UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERIA Y ARQUITECTURA

ESCUELA DE INGENIERIA ELÉCTRICA

Servicio web de datos de consumo de energía del campus central de la Universidad de El Salvador.

PRESENTADO POR:

MAURICIO ALEJANDRO FLORES BERARDI

PARA OPTAR AL TITULO DE:

INGENIERO ELECTRICISTA

CIUDAD UNIVERSITARIA, ABRIL 2014

UNIVERSIDAD DE EL SALVADOR

RECTOR :

ING. MARIO ROBERTO NIETO LOVO

SECRETARIA GENERAL :

DRA. ANA LETICIA ZAVALETA DE AMAYA

FACULTAD DE INGENIERIA Y ARQUITECTURA

DECANO :

ING. FRANCISCO ANTONIO ALARCÓN SANDOVAL

SECRETARIO :

ING. JULIO ALBERTO PORTILLO

ESCUELA DE INGENIERIA ELÉCTRICA

DIRECTOR :

ING.JOSÉ WILBER CALDERÓN URRUTIA

UNIVERSIDAD DE EL SALVADOR

FACULTAD DE INGENIERIA Y ARQUITECTURA

ESCUELA DE INGENIERIA ELECTRICA

Trabajo de Graduación previo a la opción al Grado de:

INGENIERO ELÉCTRICISTA

Título :

Servicio web de datos de consumo de energía del campus central de la Universidad de El Salvador.

Presentado por :

MAURICIO ALEJANDRO FLORES BERARDI

Trabajo de Graduación Aprobado por:

Docente Director :

Dr. CARLOS EUGENIO MARTÍNEZ CRUZ

San Salvador, Abril 2014

Trabajo de Graduación Aprobado por:

Docente Director :

Dr. CARLOS EUGENIO MARTÍNEZ CRUZ

ACTA DE CONSTANCIA DE NOTA Y DEFENSA FINAL

En esta fecha, 27 de marzo de 2014, en la Sala de Lectura de la Escuela de Ingeniería Eléctrica, a las 3:00 horas, en presencia de las siguientes autoridades de la Escuela de Ingeniería Eléctrica de la Universidad de El Salvador:

1. Ing. José Wilber Calderón Urrutia
   Director — Firma: _____

2. Ing. Salvador de Jesús German
   Secretario — Firma: _____

Y, con el Honorable Jurado de Evaluación integrado por las personas siguientes:

1- Ing. Walter Leopoldo Zelaya Chicas — Firma: _____

2- Ing. Omar Otoniel Flores Cortez — Firma: _____

Se efectuó la defensa final reglamentaria del Trabajo de Graduación:

Servicio web de datos de consumo de energía del campus central de la Universidad de El Salvador.

A cargo del Bachiller:

- Flores Berardi, Mauricio Alejandro

Habiendo obtenido en el presente Trabajo una nota promedio de la defensa final, de: 9.0

(NUEVE PUNTO CERO)

Agradecimientos

A Dios

Por llevarme en sus brazos en los momentos más difíciles.

A mi familia

En especial a mi madre y a mi abuelita por su infinito amor y paciencia.

Al personal de la Escuela de Ingeniería Eléctrica en especial al Doc. Carlos Martínez por su apoyo.

A los amigos y compañeros de la escuela de Ingeniería Eléctrica.

Y a mis seres queridos que ya no están entre nosotros, pero si en mi corazón.

Contenido

Índice de Tablas:

Indice de Figuras:

Indice de códigos.

1 CAPITULO I

1.1 INTRODUCCION

Desde el mes de junio del año 2012, en el campus central de la universidad de El Salvador, se lleva a cabo un proyecto de recolección de datos, en 30 subestaciones eléctricas. Dicha información es proporcionada por medidores que poseen características de conectividad LAN.Toda esta información es recolectada en una base de datos. En algunos casos pueden llegar a ser 68 campos guardados por minuto, por cada subestación. Esto genera una gran cantidad de información, que puede ser utilizada para mejorar el rendimiento de la red eléctrica actual, detectar fallas en la red, mejorar el diseño de la red, individualizar valores de consumo y contribuir con la formación académica de los estudiantes de la universidad.

Actualmente la Universidad de El Salvador no posee una herramienta que sirva para lograr una adecuada interpretación de los datos. Por ello y ante el auge de las técnicas de programación y montaje de servidores web cloudcomputing o computación en la nube, se planteó la creación de un Servicio web de datos de consumo de energía del campus central de la Universidad de El Salvador.

Este trabajo de graduación introduce el concepto de programación agil, base para el desarrollo de software como servicio. Esta idea común en los entornos de desarrollo de software resulta, asombrosamente, ajena a los métodos que se enseñan en la FIA.

1.2 INTERÉS Y MOTIVACIÓN DEL TRABAJO DE GRADUACIÓN

Desde el año 2010 la universidad de California, San Diego (UC San Diego), posee un sistema de monitoreo de energía, llamado EnergyDashboard (http://energy.ucsd.edu/index.html). Este sistema surgió a iniciativa del laboratorio de sistemas microelectrónicos embebidos, ("MESL" por sus siglas en ingles). Donde existe un despliegue de equipos de medición, en todos los edificios de la universidad, generando datos en tiempo real, que han servido para hacer estudios del comportamiento energético de la universidad y de cada uno de los edificios.[1]

La Universidad de El Salvador posee una red de medidores, que al igual que en la universidad de San Diego, genera datos del comportamiento eléctrico, pero no posee ninguna herramienta que sirva para hacer análisis y estudios de dichos datos.

El principal interés de este trabajo de graduación es utilizar aprovechando el desarrollo de sistemas similares al desplegado por la universidad de San Diego, utilizando tecnologías alternativas de computación, y crear una herramienta útil en el campo de la ingeniería eléctrica.

Las capacidades de un país en aprovechar las tecnologías de la información y comunicación, son un claro indicador de desarrollo tecnológico, educacional y económico [2]. La computación en la nube, es una herramienta atractiva para muchas empresas de la industria informática. Es por ello que un país en desarrollo como el nuestro, debe aprender a aprovechar esta tecnología y hacer más productivas sus infraestructuras informáticas.

1.3 OBJETIVOS DEL TRABAJO DE GRADUACIÓN

Objetivo general:

- Crear un servicio web de visualización de la información generada por la red de medidores de energía eléctrica de la UES.

Objetivos específicos:

- Desarrollar aplicación de visualización de datos en un entorno *cloudcomputing*
- Aplicar técnicas en desarrollo de software como servicio
- Introducir conceptos de programación ágil
- Continuar con los desarrollos iniciados en la Escuela de Ingeniería Eléctrica bajo las plataformas Ruby onRails.

1.4 ORGANIZACIÓN.

Este trabajo de graduación se divide en 4 capítulos. En el primer capítulo, se presenta los intereses y motivación del proyecto, los objetivos del trabajo de graduación y la organización del documento.

En el segundo capítulo, introducimos los conceptos básicos de cloudcomputing, el desarrollo ágil de software (Agile Web Development), rubyonrails y la plataforma de servicio Heroku, además de describir la manera de hacer un despliegue completo en Heroku.

El tercer capítulo, es una descripción del procedimiento interactivo de la creación del servicio web. Se describe el estado de la red de medidores y de la base de datos spud, se documentan los primeros procedimientos para la creación del servicio web, la conexión con la base de datos, la creación de las tablas, con controladores y vistas, y la descripción final del servicio web.

Finalmente en el quinto capítulo se muestran las conclusiones y líneas futuras para este trabajo de graduación.

2 CAPITULO II

2.1 COMPUTACIÓN EN LA NUBE O CLOUD COMPUTING.

La computación en nube es toda una estructura computacional basada en Internet con centros de datos remotos que gestionan servicios de información y aplicaciones.[3]Esto permite que usuarios, empresas, instituciones utilicen aplicaciones de software sin necesidad de instalar programas en sus terminales. Pudiendo ser utilizadas en cualquier computadora con acceso a Internet.

Esta nueva tecnología es mucho más eficiente en el manejo de recursos, como almacenamiento memoria, procesamiento y ancho de banda, porque provee los servicios solamente en el momento de ser necesarios. Estas características permiten que cualquier aplicación para cualquier tipo de finalidad, posea los recursos necesarios para ser desplegada, aumentando la competitividad de instituciones y usuarios que posean recursos limitados en el desarrollo de un sistema utilitario.

Características:

Rápido despliegue: Los servicios en la nube, ahorran el proceso de adquisición de hardware y la instalación de infraestructura, ahorrando costos, tiempo y recursos humanos, cualidad que le hace atractiva para empresas con recursos limitados/escasos.

Soporte de software confiable: La mayoría de los proveedores mantienen actualizado su software y cualquier eventualidad es superada con rapidez, ya que no es necesario un mantenimiento terminal por terminal.

Elasticidad: se adapta rápidamente a negocios que crecen rápidamente o que posean picos estacionales, ya que está diseñada para manejar fuertes aumentos en la carga de trabajo. Sirviendo los recursos necesarios en el instante.

Movilidad/accesibilidad: La computación en la nube está diseñada para ser utilizada a distancia, así que el usuario tendrá acceso a los sistemas en cualquier lugar donde se encuentre.

Económica: Sepaga solamente por lo que realmente se utiliza, eliminando así gastos en infraestructura innecesaria.

2.1.1 La estructura general para representar su arquitectura.

- Software como servicio: es la capa más externa y es la que interactúa con los clientes/usuarios.
- Plataforma como servicio: la capa media, es el medio donde está operando el software a nivel virtual (servidor basado en alguna plataforma de desarrollo).
- Infraestructura como servicio: la capa inferior y es un medio de entregar almacenamiento básico y capacidades de cómputo como servicios en la red. Es la parte física de la estructura de servidores, sistemas de almacenamiento, conexiones, enrutadores, y otros sistemas.

Ejemplos de aplicaciones:

- Dropbox - desarrollado por Dropbox
- Google Drive - desarrollado por Google
- iCloud - desarrollado por Apple
- SkyDrive - desarrollado por Microsoft
- Ubuntu One - desarrollado por Linux Ubuntu

2.1.2 Software como un servicio:

Software como Servicio (del inglés: **Software as a Service**, **SaaS**): Es un modelo de software que rompe la tradición y la necesidad de instalar un programa determinado en una PC, donde cada PC gasta capacidad de procesamiento en utilizar dicho software, tanto el código fuente como los datos almacenados, son administrados y alojados por compañías llamadas compañíasde tecnologías de información y comunicación o TIC. Por lo general el software es utilizado por medio de una navegador web, a través de internet.

Ventajas

- Acceso y administración a través de una red o internet.
- Todas actividades de administración se realizan desde ubicaciones centrales.
- Actualizaciones centralizadas (en el servidor donde está alojado el software).
- No es necesario que el cliente cuente con un área especializada de soporte técnico para el sistema.
- La responsabilidad de la operación recae en la empresa de tecnologías de información.
- La empresa TIC provee los medios seguros de acceso en los entornos de la aplicación.
- La política de compra de licencias queda obsoleta y la forma de contrato del servicio es por medio del pago de alquiler o renta por el uso del software.
- El sistema operativo es indiferente.

Inconvenientes

- El usuario no tiene acceso al programa o código fuente. (dependiendo de la modalidad del contrato de servicios).
- El servicio requiere internet o en algunos casos una red local.

2.1.3 Agile Web Development:

Es una filosofía de programación que da mucha importancia al desarrollo de software con métodos de programación agiles que permitan un rápido desarrollo y una rápida evolución de los sistemas creados, de tal manera que el producto final es altamente eficiente, y adaptado a la comodidad del usuario[4].

El método por excelencia es el método interactivo, una iteración es un ciclo de vida del proyecto e incluye: la planificación, el análisis de requerimientos, el diseño, la codificación, la revisión y

documentación. Una iteración no debe agregar demasiada funcionalidad, pero la meta es tener una «versión preliminar» (sin errores) al final de cada iteración. Al final de cada iteración el equipo vuelve a evaluar las prioridades del proyecto.

Los métodos ágiles también enfatizan que el software funcional es la primera medida del progreso. Combinado con la preferencia por las comunicaciones cara a cara, generalmente los métodos ágiles son criticados y tratados como "indisciplinados" por la falta de documentación técnica.

2.1.4 Ruby onRails:

Ruby es un lenguaje de programación interpretado, reflexivo y orientado a objetos, distribuido bajo una licencia de software libre, creado por el programador japonés Yukihiro "Matz"Matsumoto, quien lo presentó públicamente en 1995. Combina una sintaxis inspirada en Python y Perl. Comparte también funcionalidad con otros lenguajes de programación[5].

Rails es un framework de aplicaciones web que incluye todo lo necesario para crear aplicaciones web de acuerdo con el Modelo-Vista-Controlador (MVC).Entender el patrón MVC es la clave para la comprensión Ruby onRails. MVC divide su aplicación en tres capas, cada una con una función específica[6].

- La capa de la Vista: se compone de "plantillas" que son responsables de proporcionar representaciones apropiadas de los recursos de la aplicación. Las plantillas pueden venir en una variedad de formatos, pero la mayoría de las plantillas de vista son HTML.

