
Linkedun Documentación Técnica

Publicación 1

Manual del desarrollador

dic. 01, 2016

1. AppServer	1
1.1. Documentación técnica del AppServer	1
1.1.1. Objetivo	1
1.1.2. Distribución	1
1.1.3. Compilación	2
1.1.4. Ejecución	2
1.1.5. Contacto	4
1.1.6. Referencias	4
1.2. Especificación Técnica APIRest	5
1.2.1. Objetivo	5
1.2.2. Servicio de autenticación	5
1.2.3. Servicio de búsqueda de profesionales	6
1.2.4. Servicio de actualización de ubicación	6
1.2.5. Servicio de vista de mis contactos	7
1.2.6. Servicio de agregar contacto	7
1.2.7. Servicio de vista respuesta de solicitudes de contactos	8
1.2.8. Servicio de vista de solicitudes pendientes de contactos	8
1.2.9. Servicio de recomendación	9
1.2.10. Servicio de consulta de usuarios más populares	10
1.2.11. Servicio de conversaciones	10
1.2.12. Servicio de envío de mensajes	11
1.2.13. Servicio de consulta de perfil de otros usuarios	11
1.2.14. Servicio de consulta de perfil	12
1.2.15. Servicio de administración de perfil de usuario	13
1.2.16. Servicio de administración de skills de usuario	13
1.2.17. Servicio de administración de agregar un skill al usuario	14
1.2.18. Servicio de administración de eliminación de un skill de usuario	14
1.2.19. Servicio de administración de obtención de un skill en particular	14
1.2.20. Servicio de administración de job positions de usuario	15
1.2.21. Servicio de administración de agregar un job positions al usuario	15
1.2.22. Servicio de administración de eliminación de un job position de usuario	16
1.2.23. Servicio de administración de obtención de un jobPosition en particular	16
1.2.24. Servicio de registración de token Firebase Cloud Messages	16
1.3. Pruebas Unitarias y funcionales	17
1.3.1. Objetivo	17
1.3.2. Realización	17
1.3.3. Ejecución	17
1.3.4. Coverage	17
1.3.5. Referencias	17
2. Aplicación Android	19

3. SharedServer	21
3.1. Objetivo	21
3.2. Requisitos:	21
3.3. Distribución	21
3.4. Construcción de Imágenes Docker (Builds)	21
3.5. Ejecución	22
3.6. Estructura	22
3.7. Base de datos	22
3.8. Administracion web	23
3.9. Contacto	23
3.10. Referencias	23

Documentación técnica del AppServer

Objetivo

Documentar información técnica del app server para poder compilar, ejecutar, debuggear, extender y mejorar el Application Server. Requisitos

El servidor de la aplicación Linkedun al cuál se consultan todos los servicios está desarrollado en C++, standard de 2011 (C++11) y utiliza las siguientes bibliotecas:

- Mongoose cpp[1] como web server para responder todas las consultas realizadas por la aplicación Android.
- Jsoncpp [2] biblioteca de JSON para C++.
- Curlpp [3] versión 0.7.3 o superior.
- GoogleTest [4] con tests unitarios que verifican que no se haya modificado funcionalidad existente. No es estrictamente necesario para continuar con el desarrollo de la aplicación, pero es ampliamente recomendado realizar tests y continuar con el desarrollo TDD [5].
- Leveldb [6] la base de datos clave-valor utilizada para guardar los datos de usuarios y la interacción entre los mismos.
- Log4Cpp [7] biblioteca de logging para C++.
- lcov & gcov [8] aplicaciones para verificar el code coverage que tienen los tests sobre la aplicación.
- CMake [9] para compilar el proyecto entero con todas las dependencias y también compilar y ejecutar los tests unitarios.

El código distribuido únicamente compila estáticamente la biblioteca de mongoose para asegurar que no existan problemas de compatibilidad al instalar, pero el resto de las bibliotecas se tienen que instalar y linkear dinámicamente con el ejecutable. Se utiliza bibliotecas de terceros en todo el código, especialmente las bibliotecas de mongoose y jsoncpp. En el caso de la base de datos y la ejecución de peticiones http al otro servidor mediante curlpp se hizo una interfaz para lograr separar la utilización de la biblioteca y poder reemplazar dicha interfaz en caso de querer cambiar alguna biblioteca de terceros, es decir, usar alguna otra biblioteca de peticiones http e invocar directamente un comando del sistema operativo por ejemplo.

