FACULTAD DE INGENIERÍA Y CIENCIAS AGROPECUARIAS

IMPLEMENTACIÓN DE UNA PLATAFORMA QUE CONTRIBUYA CON EL MOVIMIENTO DE DATOS ABIERTOS SOBRE EL TRANSPORTE PÚBLICO EN QUITO

Trabajo de Titulación presentado en conformidad con los requisitos establecidos para optar por el título de Ingeniero en Sistemas de Computación e Informática.

Profesora Guía

MSc. Verónica Fernanda Falconí Ausay

Autor

Cristian Daniel Pinto Sánchez

Año

2016

DECLARACIÓN DE LA PROFESORA GUÍA

“Declaro haber dirigido este trabajo a través de reuniones periódicas con el estudiante Cristian Daniel Pinto Sánchez, orientando sus conocimientos y competencias para un eficiente desarrollo del tema escogido y dando cumplimiento a todas las disposiciones vigentes que regulan los Trabajos de Titulación”.

_____

Verónica Fernanda Falconí Ausay

Magister en Ciencias de la Computación y Comercio Electrónico

C.C. 0502395270

DECLARACIÓN DE AUTORÍA DEL ESTUDIANTE

“Declaró que el presente trabajo es de mi autoría, que toda la información citada en el mismo es tomada de las fuentes de información adecuadas y que durante su ejecución se respetaron las disposiciones legales que protegen los derechos de autor vigentes”.

_____

Cristian Daniel Pinto Sánchez

C.I. 1717527855

RESUMEN

Los datos abiertos son el resultado de la cooperación entre instituciones e individuos que han liberado información con la finalidad de alentar a investigadores, desarrolladores y emprendedores a utilizar y redistribuir los datos. Sin embargo, no todo tipo de información se puede encontrar en formato de datos abiertos y este es el caso con los datos de transporte público de Quito, específicamente de paradas y rutas.

Este trabajo habilitará a voluntarios generar y exponer datos sobre paradas y rutas en la ciudad de Quito. El resultado que este trabajo propone es un MVP (producto mínimo viable) que permite evaluar el correcto funcionamiento de los componentes que componen la plataforma.

El producto fue realizado utilizando prácticas ágiles como TDD (desarrollo guiado por pruebas), integración continua y entrega continua. Así mismo, el desarrollo fue soportado por tecnologías modernas como Python 3.4, CouchDB, ES2015 y Docker.

Para la generación de los datos, se creó una interfaz web responsiva que puede ser accedida a través de cualquier navegador de un teléfono inteligente (http://quitoabierto.org/). En esta interfaz se puede visualizar el mapa de la zona con la ubicación actual del voluntario para facilitar el levantamiento de información.

La API pública de la aplicación fue creada específicamente para brindar acceso a desarrolladores que quieran tomar ventaja de los datos generados a través de la plataforma. Documentación de la API puede ser encontrada aquí: http://quitoabierto.org/api-docs/

La culminación de este primer MVP abre la puerta a crear nuevas funcionalidades y adaptaciones de la herramienta en el futuro.

ABSTRACT

Open data is the result of the cooperation between institutions and individuals that have published information to encourage researchers, developers and entrepreneurs to use and re-distribute the data. However, not every type of information is available in a open data format and that is the case with public transportation information in Quito, specifically about stops and routes.

This work will enable volunteers to generate and expose data about stops and routes in Quito. The result that this work proposes is an MVP (minimum viable product) that will allow evaluate the proper behaviour of the components of the platform.

This product was developed using agile practices such as TDD (Test driven development), continuous integration and continuous delivery. Also, the development was supported by modern technologies like Python 3.4, CouchDB, ES2015 and Docker.

To generate the data, a responsive web interface was created that can be accesed through any browser in a smartphone (http://quitoabierto.org/). A map of the area and the volunteer's current location can be seen from this interface so the data gathering could be done easily.

The public API was specifically created so that developers can have access to the data generated through the platform and take advantage of it. API documentation can be found here: http://quitoabierto.org/api-docs/

By finishing this first MVP, there is an open door for creating new functionalities and adaptations to the tool in the future.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Antecedentes

Datos abiertos

De acuerdo al Manual de Datos Abiertos, "son datos que pueden ser usados, reusados y redistribuidos libremente por cualquiera - sujeta únicamente, a lo sumo, a la obligación de atribuir y compartir por igual“ (Dietrich D. et al. sf. Open Data Handbook. Recuperado el 1 de Junio del 2015, de http://opendatahandbook.org/guide/en/what-is-open-data/). Esto significa que los datos deben estar disponibles y ser accesibles preferentemente a través de internet, de una manera conveniente y en la que puedan ser modificados fácilmente. Es necesario, explica el Manual, que los datos estén visibles, por ejemplo a través de una API(Application Program Interface) y recomienda que se enumeren los datasets abiertos en un catálogo central.

Asimismo, los datos abiertos deben ser publicados bajo términos de que permitan reutilizarlos y redistribuirlos, lo que incluye su integración con otros conjuntos de datos. Esto es posible a través de la utilización de un licenciamiento abierto 'explícito'.

Otra de sus características es que deben permitir un acceso y participación universal, esto quiere decir que los datos no deben ser restringidos para ningún sector, persona o grupo. Además, el Manual de Datos Abiertos indica que no es permitido limitar el uso de los datos para propósitos específicos, por ejemplo de salud o educación.

Historia de los datos abiertos

Fue en Diciembre del 2007 que varios pioneros de los datos abiertos se juntaron para definir lo que serían los principios de este movimiento que, según Luke Fretwell, empresario y escritor enfocado en la tecnología, inauguró "una

nueva era de innovación democrática y oportunidad económica.” (Fretwell L. 2014 parr. 1. A brief history of open data. FCW: The business of federal technology. Recuperado el 11 de Julio del 2015, de http://fcw.com/articles/2014/06/09/exec-tech-brief-history-of-open-data.aspx)
Los principios en los que se basa este movimiento dicen que los datos deben ser completos, primarios, oportunos, de fácil acceso, que puedan ser procesados por una máquina, sin discriminación, sin propietarios y libres de licencia.

A partir de ese momento, varios gobiernos han adoptado la iniciativa de datos abiertos y han lanzado plataformas para alentar a investigadores y emprendedores a tomar esta información con alto potencial para hacer nuevos descubrimientos y crear nuevas oportunidades.

Una de las formas de fomentar el uso de los datos abiertos ha sido a través de eventos como Hackathon. En Brasil se ha utilizado varias veces esta iniciativa para crear aplicaciones que puedan aprovechar el potencial de los datos.

Datos abiertos en Ecuador

Aunque aún son pocas, existen iniciativas en Ecuador que intentan hacer de los datos abiertos una realidad. Por ejemplo, el Municipio de Quito empezó una iniciativa de datos abiertos a través de una plataforma llamada Junar que hospeda los datos sobre distintas áreas (salud, transporte, ambiente, etc.) y se encarga de proporcionar acceso a los mismos.
Asimismo, el Gobierno está fomentando la iniciativa de datos abiertos en Ecuador, a través del portal web datosabiertos.ec. Este sitio ofrece información en formato CSV, para que pueda ser leída por una máquina.

Crowdsourcing

Para la puesta a disposición de los datos por parte de gobiernos o instituciones,

generalmente los datos ya se encuentran disponibles. Pero si se quiere generar o recolectar datos inexistentes se pueden utilizar técnicas como Crowdsourcing, la cual es “un tipo de actividad participativa en línea en la que un individuo, una institución, una organización sin fines de lucro o una compañía propone a un grupo de individuos con diferente conocimiento, heterogeneidad y número, a través de una llamada flexible y abierta, el compromiso voluntario de cumplir con una tarea”. (Estellés-Arolas E. y Gonzáles-Ladrón-de-Guevara F. 2012. p. 9)

Crowdsourcing combina la cooperación, agregación, trabajo en equipo, consensos y la creatividad de varias personas que buscan cumplir un mismo objetivo.

La comunidad puede trabajar de manera colaborativa para generar datos relevantes sobre un tema específico. Waze es una aplicación de navegación y mapas que ha generado gran parte de sus datos con la ayuda de sus usuarios. Lo mismo sucede con aplicaciones como Foursquare o Airbnb.

Se puede hacer uso de esta técnica colaborativa para recolectar información que podría ser expuesta a manera de datos abiertos.

Visualización

La visualización e interacción con los datos es lo que trae el verdadero valor a la información. Esto se puede evidenciar en el proyecto Ocorrências nas Rodavias Federais. Esta plataforma muestra, a través de gráficas y mapas de calor, los datos de accidentes de tránsito; como por ejemplo, los tipos de vehículos involucrados, la severidad de los daños y la frecuencia en la que este tipo de sucesos ocurrieron en Brasil. Esta información fue recolectada y publicada por el Ministerio de Justicia brasileño, el mismo que organizó una Hackathon para la construcción de aplicaciones que permiten una visualización fácil y amigable para todos los ciudadanos.

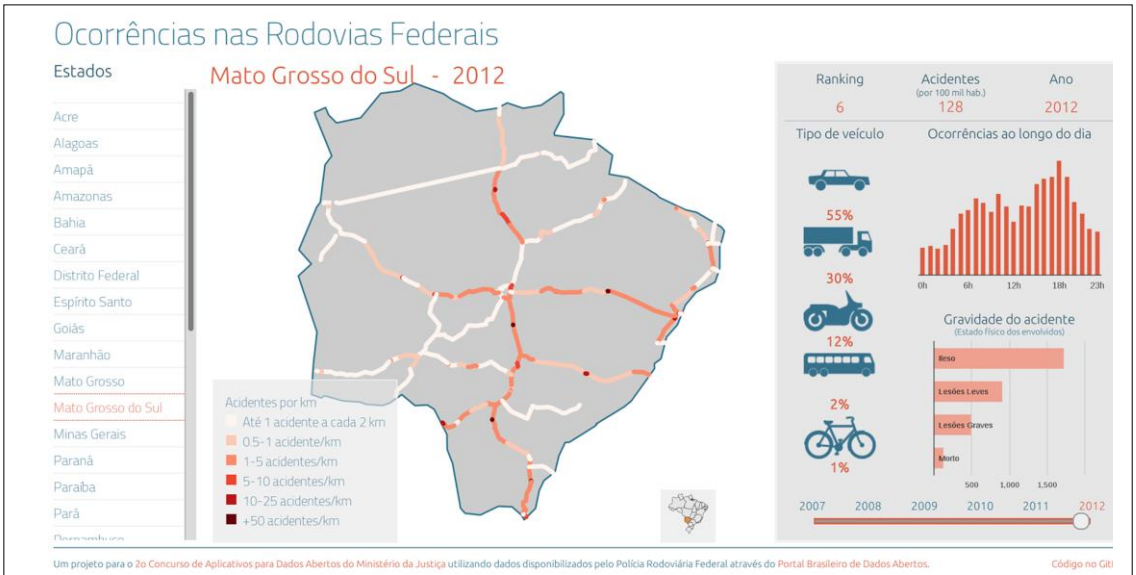


Figura 1: Visualización de accidentes de tránsito en Brasil

Tomado de: Analyticsm (2012). Mapa de Ocurrencias en las Carreteras Federales. Recuperado el 8 de Junio de 2016 de: http://analyticsmj.github.io/prf/

Alcance

La información de transportación pública es particularmente útil, especialmente para visitantes o para personas que desconocen cómo movilizarse por la ciudad. Es por eso que para la realización de este proyecto el enfoque es la generación de datos referentes a movilización, específicamente datos de paradas y rutas de varias líneas de buses de inicio a fin en la ciudad de Quito.