- El modelo: representa el modelo de dominio (por ejemplo, Cuenta, producto, persona, Post) y encapsula la lógica que es específica para la aplicación. En Rails, las bases de datos son respaldadas por las clases del modelo que se derivan de ActiveRecord:: Base. Active Record permite presentar los datos de las filas de base de datos como objetos y embellecer estos objetos de datos, con los métodos de la lógica del negocio. Aunque la mayoría de los modelos de Rails están respaldados por una base de datos, los modelos también pueden ser clases ordinarias de Ruby, o clases de Ruby que implementan un conjunto de interfaces que proporciona el módulo ActiveModel.

- El Controlador es responsable de manejar las peticiones HTTP y dar una respuesta adecuada. Normalmente esto significa volver HTML, pero los controladores Rails también puede generar XML, JSON, PDFs, vistas específicas para móviles etc. Controladores para manipular modelos y hacer plantillas de vista con el fin de generar la adecuada respuesta HTTP.

Gemas.

Las gemas son códigos añadidos (librerías en otros lenguajes de programación)que pueden ser utilizados paraincorporar funcionalidades específicas o herramientas para el desarrollo de las aplicaciones, como servidores, gestores de bases de datos, etc.

En el capítulo siguiente tomaremos a fondo el uso de rubyonrails en el despliegue de aplicaciones.

PostgreSQL como gestor de bases de datos*[7].*

PostgreSQL inició como un proyecto de la universidad de california en Berkeley en 1982, es un sistema gestor de bases de datos orientado a objetos con licencia BSD (*Berkeley Software Distribution*)que es una licencia de software libre permisiva, proporcionada por la Universidad de Berkeley.

Existe una amplia documentación en la web sobre la utilización de postgreSQL en proyectos de software como servicio. Esto lo hace actractivo para muchos desarrolladores, a tal punto de ser el gestor de bases de datos por defecto de Heroku (plataforma de servicio).

Es completamente compartible con rubyonrails y posee alto desempeño en el manejo de grandes cantidades de datos. La sintaxis es casi idéntica a mysql con la diferencia que posee características superiores en el manejo bases de datos extensas.

2.2 HEROKU COMO PLATAFORMA DE SERVICIO.

Heroku es una de las primeras plataformas de servicio en la nube, fue desarrollada en el año 2007, al inicio solo soportaba el lenguaje de programación ruby ahora se ha extendido el soporte a lenguaje java, node.js, scala, clojure, python y php.Los fundadores son: James Lindenbaum, Adam Wiggins, y Orion Henry , desde el año 2010 Salesforce.com adquirió Heroku como una subsidiaria, y desde entonces se da soporte a la mayoría de gestores de bases de datos como postgreSQL[8].

Heroku ofrece servicios gratuitos de hosting y almacenamiento de datos.Esto tiene el inconveniente de tener limitados los recursos en la infraestructura de servicio.Heroku ocupa una unidad llamada Dyno que es la unidad básica de procesamiento y el costo es de $0.05 USD por hora para el paquete básico y $0.10 USD en el paquete extendido[9].

Heroku permite despliegues de servicios de forma gratuita siempre y cuando no exceda los requeriemientos del paquete básico gratuito, en caso de necesitar mayor capacidad fácilmente se puede contratar un paquete con mejores características.

La descripción de los paquetes son los siguientes:

I dyno-web (paquete básico gratuito):

- 512 MB de RAM
- 1x CPU Share
- 10,000 registros en la base de datos con postgresql.

2dynos-webs (paquetebásico):

- 512 MB de RAM
- 1x CPU Share
- $ 0.05/dyno-hour
- El número de registros depende del paquete contratado para postgresql

I dyno-web (extendido):

- 1024MB RAM
- 2x CPU Share
- $ 0.10/dyno-hour
- El número de registros depende del paquete contratado para posrgresql

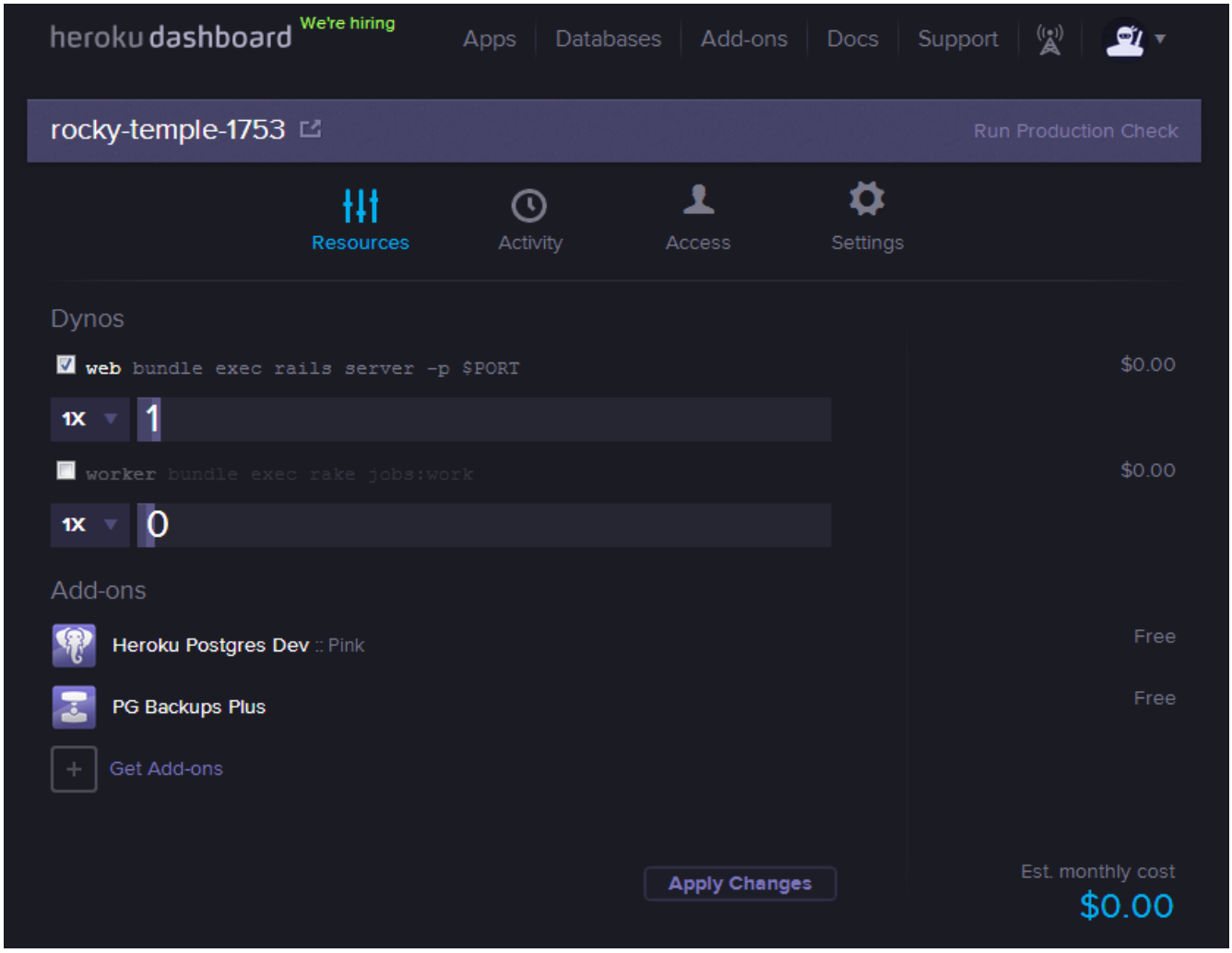

Figura 2. 1: Recursos disponibles para el servidor básico gratuito.

En la Figura 2.1 vemos la manera de configurar los recursos para las aplicaciones desplegadas en heroku y los costos de utilizarlos, esta manera de utilizar los recursos se llama escalabilidad, puesto que podemos adquirir más recursos solo el tiempo que sea necesario. En la Figura 2.1 podemos observar que el paquete gratuito posee un dyno-web, pero además de esto podermos adquirir paquetes extra para nuestro servicio, uno de estos paquetes es el gestor de bases de datos HerokuPostgresDev que además de ser gratuito, es el recomendado por Heroku, además de otros paquetes que pueden ser gratuitos siempre y cuando nos mantengamos en los requeriemientos minimos.

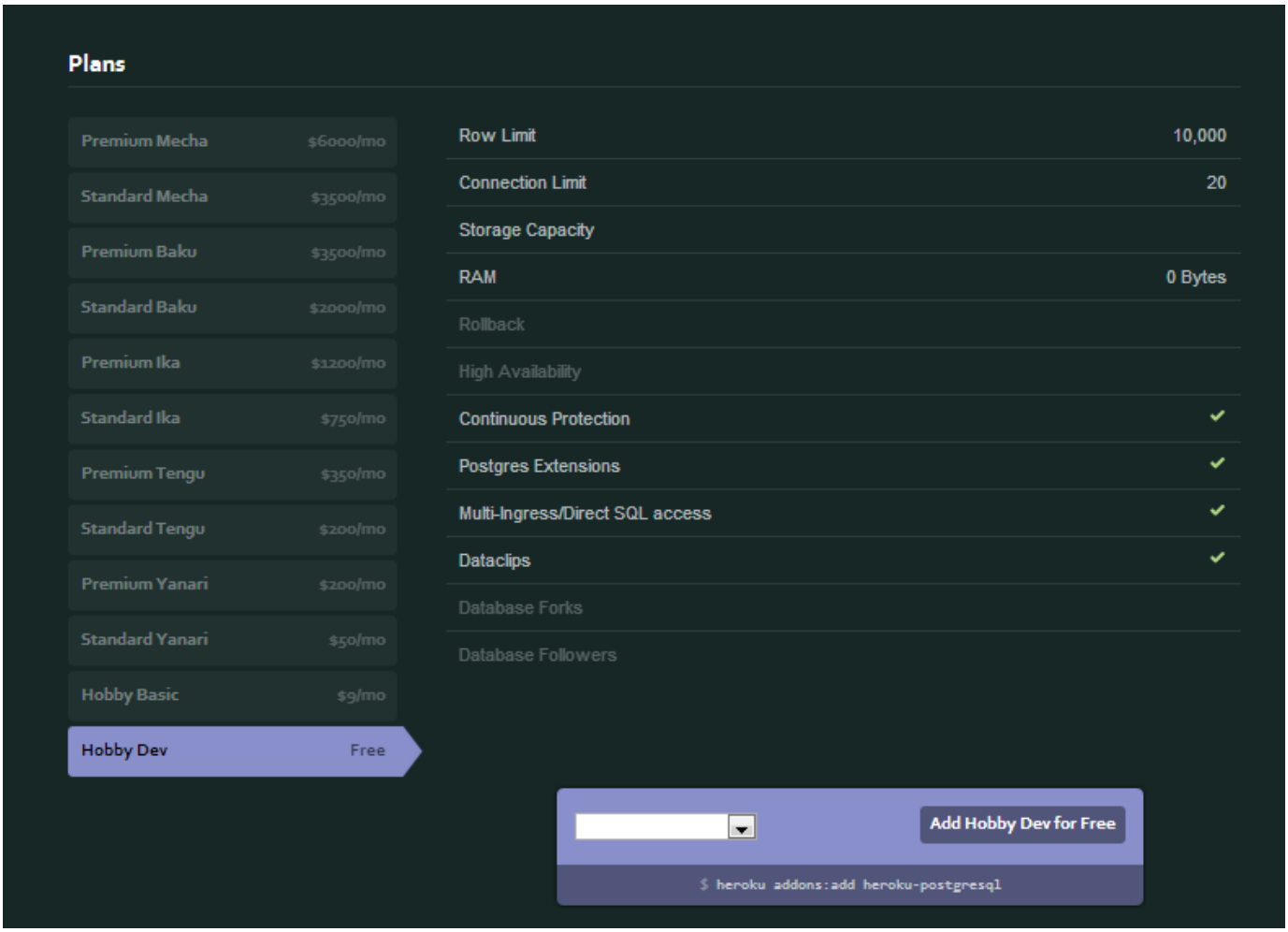

Figura 2. 2: Planes disponibles para Postgresql.

La Figura 2.2 muestra los diferentes planes de contrato de HerokuPostgres DEV, para el caso del servicio gratuito solo tenemos 10,000 líneas en la base de datos con 20 conexiones disponibles simultáneamente.

Para comenzar un despliegue en heroku con rubyonrails, postgreSQLen Windows lo primero que hay que hacer es[10]:

- Crear una aplicación usando rubyonrails y que no presente ningún error y no debe de tener más de 300MB de tamaño.
- Descargar e instalar el herokutoolbelt, disponible en la página principal de heroku (www.heroku.com).
- Luego configurar la estación de trabajo:
  $ heroku login
  Enter your Heroku credentials.

Email: schneems@example.com
Password:
Could not find an existing public key.
Would you like to generate one? [Yn]
Generating new SSH public key.
Uploading ssh public key /Users/adam/.ssh/id_rsa.pub

- Agregar las siguientes gemas en el archivo GemFile y luego instalarlas con el comando *bundleinstall*
gem 'rails_12factor'
gem 'pg'

- Especificar la versión de ruby en el top del archivo GemFile:
ruby "2.0.0"

- Ahora seguimos los siguientes comandos:
$ gitinit
$ git add .
$ git commit -m "init"
$ git status
$ heroku create
Creating serene-crag-3916... done, stack is cedar
http://serene-crag-3916.herokuapp.com/ | git@heroku.com:serene-crag-3916.git
Git remote heroku added
$ git push heroku master
-----> Compiled slug size: 34.8MB
---->Launching... done, v5
http://serene-crag-3916.herokuapp.com deployed to Heroku
To git@heroku.com:serene-crag-3916.git
** [new branch] master -> master*
$ heroku run rake db:migrate
$ heroku open
Opening serene-crag-3916... done

Ahora se puede accesar a nuestra aplicacion desde el navegador con la dirección:

http://serene-crag-3916.herokuapp.com

2.2.1 Migrando la base de datos

La aplicación esta corriendo en Heroku. Sin embargo la base de datos estavacía. Para llenar la base de se usa una aplicación gratuita de heroku llamada: PG Backups Plus[11].