Distribución

El código puede ser descargado de la página oficial de github [10]. Sistema operativo El desarrollo de la aplicación se ha realizado bajo el sistema operativo Linux. De querer desarrollar en Windows o Mac, deberá considerar la portabilidad de todas las dependencias a esos sistemas operativos y cómo realizar la compilación del proyecto.

Compilación

Para compilar el app server, puede seguir cualquiera de los dos métodos: 1. Utilizar la imagen docker y ejecutar el comando docker run que iniciará una imagen con todas las dependencias instaladas y tendrá listo el servidor ejecutándose[a], como se detalla en el manual de instalación de servidores [11]. 2. Instalar todas las dependencias mencionadas y realizar los siguientes pasos:

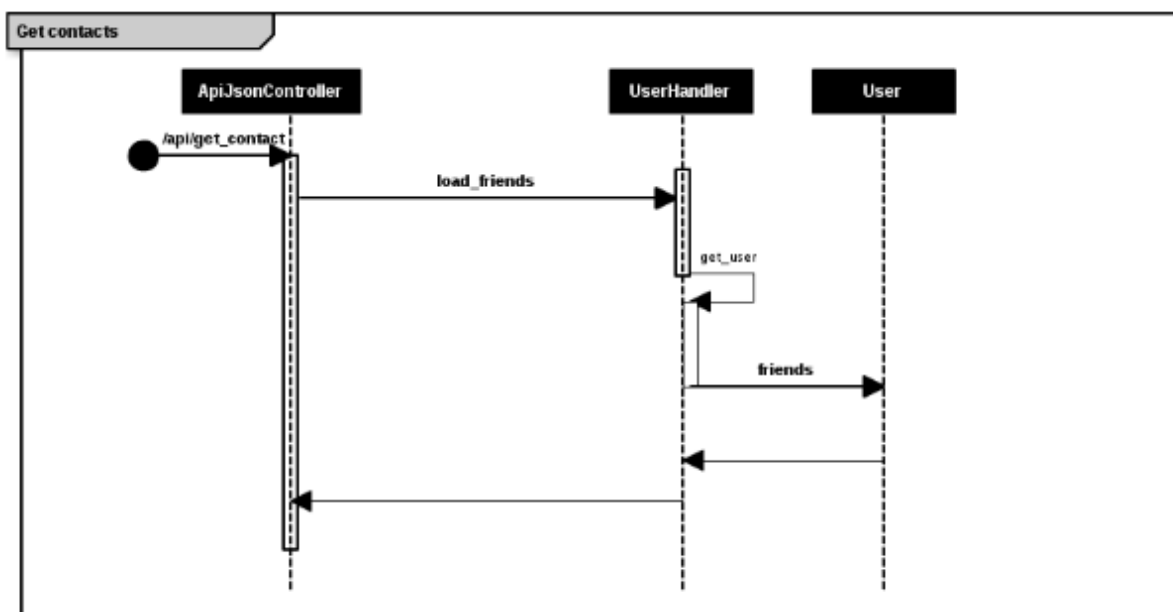
1. Ir al directorio “server”
2. Invocar el script: “. /appear.sh” que se encarga de invocar a cmake con las opciones de compilación predefinidas.

Ejecución

Debería estar en el directorio de la carpeta build y de ahí puede ejecutar el binario y con esa invocación el servidor estará escuchando conexiones en el puerto 8080. Al binario del servidor se le pueden pasar dos parámetros opcionales, en el orden mencionado, para cambiar el puerto donde el servidor recibe las peticiones y la url del shared server. Ejemplos de ejecución:

- ./src/server
- ./src/server 7000
- ./src/server 8080 localhost:5642
- ./src/server 7000 <https://guarded-sands-84788.herokuapp.com>
- ./src/server 8080 <https://guarded-sands-84788.herokuapp.com> (esta es la invocación por defecto)

Si se define el puerto donde se escucharán conexiones, se debe especificar si o si la url del shared server. Cualquier otra invocación es incorrecta y derivará en errores en la aplicación. Estructura La estructura de la aplicación es un Controller de Mongoose que envía respuestas en formato JSON (para más detalles ver la API especificada), la clase que se encarga de todas las solicitudes es ApiController y deriva las consultas en los handlers correspondientes. Un ejemplo con un diagrama de secuencia clarificará este párrafo.



