



Ciencias de la  
Computación  
FACULTAD DE CIENCIAS  
FÍSICAS Y MATEMÁTICAS  
UNIVERSIDAD DE CHILE

---

# Informe de Práctica

CC4901 – Práctica Profesional I

Empresa: NIC Chile Research Labs  
Alumno: Manuel Olguín  
Carrera: Ingeniería Civil en Computación  
RUT: 18.274.982 – 6  
E-Mail: molguin@dcc.uchile.cl  
Tel: +56 9 7463 6997  
27 de marzo de 2016  
Santiago, Chile.

---

## 1. Certificado de la Empresa

---

## 2. Observaciones

---

## Índice

<b>1. Certificado de la Empresa</b>	<b>2</b>
<b>2. Observaciones</b>	<b>3</b>
<b>3. Resumen</b>	<b>5</b>
<b>4. Introducción</b>	<b>6</b>
4.1. Lugar de Trabajo . . . . .	6
4.2. Equipo de Trabajo . . . . .	6
4.3. Software y Conceptos Importantes . . . . .	6
4.3.1. REST . . . . .	6
4.3.2. API . . . . .	7
4.3.3. PostgreSQL . . . . .	7
4.3.4. Python . . . . .	7
4.3.5. JSON . . . . .	7
4.3.6. Redis . . . . .	8
4.3.7. BeCity . . . . .	8
4.3.8. SUR - Southern Urban Observatory . . . . .	8
4.4. Situación Previa . . . . .	8
4.4.1. BeCity . . . . .	8
4.4.2. SUR . . . . .	8
4.5. Descripción General del Trabajo . . . . .	8
4.5.1. BeCity . . . . .	8
4.5.2. SUR . . . . .	9
<b>5. Trabajo Realizado</b>	<b>10</b>
5.1. BeCity . . . . .	10
5.2. SUR . . . . .	10
5.2.1. Especificaciones . . . . .	10
5.2.2. Diseño Arquitectural . . . . .	11
5.2.3. Implementación . . . . .	11
<b>6. Conclusiones</b>	<b>12</b>

## Índice de figuras

---

### 3. Resumen

El trabajo se realizó entre Agosto y Octubre del 2015 en NIC Chile Research Labs, el laboratorio de investigación y desarrollo en protocolos de internet de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Consistió en

1. rediseñar e actualizar la API del backend de BeCity, una aplicación Android desarrollada en el laboratorio, para que se adhiriera al modelo REST.
2. diseñar e implementar un microservicio de manejo de imágenes para el proyecto SUR.

BeCity es una aplicación móvil Android orientada a ciclistas, la cual pretende facilitar el tránsito del usuario por la ciudad. Cuenta con un servicio de búsqueda de rutas con ciclovías, y permite grabar los recorridos del usuario, guardando estadísticas de velocidad, distancia recorrida, calorías quemadas, etc. Funciona bajo el paradigma cliente-servidor: existe un servicio central (el backend) que almacena y maneja los datos, y la aplicación Android (el cliente) se comunica con este servicio para actualizar y obtener información. Esta comunicación se realiza mediante una API (Application Programming Interface) usando HTTP (HyperText Transfer Protocol), sobre la cual se realizó el trabajo previamente mencionado.

El rediseño de esta API tenía como fin su modernización y adaptación a estándares modernos, en específico el modelo REST (REpresentational State Transfer)

SUR (Southern URban Observatory) es una plataforma que pretende centralizar todos los proyectos desarrollados en NICLabs. En este sentido, SUR se plantea como un backend unificado para los distintos servicios, ofreciendo funcionalidades comunes como autenticación de usuarios y manejo de recursos como imágenes, y generando una integración más estrecha entre los servicios.

El trabajo realizado en SUR consistió en el diseño y posterior implementación de un pequeño servidor de imágenes, capaz de administrar recursos gráficos de manera eficiente, escalable y simple. Dicho servicio se encuentra actualmente estable y funcionando en la plataforma SUR.

---

## 4. Introducción

### 4.1. Lugar de Trabajo

NIC Chile Research Labs (en adelante, NICLabs)[1], es el laboratorio de redes de la Facultad de Ciencias Físicas y Matemáticas de la Universidad de Chile. Fundado en 2007, sus principales áreas de investigación y desarrollo son protocolos de internet y seguridad informática.

El trabajo detallado en el presente informe se desarrolló entre octubre y diciembre del año 2015, de manera presencial en las instalaciones del laboratorio ubicadas en Av. Blanco Encalada 1975, Santiago, Chile.

### 4.2. Equipo de Trabajo

NICLabs cuenta tanto con desarrolladores e investigadores de tiempo completo, así como con estudiantes de pre- y postgrado. En específico, el trabajo se desarrolló en una sala compartida con otras 6 personas de distintas especialidades (diseñadores, ingenieros y estudiantes de pregrado).