Se crea un archivo de respaldo en el intérprete de postgreSQL de la siguiente manera:

C:\Sites\servicioues>pg_dump -Fc --no-acl --no-owner -h localhost -U admin servicioues_development>servicioues_development.dump

Este archivo se sube en algún servicio de almacenamiento de datos como dropbox y ejecutar el siguiente comando en la consola de heroku:

herokupgbackups:restore enigmatic-castle-2217::white 'https://dl.dropboxusercontent.com/s/e3kbo456l4a79t3/servicioues_development.dump' –app enigmatic-castle-2217

Donde enigmatic-castle-2217::white es el nombre de la base de datos en heroku y *'https://dl.dropboxusercontent.com/s/e3kbo456l4a79t3/servicioues_development.dump'*es el enlace de descarga del archivo servicioues_development.dump.

Y listo, la aplicación está desplegada con las bases de datos completas.

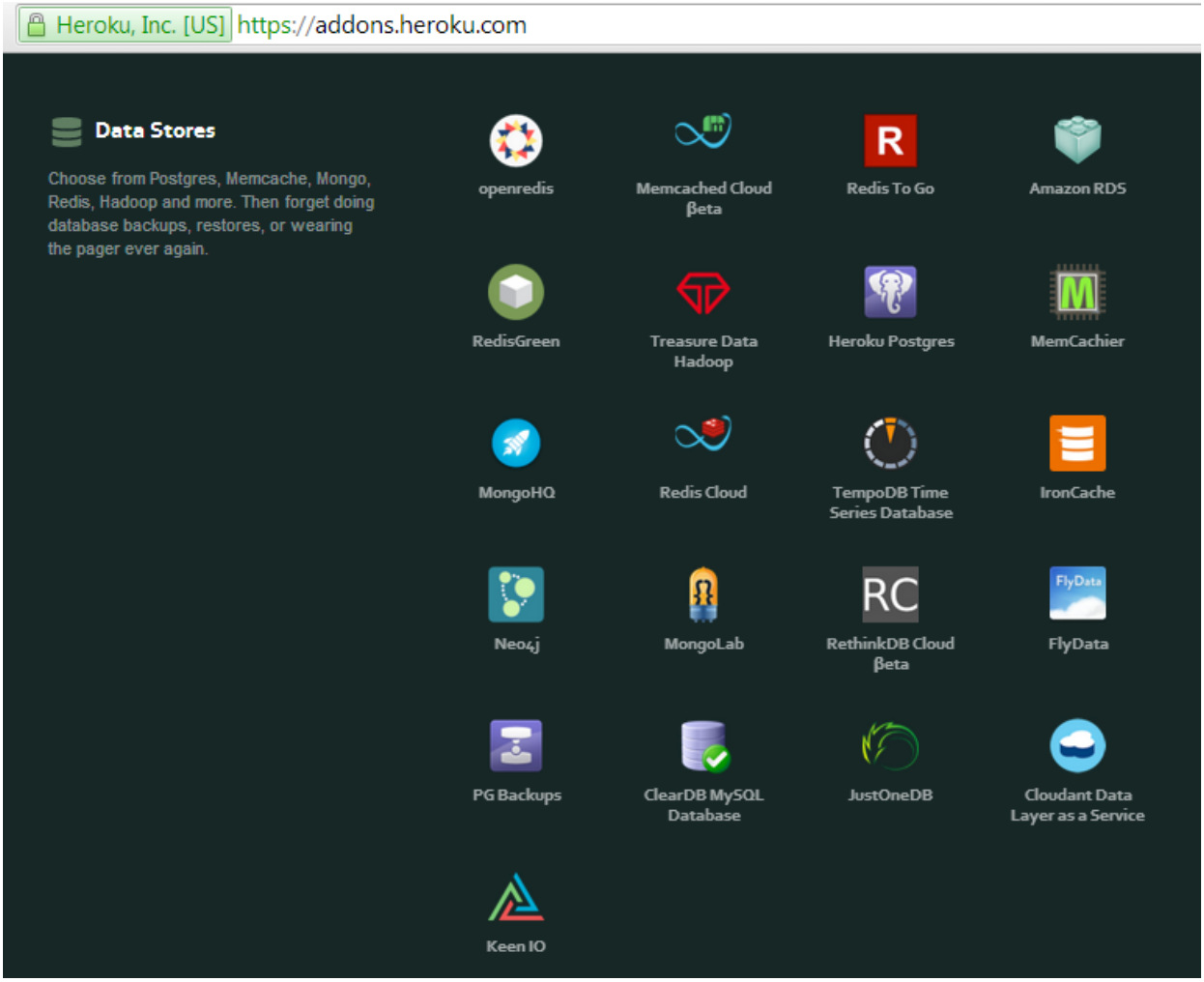

Figura 2. 3: Gestores de datos soportados por heroku.

En la Figura 2.3 vemos los diferentes gestores de bases de datos soportados por heroku, aunque por defecto heroku utiliza Postgresql.

3 CAPITULO III

3.1 SERVICIO DE MONITOREO

3.1.1 Red de medidores en el campus universitario.

Dentro del campus universitario se encuentra desplegada una red mesh. La red interconecta 30 medidoresinstalados en subestaciones. La tabla 3.1 muestra la descripción de la red de medidores.

Donde el nombre de la subestación representa a la facultad o el edificio al cual suministra energía eléctrica.La red cuentra con tres modelos de medidores el SHARK 100-s, SHARK 200-s y SHARK 200, con capacidad de lectura de 68 y 64 variables.Pero debido al pago de licencias, en la mayoría de los casos, estos no monitorean todas las variables posibles.

El algoritmo de interrogación de los medidores captura una medición por cada minuto. Pero por problemas de calidad de transmisión de la red mesh no todas las mediciones son exitosas. En algunos casos se pierden hasta el 45% de los datos, como en las subestaciones de economia2 y economia6. La subestación de mecánica complejo posee la mayor tasa de perdidas con un 53.33% de perdidas de datos.

Para calcular el porcentaje de éxito de la interrogación de los medidores, entendemos que el porcentaje de éxito ideal de una subestación seria de 100% para un total de 60 mediciones en 60 minutos. Si contamos el número de mediciones exitosas divididas entre las 60 mediciones y lo multiplicamos por 100, obtenemos el porcentaje de éxito de interrogación para una hora. La ultima columna de la la Tabla 1representa dicho porcentaje que fue calculado tomando la fecha y hora del 16 de agosto de 2013 entre la 1 y las 2 de la tarde para todas las subestaciones, con la excepción de las subestaciones de artes, economía 5 y rectoría que por encontrarse fuera de servicio no fue posible hacer el cálculo en la fecha indicada, usando la última fecha de mediciones registrada, siempre entre la 1 y las 2 de la tarde.

Se escogió el mismo día y la misma hora para tener las mismas condiciones climáticas y evaluar el desempeño de la red en igualdad de condiciones para todas las subestaciones.

Nº	Subestación	Modelo de Medidor	Numero de variables monitoreadas	Numero de variables que responden	Frecuencia de interrogación	% de éxito de la Interrogación para una hora
1	Agronomía	SHARK 100-s	68	25	1 minuto	96.67
2	Agronomía Decanato	SHARK 100-s	68	25	1 minuto	76.67
3	Agronomía Galera	SHARK 100-s	68	25	1 minuto	91.66
4	Agronomía Química	SHARK 100-s	68	68	1 minuto	91.66
5	Artes	SHARK 200-s	64	64	1 minuto	*86.66
6	Auditórium Mármol	SHARK 100-s	68	25	1 minuto	71.67
7	Cafetines	SHARK 100-s	68	25	1 minuto	91.66
8	Comedor Ues	SHARK 100-s	68	25	1 minuto	85
9	Derecho	SHARK 200-s	64	64	1 minuto	53.33
10	Economia1	SHARK 100-s	68	25	1 minuto	60
11	Economia2	SHARK 100-s	68	25	1 minuto	55
12	Economia3	SHARK 100-s	68	25	1 minuto	61.67
13	Economia4	SHARK 100-s	68	25	1 minuto	60
14	Economia5	SHARK 200-s	64	64	1 minuto	*80
15	Economia6	SHARK 100-s	68	25	1 minuto	55
16	Humanidades1	SHARK 100-s	68	25	1 minuto	31.67
17	Humanidades2	SHARK 100-s	68	25	1 minuto	28.33
18	humanidades3	SHARK 100-s	68	25	1 minuto	91.67
19	Mecánica Complejo	SHARK 100-s	68	25	1 minuto	46.67
20	Medicina	SHARK 200-s	64	64	1 minuto	66.67
21	Odontología Imprenta	SHARK 100-s	68	25	1 minuto	83.33
22	Odontologia1	SHARK 200-s	64	64	1 minuto	95
23	Odontologia2	SHARK 200-s	64	64	1 minuto	90
24	Odontologia3	SHARK 200-s	64	64	1 minuto	81.67
25	Periodismo	SHARK 100-s	68	25	1 minuto	91.67
26	Primario Fia	SHARK 200	64	64	1 minuto	93.33
27	Psicología	SHARK 100-s	68	68	1 minuto	60
28	Química	SHARK 200-s	64	64	1 minuto	100
29	Química Imprenta	SHARK 100-s	68	25	1 minuto	85
30	Rectoría	SHARK 200-s	64	64	1 minuto	*90
31	Humanidades4	-	-	-	-	**

Tabla 3. 1: Descripción de la red de medidores.

*Medidores fuera de operación.

**Medidor operativo desde febrero de 2014.

La Tabla 3.1 describe las subestaciones que forman la red de medidores de la Universidad de El Salvador, el modelo del medidor instalado, el número de variables monitoreadas, y el número de variables efectivamente monitoreadas, la frecuencia de interrogación y la efectividad de la interrogación.

3.1.2 La base de datos spud.

La base de datos nombrada SPUD, es la base de datos donde los se guardan los datos. El servidor que se encuentra montado en el laboratorio de comunicaciones de la escuela de ingeniería eléctrica. Dicha base de datos crece a razón de 2GB por año aproximadamente.Dentro de la base de datos se encuentran 30 tablas que corresponden a los 30 medidores de la red. Cada tabla posee entre 68 y 64 campos, dependiendo del modelo de los medidores conectados. Siendo el modelo SHARK 100-s el que más campos posee.

Las tablas tienen tres tipos de datos.El timestamp para almacenar la fecha y la hora de la medición en un solo campo. El float para valores que usan decimales y requieren precisión, como voltajes y corrientes. Y el integer (int(11)) para valores enteros y más específicamente, para los involucrados en el consumo de energía.

Una contradicción encontrada en la declaración de la tabla es que todas las variables están declaradas como no nulas (NOT NULL).Pero, al mismo tiempo su valor por defecto es NULL.No puede decirse que esto sea un error, pero su consecuencia es volver más pesadas las bases de datos.La declaración de la base de datos tal y como esta, no deja campos sin valores, ocupando direcciones de memoria con datos que no son útiles,Vease tabla 3.2. En subestaciones donde solo se monitorean 25 variables generan un serio problema con respecto al peso de las tablas, pues los demás campos se rellenan con ceros cuando es más conveniente que fueran NULL.

También es de hacer notar la ausencia de un campo identificador (Id), que sirva como llave primaria para hacer búsquedas más fácilmente. Asi como también la falta de campos que registren la fecha de creación y actualización de las filas.Muchas búsquedas pueden relacionarse con estos campos.

Field	Type	Null	Key	Default	Extra
Fecha_hora	timestamp	NO		CURRENT_TIMESTAMP	
***Reales**	float	NO		NULL	
***Enteros**	int(11)	NO		NULL	

Tabla 3. 2: Descripción de las variables de las diferentes tablas de la base de datos spud.

Como resumen, puede decirse que la base de datos spud, posee valores muy confiables para describir el comportamiento eléctrico de las subestaciones monitoreadas.Sin embargo en los casos de los medidores que solo muestrean 25 campos generan un exceso de peso debido los campos mal declarados, generando inconvenientes a la hora de consultar estas tablas.

Las tablas contenidas en spud sobrepasan el número de registros permitidos para poder lograr un despliegue con el paquete gratuito en heroku.Para lograr un despliegue funcional es necesario utilizar el paquete para postgresql: Standard Yanari $50/mes que nos permite un número ilimitado de registros, 60 conexiones simultaneas, 64GB de almacenamiento y 410 MB de RAM.

3.2 CREACIÓN DEL SERVICIO WEB DE DATOS DE CONSUMO DE ENERGÍA DEL CAMPUS CENTRAL DE LA UNIVERSIDAD DE EL SALVADOR.

La finalidad del servicio de monitoreo es el de crear una herramienta para la visualización en la web de las diferentes variables. Se escogió rubyonrails, que es un framework de aplicaciones web de código abierto escrito en el lenguaje de programación Ruby. Ruby onrails maneja una sintaxis fácil de comprender y sencilla de aplicar.Actualmente se encuentra en el puesto 14 del índice TIOBE [12]que mide la popularidad de los lenguajes de programación.