Los handlers, los cuáles son consultados por el ApiController son:

- UserHandler: se encarga de la interacción entre los usuarios, las peticiones de contactar, votar, responder consultas y otros servicios.
- UserList: se encarga de mantener un listado actualizado de los usuarios, tanto darlos de alta en un registro, como consultar para ver qué usuarios están disponibles en el sistema.

- **GoogleCloudMessagesHandler**: se utiliza para enviar notificaciones cuando un usuario le envía un mensaje a otro.
- **DatabaseHandler**: se encarga de mantener realizar las operaciones de lectura y escritura en la base de datos. Simplemente tiene 3 métodos los cuales permiten realizar las operaciones necesarias.

Las entidades a destacar del sistema son:

- **User**: representa a los usuarios y es la clase fundamental dado que es una aplicación de interacción entre usuarios.
- **Chat**: representa un chat entre dos personas. El mismo tiene una lista de mensajes para mostrar que fueron enviando ambos y se puede reconstruir la conversación entera.
- **Message**: representa un mensaje enviado por un usuario. Se utiliza un timestamp guardado como string de forma de identificar el orden de los mensajes dentro del chat.
- **HerokuService**: deriva todas las consultas que deben ser realizadas al otro servidor y devuelve las respuestas a la aplicación de android tal cual son recibidas.
- **Log**: loguea toda la actividad del sistema en un archivo. Es una interfaz simple que utiliza Log4cpp para la finalidad de loguear.

Base de datos Debido a un requisito no funcional, se utilizó una base de datos clave-valor. Los usuarios, su información y las interacciones se persisten en la base de datos con una clave identificatoria según si es usuario o con un identificador especial para referirse a alguna entidad como la lista de usuarios. Las entidades de dominio están todas serializadas como JSON y se almacenan de esa manera en la base de datos. ACLARACIÓN: cuando se encuentra algo entre los caracteres "<>" significa que se reemplaza la cadena dentro por el valor de la entidad. Por ejemplo, si tengo user-<user-id> una clave de un usuario en particular será user-johndoeid.

```
Entidad
  Clave
  Valor
  User
  user-<user-id>
  {
    "user" :
    {
      "city" : "CABA",
      "dob" : "11/07/1991",
      "email" : "tomas@gmail.com",
      "fb_id" : "",
      "friends" : [],
      "job_positions" : [],
      "latitude" : "",
      "longitude" : "",
      "name" : "Tomas Mussi",
      "profile_photo" : "QdHVuZw==",
      "requests" : [],
      "skills" : [],
      "summary" : "Estudiante UBA.",
      "votes" : []
    }
  }
}

UserList
users
{
  "users": [
    "fb_id_tomas",
    "fb_id_luis"
  ]
}

Chat
```

```
chat-<userid-1>-<userid2>
{
  "messages" :
  [
    {
      "message" : "como va?",
      "receiver_id" : "recv",
      "sender_id" : "send",
      "timestamp" : "2016-11-14 15:28:43"
    },
    {
      "message" : "hola",
      "receiver_id" : "recv",
      "sender_id" : "send",
      "timestamp" : "2016-11-14 13:57:23"
    }
  ]
}
```

Respecto a un chat en especial, el <userid-1> será el del iniciador de la conversación, si uno quiere recuperar esa conversación y se desconoce quién inició la conversación, deberá probarse las dos combinaciones de ids para poder saber si existía una conversación previa entre los usuarios o hay que crear un chat nuevo. Esta resolución es engorrosa y en la próxima versión se contempla realizar un ordenamiento alfabético de los ids de usuario para no tener que probar las permutaciones de ids. API El app server respeta la API-REST[12] de interacción entre la aplicación Android y el servidor.

Contacto

Por cualquier inquietud, duda, consulta, usted puede enviar un email a la casilla de correos linkedunservicios@gmail.com y se le responderá a la brevedad. También puede contribuir al proyecto, reportar bugs, etc en la página de github del proyecto.