Para la recolección de los datos se implementará una aplicación web la cual utilizará la ubicación actual para mapear las paradas de las líneas de buses. Esta aplicación será utilizada por un grupo de voluntarios, quienes se encargarán de subir nueva información.

La aplicación no tendrá visualización de las rutas ya que su única responsabilidad es la generación de datos.

Se creará una RESTful API para que los datos puedan ser consumidos a través de HTTP y el formato de transferencia de los datos será JSON. La API estará debidamente documentada, con los verbos y datasets disponibles en un sitio web.

También se creará una aplicación web que demuestre cómo se pueden utilizar los datos obtenidos, en este caso se visualizarán las paradas más cercanas a

la posición actual del usuario.

Justificación

Los datos abiertos permiten la disponibilidad de un conocimiento, que finalmente puede ser compartido con el objetivo de beneficiar a las comunidades y los individuos. De acuerdo a Stefan Kulk y Bastiaan Van Loenen, de la Universidad de Tecnología de Delft, hay una creciente tendencia a publicar todo tipo de datos en internet. Tanto gobiernos nacionales como locales, empresas pertenecientes a distintas industrias, organizaciones no gubernamentales, entre otras, están exponiendo datos y permitiendo que individuos y compañías los reutilicen libremente para sus propios fines. Entre los ejemplos de uso de datos abiertos que estos autores mencionan están la generación estadísticas sobre crímenes, consumo de energía o permisos de construcción. (Kulk S. y Van Loenen B. 2012, p. 1 Brave New Open Data World?. Recuperado el 1 de Junio del 2015, de http://www.gsdi.org/gsdiconf/gsdi13/papers/138.pdf)

El Ecuador, en el 2014 se posicionó en el puesto #43 del ranking del Índice Mundial de Datos Abiertos (http://index.okfn.org/place/ecuador/), el cual estimó que el 44% de la información considerada como pública es de libre acceso. Sin embargo, la mayor parte no es legible por una máquina, es decir, los archivos que la contienen son digitales pero no pueden ser procesados, analizados o transformados fácilmente por una computadora. Se tratan de archivos en formato DOC o PDF, publicados en las páginas web de las instituciones públicas; o de geoportales y aplicaciones que, aunque permiten la visualización de los datos, no ofrecen un acceso directo a los mismos.

Los datos expuestos por el Municipio de Quito poseen varios problemas; entre ellos, la mala documentación, ya que no especifica, entre otras cosas, los datasets que se pueden consultar. Además, no garantiza disponibilidad ya que no está preparada para manejar cargas pesadas y es inestable.

En el caso de la información que ha sido puesta a disposición por el gobierno del Ecuador, si bien esta puede ser leída por una máquina, su utilidad es limitada porque depende de la interacción humana para descargarla y no permite hacer búsquedas inteligentes, dado que no es posible filtrar los datos antes de descargar un archivo. Por otro lado, la información aún es escasa; por ahora, solo están disponibles dos datasets de datos relacionados con el presupuesto del Estado.

Para que la ciudadanía sea capaz de crear sus propios datos, se propone construir una plataforma que facilite la generación de la información. El potencial de los datos solo se ve limitado por la creatividad de las personas para crear diferentes visualizaciones o indicadores.

Objetivos

Objetivo General

Implementar una plataforma para la recolección de información que contribuya con la disponibilidad y accesibilidad de datos abiertos relacionados al servicio de transporte público de buses, en la ciudad de Quito.

Objetivos Específicos

Configurar una plataforma para el almacenamiento de los datos en un servicio en la nube.

Construir una API REST para proporcionar el libre acceso de los datos almacenados.

Crear un sitio web para la evaluación del funcionamiento de la plataforma a través de la visualización de los datos de las paradas de las líneas de buses.

Verificar el correcto funcionamiento de la aplicación mediante pruebas automatizadas.

Escribir la documentación pertinente para que los usuarios conozcan la manera

de interactuar con la API.

IMPLEMENTACIÓN DE LA SOLUCIÓN

Este trabajo se originó a partir del planteamiento de la siguiente declaración, la cual expresa la visión del producto que se construyó:

Para comunidades y organizaciones que trabajan en proyectos de movilidad que no tienen acceso a datos de transporte público, Quito Abierto es una plataforma de datos abiertos, que permite levantar y exponer datos de las paradas y rutas de las líneas de buses en Quito.
A diferencia de Junar, la plataforma de datos abiertos del Municipio de Quito, este producto es de código abierto, modificable, de arquitectura escalable y de fácil despliegue.

El resultado de este trabajo se considera como el producto mínimo viable para satisfacer los objetivos planteados.

1. Prácticas metodológicas

Para el desarrollo de los componentes de la plataforma, se utilizó el agilismo como guía práctica y metodológica, con especial enfoque en la calidad, automatización e integración y entrega continua.

El agilismo no limita a las personas a quedarse con una metodología o con otra. En ThoughtWorks, empresa con más de 20 años de experiencia en desarrollo de software y pionera en agilismo, se recomienda tomar las prácticas de varias metodologías y adaptarlas a las realidades de los proyectos.

La selección de estas prácticas se basaron en la experiencia desarrollando proyectos en varios campos.
Entrega tan rápido como sea posible (Lean)
Para disminuir la incertidumbre que existe en los proyectos de software. Lean

recomienda hacer entregas tan rápido como sea posible. Esto permite recibir retroalimentación desde etapas tempranas del desarrollo y poder hacer cambios justo a tiempo.

Muro Kanban (Kanban)
El muro es una representación de un flujo continuo que hace fácil visualizar el estado de las tareas.

Historias de usuario (XP)
Las tareas o requerimientos están representadas como historias de usuario que permiten utilizar lenguaje común. Las historias son visibles a través de un muro virtual.

TDD (XP)
TDD permite escribir el código estrictamente necesario para cumplir con un caso de prueba. Al escribir las pruebas primero se cuenta con una gran cobertura de código y se garantiza el funcionamiento del software.

Integración continua (XP)
Es la práctica que permite a los desarrolladores integrar su código en un repositorio central, correr pruebas automatizadas y brindar retroalimentación en caso de que algo esté “roto”.

1.1 Muro Kanban en Trello

El muro kanban ayuda a visualizar las etapas por las que una historia de usuario puede pasar hasta ser completada. La herramienta Trello fue utilizada para crear un muro virtual que puede ser compartido con otras personas como los interesados del proyecto y otros desarrolladores.

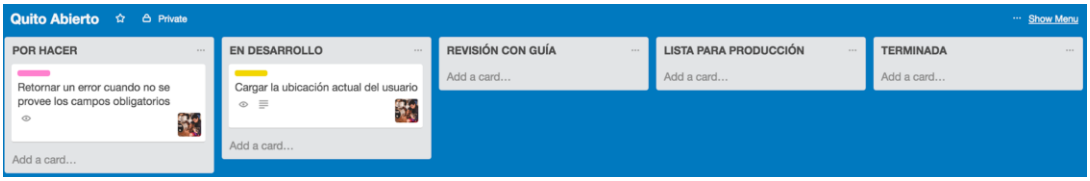


Figura 2: Muro virtual en Trello

Tomado de: Trello. (s.f). Recuperado el 8 de Junio de 2016 de: http://trello.com

Las columnas en el muro virtual son las siguientes:

Por hacer

Aquí están todas las historias de usuario listas para ser tomadas.

En desarrollo

En esta columna están las historias que actualmente están siendo desarrolladas.

Revisión con guía

Cuando el desarrollo termina, se hará una revisión junto con la profesora guía.

Lista para producción

Una vez que la guía haya dado su retroalimentación y las recomendaciones hayan sido aplicadas, las historias se considerarán aceptadas y en espera para ser desplegadas a producción.

Terminada

La historia se encuentra en el ambiente de producción.

1.2. Historias de Usuario

Durante el desarrollo de las aplicaciones se utilizó historias de usuario para reflejar los requerimientos técnicos y de negocio.

Las historias de usuario para el componente web de carga de datos tienen la etiqueta “carga-web” y son de color amarillo.

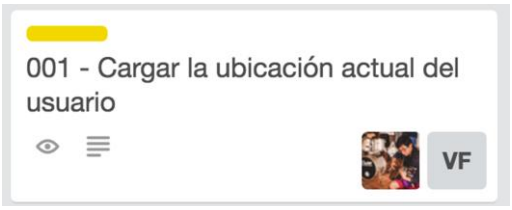


Figura 3: Historia de Usuario para la carga web

Tomado de: Trello. (s.f). Recuperado el 8 de Junio de 2016 de: http://trello.com

Las historias de usuario para la api de datos tienen la etiqueta “api-datos” y son de color rosado.

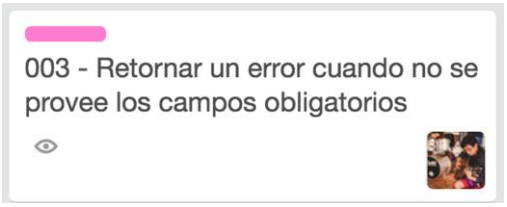


Figura 4: Historia de Usuario para la api de datos

Tomado de: Trello. (s.f). Recuperado el 8 de Junio de 2016 de: http://trello.com

Las historias o tareas técnicas pueden ser para todos los componentes, tiene la etiqueta “tarea técnica” y son de color azul.

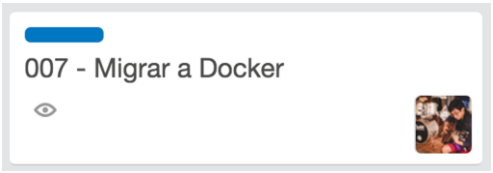


Figura 5: Historias técnicas

Tomado de: Trello. (s.f). Recuperado el 8 de Junio de 2016 de: http://trello.com

Las historias de usuario para la visualización tienen la etiqueta “visualización” y son de color verde.