La estructura del sitio se basa en paradigma de modelo-vista-controlador, facilitando la estructura del servicio. El primer paso es crear la estructura del servicio y montarla en el servidor que por defecto viene instalado con rubyonrails llamado WEBrik. Esto se hace de forma muy fácil, solo es necesario ejecutar un comando en la commandprompt de rubyonrails, especificando el nombre del servicio y el sistema de gestión de bases de datos utilizado. El comando es el siguiente[6]:

rails new myapp --database=SGBD

Y para nuestro caso el comando seria:

rails new servicioues --database=postgresql

Donde "rails new" es la petición para crear un nuevo proyecto de rubyonrails, con el nombre de "servicioues" que usa el sistema gestor de base de datos postgresql.

Dicho comando crea la estructura necesaria para la creación del proyecto dentro de una carpeta llamada servicioues[13]. Ver Tabla 3.

Carpeta	Descripción
app/	Es la capeta donde están definidos las diferentes vistas, controladores y modelos.
config/	Contiene la configuración de la aplicación.
db/	Contiene los archivos de configuración de las migraciones de la base de datos, así como también la estructura de las tablas.
doc/	Documentación de la aplicación.
lib/	Módulos de biblioteca.
log/	Contiene los logs de la aplicación.
public/	Contiene todos los archivos extras necesarios, como imágenes y pequeños scripts.
script/rails	Contiene un script proporcionado por rails para generar código, abrir una consola de sesión o iniciar el servidor web local.
test/	Pruebas de aplicación.
tmp/	Archivos temporales.
vendor/	Contiene la tercera parte del código, tales como plugins y gems.
Rakefile	Utilidad disponible a través del comando rake.
Gemfile	Gems requeridos para la aplicación.
config.ru	Un archivo de configuración para rack middleware.
.gitignore	Patrones que ignora git.

Tabla 3. 3: Estructura de carpetas creadas dentro de la carpeta de la aplicación.

La Tabla 3.3 representa la distribución de las capetas y archivos que forman la estructura estándar de una aplicación de rubyonrails.

3.2.1 Creando los modelos, las vistas, controladores y relacionando la base de datos.

Hasta este momento tenemos un servicio llamado servicioues, pero sin ninguna base de datos relacionada, y aun así podemos levantar el servicio con el comando:

rails s

Para ingresar al servicio solo basta con ir a: http://localhost:3000/servicioues en un navegador. Ver figura 3.1.

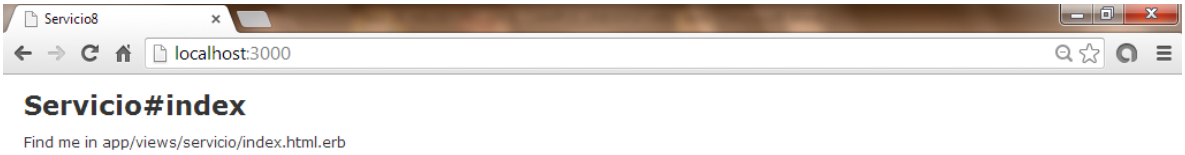

Figura 3. 1: Primera vista del servicio.

En este punto de la creación del servicio es necesario crear los modelos, las vistas y los controladores. La complicación parte de que la fuente de datos del servicio es la base de datos spud, que posee 30 tablas con los registros de los medidores. Esto implica crear una nueva base de datos que posea las mismas 30 tablas, con los mismo campos pero generada a través de rubyonrails. Para evitar problemas de comunicación e interpretación, dicha base de datos estará vacía en principio y será necesaria la exportación de los datos de las tablas de spud.

Para hacer esto es necesario instalar y crear una nueva base de datos en el gestor de bases postgresql, para ello utilizamos el programa administrador de posgreSQL: pgAdmin III. Ver Figura 3.2.

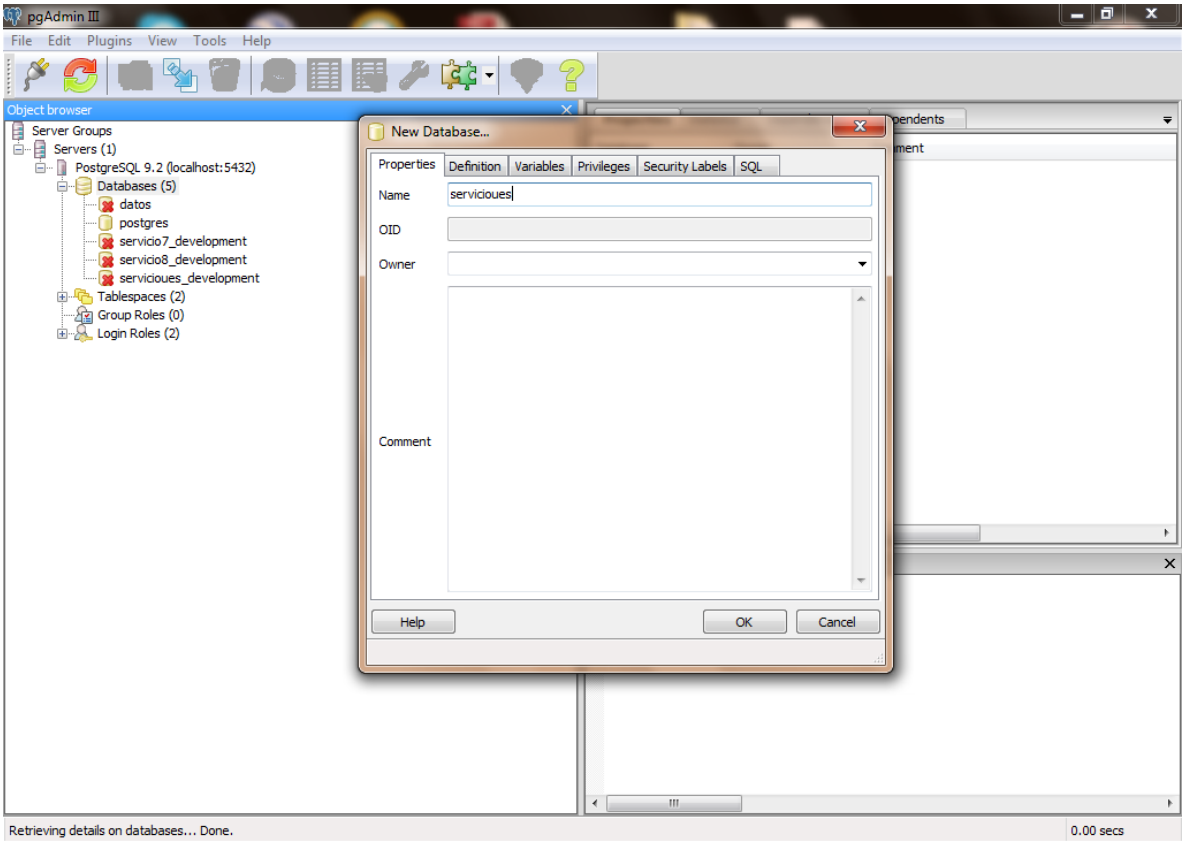

Figura 3. 2: Creacion una nueva base de datos en PgAdmin III.

La Figura 3.2, muestra la ventana de creación de una nueva base de datos en pgAdmin III.

3.2.2 Relacionando la base de datos

Basta con crear una base de datos vacía, llamada servicioues_development para poder ser conectada con ruby. Ahora configuramos el archivo database.yml que maneja la conexión a las bases de datos, este archivo se encuentra en la dirección: *C:/Sites/servicioues/config/database.yml.*

El contenido del archivo es el siguiente:

```
# PostgreSQL. Versions 8.2 and up are supported.
#
# Install the pg driver:
#   gem install pg
# On Mac OS X with macports:
#   gem install pg -- --with-pg-config=/opt/local/lib/postgresql84/bin/pg_config
# On Windows:
#   gem install pg
#       Choose the win32 build.
#       Install PostgreSQL and put its /bin directory on your path.
```

```
# Configure Using Gemfile
# gem 'pg'
#
development:
adapter: postgresql
encoding: unicode
database: servicioues_development//base de datoscreada en pgAdmin
pool: 5
username: admin
password: admin

  # Connect on a TCP socket. Omitted by default since the client uses a
  # domain socket that doesn't need configuration. Windows does not have
  # domain sockets, so uncomment these lines.
  #host: localhost
  #port: 5432

  # Schema search path. The server defaults to $user,public
  #schema_search_path: myapp,sharedapp,public

  # Minimum log levels, in increasing order:
  #   debug5, debug4, debug3, debug2, debug1,
  #   log, notice, warning, error, fatal, and panic
  # The server defaults to notice.
  #min_messages: warning

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
adapter: postgresql
encoding: unicode
database: servicioues_test
pool: 5
username: admin
password: admin

production:
adapter: postgresql
encoding: unicode
database: servicioues_production
pool: 5
username: admin
password: admin
```

Código 3. 1: Configuración de la conexión del servicio con la base de datos.

Teniendo ya configurada la conexión con la base de datos, procedemos a crear las tablas y aplicar las migraciones, para ello ocupamos el comando “scaffold” el cual crea un controlador, una vista y un modelo para cada tabla. La sintaxis general es la siguiente[13]:

Railsgeneratescaffoldnombre_de_la_tabla campo1:type campo2:type ……..etc.

Como la base de datos spud contiene dos tipos diferentes de tablas generamos dos tipos diferentes de scaffolds dependiendo de las variables monitoreadas:

```
railsgeneratescaffold Agronomía
Fecha_hora:timestampVa:floatVb:floatVc:floatVab:floatVbc:floatVca:floatIa:floatIb:floatIc:float
P:float Q:float S:float FP:float F:float
Ineutro:floatWhRec:integerWhDes:integerWhNet:integerWhTot:integerVArhPos:integerVArhNeg
:integerVArhNet:integerVArhTot:integerVAhTot:integerTHDVa:floatTHDVb:floatTHDVc:floatTHDI
a:float THDIb:float THDIc:float Ia0:integer Ia1:integer Ia2:integer Ia3:integer Ia4:integer
Ia5:integer Ia6:integer Ia7:integer Va0:integer Va1:integer Va2:integer Va3:integer Ib0:integer
Ib1:integer Ib2:integer Ib3:integer Ib4:integer Ib5:integer Ib6:integer Ib7:integer Vb0:integer
Vb1:integer Vb2:integer Vb3:integer Ic0:integer Ic1:integer Ic2:integer Ic3:integer Ic4:integer
Ic5:integer Ic6:integer Ic7:integer Vc0:integer Vc1:integer Vc2:integer Vc3:integers
```

Código 3. 2: ***Scaffold de la tabla agronomía modelo para los medidores SHARK 100s.***

```
railsgeneratescaffold Artes
Fecha_hora:timestampVa:floatVb:floatVc:floatVab:floatVbc:floatVca:floatIa:floatIb:floatIc:float
P:float Q:float S:float FP:float F:float
Ineutro:floatPa:floatPb:floatPc:floatQa:floatQb:floatQc:floatSa:floatSb:floatSc:floatFPa:floatFPb:fl
oatFPc:floatWhRec:integerWhDes:integerWhNet:integerWhTot:integerVArhPos:integerVArhNeg:
integerVArhNet:integerVArhTot:integerVAhTot:integerWhReca:integerWhRecb:integerWhRecc:i
ntegerWhDesa:integerWhDesb:integerWhDesc:integerWhNeta:integerWhNetb:integerWhNetc:i
ntegerWhTota:integerWhTotb:integerWhTotc:integerVArhPosa:integerVArhPosb:integerVArhPos
c:integerVArhNega:integerVArhNegb:integerVArhNegc:integerVArhNeta:integerVArhNetb:intege
rVArhNetc:integerVArhTota:integerVArhTotb:integerVArhTotc:integerVAhTota:integerVAhTotb:i
ntegerVAhTotc:integer
```

Código 3. 3: ***Scaffold de la tabla artes modelo para los medidores SHARK 200s.***

Scaffold de 68 campos para subestaciones con medidor SHARK 100S y scaffold de 64 campos para subestaciones con medidor SHARK 200S. Ver tabla 1.

Este comando genera 9 nuevos archivos por cada tabla creada, para el caso de la tabla de agronomía los archivos son:

- 20131028040555_create_agronomia.db: que genera la migración de la tabla a la base de datos servicioues_development.
- Agronomium.rb: que genera el modelo para hacer las consultas a la tabla.
- Routes.rb: genera una ruta que apunta a los controladores de la tabla.
- Agronomía_controller.rb: donde se generan automáticamente todas las funciones del GRUD y donde podemos añadir las funciones que se requieran.
- Index.html.erb, edit.html.erb, show.html.erb, new.html.erb y _form.html.erb: son los archivos de las vistas de los diferentes formularios que apuntan a las funciones de ver, editar, crear y enlistar que por defecto se generan en el controlador (agronomía_controller.rb).

Ahora tenemos definidos los modelos, las vistas y los controladores de cada tabla, solo falta migrar esta estructura a una base de datos para lograr la conectividad, para lograr esto usamos el comando:

Rakedb:migrate.

Con este comando creamos las tablas definidas en rubyonrails en la base de datos servicioues_development creada en postgreSQL.