Referencias

- [1] Mongoose-cpp: <https://github.com/Gregwar/mongoose-cpp>
- [2] Jsoncpp: <https://github.com/open-source-parsers/jsoncpp>
- [3] Curlpp: <http://www.curlpp.org/>
- [4] Google Test: <https://github.com/google/googletest>
- [5] Test Driven Development: https://en.wikipedia.org/wiki/Test-driven_development
- [6] Level DB: <https://github.com/google/leveldb>
- [7] Log4Cpp: <http://log4cpp.sourceforge.net/>
- [8] Gcov: <https://gcc.gnu.org/onlinedocs/gcc/Gcov.html>
- [9] Cmake: <https://cmake.org/>
- [10] Código del repositorio oficial de Linkedun: <https://github.com/tomasmussi/taller2>
- [11] Manual de instalación de servidores: https://docs.google.com/document/d/1kZQ_5mHo0CJr2s9m6N1SgbrNDUm87tNNwNeo0STK0o/edit#heading=h.vo1cx285zg1d

Especificación Técnica APIRest

Objetivo

Definir el protocolo de comunicación entre la aplicación android y el app server que tenemos que implementar.
Funcionalidad: recordar que la funcionalidad pedida en el trabajo es:

- Loguearme en el sistema
- Ver/modificar mi perfil
- Buscar a otro usuario (1) (lookup)
- Invitar a conectar a otro usuario (add_contact)
- Aceptar/cancelar una solicitud de conectar (accept_contact)
- Get my contacts... (get_contacts)
- Recomendar a un usuario (votar por un usuario)
- Consultar usuarios más populares
- Chatear con usuarios con los que haya conectado
- Ver/modificar Skills
- Ver/modificar job positions
- Cambiar mi ubicación actual

Servicio de autenticación

```

LoginDTO
URL
    URL: http://<appserver>/api/fb_login
    HTTP VERB
    GET
    PARAMETROS
    {
        "api_sec" : "api_sec_hash",
        "fb_user_id": "a_fb_user_id"
    }
    RESPUESTA
    {
        "data":
        {
            "token": "d975c4e93772b4fcbdf65ce62a1e14f6",
            "user_exists": "false"
        }
        ,
        "errors": [
        ]
    }
    RESPUESTA
    {
        "data": null,
        "errors": [
            "status" : "ERROR",
            "message" : "Facebook user ID invalido"
        ]
    }
    
```

Servicio de búsqueda de profesionales

```
URL
    URL: http://<appserver>/api/lookup
    HTTP VERB
    GET
    PARAMETROS
    {
        "token": "hash",
        "query" : "tomas"
    }
    RESPUESTA
    {
        "data": [
            [
                {
                    "distance": "3.978116",
                    "fb_id": "tomas",
                    "is_contact": "false",
                    "name": "Tomas mussi",
                    "photo": "bas64encode..=="
                }
            ]
        ],
        "errors": [
        ]
    }
}
```

Servicio de actualización de ubicación

```
URL
    URL: http://<appserver>/api/location
    HTTP VERB
    POST
    PARAMETROS
    {
        "token": "hash",
        "latitude": "-34.595241",
        "longitude": "-58.402460"
    }
    RESPUESTA
    HTTP CODE 200
    {
        "data":
        {
            "message": "Ubicacion de usuario actualizada",
            "status": "OK"
        }
        ,
        "errors": [
        ]
    }

    HTTP CODE 400
    {
        "data": [
        ],
        "errors": [
        ]
    }
```

```
{
    "message": "Latitud o longitud vacios",
    "status": "ERROR"
}

]
```

Servicio de vista de mis contactos

```
URL
URL: http://<appserver>/api/get_contacts
HTTP VERB
GET
PARAMETROS
{
    "token": "hash"
}

RESPUESTA
{
    "data": [
        {
            "distance": "3.978116",
            "fb_id": "tomas",
            "is_contact": "true",
            "name": "Tomas mussi",
            "photo": "bas64encode..=="
        }
    ],
    "errors": [
    ]
}
```

Servicio de agregar contacto

```
URL
URL: http://<appserver>/api/contact/
HTTP VERB
POST
PARAMETROS
{
    "token": "hash",
    "contact_fb_id" : "a_user_fb_id"
}

RESPUESTA
{
    "data": [
        {
            "message": "Enviada solicitud a contacto",
            "status": "OK"
        }
    ],
    "errors": [
    ]
}
```

```
RESPUESTA ERROR
{
  "data": [],
  "errors": [
    {
      "status" : "ERROR",
      "message" : "El usuario no existe"
    }
  ]
}
```