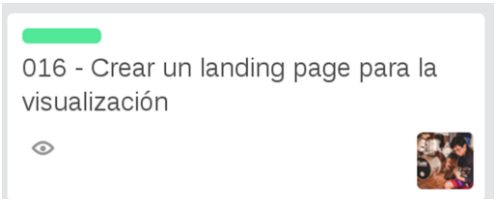


Figura 6: Historia de Usuario para la visualización
Tomado de: Trello. (s.f). Recuperado el 8 de Junio de 2016 de: http://trello.com

El muro virtual de Trello es público y puede ser visualizado en la siguiente dirección web**:** https://trello.com/b/Zs1qwNJY/quito-abierto

1.3. Principios de la Entrega Continua

Durante todo el proceso, la construcción de los componentes fueron apoyados por las prácticas de integración y entrega continua, las cuales permiten implementar y desplegar pequeñas partes del código en cortos periodos de tiempo.

Estos son los principios que guiaron el desarrollo:

- El proceso de despliegue debe ser repetible y confiable.
- Automatizar todo.
- Mantener todo en el sistema de control de versión.
- Finalizado significa desplegado.
- La calidad es parte del proceso.

1.3.1. Despliegues

Los despliegues a los ambientes se los hace utilizando la tubería de despliegue o *Deployment Pipeline*. El código es desplegado automáticamente al ambiente de prueba cuando se envían cambios al repositorio remoto, y llega al ambiente de producción con tan solo aplastar un botón.

1.3.2. Automatización

Los procesos que fueron automatizados durante el desarrollo son:

- Pruebas (unitarias, integración, funcionales, aceptación, etc.)
- Descarga de librerías para los diferentes lenguajes (pip, npm, bundle, etc.)
- Arquitectura (Docker)
- Despliegues (SnapCI)

1.3.3. Control de versión

Todo el código realizado se encuentra versionado utilizando Git y se encuentra disponible en los repositorios remotos de GitHub.

Los tres repositorios son públicos y cualquier persona puede contribuir en ellos.

API Datos: https://github.com/QuitoAbierto/api-datos
Carga Web: https://github.com/QuitoAbierto/carga-web
Visualización: https://github.com/QuitoAbierto/visualizacion

2. Herramientas y lenguajes utilizados

En las siguientes tablas se puede ver la lista completa de herramientas y lenguajes que se utilizó y además comparaciones con otras alternativas existentes en el mercado. Las comparaciones fueron realizadas utilizando los siguientes parámetros:

Tabla 1. Definición de parámetros para comparar herramientas

Parámetro	Porcentaje	Observación
Madurez de la herramienta	20%	Se evaluará la madurez de la herramienta basada en los años en el mercado y el nivel de documentación existente. **0-7%:** Grado de madurez bajo **8-15%:** Grado de madurez medio **16-20%:** Grado de madurez alto
Facilidad de aprendizaje	20%	Este parámetro representa que tan facil de aprender es la herramienta. **0-7%:** Facilidad de aprendizaje baja **8-15%:** Facilidad de aprendizaje media **16-20%:** Facilidad de aprendizaje alta
Experiencia	40%	En este parametro se coloca el porcentaje de recomendación de uso de la herramienta basado en la experiencia. **0-13%:** No se recomienda su uso **14-27%:** Se recomienda evaluar el uso de la herramienta **28-40%:** Se recomienda adoptar el uso de la herramienta
Mantenibilidad	20%	La mantenibilidad se evalua basado en la facilidad de la herramienta a soportar cambios y a la realización de pruebas. **0-7%:** Grado de mantenibilidad bajo **8-15%:** Grado de mantenibilidad medio **16-20%:** Grado de mantenibilidad alto
Total	100%	

2.1. Herramientas para el frontend

Lenguaje/Framework

En la siguiente tabla se realiza la comparación ente JavaScript (ES2015) frente al framework AngularJS.

Tabla 2. Comparación entre tecnologías del frontend

Parámetro	%	JavaScript (ES2015)	AngularJS
Madurez de la herramienta	20	15	20
Facilidad de aprendizaje	20	18	10
Experiencia	40	30	5
Mantenibilidad	20	20	15
Total	100	**83**	**50**

El resultado indica que ES2015 constituye una mejor opción para realizar este trabajo. La mala experiencia utilizando AngularJS en varios proyectos durante los últimos años lo descartan para ser utilizado en este desarrollo.

Mapas

En la siguiente tabla se realiza una comparación entre las herramientas para interactuar con los mapas en el frontend.

Tabla 3. Comparación entre herramientas para trabajar con mapas

Parámetro	%	Leaflet.js	OpenLayers
Madurez de la herramienta	20	15	20
Facilidad de aprendizaje	20	20	5
Experiencia	40	14	0
Mantenibilidad	20	18	15
Total	100	**67**	**40**

La decisión de elegir Leaflet.js por sobre OpenLayers se basó en simplicidad. Si bien es cierto que OpenLayers ofrece una mayor cantidad de funcionalidades, Leaflet.s es una alternativa mucho más liviana y más fácil de empezar a trabajar con ella.

2.2. Herramientas para el backend

Lenguaje/Framework

En la siguiente tabla se realiza una comparación entre Python y Ruby.

Tabla 4. Comparación entre lenguajes de programación

Parámetro	%	Python	Ruby
Madurez de la herramienta	20	20	20
Facilidad de aprendizaje	20	20	15
Experiencia	40	35	25
Mantenibilidad	20	20	20
Total	100	**95**	**80**

Al tener más experiencia con Python se considera una mejor opción para construir la API, sin embargo Ruby es una herramienta poderosa al escribir pruebas de aceptación por lo que también se consideró su uso en la construcción de la plataforma.

Framework para construir la API

En la siguiente tabla se comparan los frameworks disponibles para crear APIs.

Tabla 5. Comparación entre frameworks para Python

Parámetro	%	Flask	Django
Madurez de la herramienta	20	20	20
Facilidad de aprendizaje	20	20	15
Experiencia	40	35	10
Mantenibilidad	20	20	15
Total	100	**95**	**60**

Se cuenta con gran experiencia construyendo APIs con Flask, es por esto que el micro framework es una mejor opción.

Base de datos

En la siguiente tabla se ve la comparación entre las bases de datos CouchDB y PostgreSQL.

Tabla 6. Comparación entre bases de datos

Parámetro	%	CouchDB	PostgreSQL
Madurez de la herramienta	20	20	20
Facilidad de aprendizaje	20	18	10
Experiencia	40	25	10
Mantenibilidad	20	20	15
Total	100	**83**	**55**

Una base de datos no relacional se adapta de mejor manera para almacenar objetos complejos que son representaciones fieles del negocio y es por eso que se decidió utilizar CouchDB.

Infraestructura

En la siguiente tabla se comparan los servicios Heroku y DigitalOcean para alojar aplicaciones.

Tabla 7. Comparación entre servicios de alojamiento de aplicaciones

Parámetro	%	DigitalOcean	Heroku
Madurez de la herramienta	20	18	20
Facilidad de aprendizaje	20	20	10
Experiencia	40	35	20
Mantenibilidad	20	20	10
Total	100	**93**	**60**

Durante los últimos años se han realizado varios proyectos en ambos servicios pero la buena experiencia utilizando DigitalOcean lo hacen una mejor opción en este caso.

2.3. Herramientas para desarrollo de pruebas de aceptación

En la siguiente tabla se realiza una comparación entre las herramientas Cucumber y Lettuce para hacer pruebas de aceptación automatizadas.

Tabla 8. Comparación entre herramientas de pruebas de aceptación

Parámetro	%	Cucumber	Lettuce
Madurez de la herramienta	20	20	15
Facilidad de aprendizaje	20	15	15
Experiencia	40	35	30
Mantenibilidad	20	15	15
Total	100	**85**	**75**

Se decidió seleccionar Cucumber para poder utilizar Capybara y aprovechar la facilidad de interacción con el navegador web.

2.4. Otras herramientas

Otras herramientas que fueron utilizadas en el desarrollo y el despliegue de ls aplicaciones fueron las siguientes:

- Docker
- NGINX
- SnapCI

2.5. Método de persistencia

Para almacenar los datos se utilizó la base de datos documental NoSQL o no relacional CouchDB.
CouchDB ofrece almacenamiento de documentos de tipo JSON y no posee ningun esquema o colecciones.

Para almacenar datos de paradas y rutas se creó una propiedad en el documento a almacenar llamada *type.*

Un documento de tipo parada se almacena en la base de datos de la siguiente manera:

{
 "_id": "a5b271d05265073da8b1049120000c50",
 "_rev": "1-2079a9bdaf1d64e65acdac6f01debfce",
 "name": "Parada #2",
 "description": "Cerca de la carolina",
 "line": "Alborada"
 "type": "parada",
 "location": {
  "lat": -0.18353252018746405,
  "lng": -78.47709596157074
 }
} *

* GeoJSON omitido

Un documento de tipo ruta se almacena en la base de datos de la siguiente manera.

{
 "_id": "b5b271d0d2s5043sa8b10as120000e80",
 "_rev": "1-0d2s5043sa8b10as120000e8rr3466",
 "name": "Ruta #2",
 "description": "La ruta que va al sur",
 "type": "rotue",
 "location": {
  "lat": -0.18353252018746405,
  "lng": -78.47709596157074
 }
} *

* GeoJSON omitido

CouchDB utiliza dos propiedades del documento almacenado para poder identificarlo: _id y _rev.

_id es un número único que identifica al documento y _rev es un número único que identifica a la versión del documento.

Para realizar búsquedas en CouchDB se utilizan vistas escritas en JavaScript, esto difiere mucho de las consultas SQL que se realizan en las bases de datos relacionales.

Una vista para buscar todas las paradas en la base de datos se vería de la siguiente forma:

```
function(doc) {
  if(doc.type === 'parada') {
    emit(doc._id, doc);
  }
}
```

Esto retorna todos los documentos de la base de datos que tengan una propiedad type y que sea igual a “parada”.

3. Desarrollo de componentes

En esta sección se detalla el proceso de desarrollo de los componentes construidos durante la realizacioń del proyecto.

3.1. Componente web para carga de datos

Este componente es la interfaz mediante la cual los voluntarios ingresan los datos a la plataforma. Se encuentra construido con las tecnologías de front end descritas en la sección 2.1.

Para la construcción de las funcionalidades necesarias se utilizó la técnica BDD (desarrollo guiado por comportamiento) utilizando pruebas automatizadas de aceptación descritas en la sección 2.3.

3.1.1. Diagrama de la arquitectura

Este componente está desplegado en Digital Ocean por medio de contenedores de Docker y utiliza HTTP para enviar y recibir datos.