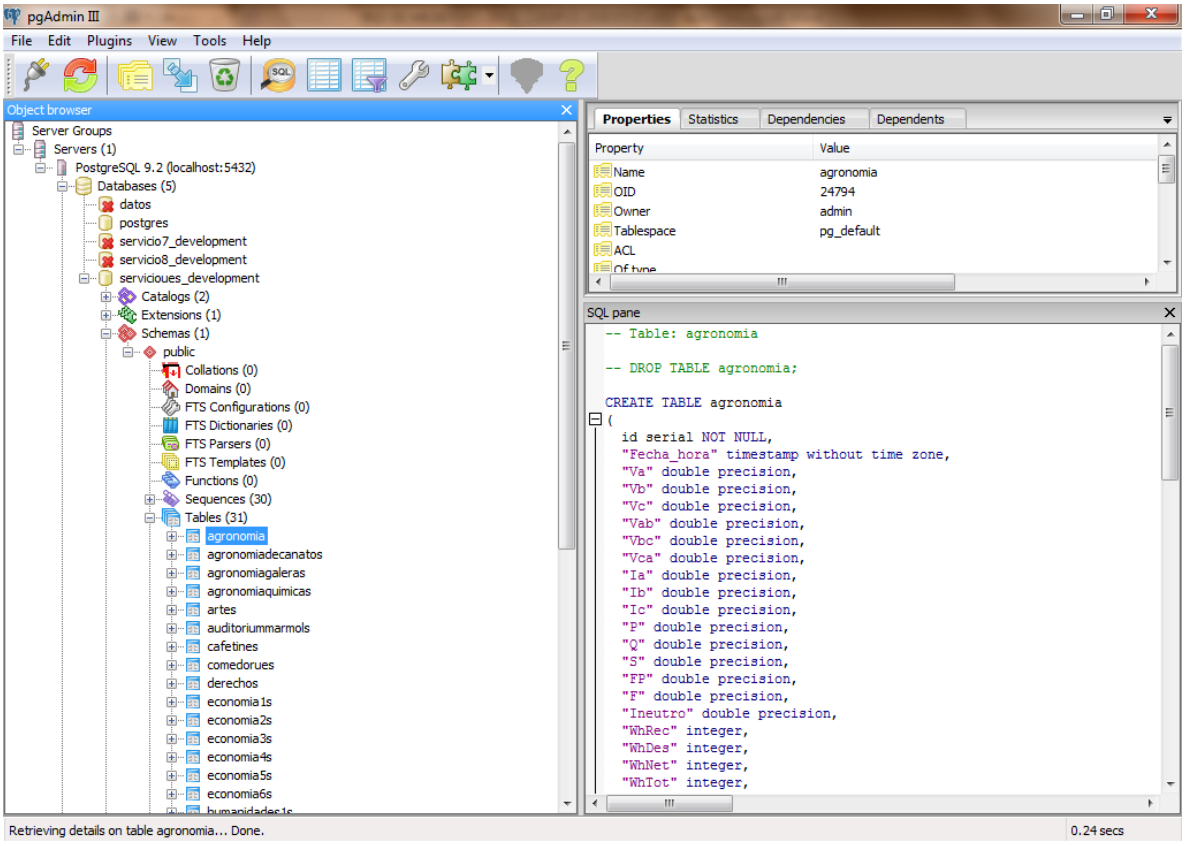

Figura 3. 3: Estructura y declaración de la tabla agronomía en postgresql.

En la Figura3.3 corroboramos la creación de las tablas dentro del gestor de bases de datos.

3.3 COMENZANDO EL DESARROLLO.

3.3.1 El controlador básico.

Las funciones estándar están definidas dentro del controlador y son las siguientes[6]:

- Índex: Esta función enlista todos los registros de la tabla y los cambia para poder ser interpretados por HTML y JAVA, y lo renderiza en el archivo index.html.erb. Esta es la principal función dentro del controlador, ya que carga en una sola variable (@agronomia) todos los registros para poder ser manejados.

- Show: Busca y muestra un registro a través de su id. Y lo guarda en la variable @agronomium cambia el formato del contenido de @agronomium a un formato interpretado por HTML y JAVA, y lo renderiza en el archivo show.html.erb.

- New: Crea un Nuevo registró en la tabla y llama a un formulario web para ser llenado.

- Edit: Encuentra por medio de su id a un registro a ser modificado.

- Create: Guarda un registro en la base de datos usando la función new y el formulario relacionado.

- Update: Actualiza un registro usando la función edit.

- Destroy: Elimina un registro de la tabla.

El código del controlador es el siguiente:

```
classAgronomiaController<ApplicationController
  # GET /agronomia
  # GET /agronomia.json
def index
    @agronomia = Agronomium.all

respond_to do |format|
     format.html # index.html.erb
format.json {render json: @agronomia}
end
end

  # GET /agronomia/1
  # GET /agronomia/1.json
def show
    @agronomium = Agronomium.find(params[:id])

respond_to do |format|
     format.html # show.html.erb
format.json {render json: @agronomium}
end
end

  # GET /agronomia/new
  # GET /agronomia/new.json
def new
    @agronomium = Agronomium.new

respond_to do |format|
     format.html # new.html.erb
format.json {render json: @agronomium}
end
end
```

```
 # GET /agronomia/1/edit
def edit
   @agronomium = Agronomium.find (params [: id])
end

 # POST /agronomia
 # POST /agronomia.json
def create
   @agronomium = Agronomium.new(params[:agronomium])

respond_to do |format|
if @agronomium.save
      format.html { redirect_to @agronomium, notice: 'Agronomium was successfully created.' }
format.json{ renderjson: @agronomium, status: :created, location: @agronomium }
else
      format.html { render action: "new" }
format.json{ renderjson: @agronomium.errors, status: :unprocessable_entity }
end
end
end

 # PUT /agronomia/1
 # PUT /agronomia/1.json
def update
   @agronomium = Agronomium.find(params[:id])

respond_to do |format|
if @agronomium.update_attributes(params[:agronomium])
      format.html { redirect_to @agronomium, notice: 'Agronomium was successfully updated.' }
format.json{ head :no_content }
else
      format.html { render action: "edit" }
format.json{ renderjson: @agronomium.errors, status: :unprocessable_entity }
end
end
end

 # DELETE /agronomia/1
 # DELETE /agronomia/1.json
def destroy
   @agronomium = Agronomium.find(params[:id])
   @agronomium.destroy

respond_to do |format|
    format.html { redirect_toagronomia_url }
format.json{ head :no_content }
end
end
end
```

Código 3. 4: Controlador para la tabla agronomia.

3.3.2 El modelo básico.

Un modelo no es más que la definición de la estructura de la base de datos en rubyonrails, y su existencia es requerida por ActiveRecord, que es una gema utilizada para la conectividad con las bases de datos, su código no es más que una declaratoria de conectividad:

```
classAgronomium<ActiveRecord::Base
end
```

Código 3. 5: Modelo para la tabla de agronomía.

3.3.3 Las vistas básicas.

Las vistas básicas son un conjunto de archivos HTML vinculados a las funciones definidas en el controlador, creando un espacio visual para interactuar con la tabla.

Formadas por 5 archivos:

- Index.html.erb
- New.html.erb
- Edit.html.erb
- Show.html.erb
- _form.html.erb

Cada archivo es nombrado igual a la función con la que interactúa, con la excepción de _form.html.erb, que es un archivo de respaldo, pues en él, está definido un formulario con todos los campos necesarios para ser usado con las funciones new, edit y show. Los códigos de estos archivos pueden consultarse en el anexo A.

Todos estos archivos nos permiten tener un control básico de las tablas pero en el caso de graficar dichos datos, es necesario hacer modificaciones considerables en los códigos, ya que la herramienta usada para graficar está escrita en JavaScript y necesitamos crear una variable de enlace para lograr la interacción de los dos lenguajes de programación.

Así como limitar la cantidad de campos cargados en la variable, para evitar sobre cargar la capacidad de memoria del navegador.

Esto se logra modificando la forma de búsqueda y carga de las variables tanto en el modelo como en el controlador, en el modelo básico, todos los campos y registros se cargan al mismo tiempo y eso para nuestro caso es ineficiente pues solo necesitaremos cargar los campos de fecha_hora y el campo a graficar para un determinado rango de tiempo.

Esto lo hacemos incorporando y modificando algunas funciones en el controlador y en el modelo, las modificaciones son las siguientes:

3.3.4 Modificaciones en el controlador.

Modificaciones de la función Index:

La modificación más sustancial se da en la forma de cargar los datos en la variable @agronomia, que es lavariable de enlace, ya que esta variable carga en la memoria RAM y en cache del navegador todos los campos seleccionados; si dicha variable es demasiado grande, desborda la memoria RAM y el servicio falla; por eso es necesario restringir la cantidad de campos y registros a cargar en la variable de enlace, ya que si usamos la sintaxis por defecto cargaremos todo el peso de la tabla de datos y desbordaremos la memoria.

Usando la función params[] capturamos el contenido de las variables :pot :fecha y :fecha2 que alojan el valor y los rangos involucrados enla limitación de campos y registros a utilizar.

Esto se muestra en el código 3.6.

```
defindex

ifparams[:fecha1] &&params[:fecha2]
    @pot = eje(params[:pot])
@agronomia=Agronomium.select("#{eje(params[:pot])}").select('"Fecha_hora"').search(unir(params[:fecha1]),
unir(params[:fecha2]))

else
    @pot=eje('1')
    @agronomia = Agronomium.select('"Va"').select('"Fecha_hora"').where(Fecha_hora: '2013-05-09
00:00:00'..'2013-05-16 23:59:59')
end
respond_to do |format|
    format.html # index.html.erb
format.json{ renderjson: @agronomia }
end
end
```

Código 3. 6: Definición de la función índex modificada.

Donde generamos una validación que identifica si existen valores en las variables de entrada.De existir estos valores creamos una variable llamada @pot. Esta guarda el campo a graficar usando la función “eje”, luego invocamos al modelo “Agronomium” y hacemos la petición de dos campos usando la función “select”.El primer campo seleccionado en el formulario y el segundo campo esel de fecha _hora por defecto.Después buscamos usando la función “search”Vease código 3.9; que está definida en el modelo; el rango seleccionado de registros para no cargar completamente la tabla.

Definición de la función eje:

Esta función ejecuta una identificación del campo a graficar y hace una normalización de los caracteres para poder ser interpretados por el modelo.

El código es el siguiente:

```
defeje(pot)
if pot == '1'
'"Va"'
elsifpot == '2'
'"Vb"'
elsif pot == '3'
    '"Vc"'
.
.
.
.
.
elsif pot == '66'
    '"Vc3"'
else
    '"WhTot"'
end
end
```

Código 3. 7: Definición de la función eje.

Definición de la función unir:

Estafunción crea una cadena de caracteres a partir de los campos seleccionados en el formulario de la vista para la búsqueda por fecha, si algún de estos formularios falla, por defecto asume la fecha de '2012-07-08 00:00:00'. El código es el siguiente:

```
def unir(fecha)
if fecha
    a=fecha.values
b=a[0]+"-"+a[1]+"-"+a[2]+" "+a[3]+":"+a[4]+":00"
else
    '2012-07-08 00:00:00'
end
end
end
```

Código 3. 8: Definición de la función unir.

3.3.5 Modificaciones en el modelo.

La única modificación del modelo fue la creación dela función search que hace una búsqueda a partir de dos fechas establecidas, ya que la búsqueda por defecto del modelo se hace por medio de su id.

```
classAgronomium<ActiveRecord::Base
defself.search(fecha1 , fecha2)
if fecha1 && fecha2
where(Fecha_hora:  "#{fecha1}".."#{fecha2}"
else
select('"P"').select('"Fecha_hora"').where(Fecha_hora: '2012-08-01 00:00:00'..'2012-08-10 00:00:00')
end
end
end
```

Código 3. 9: Código de la función search.

3.3.6 Modificaficaciones de los helpers

Los helper son módulos que nos permiten hacer funciones no relacionadas con los controladores y modelos, en nuestro creamos una función que convierta los datos de la variable @agronomia en datos que puedan ser interpretador por HTML, el código es el siguiente:

```
moduleApplicationHelper
        defconvert_to_amcharts_json(data_array)
                data_array.to_json.gsub(/\"text\"/, "text").html_safe
        end
end
```

Código 3. 10: Función convert_to_amcharts_json.

Esta función hace una conversión de un arreglo de caracteres en un hash, extrayendo solo los valores útiles, cambia el formato para ser compatible con HTML y que puede ser llamada por todos los controladores de la aplicación.