Servicio de vista respuesta de solicitudes de contactos

```
URL
URL: http://<appserver>/api/contact/response
HTTP VERB
POST
PARAMETROS
{
  "token": "hash",
  "contact_fb_id" : "a_user_fb_id",
  "accept": "true" / "false"
}

RESPUESTA
{
  "data": [
    {
      "message": "Respondida solicitud a contacto",
      "status": "OK"
    }
  ],
  "errors": [
  ]
}

RESPUESTA ERROR
{
  "data": [],
  "errors": [
    {
      "status" : "ERROR",
      "message" : "El usuario no existe"
    }
  ]
}
```

Servicio de vista de solicitudes pendientes de contactos

```
URL
URL: http://<appserver>/api/contact/
HTTP VERB
GET
PARAMETROS
{
  "token": "hash"
}

RESPUESTA
{
  "data": [
```

```
[
  {
    "fb_id":"tomas",
    "is_contact":"false",
    "name":"Tomas mussi",
    "photo":"bas64encode..=="
  },
  {
    "fb_id":"luis",
    "is_contact":"false",
    "name":"Luis Arancibia",
    "photo":"bas64encode..=="
  }
],
"errors":[]
]
}

RESPUESTA ERROR
{
  "data":[],
  "errors": [
    "status" : "ERROR",
    "message" : "El usuario no existe"
  ]
}
```

Servicio de recomendación

```
URL
URL: http://<appserver>/api/vote
HTTP VERB
POST
PARAMETROS
{
  "token": "hash",
  "contact_fb_id" : "a_user_fb_id"
}

RESPUESTA
{
  "data": [
    {
      "message": "Enviada votación a contacto",
      "status": "OK"
    }
  ],
  "errors": [
  ]
}

RESPUESTA ERROR
{
  "data": [
  ],
```

```
    "errors":[
      "message":"Usuario [a_user_fb_id] no existe o no es valido",
      "status":"ERROR"
    ]
  }
```

Servicio de consulta de usuarios más populares

```
URL
  URL: http://<appserver>/api/vote/popular
  HTTP VERB
  GET
  PARAMETROS
  {
    "token": "hash",
  }

  RESPUESTA
  {
    "data":[
      [
        {
          "fb_id":"tomas",
          "is_contact" : "true",
          "name" : "Tomas Mussi",
          "photo" : "dsjagkjasg...==",
          "votes":1
        },
        {
          "fb_id":"aran.com.ar@gmail.com",
          "is_contact" : "false",
          "name" : "Luis Arancibia",
          "photo" : "dsjagkjasg...==",
          "votes":0
        },
        {
          "fb_id":"a-fb-user-id",
          "is_contact" : "false",
          "name" : "User Name",
          "photo" : "dsjagkjasg...==",
          "votes":0
        }
      ]
    ],
    "errors":[
    ]
  }
```

Servicio de conversaciones

```
URL
  URL: http://<appserver>/api/message/
  HTTP VERB
  GET
  PARAMETROS
  {
    "token": "hash",
    "contact_fb_id": "tomas"
  }
```

```

RESPUESTA
{
  "data":
  [
    {
      "message": "como va?",
      "receiver_id": "tomas",
      "sender_id": "aran.com.ar@gmail.com",
      "timestamp": "2016-11-05 18:48:03"
    },
    {
      "message": "hola",
      "receiver_id": "aran.com.ar@gmail.com",
      "sender_id": "tomas",
      "timestamp": "2016-11-05 18:44:22"
    }
  ]
  ,
  "errors": [

]
}

```

Servicio de envío de mensajes

```

URL
URL: http://<appserver>/api/message
HTTP VERB
POST
PARAMETROS
{
  "token": "hash",
  "contact_fb_id" : "tomas",
  "message" : "Hola, como te va?"
}

RESPUESTA
{
  "data":
  {
    "message": "Mensaje enviado exitosamente",
    "status": "OK"
  }
  ,
  "errors": {}
}

```

Servicio de consulta de perfil de otros usuarios

```

URL
URL: http://<appserver>/api/profile/others
HTTP VERB
GET
PARAMETROS
{
  "token": "hash",
  "contact_fb_id": "other_fb_id"
}

RESPUESTA

```

```
{
  "data": [
    {
      "user": {
        "city": "",
        "contacts": 1,
        "dob": "11/07/1991",
        "email": "",
        "job_positions": [

        ],
        "name": "Tomas Mussi",
        "profile_photo": "",
        "requests": [

        ],
        "skills": [

        ],
        "summary": "A summary"
      }
    ]
  ],
  "errors": [
  ]
}
```