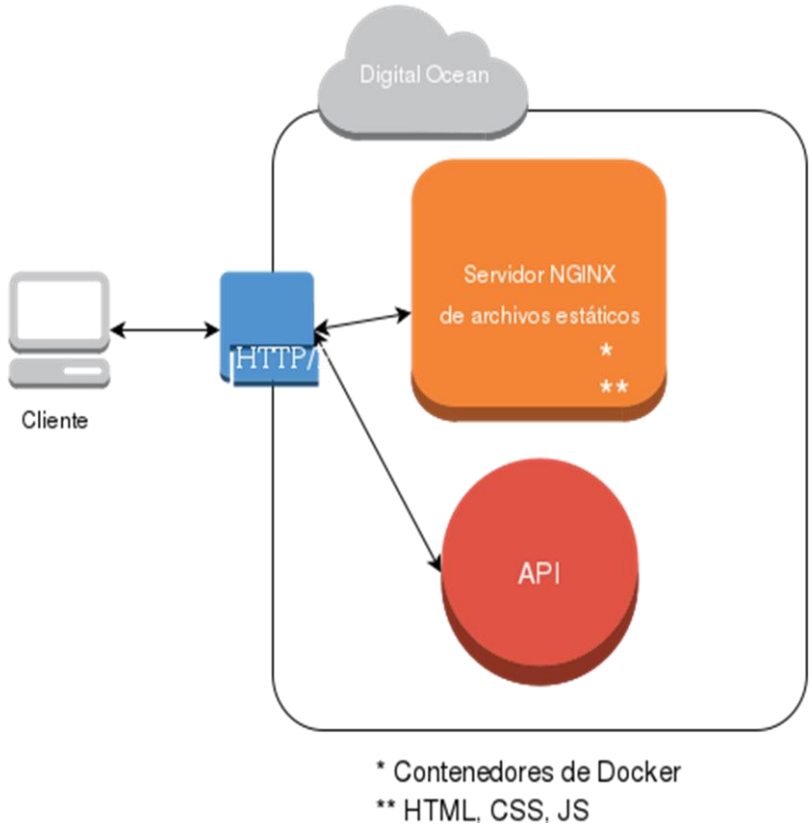


Figura 7: Arquitectura de la carga web

3.1.2. Ambientes

Esta aplicación cuenta con cuatro ambientes:

- Local o desarrollo
- Integración continua
- Ambiente de pruebas
- Ambiente de producción

Los ambientes de pruebas y de producción están alojados en máquinas en la nube utilizando el servicio Digital Ocean

3.2. API para exposición y carga de datos

Esta es la interfaz que permite almacenar y exponer los datos a través de HTTP. Para su construcción fueron utilizadas las herramientas de backend descritas en la sección 2.2.

El desarrollo guiado por pruebas o TDD fue utilizado durante la construcción de todas las funcionalidades del componente.

3.2.1. Diagrama de la arquitectura

Este componente al igual que el componente de carga web está desplegado en Digital Ocean por medio de contenedores de Docker. Sus principales componentes son Flask para construir la API y la base de datos CouchDB.

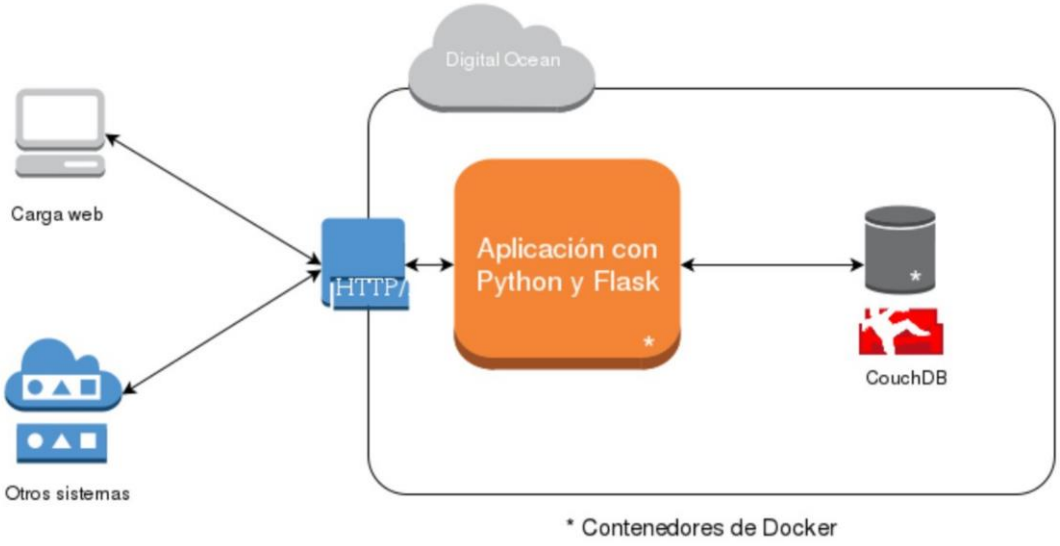


Figura 8. Diagrama de la arquitectura

3.2.2. Ambientes

La API cuenta con cuatro ambientes:

- Local o desarrollo
- Integración continua
- Ambiente de pruebas
- Ambiente de producción

Al igual que el componente web, los ambientes de pruebas y de producción están alojados en máquinas en la nube utilizando el servicio Digital Ocean

4. Proceso de despliegue

El proceso de despliegue se encuentra totalmete automatizado y puede ser visualizado a través de la tubería de despliegue.

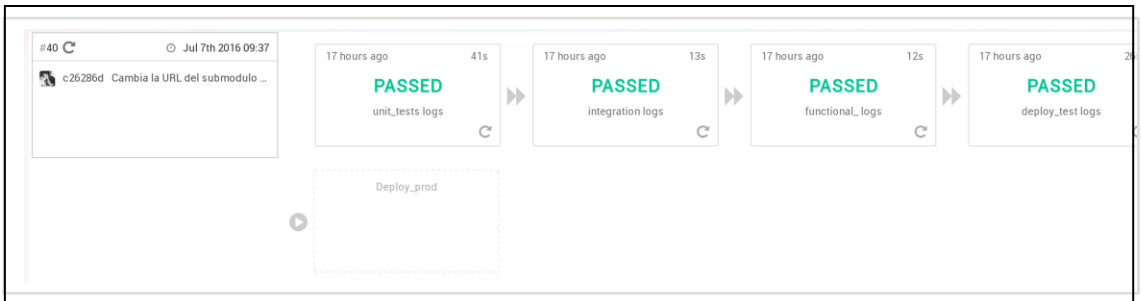


Figura 9. Proceso de despliegue

La tubería o *pipeline* de despliegue es uno de los principales habilitantes de la entrega continua.

Cada componente cuenta con su propia tubería de despliegue ya que cada uno tiene características diferentes, como los lenguajes en los que están escritos y los tipos de pruebas realizadas.

4.1. Tubería para la carga web

Esta tubería tiene los siguientes pasos:

Despliegue al ambiente de pruebas

Este paso es el punto de entrada y se ejecuta cada vez que se promueven cambios al repositorio remoto. Este paso realiza el despliegue hacia el ambiente de pruebas en Digital Ocean, un proveedor de infraestructura en la nube.

Pruebas E2E (de punto a punto)

En este paso automático se corren las pruebas E2E que se ejecutan en el ambiente de pruebas.

Despliegue al ambiente de producción
A diferencia de los dos pasos previos, este es un paso manual, es decir, requiere únicamente aplastar un botón para poder desplegar los cambios al ambiente de producción en Digital Ocean.

4.2. Tubería para la API de datos

Esta tubería tiene los siguientes pasos:

Pruebas de integración
Este es el punto de entrada y se ejecuta cada vez que se promueven cambios al repositorio remoto. Aquí se ejecutan las pruebas de integración en el ambiente local de SnapCI.

Pruebas funcionales
En este paso automático se ejecutan las pruebas funcionales en el ambiente local de SnapCI.

Despliegue al ambiente de pruebas
Este paso despliega automáticamente los cambios al ambiente de pruebas en Digital Ocean.

Despliegue al ambiente de producción
De igual manera que en el pipeline de la carga web, este es un paso manual. Requiere únicamente aplastar un botón para poder desplegar los cambios al ambiente de producción en Digital Ocean.

5. Pruebas

Durante todo el desarrollo se utilizó la técnica TDD (desarrollo guiado por pruebas). Kent Beck manifiesta que “seguir TDD religiosamente debería dar como resultado una cobertura de sentencias del 100 porciento” (Beck K. 2003. p. 52) por lo que se garantiza una alta cobertura en el código.

Todas las pruebas están automatizadas y se ejecutan varias veces al día en la máquina de desarrollo y en el ambiente de integración continua.

5.1. Proceso de pruebas

Para entender el proceso en general, es necesario saber cómo escribir una prueba automatizada y cuáles son las partes que la componen para posteriormente poder ser ejecutada y saber cuál es su propósito con tan solo una mirada al código.

Las partes principales de una prueba común en este trabajo son:

- Título
- Preparación
- Acción
- Aserción

Título
El título de la prueba detalla cuál es el objetivo de la prueba, describiendo muchas veces la acción a ejecutar y el resultado esperado.

Preparación
En la preparación se prepara el ambiente o se crean los actores necesarios para la prueba. Puede ser la creación de la clase que se quiere probar o visitar una de las páginas de la plataforma.

Acción

Es aquí en donde se ejecuta la acción que está siendo probada. Puede ser la llamada a un método con ciertos parámetros o hacer click en un botón.

Aserción

En este punto se compara el resultado obtenido con el resultado esperado y determina el éxito la falla de la prueba.

5.1.1 Pruebas en Python

Las pruebas en Python están escritas con el *framework* **unittest** y las herramientas de **nosetests** y en algunos casos utilizan el cliente de pruebas de **Flask** para facilitar las llamadas HTTP.

```
1. # Título de la prueba
2. def test_returns_error_when_required_fields_are_not_provided(self):
3.     # Preparación
4.     self.test_app = app.test_client()
5.     invalid_stop = {'invalid_key': 'value'}
6.
7.     # Acción
8.     response = self.test_app.post('/api/parada',
9.         data=json.dumps(invalid_stop))
10.    error = json.loads(response.data.decode("utf-8"))['error']
11.
12.    # Aserción
13.    assert_equal(400, response.status_code)
14.    assert_equal('Parada mal formateada', error)
```

En el ejemplo anterior se puede ver claramente todas las partes de la prueba. En la línea 2 se encuentra el título de la prueba que en este caso explica cuál es el comportamiento esperado dados ciertos parámetros. En las líneas 4 y 5 se evidencia la preparación, en este caso crear el cliente de pruebas de Flask y una variable representando una parada inválida. En la línea 8 se ejecuta la acción a ser probada, en este caso es hacer una llamada HTTP de tipo POST al repositorio de paradas con una parada inválida. En las líneas 13 y 14 se puede observar la aserción, que en este caso espera una respuesta con código 400 desde el servidor y un mensaje diciendo “Parada mal formateada”.

