs.com/package/benchmark-pages

local authorities, advanced acknowledge service when reported problems are solved and detailed benchmarking. In addition, the platform can be extended to cover different types of reports.

6. ACKNOWLEDGEMENTS

The authors want to thank Randall Degges for his help with Heroku internals and Oscar Sanjinez for his work in the early version of HeyCity's Android App.

7. BIBLIOGRAPHY

[1] P. M. Mell and T. Grance. "SP 800-145. The NIST Definition of Cloud Computing," National Institute of Standards & Technology, Gaithersburg, MD, USA, Tech. Rep., 2011.

[2] B.P. Rimal et al. "A Taxonomy and Survey of Cloud Computing Systems," in *Proceedings of the 2009 Fifth International Joint Conference on INC, IMS and IDC*, Washington, DC, USA, 2009, pp. 44-51. [Online]. Avaible: http://dx.doi.org/10.1109/NCM.2009.218

[3] R. T. Fielding, "Architectural Styles and the Design of Network-based Software Architectures," University of California, Irvine, Ph.D. dissertation ISBN: 0-599-87118-0, 2000.

[4] S. Tilkov and S. Vinoski, "Node.js: Using JavaScript to build high-performance network programs," *IEEE Internet Computing*, vol. 14, no. 6, pp. 80-83, 2010.

[5] Object Labs Corporation. MongoLab | MongoDB. [Online]. Available: http://www.mongodb.com/partners/cloud/mongolab

[6] S. Kächele et al. "Beyond IaaS and PaaS: An Extended Cloud Taxonomy for Computation, Storage and Networking," in *Proceedings of the 2013 IEEE/ACM 6th International Conference on Utility and Cloud Computing*, Washington, DC, USA, 2013, pp. 75-82. [Online]. Avaible: http://dx.doi.org/10.1109/UCC.2013.28

[7] Heroku Inc. How Heroku Works. [Online]. Avaible: https://devcenter.heroku.com/articles/how-heroku-works

[8] R. Degges, The Heroku Hacker's Guide. Release 1.0, 2012.

[9] Object Labs Corporation. MongoLab. [Online]. Avaible: https://mongolab.com/

[10] C. Cheng. "What is Socket.IO?" [Online]. Available: http://learn-gevent-socketio.readthedocs.org/en/latest/socketio.html