Servicio de consulta de perfil

```
URL
URL: http://<appserver>/api/profile
HTTP VERB
GET
PARAMETROS
{
  "token": "hash"
}

RESPUESTA
{
  "data" [
    {
      "user": {
        "city" : "Ciudad de Buenos Aires",
        "contacts": 1,
        "dob" : "11/07/1991",
        "email": "tomasmussi@gmail.com",
        "job_positions": [

        ],
        "name": "",
        "profile_photo" : "QURQIEdtYkgK...mdHVuZw==",
        "requests": [

        ],
        "skills": [
          "c",
          "c++"
        ],

```



```

        "summary" : "Estudiante de ingenieria informatica de la UBA."
    }
}
], [b]
"errors": [
]
}

```

Servicio de administración de perfil de usuario

```

URL
URL: http://<appserver>/api/profile/
HTTP VERB
POST
PARAMETROS
{
    "token": "hash",
    "name": "Tomas Mussi Reyer",
    "summary" : "Estudiante de ingenieria informatica de la UBA.",
    "profile_photo" : "QRQIEdtYkgK...mdHVuZw==",
}

RESPUESTA
{
    "data": [
        {
            "message": "Usuario modificado exitosamente",
            "status": "OK"
        }
    ],
    "errors": []
}

```

Servicio de administración de skills de usuario

```

URL
URL: http://<appserver>/api/skills
HTTP VERB
GET
PARAMETROS

RESPUESTA
{
    "skills": [{
        "name": "c",
        "description": "c programming language",
        "category": "software"
    }, {
        "name": "PMI",
        "description": "Project Management Institute",
        "category": "management"
    }],
    "metadata": {
        "version": "0.1",
        "count": 2
    }
}

```

Servicio de administración de agregar un skill al usuario

```
URL
    URL: http://<appserver>/api/skills
    HTTP VERB
    POST
    PARAMETROS
    {
        "token": "hash",
        "skill": "c"
    }
    RESPUESTA
    {
        "data": [
            {
                "message": "Skill agregado",
                "status": "OK"
            }
        ],
        "errors": []
    }
```

Servicio de administración de eliminación de un skill de usuario

```
URL
    URL: http://<appserver>/api/skills
    HTTP VERB
    DELETE
    PARAMETROS
    {
        "token": "hash",
        "skill": "c"
    }
    RESPUESTA
    {
        "data": [
            {
                "message": "Skill eliminado",
                "status": "OK"
            }
        ],
        "errors": []
    }
```

Servicio de administración de obtención de un skill en particular

```
URL
    URL: http://<appserver>/api/skill
    HTTP VERB
    GET
    PARAMETROS
    {
        "token": "hash",
        "name": "c",
    }
    RESPUESTA
    {
        "data": [
```

```
{
    "name": "c",
    "description": "c programming language",
    "category": "software"
}
],
"errors": []
}
```

Servicio de administración de job positions de usuario

```
URL
URL: http://<appserver>/api/job_positions
HTTP VERB
GET
PARAMETROS

RESPUESTA
{
    "job_positions": [{
        "name": "developer",
        "description": " a software developer"
        "category": "software"
    }, {

        "name": "project manager",
        "description": " a project manager"
        "category": "management"
    }, {
        "name": "dj",
        "category": "music"
    }],
    "metadata": {
        "version": "0.1",
        "count": 3
    }
}
```

Servicio de administración de agregar un job positions al usuario

```
URL
URL: http://<appserver>/api/job_positions
HTTP VERB
POST
PARAMETROS
{
    "token": "hash",
    "job": "developer"
}

RESPUESTA
{
    "data": [
        {
            "message": "Job position agregado",
            "status": "OK"
        }
    ]
}
```

```
    ],  
    "errors": []  
}
```

Servicio de administración de eliminación de un job position de usuario

```
URL  
URL: http://<appserver>/api/job_positions  
HTTP VERB  
DELETE  
PARAMETROS  
{  
    "token": "hash",  
    "job": "developer"  
}  
  
RESPUESTA  
{  
    "data": [  
        {  
            "message": "Job position eliminado",  
            "status": "OK"  
        }  
    ],  
    "errors": []  
}
```

Servicio de administración de obtención de un jobPosition en particular

```
URL  
URL: http://<appserver>/api/job_position  
HTTP VERB  
GET  
PARAMETROS  
{  
    "token": "hash",  
    "name": "developer",  
}  
  
RESPUESTA  
{  
    "data": [  
        {  
            "name": "developer",  
            "description": " a software developer"  
            "category": "software"  
        }  
    ],  
    "errors": []  
}
```