Al ejecutar esta prueba por primera vez se podrá observar cómo el código no cumple con el comportamiento esperado:

```
$ invoke test

======================================================
FAIL: test_returns_error_when_required_fields_are_not_provided
------------------------------------------------------
AssertionError: 400 != 404
------------------------------------------------------
Ran 1 test in 0.101s
FAILED (failures=1)
```

Una vez implementado el código para poder pasar esta prueba se obtendrá un resultado favorable al volver a ejecutarla.

```
$ invoke test

test_returns_error_when_required_fields_are_not_provided
.
------------------------------------------------------
Ran 1 test in 0.091s
OK
```

5.1.2. Pruebas de aceptación en Gherkin y Ruby

Para estas pruebas se utilizó el proceso BDD (desarrollo guiado por comportamiento) para lo cual fue necesario integrar herramientas diseñadas para este propósito:

- Cucumber
- Capybara
- Gherkin
- Ruby
- Minitest

Las pruebas tienen 2 componentes importantes:

- Archivo de caracteristica o *feature*
- Archivo de pasos o *steps*

Archivo de característica

El archivo de caracteristica describe cuáles son los escenarios que se desean probar y cuáles son los pasos a seguir para reproducir cada escenario. Está escrito en el lenguaje Gherkin el cual soporta varios idiomas por lo que fue posible escribir los escenarios en español. Estos archivos son considerados documentación viva y describen el comportamiento de cada funcionalidad.

```
1. Escenario: Guardar una parada
2.   Dado que ingreso a la aplicación
3.   Cuando estoy en la página de mapeo de paradas
4.   Y selecciono una ubicación
5.   Y lleno el formulario con los siguientes datos:
6.     | línea    | nombre             | descripción        |
7.     | Alborada | Parada la carolina | Cerca de la tribuna |
8.   Y envío el formulario
9.   Entonces veo el mensaje “Parada guardada exitosamente”
```

Archivo de pasos

El archivo de pasos está escrito en un lenguaje de programación, en este caso Ruby y su función es interpretar lo que el archivo de caracteristica describe y convertir los pasos descritos en acciones ejecutables a través de código.

```
1.  Cuando(/^selecciono una ubicación$/) do
2.    find('#my-map').click
3.    find('#next-button').click
4.  end
5.
6.  Cuando(/^lleno el formulario con los siguientes datos:$/) do |table|
7.    data = table.hashes[0]
8.    select data['linea'], :from => 'line-field'
9.    fill_in 'name-field', :with => data['nombre']
10.   fill_in 'description-field', :with => data['descripción']
11. end
12.
13. Cuando(/^envío el formulario$/) do
14.   click_link('submit')
15. end
```

En este caso el archivo de pasos interactúa con el navegador web automatizando acciones como hacer click en botones, llenar formularios y navegar a diferentes páginas.

La lista completa de pruebas de aceptación se encuentra en el Anexo 3.

5.2. Reportes de pruebas

Al ser ejecutadas, las pruebas generan reportes que permiten visualizar los fallos, errores, omisiones, pruebas exitosas y la cobertura obtenida.

Tabla 9. Información general sobre pruebas ejecutadas

Información general

Class	Fail	Error	Skip	Success	Total
test.functional.test_api.TestApi	0	0	0	9	9
test.integration.test_route_repository.TestRouteRepository	0	0	0	3	3
test.integration.test_route_service.TestRouteService	0	0	0	3	3
test.integration.test_stop_repository.TestStopRepository	0	0	0	5	5
Total	0	0	0	20	20

Figura 10. Detalle de pruebas ejecutadas

Tabla 10. Detalle de cobertura

Coverage report: 99%

Module ↓	*statements*	*missing*	*excluded*	*coverage*
app.py	0	0	0	100%
app/errors.py	2	0	0	100%
app/route_repository.py	12	0	0	100%
app/route_service.py	21	0	0	100%
app/scripts.py	0	0	0	100%
app/scripts/get_db.py	11	1	0	91%
app/stop_repository.py	13	0	0	100%
app/stop_service.py	27	0	0	100%
Total	**86**	**1**	**0**	**99%**

6. Funcionalidades construidas

Las lista completa de funcionalidades de los componentes están descritas en las historias de usuario (Anexo 2). En esta sección se detallan las funcionalidades críticas de la plataforma.

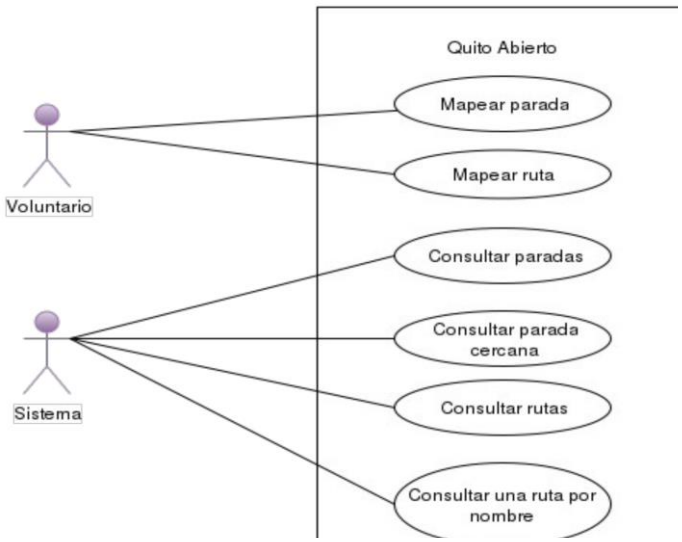


Figura 11: Diagrama de casos de uso

Para poder ver de manera rápida las funcionalidades con sus respectivos actores se puede tomar en cuenta el siguiente diagrama de casos de uso.

Sobre los casos de uso

En la metodología ágil los casos de uso son una herramienta opcional que pueden ayudar a dar claridad en la realización de grandes sistemas. En este caso al ser una aplicación pequeña se decidió únicamente utilizar historias de usuario.

6.1. Funcionalidades de la carga web

Estas son las funcionalidades que permiten el levantamiento de los datos de paradas y rutas.

Para ver la aplicación en vivo se debe ingresar al sitio web http://quitoabierto.org.

Figura 12: Pantalla principal de la aplicación.
Tomado de: Quito Abierto. (s,f). Recuperado el 12 de Junio de 2016 de: http://quitoabierto.org

Aquí se puede seleccionar una de las dos opciones presentadas:

- Mapear paradas
- Mapear rutas

6.1.1. Mapear una parada

Para mapear una parada se debe escoger la opción de “Mapear Paradas” de la pantalla principal (Figura 12).

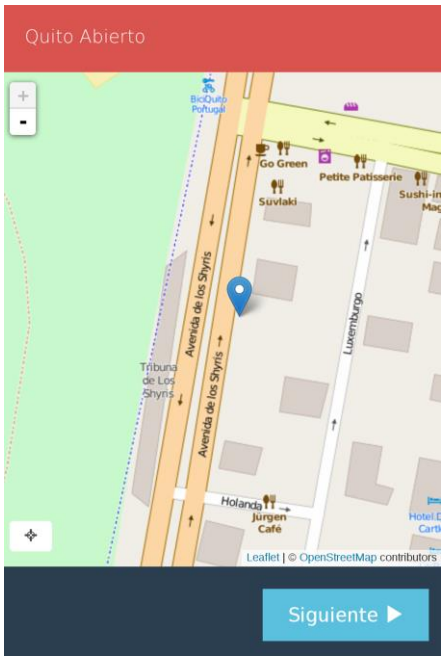


Figura 13: Pantalla de selección de ubicación de la parada.
Tomado de: Quito Abierto. (s,f). Recuperado el 12 de Junio de 2016 de: http://quitoabierto.org

Al seleccionar esta opción se presentará una pantalla en la cual se debe seleccionar la ubicación de la parada que se quiere levantar (Figura 13).

Al aplastar el botón “Siguiente” se presenta una nueva pantalla (Figura 14) en la cual se debe especificar información básica de la parada:

- Línea de bus
- Nombre
- Descripción

Una vez completada la información de la parada y al aplastar el botón “Guardar” se presenta una pantalla de confirmación (Figura 15).

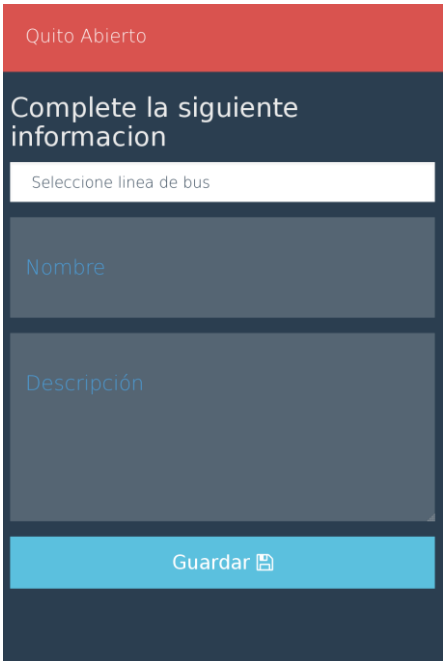


Figura 14: Pantalla de información básica de la parada.
Tomado de: Quito Abierto. (s,f). Recuperado el 12 de Junio de 2016 de: http://quitoabierto.org

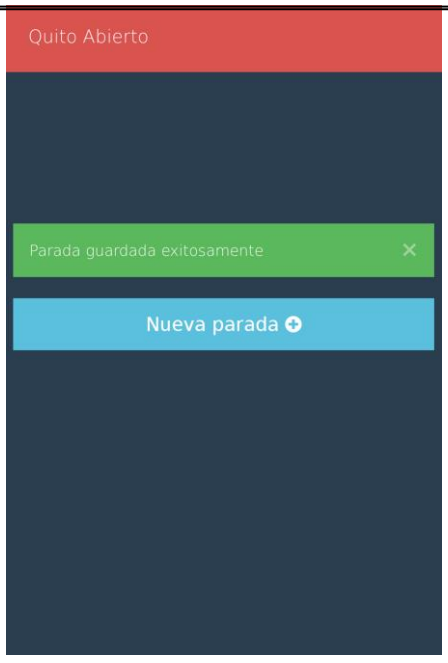


Figura 15: Pantalla de confirmación.
Tomado de: Quito Abierto. (s,f). Recuperado el 12 de Junio de 2016 de: http://quitoabierto.org

6.1.2. Mapear una ruta

Para mapear una parada se debe escoger la opción de “Mapear Rutas” de la pantalla principal (Figura 12).

Al seleccionar esta opción se presentará una pantalla en la cual se debe proveer la información básica para empezar a mapear una ruta (Figura 16).