3.3.7 Modificaciones en la vista.

El único archivo de la vista que fue necesario modificar fue el de index, la modificación consiste en 2 partes; ver códigos 10 y 11; primero en insertar una herramienta que visualizar datos, para ello ocupamos las herramientas de amcharts que es una herramienta que corre con JavaScript y compatible con rubyonrails; ver anexo A. Y segundo la creación de un formulario que sirva para controlar los rangos de fecha y las variables a graficar[14], [15].

```
<h3>Grafica para agronomía</h3>

<h1><%= form_tag({:controller => "agronomia", :action => "index"}, :method => :get) do %>
<%= label_tag(:fecha1, "Desde la Fecha:") %><br />
<%= datetime_select "fecha1", :fecha1 %><br /><br />
<%= label_tag(:fecha2, "Hasta la Fecha: ") %><br />
<%= datetime_select "fecha2", :fecha2 %><br />
<br /><%= label_tag(:pot, "Valor a Graficar:") %>
<%= select_tag(:pot, options_for_select([
    ['Va', 1],
    ['Vb', 2],
    ['Vc', 3],
    ['Vab', 4],
    ['Vbc', 5],
    ['Vca', 6],
    ['Ia', 7],
    ['Ib', 8],
    ['Ic', 9],
    ['P', 10],
    ['Q', 11],
    ['S', 12],
    ['Fp', 13],
    ['F', 14],
    ['Ineutro', 15],
    ['WhRec', 16],
    ['WhDes', 17],
    ['WhNet', 18],
    ['WhTot', 19],
    ['VArhPos', 20],
    ['VArhNeg', 21],
    ['VArhNet', 22],
    ['VArhTot', 23],
```

Continua ….

```
    ['VAhTot', 24],
    ['THDVa', 25],
    ['THDVb', 26],
    ['THDVc', 27],
    ['THDIa', 28],
    ['THDIb', 29],
    ['THDIc', 30],
    ['Ia0', 31],
    ['Ia1', 32],
    ['Ia2', 33],
['Ia3', 34],
    ['Ia4', 35],
    ['Ia5', 36],
    ['Ia6', 37],
    ['Ia7', 38],
    ['Va0', 39],
    ['Va1', 40],
    ['Va2', 41],
    ['Va3', 42],
    ['Ib0', 43],
    ['Ib1', 44],
    ['Ib2', 45],
    ['Ib3', 46],
    ['Ib4', 47],
    ['Ib5', 48],
    ['Ib6', 49],
    ['Ib7', 50],
    ['Vb0', 51],
    ['Vb1', 52],
    ['Vb2', 53],
    ['Vb3', 54],
['Ic0', 55],
    ['Ic1', 56],
    ['Ic2', 57],
    ['Ic3', 58],
    ['Ic4', 59],
    ['Ic5', 60],
    ['Ic6', 61],
    ['Ic7', 62],
    ['Vc0', 63],
    ['Vc1', 64],
    ['Vc2', 65],
    ['Vc3', 66]],1)) %>


<%= submit_tag "Graficar", :name => nil %>
<% end %></h1>
```

Código 3. 11: Formulario de control de variables y rango a graficar del índex.

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<title>amCharts examples</title>
<link rel="stylesheet" href="assets/style.css" type="text/css">
<script src="assets/amcharts.js" type="text/javascript"></script>
<script type="text/javascript">
var chart;

varchartData= <%= convert_to_amcharts_json(@agronomia) %>;

AmCharts.ready(function () {
        // SERIAL CHART
chart = new AmCharts.AmSerialChart();
chart.pathToImages = "assets/";
chart.zoomOutButton = {
backgroundColor: "#FFFFFF",
backgroundAlpha: 0.15
        };
chart.dataProvider = chartData;
chart.categoryField = "Fecha_hora";
chart.balloon.bulletSize = 5;

        // listen for "dataUpdated" event (fired when chart is rendered) and call zoomChart method
when it happens
        //chart.addListener("dataUpdated", zoomChart);

        // AXES
        // category
varcategoryAxis = chart.categoryAxis;
        //categoryAxis.parseDates = true; // as our data is date-based, we set parseDates to true
categoryAxis.minPeriod = "mm"; // our data is daily, so we set minPeriod to DD
categoryAxis.dashLength = 1;
categoryAxis.gridAlpha = 0.5;
categoryAxis.position = "top";
categoryAxis.axisColor = "#FFFFFF";
categoryAxis.gridColor = "#FFFFFF";
categoryAxis.color = "#FFFFFF";
categoryAxis.labelRotation = 90;

varvalueAxis = new AmCharts.ValueAxis();
valueAxis.axisAlpha = 0;
valueAxis.dashLength = 1;
valueAxis.gridColor = "#FFFFFF";
valueAxis.color = "#FFFFFF";
chart.addValueAxis(valueAxis);
```

Continua ….

```
            // GRAPH
var graph = new AmCharts.AmGraph();
graph.title = <%= @pot.html_safe %>;
graph.valueField = <%= @pot.html_safe %>;

graph.bullet = "round";
graph.bulletBorderColor = "#FFFFFF";
graph.bulletBorderThickness = 2;
graph.lineThickness = 2;
graph.lineColor = "#5fb503";
graph.negativeLineColor = "#efcc26";
graph.hideBulletsCount = 50; // this makes the chart to hide bullets when there are more than 50 series
in selection
chart.addGraph(graph);

            // CURSOR
chartCursor = new AmCharts.ChartCursor();
chartCursor.cursorPosition = "mouse";
chartCursor.pan = true; // set it to fals if you want the cursor to work in "select" mode
chart.addChartCursor(chartCursor);

            // SCROLLBAR
varchartScrollbar = new AmCharts.ChartScrollbar();
chartScrollbar.graph = graph;
chartScrollbar.scrollbarHeight = 40;
chartScrollbar.color = "#FFFFFF";
            //chartScrollbar.autoGridCount = true;
chart.addChartScrollbar(chartScrollbar);

             // LEGEND
var legend = new AmCharts.AmLegend();
legend.bulletType = "round";
legend.equalWidths = false;
legend.valueWidth = 120;
legend.color = "#FFFFFF";
chart.addLegend(legend);


chart.write("chartdiv");
         });
```

Continua ….

```
functionzoomChart() {
        // different zoom methods can be used - zoomToIndexes, zoomToDates, zoomToCategoryValues
chart.zoomToIndexes(chartData.length - 40, chartData.length - 1);
        }