En cuanto a los proyectos descritos en el presente informe, ambos se encontraban bajo la dirección de Felipe Lalanne[2], investigador y desarrollador experimentado. Cabe destacar que si bien existe un equipo de trabajo formalmente conformado para BeCity, éste sólo maneja y actualiza la versión actual del software, y por ende el practicante no se unió directamente a éste, ya que su trabajo consideraba el rediseño completo del backend. Por otro lado, no existe grupo de trabajo para SUR, y el practicante trabajó directamente bajo la tutela del supervisor.

### 4.3. Software y Conceptos Importantes

En términos generales, ambos proyectos se desarrollaron en el lenguaje de programación Python[4], utilizando además JetBrains Pycharm 5 Professional Edition[5] como IDE (*Integrated Development Environment* - Entorno de Desarrollo Integrado) principal.

#### 4.3.1. REST

El modelo REST (por sus siglas en inglés, “Representational State Transfer”) es una serie de restricciones arquitecturales que, al aplicarse a un servicio web, induce una serie de propiedades deseables (e.g. escalabilidad, estabilidad, rendimiento, etc). Al estar relacionado sólo con la estructura arquitectural del software, no especifica detalles de implementación, y es independiente del lenguaje de programación escogido.

A continuación, se expondrán brevemente las restricciones impuestas por el modelo REST:

1. **Modelo Cliente - Servidor** Debe existir una clara separación de responsabilidades mediante una interfaz uniforme. Por ejemplo, el servidor debe encargarse del almacenamiento y procesamiento de los datos, pero no de la representación visual de éstos, y vice-versa en el caso del cliente.
2. **Statelessness - Ausencia de Estados** El servidor no debe guardar información del estado del cliente entre solicitudes. El estado es almacenado por el cliente; cada solicitud debe ser independiente y contener toda la información necesaria para realizarse.
3. **Caché** El servidor debe poder guardar solicitudes en caché para poder responder futuras solicitudes de manera más expedita.

---

4. **Transparencia de Capas** El cliente no debe poder determinar si está conectado directamente al servidor o a un servicio intermedio (proxy, etc).

5. **Uniformidad de la Interfaz** A su vez puede separarse en:

- *Separación de recursos y su representación* La representación externa de los recursos (datos) debe ser independiente de como son almacenados por el servidor. A su vez, estas representaciones contienen toda la información necesaria para poder modificar el recurso original.
- *Identificación Uniforme de Recursos* Por ejemplo, mediante URI's[6].
- *Mensajes Autodescriptivos* Cada mensaje incluye información de cómo procesarse.

#### 4.3.2. API

Una API (Application Programming Interface), en el contexto de servicios web, es una interfaz de interacción entre los componentes de un modelo servidor-cliente. Específicamente, consiste en un conjunto de puntos de enlace (endpoints) públicos en el servidor, a través de los cuales un cliente puede comunicarse e interactuar con el servicio central, utilizando verbos estándar de HTTP (GET, PUT, POST y DELETE).

#### 4.3.3. PostgreSQL

PostgreSQL[3] es una de las principales implementaciones modernas del lenguaje SQL (Structured Query Language - Lenguaje Estructurado de Consultas) para bases de datos. Es un sistema de base de datos relacional ampliamente utilizado en la industria, y en específico, en BeCity se ocupa para el almacenamiento y consulta de los datos proporcionados por los usuarios.

#### 4.3.4. Python

Python es un lenguaje de programación interpretado, dinámico y de alto nivel, con énfasis en la fácil lectura de su código y su brevedad sintáctica. Es un lenguaje multipropósito, apto para programas de pequeña y gran escala, y con soporte para múltiples paradigmas de programación (orientado a objetos, funcional, imperativo, etc).

En el contexto del trabajo realizado, Python, principalmente en su versión 2.7 (aunque en algunas instancias se ocupó también la versión más nueva del lenguaje, 3.5), fue el lenguaje escogido para el desarrollo de los servicios. Esto, ya que debido a su popularidad, cuenta con una gran cantidad de librerías bien documentadas y mantenidas que facilitaron el desarrollo.

#### 4.3.5. JSON

JSON (JavaScript Object Notation), es un estándar abierto de codificación de datos para la comunicación entre servidor y cliente, en la cual los datos se codifican en pares atributo-valor en "cleartext" (es decir, texto legible por humanos). Por ejemplo, el siguiente extracto ilustra una posible codificación de la FCFM en JSON:

```
1 {  
2   "Nombre": "Facultad de Ciencias Fisicas y Matematicas",  
3   "Universidad": "Universidad de Chile",  
4   "Direccion": {  
5     "Calle": "Beauchef",  
6     "Numero": 850,  
7     "Comuna": "Santiago",
```

---

```
8     "Region" : " Metropolitana" ,
9     " Pais" : " CHILE"
10 }
11 }
```

#### 4.3.6. Redis

Redis puede describirse como un “almacén de estructuras de datos en disco”, y se utiliza comúnmente como base de datos y transmisor de mensajes; además, en conjunto con la librería *RQWorker*[cite here] de Python, se puede utilizar como una cola de ejecución de tareas. Éste fue el uso que se le dió en el servidor de imágenes, para ejecutar ciertas tareas de tiempo de ejecución más largo de manera diferida.