- Nombre
- Descripción

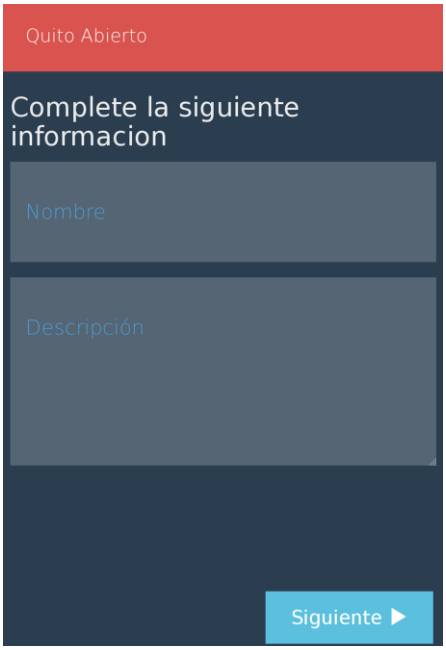


Figura 16: Pantalla de información básica para mapear una ruta.
Tomado de: Quito Abierto. (s,f). Recuperado el 12 de Junio de 2016 de: http://quitoabierto.org

Al aplastar el botón “Siguiente” se presenta una nueva pantalla (Figura 17) en la cual se cargará automáticamente la ubicación actual. Para guardar los puntos de la ruta se debe aplastar el botón “Empezar”.

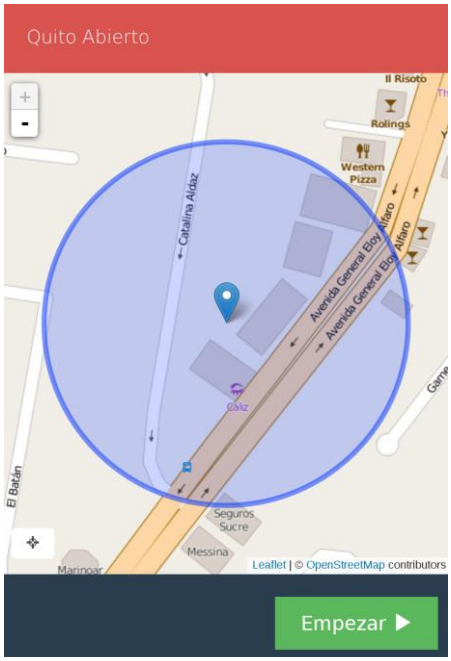


Figura 17: Pantalla de mapeo de puntos de ruta.
Tomado de: Quito Abierto. (s,f). Recuperado el 12 de Junio de 2016 de: http://quitoabierto.org

6.2. Funcionalidades de la API

Además de las funcionalidades construidas para el correcto funcionamiento de la carga web, la API posee otras funcionalidades que son públicas y están destinadas a ser usadas por cualquier persona o sistema. El acceso es a través de HTTP y la transferencia de los datos se lo hace con el formato JSON.

6.2.1. Obtener información de paradas

Para poder acceder a la información de paradas se debe mandar una petición al servidor con el método GET y hacia la ruta /api/parada.

```
$ curl -XGET http://quitoabierto.org:5000/api/parada
```

La respuesta a esta petición es una lista de paradas que además de la información básica llevan en su estructura un objeto geoJSON.

```
[{
   "line": "Alborada",
   "name": "Parada #54",
   "description": "Una descripcion",
   "type": "parada",
   "geoJSON": {
      "type": "Feature",
      "geometry": {
         "coordinates": [-0.183599038596892, -78.48205268383026],
         "type": "Point"
      },
      … // propiedades omitidas
   },
   … // propiedades omitidas
}]
```

6.2.2. Obtener información de la parada más cercana

Este *endpoint* se utiliza para obtener la parada más cercana a las coordenadas que se proveen.

```
$ curl -XPOST http://quitoabierto.org:5000/api/parada/cercana
 -d '{"location": {"lat": 100, "lng": 100}}'
```

La respuesta es la información de la parada más cercana que se pudo encontrar.

```
{
        "stop":{
        "name":"Parada #32",
        "line":"Alborada",
        "description":"Una parada cerca de la carolina",
        "type":"parada",
        "geoJSON":{
        "type":"Feature",
        "geometry":{
        "coordinates":[
                -78.48271787166595,
                -0.18859650410379172
        ],
        "type":"Point"
        },
        … // propiedades omitidas
        },
        … // propiedades omitidas
        }
}
```

6.2.3. Obtener información de rutas

Este endpoint retorna todas las rutas almacenadas en la base de datos.

```
$ curl -XGET http://quitoabierto.org:5000/api/route
```

La respuesta es la lista de rutas disponibles en el sistema.


```
[
   {"locations": [[1, 1], [2, 2]], "name": "ruta1"},
   {"locations": [[5, 5], [6, 6]], "name": "ruta2"}
]
```


6.2.4. Obtener información de una ruta

A través de este endpoint se pueden obtener todas las rutas almacenadas en la base de datos.

```
$ curl -XGET http://test.quitoabierto.org:5000/api/route/ruta35
```

La respuesta es la información básica de la ruta y todos los puntos asociados a ella.


```
{
        "route":{
        "name":"ruta35",
        "description": "La ruta que va al sur",
        "locations":[
        {
                "lng":-78.4771829,
                "lat":-0.18443016
        },
        {
                "lng":-78.47720136,
                "lat":-0.18439812
        }
        ]
        }
}
```

7. Conclusiones y recomendaciones

7.1. Conclusiones

Esta plataforma es un primer prototipo que ayudará a la recolección de datos de transporte público en Quito. Se considera concluido el primer MVP que habilita a los voluntarios a realizar el levantamiento de información de paradas y rutas.

El uso de TDD permitió alcanzar un porcentaje de 99% de líneas de código cubiertas con pruebas y al ser automatizadas brindan confianza al hacer cambios en el futuro

El uso de Docker permitió mimetizar la configuración entre los ambientes de desarrollo, integración continua, QA y producción por lo que se puede tener certeza que los componentes se comportarán de la misma manera independientemente del ambiente en el que estén desplegados.

La documentación, el proceso automatizado de aprovisionamiento de ambientes y la alta cobertura de código harán que agregar nueva funcionalidad y cambios en las ya existentes, sea un proceso confiable.

7.2. Recomendaciones

Se recomienda socializar la plataforma con las comunidades de tecnología en Quito y próximamente en Ecuador en búsqueda de retroalimentación para seguir creando funcionalidades y agregar mejoras.

El uso de entrega continua permitió desplegar cambios al ambiente de producción desde los primeros días de desarrollo, por lo que se recomienda su uso en cualquier proyecto de software.

Se recomienda el uso de contenedores tanto para desarrollo como para despliegue de servicios. Los contenedores aseguran que el código se comportará exactamente de la misma manera en todos los ambientes.

Es recomendable el uso de herramientas livianas como Flask y Leaflet.js para construir aplicaciones ya que ofrecen una manera rápida de empezar el desarrollo y una gran flexibilidad en el futuro.

Se recomienda explorar las posibilidades de uso de la plataforma para otro tipo de medio de transporte como aquellos que movilizan a personas a las zonas rurales de la ciudad de Quito.

REFERENCIAS

Open Data Handbook (2014). ¿Qué son los datos abiertos? Recuperado el 1 de Junio de 2016, de: http://opendatahandbook.org/guide/en/what-is-open-data/

Brabham, D.C. (2013). Crowdsourcing: Series de Conocimiento Esencial (p.1). Massachusetts, Estados Unidos: MIT Press.

Kulk, S. y Van Loenen, B. (2012) ¿Nuevo Mundo Valiente de Datos Abiertos? (p.1) Recuperado el 1 de Junio de 2016, de: http://www.gsdi.org/gsdiconf/gsdi13/papers/138.pdf

Fry, B. (2008). Visualizando los Datos: Explorando y Explicando los Datos con el Ambiente Procesado. (p.18). Estados Unidos: O'REILLY

Global Open Data Index. (2014). Índice de Datos Globales Abiertos por Conocimiento. Obtenido el 4 de junio de 2016, de: http://index.okfn.org/place/ecuador

FCW: The Business Of Federal Technology. (2014). Una Breve Historia de los Datos Abiertos. Recuperado el 11 de Julio de 2016, de: http://fcw.com/articles/2014/06/09/exec-tech-brief-history-of-open-data.aspx

Rasmusson, J. (2010). The Agile Samurai. (p.17). Estados Unidos: Pragmatic Bookshelf.

Estellés-Arolas E. y Gonzáles-Ladrón-de-Guevara F. (2012). Hacia una definición integrada del crowdsourcing. (p.9). Estados Unidos: Journal of Information Science.

Beck, K. (2003). Desarrollo Conducido con Pruebas por Ejemplo. (p.52). Boston, Estados Unidos: Addison-Wesley.

ANEXOS

Anexo 1: Glosario Técnico

Docker: Es una herramienta para crear e interactuar con contenedores.

Contenedor de docker: Los contenedores envuelven un pedazo de Software con todo lo que necesita para correr: código, herramientas, librerías, etc.

Digital Ocean: Es un servicio en línea que ofrece máquinas en la nube para desarrolladores.

Heroku: Es una plataforma como servicio (PaaS) que ofrece alojamiento de aplicaciones.

TravisCI: Es un servicio en línea que permite realizar integración continua.

SnapCI: Es un servicio en línea que permite realizar integración y entrega continua.

Trello: Es un servicio en línea que permite crear muros virtuales utilizando el paradigma Kanban.

NoSQL: Es un término para identificar a las bases de datos que no son relacionales y que permiten almacenar la información de maneras diferentes a las tablas en las bases de datos relacionales.

Micro framework: Se refiere a herramientas minimalistas para construir aplicaciones web.

MVP: Producto mínimo viable.