functionsetPanSelect() {
if (document.getElementById("rb1").checked) {
chartCursor.pan = false;
chartCursor.zoomable = true;
        } else {
chartCursor.pan = true;
        }
chart.validateNow();
}
</script>
</script>
</head>
<body>
<div id="chartdiv" style="width: 100%; height: 500px;"></div>
<div style="margin-left:35px;">
<input type="radio" name="group" id="rb1" onclick="setPanSelect()">Select
<input type="radio" checked="true" name="group" id="rb2" onclick="setPanSelect()">Pan
</div>
</body>
</html>
```

Código 3. 12: Script de amcharts usado para general el grafico.

Los códigos 3.11 y 3.12, forman en conjunto el índex modelo de todas las subestaciones.

Hasta el momento, tenemos un servicio funcional, pero con una mala apariencia, rubyonrails nos permite modificar la apariencia de un proyecto insertando código css y JavaScript, pero no de la manera tradicional, para ello modificamos los archivos que se encuentran en la carpeta asset/stylesheets y asset/javascripts. Insertando código para las diferentes etiquetas del html. Como amcharts posee su propio código solo es necesario descargar desde la página el archivo amscharts.js y copiarlo en asset/javascripts. Estos códigos pueden consultarse en el anexo A.

3.3.8 El diseño de la página de inicio.

La página de inicio fue pensada para proporcionar un rápido acceso a las vistas de las subestaciones, donde fue necesario colocar un menú quehace un despliegue de las 30 subestaciones, un mapa de ubicación de googlemapsy un vínculo que retorne a la página de inicio[14].

El menú de acceso es operacional en todas las vistas del servicio, pero en el caso del mapa de ubicación, que está presente en la página de inicio.

En el archivo views/layouts/application.html.erb se inserta el código que llama al api de JavaScript de googlemaps y se colocan los marcadores respectivos y en el archivo views/servicioues/index.html.erb se coloca el inicializador del mapa y los marcadores de ubicación. Para consultar los códigos de los archivos application.html.erb e index.html.erb ver anexo A.Para consultar como utilizar el JavaScript de googlemaps ver el anexo B[14].

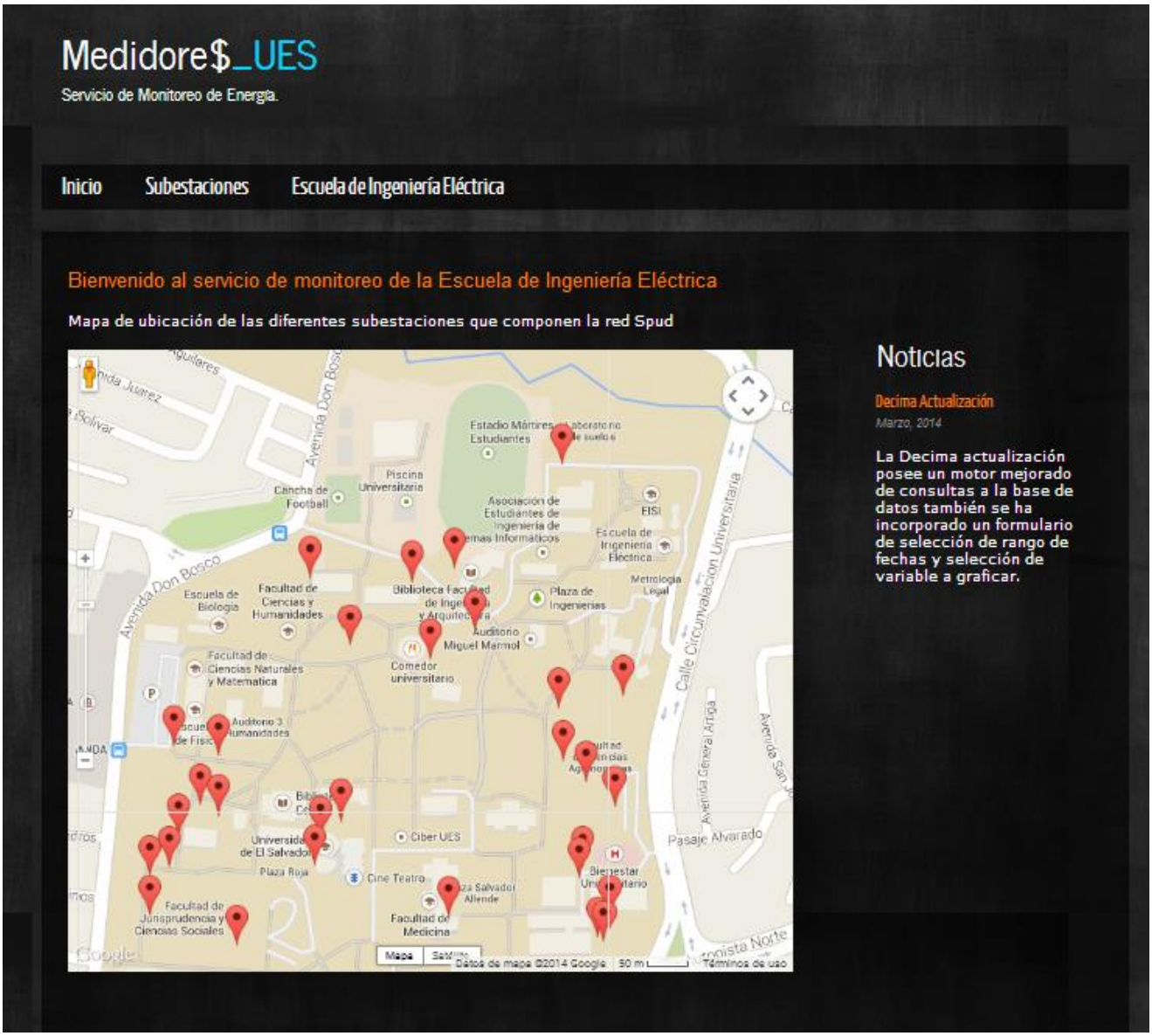

Figura 3. 4: Vista de la página de inicio del servicio de monitoreo.

En la Figura 3.4 podemos ver la apariencia de la página de inicio, que consta de una barra de inicio donde se encuentra el menú de subestaciones, un mapa de ubicación y un pequeño apartado para comentar las actualizaciones y acontecimientos relevantes.

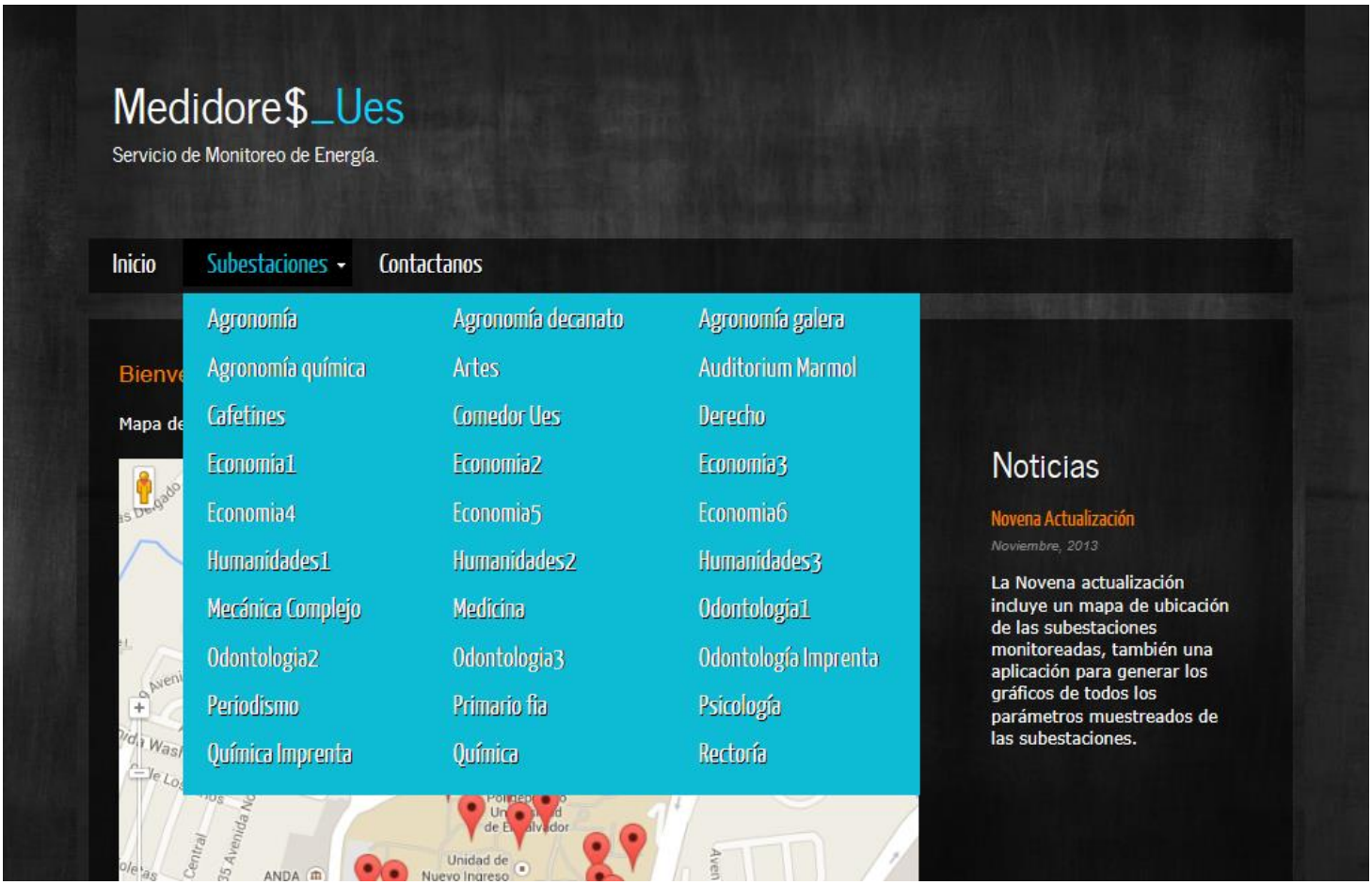

Figura 3. 5: Vista del menú subestaciones.

En la Figura 3.5 vemos el despliegue del menú subestaciones, con las 30 subestaciones ordenadas en 3 columnas.

3.3.9 Gráficas de Amcharts

AmCharts es una librería escrita en javascripts que genera graficas dinámicas.Soporta un gran número de gráficos principalmente gráficos de columnas, barras, líneas, áreas etc[15].

Todas las gráficas están disponibles para descargas, con características limitadas, y con un link promocional en la parte superior-izquierda: chart by amCharts.com . Para quitar ese enlace debemos pagar 99 euros.

Lo interesante de estas utilidades es que son totalmente configurables y trabaja con los datos obtenidos de formatos XML o archivos de texto TXT.

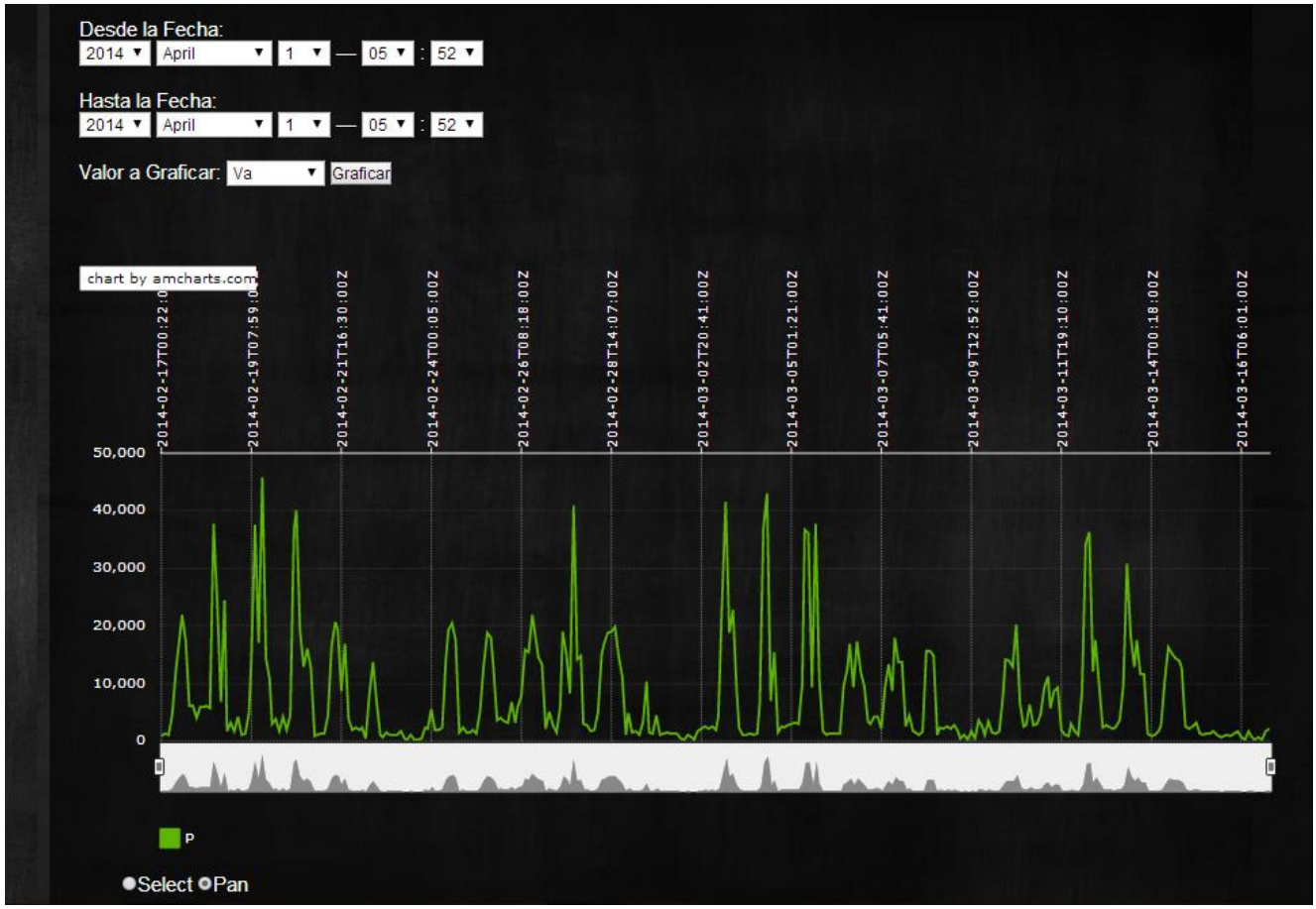

Figura 3. 6: Vista estándar de los generadores de gráficas.

En la Figura 3.6 vemos la vista para el medidor agronomía que sirve de modelo para las demás subestaciones, donde tenemos un formulario que sirve de control de rango de tiempo y de variable a graficar, además de la presentación del grafico

3.3.10 Llenando la base de datos.

El último paso necesario, para declarar el sistema como operativo, es llenar las tablas de la base de datos servicioues_development con los datos de spud. Los pasos son los siguientes:

Como la base de datos spud en el gestor de bases de datos Mysql que no es compatible con postgreSQL es necesario agregar el campo id en todas las tablas de la siguiente manera[16]:

alter table agronomia add id int(11) DEFAULT NULL auto_increment PRIMARY KEY first;

Cuyas propiedades le permiten ser la llave primaria para las búsquedas, además de poder autoincrementarse con cada registro que se almacene en la base de datos.

Luego creamos un archivo csv para cada tabla.

select * from agronomia into outfile "agronomia.csv" fields terminated by ',' lines terminated by '\n';

Teniendo los 30 archivos csv procedemos a insertar los datos en las tablas de servicioues_development desde pgAdmin III.

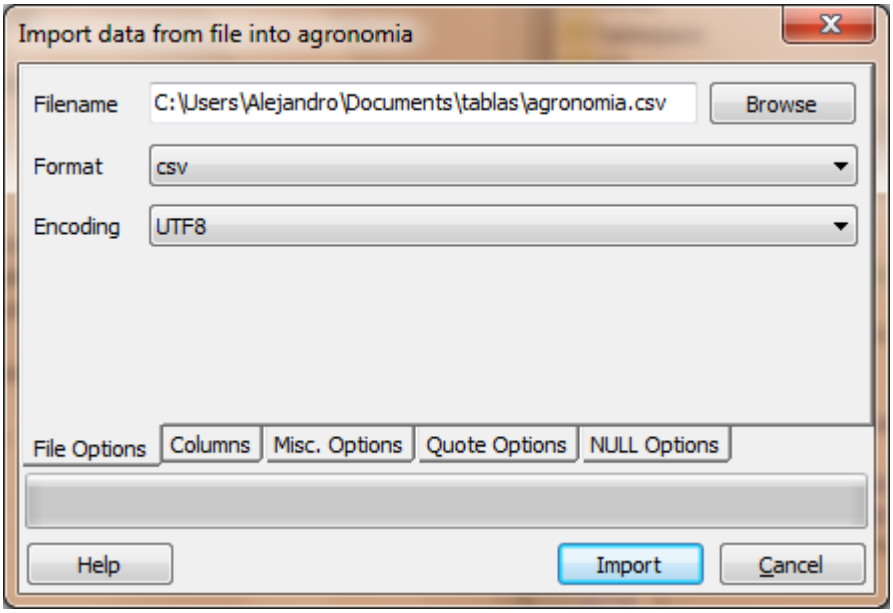

Figura 3. 7: Ventana de pgAdmin III, para importar datos a una tabla.

La Figura 3.7 nos muestra el archivo de origen de los datos, que en este caso es agronomía.csv, el formato del archivo a importar y el tipo de codificación para nuestro caso UTF8.

Ahora el servicio puede ser utilizado desde un servidor local, podemos hacer un despliegue en heroku, pero por tener el inconveniente de poseer demasiados registros, el despliegue gratuito se hace imposible, es necesario contratar un plan para postgreSQL de 50 dólares mensuales.

4 CAPITULO IV

4.1 CONCLUSIONES Y LINEAS FUTURAS.

En este trabajo de graduación, se realizó la creación de un sistema de monitoreo para la red inalámbrica de medidores dentro del campus universitario, para lo cual se experimentaron con diferentes gestores de bases de datos.Se investigó la mejor manera de definir una base de datos, para que sea compatible con el desarrollo web en la nube.Además análizaron los diferentes parámetros eléctricos monitoreados por la red.

Después de implementar y evaluar el servicio y la base de datos involucrada, llegamos a las siguientes conclusiones:

4.2 CONCLUSIONES.

Ruby onrails presenta problemas de velocidad al ser utilizado para consultas y procedimientos lógicos, con bases de datos muy grandes, en especial al utilizar búsquedas tipo N+1.

El desarrollo web se vuelve difícil con una base de datos no diseñada con ese propósito, en especial si estas bases de datos son de tamaños muy grandes.

El uso de herramientas para la interpretación de datos de forma visual, ayuda a tomar mejores decisiones en la resolución de problemas en una subestación eléctrica, así como para mejorar el rendimiento de esta.

Es necesario crear una base de datos diseñada para ser utilizada en el desarrollo web a partir de la base de datos spud.

Para desplegar gratuitamente el servicio en Heroku, es necesaria una base de datos con menos registros en ella, pero sin comprometer la representatividad de la misma.

Es necesario crear una base de datos dedicada únicamente al consumo de energía (Whtot) de todas las subestaciones de la universidad de El Salvador a partir de la base de datos spud, pero con un muestreo de datos fijado a las horas establecidas por la SIGET como pico, valle y resto, para evitar medir datos no necesarios y que la base de datos resultante no sea excesivamente grande y difícil de manipular.

Modificar el script que interroga a los medidores, para que pueda diferenciar entre un cero y un null, así como modificar la arquitectura de los campos de la base de datos spud, para que puedan alojar un campo null, esto ayudaría notablemente en la reducción del tamaño de la base de datos spud y la volverían compatible con el desarrollo web.

Crear un nuevo campo en la base de datos spud que sirva de identificador (id) y de llave primaria, así como dos campos que indiquen la fecha de creación del registro y la fecha de actualización del mismo. Ya que por defecto, los lenguajes de programación hacen la búsqueda de registros en las bases de datos recurriendo a estos.

4.3 LINEAS FUTURAS

Crear las bases de datos descritas en las conclusiones y utilizarlas para optimizar el rendimiento del servicio de consulta actual, y poder identificar los costes de la energía de manera más efectiva y con menos margen de error, para cada una de las subestaciones.

Usando el servicio, encontrar los fallos más frecuentes en todas las subestaciones de la red y comenzar una caracterización de cada una de ellas, así como también corregir los errores de montaje de los medidores.

Incorporar un análisis estadístico de los gráficos como valores máximos y mínimos, medidas de tendencia central y de dispersión de las muestras, para fines de evaluación de calidad de energía.

Cambiar la frecuencia de muestreo de los campos a una frecuencia que optimice el peso de la base de datos con respecto a la fidelidad de las mediciones y encontrar márgenes de error tolerable para mejorar el rendimiento del servicio y para lograr un despliegue exitoso en Heroku.

Experimentar con diferentes plataformas de servicio, y hacer una comparación del rendimiento de estas.

Crear una base de datos compatible con las condiciones de los paquetes gratuitos de Heroku.

Incorporar una función que permita exportar los gráficos a un formato de imagen.

5 Bibliografía

[1] U. SanDiego, «Energy dashboard,» [En línea]. Available: http://energy.ucsd.edu/index.html.

[2] U. N. C. T. A. Development, «Information Economy Report 2013. The Cloud Economy and Developing Countries,» 2013.

[3] computacionenlanuve.org, «computacion en la nuve,» [En línea]. Available: http://www.computacionennube.org/.

[4] G. J. Colusso Ricardo, «Desarrollo ágil de software: Una introducción a las metodologías ágiles de desarrollo de software [Internet]. Version 1. agilesintro,» 26 nov. 2011. [En línea]. Available: https://agilesintro.wordpress.com/article/desarrollo-agil-de-software-3satfj6065tbv-2/.. [Último acceso: 15 mayo 2013].

[5] A. Fox, Engineering Long-Lasting Software: An Agile Approach Using SaaS and Cloud Computing, 2013.

[6] S. Ruby, Agile Web Development with Rails, Dallas, Texas, 2011.

[7] The PostgreSQL Global Development Group, «PostgreSQL 9.2.4 Documentation,» 1996-2013. [En línea]. Available: file:///C:/Program%20Files/PostgreSQL/9.2/doc/postgresql/html/index.html. [Último acceso: 1 septiembre 2013].

[8] Wikipedia.org, «Wikipedia,» 2013. [En línea]. Available: http://es.wikipedia.org/wiki/Heroku. [Último acceso: 20 abril 2013].

[9] Heroku, «Heroku,» 2013. [En línea]. Available: https://www.heroku.com/pricing. [Último acceso: 5 Noviembre 2013].

[10] Heroku Dev Center, «Heroku Dev Center,» 2013. [En línea]. Available: https://devcenter.heroku.com/articles/getting-started-with-rails4. [Último acceso: 20 julio 2013].

[11] Heroku Dev Center, «Heroku Dev Center,» 2013. [En línea]. Available: https://devcenter.heroku.com/articles/heroku-postgres-import-export. [Último acceso: 6 octubre 2013].

[12] TIOBE SOFTWARE, «TIOBE Programming Community Index for November 2013,» 2013. [En línea]. Available: http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html. [Último acceso: 10 noviembre 2013].

[13] rubyonrails.org, «Ruby Guides,» 2013. [En línea]. Available: http://guides.rubyonrails.org/getting_started.html. [Último acceso: 16 marzo 2013].

[14] J. Athayde, The Rails View, Dallas, Texas, 2011.

[15] amcharts.com, «amcharts support,» 2013. [En línea]. Available: http://support.amcharts.com/. [Último acceso: 12 julio 2013].

[16] conclase.net, «Mysql con clase,» 2005. [En línea]. Available: http://mysql.conclase.net/curso/?cap=000#. [Último acceso: 16 abril 2013].

6 ANEXOS

6.1 ANEXO A.

6.1.1 Instrucciones de Google Developers para el uso de la Versión 3 del API de JavaScript de Google Maps

Todas las aplicaciones del API de Google Maps deben cargar el API de Google Maps mediante una clave de API.

Para crear tu clave de API:

1. Accede a la página de la consola de las API (https://code.google.com/apis/console) e inicia sesión con tu cuenta de Google.
2. Haz clic en el enlace de **servicios** en el menú de la izquierda.
3. Activa el servicio de la **versión 3 del API de Google Maps**.
4. Haz clic en el enlace de **acceso al API** en el menú de la izquierda. Tu clave de API está disponible desde la página de **acceso al API**, en la sección de **acceso al API sencilla**. Las aplicaciones del API de Google Maps utilizan la **clave para aplicaciones del navegador**.

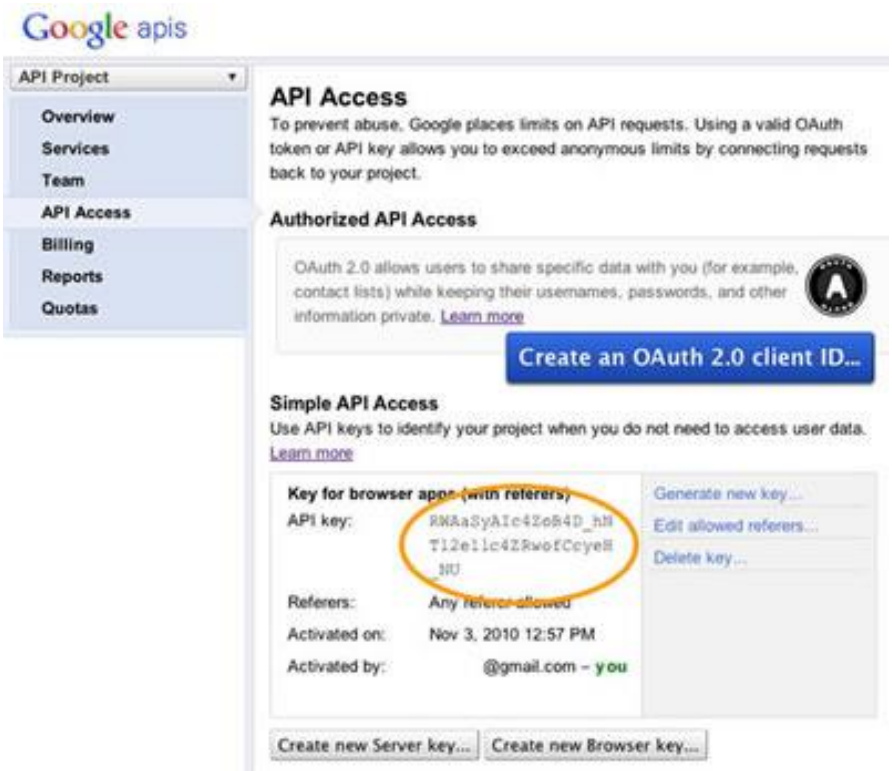

Figura A. 1: Clave de acceso al API de JavaScript de Google Maps

Cómo declarar tu aplicación como HTML5

Te recomendamos que declares un DOCTYPE verdadero en tu aplicación web. En los ejemplos que te mostramos aquí, hemos declarado nuestras aplicaciones como HTML5 mediante el sencillo DOCTYPE HTML5, tal y como se muestra a continuación:

```
<!DOCTYPEhtml>
```

Los navegadores más habituales mostrarán contenido declarado con este DOCTYPE en "modo estándar", lo que significa que tu aplicación deberá ser más compatible con los navegadores. El DOCTYPE también está diseñado para ajustarse de la mejor manera; así, los navegadores que no lo entiendan lo ignorarán y utilizarán el "modo chapucero" para mostrar el contenido.

Ten en cuenta que algunas de las CSS que funcionan en el modo chapucero no son válidas en el modo estándar. En concreto, todos los tamaños basados en porcentajes deben heredarse de los elementos de bloque principales, y si cualquiera de estos antecesores no puede especificar un tamaño, se supondrá un tamaño de 0 x 0 píxeles. Por esta razón, incluimos la siguiente declaración <style>:

```
<styletype="text/css">
  html{height:100%}
  body{height:100%;margin:0;padding:0}
  #map_canvas{height:100%}
</style>
```

Esta declaración de CSS indica que el contenedor del mapa (denominado map_canvas) debe ocupar el 100% de la altura del cuerpo de HTML. Ten en cuenta que debemos declarar de forma específica estos porcentajes tanto para <body> como para <html>.

Cómo cargar el API de Google Maps

```
<html>
  <head>
    <scripttype="text/javascript"

src="http://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&sensor=SET_TO_TRUE_OR_FALSE">
    </script>
```

La URL incluida en la etiqueta script indica la ubicación de un archivo JavaScript que carga todos los símbolos y las definiciones que necesitas para utilizar el API de Google Maps. La etiqueta script es obligatoria.

El parámetro key incluye la clave de API de tu aplicación. Para obtener más información, consulta la sección Cómo obtener una clave de API. Ten en cuenta que esta clave no es la misma clave que se utiliza en la versión 2 del API y que se debe generar a partir de la consola de las API.

Elementos DOM de mapas

```
<divid="map_canvas"style="width:100%;height:100%"></div>
```

Para que el mapa aparezca en una página web, se debe reservar un lugar para él. Normalmente, lo hacemos mediante la creación de un elemento div con nombre y la obtención de una referencia a este elemento en el modelo de objetos de documento (DOM) del navegador.

En el ejemplo anterior, definimos un objeto denominado "map_canvas" y establecemos su tamaño mediante atributos de estilo. Ten en cuenta que este tamaño se establece en "100%", lo que amplía el mapa hasta que ocupa toda la pantalla del dispositivo móvil. Es posible que sea necesario ajustar estos valores según el desplazamiento y el tamaño de la pantalla del navegador. Ten en cuenta que el mapa siempre adoptará el tamaño del elemento en el que esté contenido, por lo que siempre debes establecer un tamaño para el elemento de forma explícita.

Opciones de mapas

```
varmapOptions={
  center:newgoogle.maps.LatLng(-34.397,150.644),
  zoom:8,
  mapTypeId:google.maps.MapTypeId.ROADMAP
};
```

Antes de inicializar un mapa, debemos crear un objeto *Mapoptions* que contenga las variables de inicialización correspondientes. Este objeto no se construye, sino que se crea como un objeto literal.

```
varmapOptions={};
```

Latitudes y longitudes

Como queremos center el mapa en un punto específico, creamos un objeto LatLng para mantener esta ubicación especificando las coordenadas de ubicación en el orden {latitud, longitud}:

```
center=newgoogle.maps.LatLng(-34.397,150.644)
```

El proceso de convertir una *dirección* en un punto geográfico se conoce como *codificación geográfica*. Esta versión del API de Google Maps incluye funciones de codificación geográfica. Para obtener más información, consulta la sección Codificación geográfica del capítulo**Servicios** de esta guía.

Niveles de zoom

La propiedad zoom especifica la resolución inicial con la que se mostrará un mapa, donde un zoom de 0 se corresponde con un mapa de la Tierra totalmente alejado y los niveles de zoom acercan el mapa con una resolución más elevada.

```
zoom:8
```

Para ofrecer un mapa de todo el planeta como una única imagen, es necesario un mapa muy grande o un mapa pequeño con una resolución muy baja. Por consiguiente, las imágenes de mapa en Google Maps y el API de Google Maps se dividen en "mosaicos" de mapas y "niveles de zoom". A niveles bajos de zoom, un conjunto pequeño de mosaicos de mapas cubre una superficie amplia; a niveles de zoom más elevados, los mosaicos tienen una resolución mayor y cubren una superficie más pequeña.

Tipos de mapas

También debes establecer expresamente un tipo de mapa inicial en este momento.

```
mapTypeId:google.maps.MapTypeId.ROADMAP
```

Se admiten los siguientes tipos de mapas:

- ROADMAP, que muestra los mosaicos normales en 2D predeterminados de Google Maps.
- SATELLITE muestra imágenes de satélite.
- HYBRID muestra una mezcla de mosaicos fotográficos y una capa de mosaicos para los elementos del mapa más destacados (carreteras, nombres de ciudades, etc.).
- TERRAIN muestra mosaicos de relieve físico para indicar las elevaciones del terreno y las fuentes de agua (montañas, ríos, etc.).
- Para obtener más información sobre los tipos de mapas, consulta la sección Tipos de mapas. Sin embargo, para la mayoría de los casos lo único que necesitas saber es cómo utilizar los tipos básicos descritos anteriormente.

El objeto "Map"

```
varmap=newgoogle.maps.Map(document.getElementById("map_canvas"),
  mapOptions);
```

La clase de JavaScript que representa a los mapas es Map. Cada objeto de esta clase define un único mapa en una página. (Puedes crear más de una instancia de esta clase; cada objeto definirá un mapa independiente en la página). Creamos una nueva instancia de esta clase mediante el operador new de JavaScript.

Al crear una nueva instancia de mapa, se especifica un elemento HTML en la página como contenedor para el mapa. Los nodos HTML son elementos secundarios del objeto document de JavaScript. Se obtiene una referencia a este elemento mediante el métododocument.getElementById().

Este código permite definir una variable (denominada map) y asignar dicha variable a un nuevo objeto Map, además de transmitir opciones definidas en el objeto mapOptions literal. Estas opciones se utilizarán para inicializar las propiedades del mapa. A continuación se muestra la definición de la función Map(), conocida como *constructor*:

Constructor	Descripción
Map(mapDiv:Node, opts?:*MapOptions*)	Crea un mapa nuevo dentro del contenedor HTML en cuestión, que suele ser un elemento DIV, mediante los parámetros (opcional) que se especifiquen.

Tabla A. 1: Descripcion de la variable maps.

Cómo cargar el mapa

```
<bodyonload="initialize()">
```

Mientras se procesa una página HTML, se crea el modelo de objetos de documentos (DOM) y las imágenes y secuencias de comandos externas se reciben y se incorporan al objeto document. Para garantizar que nuestro mapa se añada a la página cuando se cargue por completo, solo ejecutamos la función que crea el objeto Map cuando el elemento <body> de la página HTML ha recibido un evento onload. De este modo, evitamos un comportamiento impredecible y obtenemos más control acerca del modo y del momento en que se dibuja el mapa.

El atributo onload de la etiqueta body es un ejemplo de un controlador de eventos. El API de JavaScript de Google Maps también proporciona varios eventos que se pueden controlar para determinar cambios de estado.