#### 4.3.7. BeCity

BeCity es una aplicación para smartphones con el sistema operativo Android, la cual tiene como fin la recolección de datos de ciclistas (por ejemplo, recorridos populares en una ciudad o comuna). Además, ayuda a los usuarios a transitar de manera segura por la ciudad, ofreciendo rutas con ciclovías e información atingente.

#### 4.3.8. SUR - Southern Urban Observatory

SUR es pretende ser un servicio de uso interno de INRIA Chile y NICLabs; una plataforma centralizada para todos los proyectos de ambos laboratorios. Pretende unificar de ésta manera muchos servicios que se encuentran replicados en los distintos proyectos (por ejemplo, manejo de usuarios) bajo una misma plataforma, integrandolo así de manera mucho más eficiente distintos proyectos y servicios otorgados por los laboratorios.

### 4.4. Situación Previa

#### 4.4.1. BeCity

En agosto del 2015, BeCity como aplicación ya se encontraba en funcionamiento, siendo distribuída a través de la tienda de aplicaciones de Google, la “Play Store”, y con su backend montado en el servidor de NICLabs. Sin embargo, si bien contaba con una API completa para la comunicación entre clientes y backend, ésta no seguía el estándar REST, y se había decidido reconstruirla desde cero siguiendo éste estándar, para asegurar su futura escalabilidad y estabilidad.

#### 4.4.2. SUR

Cuando el practicante comenzó su trabajo en NICLabs, SUR como proyecto todavía no existía, por lo que todo el trabajo se desarrolló desde cero.

### 4.5. Descripción General del Trabajo

#### 4.5.1. BeCity

El practicante diseñó e implementó parte de la nueva API, siguiendo las restricciones del modelo REST para servicios web. El trabajo se realizó por módulos de funcionalidad, siguiendo dos etapas por cada módulo:



---

1. *Diseño y Documentación*

En esta etapa, el practicante estudió la API existente del módulo escogido para entender el funcionamiento interno de éste, y los métodos que presentaba su interfaz. Luego, se diseñó y documentó la nueva versión del módulo, con especial énfasis en la nueva interfaz.

2. *Implementación y Testeo*

Luego de diseñado y documentado el módulo, se procedió a la implementación de las funcionalidades en código. Se utilizó el proceso de desarrollo “Test Driven Development”, es decir, primero se escribían tests para funcionalidades deseadas, y luego se implementaban éstas.

#### **4.5.2. SUR**

El trabajo realizado en SUR consistió en el desarrollo de un servicio de manejo de imágenes centralizado para todos los proyectos de la plataforma. El servicio debía básicamente almacenar y servir imágenes proporcionadas por los proyectos, por ejemplo para avatares de usuario. Se requería especial énfasis en la modularidad del servicio, de tal manera de que pudiese interactuar con cualquier otro proyecto que se adhiriera a la plataforma SUR en el futuro.

El trabajo consistió de dos etapas:

1. *Diseño del Servicio*

Para comenzar, se determinaron los requerimientos del servicio; las funcionalidades deseadas y la manera de interactuar con éste.

2. *Implementación*

Ya establecidos los requerimientos, se implementó el sistema deseado.

---

## 5. Trabajo Realizado

### 5.1. BeCity

### 5.2. SUR

Como se ha mencionado anteriormente, el proyecto SUR pretende ser una plataforma central para todos los proyectos de INRIA Chile y NICLabs - esto con el fin de centralizar varios servicios que actualmente se encuentran “duplicados” en todos los proyectos (por ejemplo, manejo de usuarios y de imágenes). Un servicio central como SUR permitiría que cierta información se pudiese compartir de manera transparente entre los proyectos y servicios, permitiendo por ejemplo que usuarios usen sus mismas credenciales de inicio de sesión, evitando la redundancia de tener que crear cuentas en cada uno de los servicios.

Bajo esta lógica fue que al practicante se le solicitó la implementación del servicio central de imágenes, el cual permitiese compartir de manera transparente y eficiente recursos de imágenes (por ejemplo, avatares de usuario) entre los proyectos.

#### 5.2.1. Especificaciones

El servicio en cuestión debía seguir una serie de especificaciones generales que aseguraban su eficiencia, escalabilidad y su compatibilidad con los demás servicios de la plataforma.