Anexo 2: Historias de usuario

Historias terminadas

001 - Cargar la ubicación actual del usuario

002 - Creación de ambientes para carga-web (e2e, qa, prod)

003 - Retornar un error cuando no se provee los campos obligatorios

004 - Creación de ambientes para api-datos (e2e, qa, prod)

005 - Agregar un boton para encontrar la ubicacion actual del usuario

006 - Persistir el recurso creado

007 - Migrar a Docker

008 - Mostrar mensaje de confirmación al guardar datosS

010 - No permitir enviar el formulario si no se proveen los campos obligatorios

011 - Crear endpoint para acceder a todas las paradas

012 - Automatizar la creación de las imágenes en digital ocean

013 - Almacenar ubicacion seleccionada

014 - Permitir seleccionar línea de bus

015 - Investigar herramienta para formato de pruebas de aceptación

016 - Crear un landing page para la visualización

017 - Retornar la parada más cercana dada una coordenada

018 - Mostrar todas las paradas

019 - Permitir crear una nueva parada después de guardar

020 - Guardar informacion como GeoJSON

022 - Permitir mapear una ruta

023 - Crear endpoint para almacenar una ruta

024 - Crear landing page para documentacion de la API pública

026 - Crear endpoint para obtener datos de todas las rutas

Historias no terminadas

021 - Mostrar únicamente las paradas de la línea seleccionada

025 - Mostrar la ruta de una línea

Historias fuera del alcance

009 - Hacer que el formulario sea personalizable

Anexo 3: Pruebas de aceptación

Archivo carga_de_paradas.feature

Característica: Mapeo de paradas

Escenario: Formulario de datos

Dado que ingreso a la aplicación

Cuando estoy en la página de mapeo de "paradas"

Entonces veo el mapa para seleccionar paradas

Escenario: Guardar una parada

Dado que ingreso a la aplicación

Cuando estoy en la página de mapeo de "paradas"

Y selecciono una ubicación

Y lleno el formulario con los siguientes datos:

linea	nombre	descripción
Alborada	Parada la carolina	Esta es una parada de la linea #5

Y envío el formulario

Entonces veo el mensaje "Parada guardada exitosamente"

Escenario: El voluntario no provee información

Dado que ingreso a la aplicación

Cuando estoy en la página de mapeo de "paradas"

Y selecciono una ubicación

Y envío el formulario

Entonces veo el mensaje "Todos los campos son obligatorios"

Escenario: El voluntario quiere guardar una nueva parada

Dado que ingreso a la aplicación

Cuando estoy en la página de mapeo de "paradas"
Y cargo la información de una parada
Y envío el formulario
Entonces veo la opción de crear otra parada
Cuando quiero crear una nueva parada
Entonces veo el mapa para seleccionar paradas

Característica: Mapeo de rutas
Escenario: Formulario de datos
Dado que ingreso a la aplicación
Cuando estoy en la página de mapeo de "rutas"
Entonces veo el formulario para cargar los datos

Escenario: El voluntario no provee información
Dado que ingreso a la aplicación
Cuando estoy en la página de mapeo de "rutas"
Y hago click en siguiente
Entonces veo el mensaje "Todos los campos son obligatorios"

Escenario: Guardar una ruta
Dado que ingreso a la aplicación
Cuando estoy en la página de mapeo de "rutas"
Y lleno el formulario de ruta con los siguientes datos:

nombre	descripción
Parada la carolina	Esta es una parada de la linea #5

Y hago click en siguiente
Y hago click en empezar
Entonces veo el mensaje "Mapeando ruta..."
Cuando hago click en finalizar
Entonces veo el mensaje "Ruta guardada exitosamente"

Archivo de pasos

```
Dado(/^que ingreso a la aplicación$/) do
  visit '/'
end

Cuando(/^estoy en la página de mapeo de "([^"]*)"$/) do |page|
  page_map = {
    paradas: 'stop',
    rutas: 'route'
  }
  button_id = "##{page_map[page.to_sym]}-mapping-button"
  find(button_id).click
end

Entonces(/^veo el mapa para seleccionar paradas$/) do
  find '#my-map'
end

Entonces(/^veo el formulario para cargar los datos$/) do
  find '.fields-container'
end

Cuando(/^selecciono una ubicación$/) do
  find('#my-map').click
  find('#next-button').click
end

Cuando(/^lleno el formulario con los siguientes datos:$/) do |table|
  data = table.hashes[0]
  select data['linea'], :from => 'line-field'
  fill_in 'name-field', :with => data['nombre']
```

```
  fill_in 'description-field', :with => data['descripción']
end

Cuando(/^envío el formulario$/) do
  click_link('submit')
end

Cuando(/^hago click en siguiente$/) do
  next_button = find '#next-button'
  next_button.click
end

Entonces(/^veo el mensaje "([^"]*)"$/) do |message|
  wait_for_ajax
  message_box = find '#message-box'
  assert message_box.has_content?(message), 'Message not shown!'
end

Cuando(/^cargo la información de una parada$/) do
  find('#my-map').click
  find('#next-button').click
  select 'Alborada', :from => 'line-field'
  fill_in 'name-field', :with => 'default_name'
  fill_in 'description-field', :with => 'default_description'
end

Entonces(/^veo la opción de crear otra parada$/) do
  wait_for_ajax
  new_stop_button = find '#new-stop'
  assert new_stop_button.has_content?('Nueva parada'), 'New Stop Button not
present'
end
```

```
Cuando(/^quiero crear una nueva parada$/) do
  new_stop_button = find '#new-stop'
  new_stop_button.click
end

Cuando(/^lleno el formulario de ruta con los siguientes datos:$/) do |table|
  data = table.hashes[0]
  fill_in 'name-field', :with => data['nombre']
  fill_in 'description-field', :with => data['descripción']
end

Cuando(/^hago click en empezar$/) do
  next_button = find '#next-button'
  next_button.click
end

Cuando(/^hago click en finalizar$/) do
  next_button = find '#finish-button'
  next_button.click
end
```

Anexo 4: Manual de usuario

El texto a continuación describe las funcionalidades del sistema y constituye una herramienta de ayuda para que los usuarios de la plataforma puedan interactuar con la misma.

1. Carga de datos a través de la aplicación web

El componente para hacer la carga de datos es una aplicación web construida con tecnologías de *front end* y permitirá a los voluntarios levantar información de paradas y rutas.

Para ver la aplicación en vivo se debe ingresar al sitio web http://quitoabierto.org.

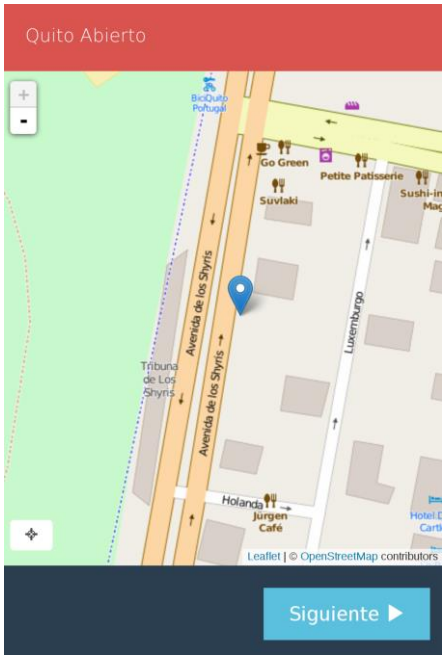
Aquí se puede seleccionar una de las dos opciones presentadas:

- Mapear paradas
- Mapear rutas

1.1. Mapear una parada

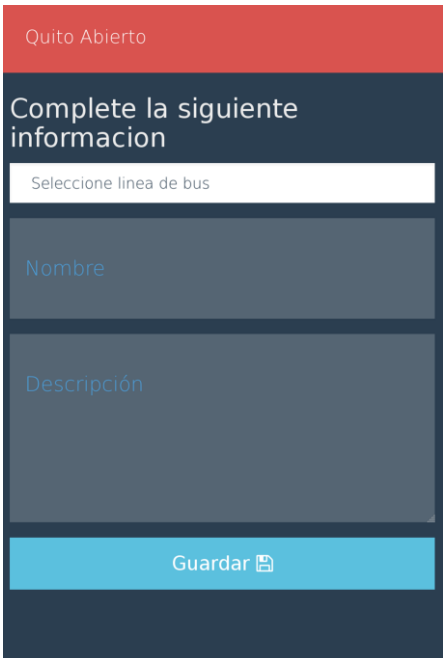
Para mapear una parada se debe escoger la opción de “Mapear Paradas” de la pantalla principal.

Al seleccionar esta opción se presentará una pantalla en la cual se debe seleccionar la ubicación de la parada que se quiere levantar.

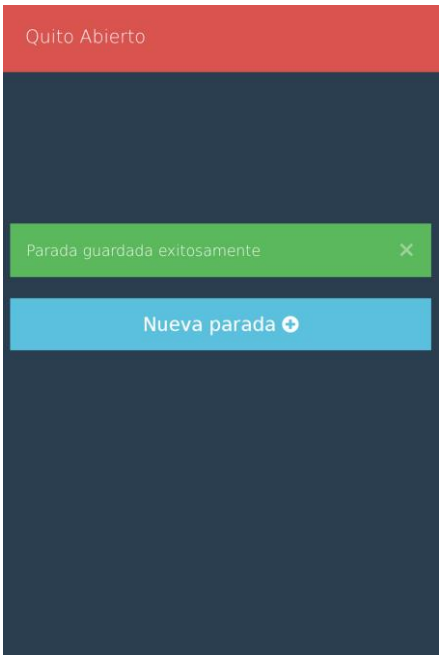


Al aplastar el botón “Siguiente” se presenta una nueva pantalla (Figura 14) en la cual se debe especificar información básica de la parada:

- Línea de bus
- Nombre
- Descripción



Una vez completada la información de la parada y al aplastar el botón “Guardar” se presenta una pantalla de confirmación (Figura 15).

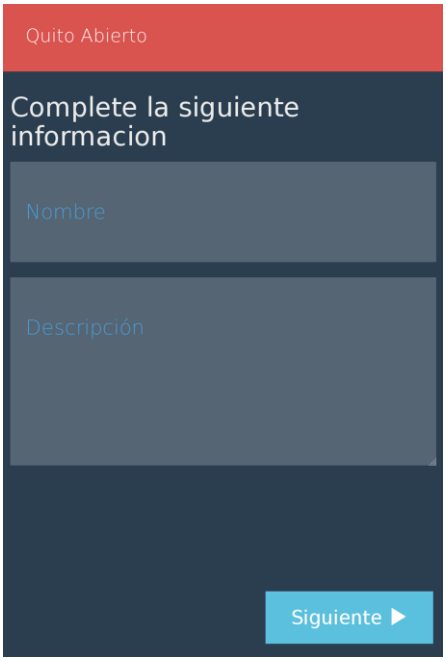


1.2. Mapear una ruta

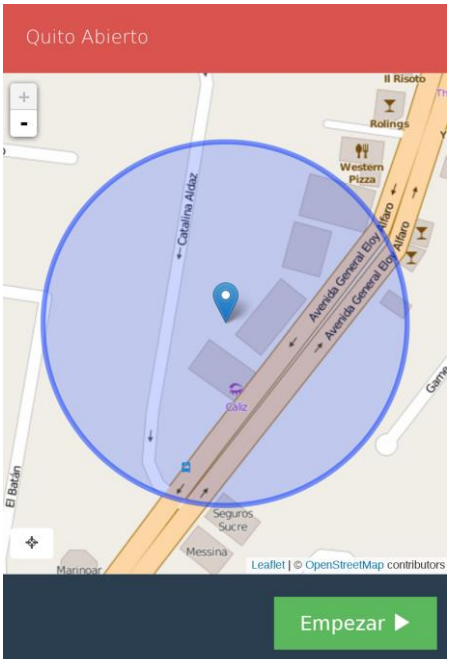
Para mapear una parada se debe escoger la opción de “Mapear Rutas” de la pantalla principal (Figura 12).

Al seleccionar esta opción se presentará una pantalla en la cual se debe proveer la información básica para empezar a mapear una ruta (Figura 16).

- Nombre
- Descripción



Al aplastar el botón “Siguiente” se presenta una nueva pantalla (Figura 17) en la cual se cargará automáticamente la ubicación actual. Para guardar los puntos de la ruta se debe aplastar el botón “Empezar”.