1. *Independencia:* La implementación del servicio de imágenes debía ser independiente y agnóstica de la implementación del resto de los servicios. Es decir, si bien podía existir interdependencia entre el servicio de imágenes, y (por ejemplo) el servicio de autenticación, ésta no podía depender de la implementación de cada uno. Esto para asegurar que en el futuro, los servicios pudiesen actualizarse o cambiarse completamente sin perturbar el resto de los servicios en la plataforma.
2. *Adherencia al modelo REST* El servicio debía basarse en el paradigma cliente-servidor, y seguir las especificaciones del modelo REST; comunicación mediante verbos HTTP, no almacenar estados, etc.
3. *Simplicidad* El código debía ser simple y conciso, y no agregar demasiada complejidad al sistema.
4. *Instantaneidad* Finalmente, la respuesta del servicio a solicitudes por parte de clientes debiese ser expedita, y no causar esperas prolongadas.

Además, se establecieron una serie de requerimientos funcionales de los servicios que debía proveer el sistema:

1. Almacenamiento de imágenes. Por simplicidad, sólo se debía aceptar imágenes en el formato *Portable Network Graphics*, **.png**.
2. Descarga de imágenes del servidor, en tamaños y resoluciones no necesariamente iguales a las originales.
3. Actualización de imágenes almacenadas.
4. Eliminación de imágenes almacenadas.

---

### 5.2.2. Diseño Arquitectural

En concordancia con las especificaciones y requerimientos anteriores, se llegó al diseño arquitectural detallado en esta sección.

El lenguaje escogido para la implementación fue nuevamente Python 2.7, principalmente por su amplio repertorio de librerías para desarrollo web, además de su simplicidad y elegancia sintáctica. Adicionalmente, es el lenguaje en el que están escritos todos los demás servicios de SUR, asegurando así una coherencia arquitectural interna de la plataforma.

Entre las librerías del lenguaje que fueron utilizadas, podemos destacar:

- Flask[7], la cual proporciona un framework básico para el desarrollo de aplicaciones web en Python. Por ejemplo, simplifica la ejecución de una aplicación Python como un servicio web asíncrono, y provee abstracciones para la implementación de endpoints (URL's) para interactuar con la aplicación.
- Pillow[10], librería de manejo de imágenes. Provee métodos de alteración de imágenes (transformaciones de tamaño, formato, etc).

Por otro lado, se descartó utilizar una base de datos para el servicio, ya que toda la información de las imágenes se puede manejar a través de *metadatos* y se almacenarse en el sistema de archivos. Agregar un sistema de bases de datos aumentaría la complejidad del sistema de manera innecesaria.

Finalmente, para poder asegurar la rapidez del servicio, se decidió realizar ciertas solicitudes de manera diferida – esto es, no se ejecutan inmediatamente, sino que quedan en una cola de solicitudes y eventualmente se resuelven. Esto se implementó para solicitudes por parte de clientes que no requieren una respuesta inmediata del servidor y que además son intensas en términos de tiempo de ejecución (por ejemplo, la carga de una nueva imagen al servicio, ya que ésta operación requiere además un procesamiento posterior). Esto se implementó mediante el uso de un servidor Redis en conjunto con el servicio de imágenes, el cual almacena y ejecuta las operaciones en la cola.

### 5.2.3. Implementación

La implementación del código se llevó a cabo siguiendo una estrategia agile y test-driven.

olo

---

## 6. Conclusiones

---

## Referencias

- [1] *NIC Chile Research Labs* - <http://niclabs.cl>
- [2] *Felipe Lalanne* - <https://github.com/pipex>
- [3] *PostgreSQL* - "The world's most advanced open source database."  
<http://www.postgresql.org/>
- [4] *The Python Programming Language* - <https://www.python.org/>
- [5] *JetBrains Pycharm* - <https://www.jetbrains.com/pycharm/>
- [6] *URI - Uniform Resource Identifier*: cadena de caracteres utilizada para identificar recursos específicos, siguiendo un esquema definido.  
[https://en.wikipedia.org/wiki/Uniform\\_Resource\\_Identifier](https://en.wikipedia.org/wiki/Uniform_Resource_Identifier)
- [7] *Flask is a microframework for Python based on Werkzeug, Jinja 2 and good intentions.*  
<http://flask.pocoo.org/>
- [8] *Restless - A lightweight REST miniframework for Python.*  
<https://restless.readthedocs.org/en/latest/>
- [9] *SQLAlchemy - The Python SQL Toolkit and Object Relational Mapper*  
<http://www.sqlalchemy.org/>
- [10] *PILLOW - Python Imaging Library fork*  
<https://pillow.readthedocs.org/en/3.1.x/>