2. Funcionalidades de la API

Además de las funcionalidades construidas para el correcto funcionamiento de la carga web, la API posee otras funcionalidades que son públicas y están destinadas a ser usadas por cualquier persona o sistema. El acceso es a través de HTTP y la transferencia de los datos se lo hace con el formato JSON.

2.1. Obtener información de paradas

Para poder acceder a la información de paradas se debe mandar una petición al servidor con el método GET y hacia la ruta /api/parada.

```
$ curl -XGET http://quitoabierto.org:5000/api/parada
```

La respuesta a esta petición es una lista de paradas que además de la información básica llevan en su estructura un objeto geoJSON.

```
[{
   "line": "Alborada",
   "name": "Parada #54",
   "description": "Una descripcion",
   "type": "parada",
   "geoJSON": {
      "type": "Feature",
      "geometry": {
         "coordinates": [-0.183599038596892, -78.48205268383026],
         "type": "Point"
      },
      … // propiedades omitidas
   },
   … // propiedades omitidas
}]
```


2.2. Obtener información de la parada más cercana

Este *endpoint* se utiliza para obtener la parada más cercana a las coordenadas que se proveen.

```
$ curl -XPOST http://quitoabierto.org:5000/api/parada/cercana
 -d '{"location": {"lat": 100, "lng": 100}}'
```

La respuesta es la información de la parada más cercana que se pudo encontrar.


```
{
        "stop":{
        "name":"Parada #32",
        "line":"Alborada",
        "description":"Una parada cerca de la carolina",
        "type":"parada",
        "geoJSON":{
        "type":"Feature",
        "geometry":{
        "coordinates":[
                -78.48271787166595,
                -0.18859650410379172
        ],
        "type":"Point"
        },
        … // propiedades omitidas
        },
        … // propiedades omitidas
        }
}
```


2.3. Obtener información de rutas

Este endpoint retorna todas las rutas almacenadas en la base de datos.

```
$ curl -XGET http://quitoabierto.org:5000/api/route
```

La respuesta es la lista de rutas disponibles en el sistema.


```
[
   {"locations": [[1, 1], [2, 2]], "name": "ruta1"},
   {"locations": [[5, 5], [6, 6]], "name": "ruta2"}
]
```


2.4. Obtener información de una ruta

A través de este endpoint se pueden obtener todas las rutas almacenadas en la base de datos.

```
$ curl -XGET http://test.quitoabierto.org:5000/api/route/ruta35
```

La respuesta es la información básica de la ruta y todos los puntos asociados a ella.


```
{
	"route":{
	"name":"ruta35",
	"description": "La ruta que va al sur",
	"locations":[
	{
		"lng":-78.4771829,
		"lat":-0.18443016
	},
	{
		"lng":-78.47720136,
		"lat":-0.18439812
	}
	]
	}
}
```


Anexo 5: Manual técnico

La información descrita a continuación ayudará a más desarrolladores a colaborar con el proyecto, presentando guías de cómo levantar el ambiente local y además contiene requerimientos y asunciones sobre el conocimiento básico del programador.

1. Requerimientos

Los únicos requerimientos para poder colaborar en el proyecto es tener instalado Docker y docker-compose ya que todos los servicios como base de datos, servidor web y servidor de aplicación están configurados como contenedores de Docker.

2. Conocimiento básico

Durante todo el documento se asume que el programador tiene conocimientos sobre las siguientes herramientas y tecnologías:

- Docker
- Docker Compose
- Python
- Flask
- Ruby
- Cucumber
- JavaScript
- Git

3. Preparar el ambiente local

Una vez instalados los requerimientos se puede descargar los dos repositorios utilizando Git:

```
$ git clone https://github.com/QuitoAbierto/api-datos.git
$ git clone https://github.com/QuitoAbierto/carga-web.git
```

4. Desarrollando en la API de datos

Para desarrollar funcionalidades en la API de datos se deben escribir pruebas unitarias, de integración y funcionales que aboguen por el buen funcionamiento del código.

4.1. Herramientas

Este servicio está construido con Python y Flask por lo que se debe tener un conocimiento medio sobre esas herramientas.

La lista completa de herramientas y frameworks utilizados es la siguiente:

- Python
- Flask
- Gunicorn
- Nosetests
- Invoke
- CoucDB

4.2. Levantando la base de datos y el servidor de aplicaciones de Python

Para entender lo que sucede al levantar el componente con docker compose se puede ver el archivo **docker-compose.yml** el cual contiene la configuración de los servicios.

```
1  version: '2'
2  services:
3    db:
4      image: couchdb
5      volumes:
6        - /var/db/:/usr/local/var/lib/couchdb
7      ports:
8        - "5984:5984"
9
10   api:
11     build: .
12     ports:
13       - "5000:5000"
14     links:
15       - db
16
```

De la línea 3 a la línea 8 del archivo se especifica la configuración del servicio de base de datos, se utilizará una imagen con el nombre couchdb, esta es la imagen oficial provista por CochDB, se creará un volumen para que los datos sean persistentes y se expondrá el puerto 5984 para que exista comunicación

hacia el contenedor.
De la línea 10 a la línea 15 se especifica las características del servicio de API. A diferencia de la base de datos, este servicio utiliza un archivo **Dockerfile** para configurar la imagen. Aquí también se expone un puerto para que exista comunicación hacia el contenedor y se crea un enlace hacia el servicio de base de datos.

```
1  FROM python:3.4
2  WORKDIR /code
3  COPY requirements.txt /code/
4  COPY config/gunicorn.py /code/config/
5  RUN pip install -r requirements.txt
6  CMD gunicorn api:app -c config/gunicorn.py
7  COPY . /code
8
```

En la línea 1 se especifica la imagen base que va a ser utilizada para construir la imagen del servicio API con la instrucción **FROM**, en este caso la imagen base es la imagen oficial de Python en la versión 3.4.
En la línea 2 se menciona el directorio de trabajo dentro de la imagen con la instrucción **WORKDIR**.
Las líneas 3, 4, 7 copian los archivos necesarios hacia la imagen para poder levantar el servicio, entre ellos están archivos de configuración y el código de la API con la instrucción **COPY**.
En la línea 5 se instalan los requerimientos de la aplicación usando la instrucción **RUN**.
Finalmente la línea más importante es la 6, en ella se especifica cual es el comando que el contenedor va a correr y se lo hace a través de la instrucción **CMD**. En este caso se levanta un servidor Gunicorn.

Para iniciar la API debemos entrar al directorio y levantar los servicios utilizando docker-compose:

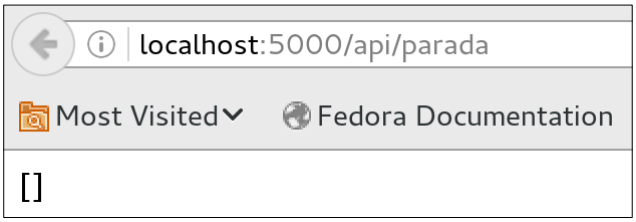
```
$ cd api-datos
$ docker-compose build
```

```
$ docker-compose up
```

Al hacer docker-compose build, docker descarga la imagen de CouchDB e inicia un contenedor con el servicio de base de datos a partir de la configuración especificada en el archivo docker-compose.yml, descarga la imagen base de Python 3.4, la configura de acuerdo al archivo Dockerfile e inicia un contenedor con el servicio de API a partir de la configuración especificada en el archivo docker-compose.yml. Este último contenedor es capaz de enlazarse al contenedor con la base de datos gracias a la configuración en el archivo docker-compose.yml.

Al hacer docker-compose up es cuando realmente se crean los contenedores a partir de las imágenes que configuramos anteriormente.

Para comprobar el correcto funcionamiento de la API se puede visitar la siguiente dirección en el navegador web: http://localhost:5000/api/parada.



Esta ruta devuelve todas las paradas almacenadas en la base de datos. Al ser esta la primera vez que iniciamos la API, retornará una lista vacía.
En otro terminal podemos confirmar que la base de datos y el servidor están corriendo utilizando el comando docker ps.

```
$ docker ps
```

Este comando muestra los contenedores que están corriendo actualmente, en este caso mostrará dos contenedores.

Para iniciar el componente en modo de desarrollo se debe utilizar el archivo

docker-compose.dev.yml.

En este pequeño archivo de configuración se crea un volumen de datos en el cual el código estará siempre actualizado en el contenedor activo haciendo que el desarrollo sea transparente y rápido.

El único paso para iniciar la aplicación en modo de desarrollo es ejecutar docker compose con este archivo como parámetro.

```
$ docker-compose -f docker-compose.yml -f docker-compose.dev.yml up
```

Ahora cualquier cambio en el código se verá reflejado en el contenedor de manera inmediata.

5. Desarrollando en la carga web

Este componente está totalmente construido con tecnologías de front end por lo que se debe tener un conocimiento medio sobre este tipo de herramientas.

5.1. Herramientas

Para realizar pruebas se utiliza Ruby como lenguaje y Cucumber con Capybara y Minitest para hacer aserciones.

La lista completa de herramientas y frameworks utilizados es la siguiente:

- HTML

- CSS3
- JavaScript (ES2015)
- LeafletJS
- Gulp
- Webpack

5.2 Levantando el servidor de archivos estáticos

Al igual que en la API, este componente utiliza docker compose para levantar el servidor. Se puede ver la configuración en el archivo **docker-compose.yml**.

```
version: '2'
services:
  web:
    build: public
    ports:
      - "80:80"

```

Este es un archivo de configuración más sencillo que utiliza un archivo **Dockerfile** dentro del directorio public para construir una imagen y además expone el puerto 80.

```
FROM nginx
COPY . /usr/share/nginx/html

```

En este archivo se define que la imagen base que va a ser utilizada será nginx con la instrucción **FROM** y se copian los archivos estáticos hacia el directorio en donde serán publicados por el servidor utilizando la instrucción **COPY**.

Para iniciar el componente web debemos entrar al directorio y levantar el servicio utilizando docker-compose:

```
$ cd carga-web
$ docker-compose build
$ docker-compose up
```

Al hacer docker-compose build, docker descarga la imagen de NGINX, copia los archivos estáticos y expone el puerto 80 de la imagen.

Al hacer docker-compose up se crea un contenedor capaz de servir archivos estáticos.

Para comprobar que todo funcione correctamente se puede visitar la dirección http://localhost

Para iniciar el componente en modo de desarrollo se debe utilizar el archivo docker-compose.dev.yml.

```yaml
version: '2'
services:
  web:
    volumes:
      - ./public:/usr/share/nginx/html

```

Al utilizar este archivo de configuración los archivos estáticos son copiados a un volumen de datos para que los cambios realizados se vean reflejados inmediatamente en el contenedor.

Anexo 6: CD del código