Servicio de registración de token Firebase Cloud Messages

```
URL  
URL: http://<appserver>/api/token_FCM  
HTTP VERB  
POST  
PARAMETROS
```

```

    {
        "token": "hash",
        "token_FCM": "hash",
    }

    RESPUESTA
    {
        "data":
    {
        "message": "El token ha sido registrado satisfactoriamente",
        "status": "OK"
    },
        "errors": []
    }

```

Pruebas Unitarias y funcionales

Objetivo

Exponer al programador cómo se realizaron tests sobre la aplicación, describir paso a paso la forma de ejecutar los tests y explicar cómo mantener y mejorar los mismos.

Realización

Los tests están realizados para verificar la funcionalidad principalmente del app server, desarrollado en C++. Dichos tests se realizaron con el framework de testing de Google [1]. Los mismos verifican el comportamiento unitario de las clases del servidor. Los tests de funcionalidad de la ApiREST se desarrollaron en python.

Ejecución

Para ejecutar las pruebas, deben seguirse los pasos de compilación del servidor especificados en la documentación técnica del app server [2], pero en vez de ejecutar el binario del servidor src/server, se debe ejecutar el binario generado testing que se encuentra dentro de la carpeta test.

Coverage

Para medir el coverage se debe ejecutar el script ./coverage.sh. Si se tiene instalado el firefox se abre automáticamente, sino se debe abrir la web generada en el archivo index en la carpeta coverage dentro de este mismo directorio. Este test queda a la espera de que uno cierre el test dado que se deben ejecutar las pruebas de la Api-Rest y las mismas necesitan que el AppServer esté corriendo. Se debe ir al directorio /server/test/ y ejecutar bash ./python/test.sh cuando finalicen los test se puede finalizar el Appserver.

Referencias

- [1]: Google test framework para c/c++: <https://github.com/google/googletest>
- [2]: Documentación técnica app server: https://docs.google.com/document/d/1Db988RfcEOvXmyQ6Ple9AO_NscSqZUGfGTnTWRT6-60/edit#heading=h.abohzj50eg4c

Aplicación Android

SharedServer

Objetivo

Documentar información técnica del *Shared Server* para poder ejecutar, extender y mejorar dicho servidor.

Requisitos:

El servidor de la aplicación *Linkedun* al cuál se consultan todos los servicios está desarrollado en *NodeJS*, utiliza los siguientes módulos:

- **pg:** V6.1.0 “PostgreSQL” [2]
- **Express:** V4.13.3 “Framework de infraestructura web rápida, minimalista y flexible para las aplicaciones Node.js.”[3]
- **Winston:** V^2.3.0 “Logger”[4]
- **pg-promise:** V ^3.2.3 “Promises interface for PostgreSQL”[5]
- **bluebird:** V^3.3.4 “Promises in NodeJS”[6]
- **body-parser:** V ^1.15.2 “Node.js body parsing middleware”[7]
- **ejs:** V2.4.1 “Embedded JavaScript templates”[8]

Distribución

El código puede ser descargado de la página oficial de *github* [10]. Sistema operativo El desarrollo de la aplicación se ha realizado bajo el sistema operativo Linux. De querer desarrollar en Windows o Mac, deberá considerar la portabilidad de todas las dependencias a esos sistemas operativos y cómo realizar la compilación del proyecto.

Construcción de Imágenes Docker (Builds)

Este software está diseñado para ejecutarse en un contenedor Docker. Para ello, primero necesitamos construir la imagen de PostgreSQL para la base de datos que le da infraestructura a este servidor.

Posicionarse dentro de la carpeta `/database/` dentro de la carpeta principal del proyecto descargado por git. Luego ejecutar el siguiente comando:

- `docker build -t sharedserver-database .`

Esto generará la dockerización de la base de datos, luego hay que posicionarse en la carpeta principal (`taller2/SharedServerHeroku/`) y ejecutar el siguiente comando:

- docker-compose up

Este comando dockerizara NodeJs y unirá las dos aplicaciones (PostgreSQL y NodeJs) quedando el servidor operativo. Para detener la ejecución del servidor ejecutar Control + C.

Ejecución

A través del comando:

- docker-compose up

Ejecutado en la carpeta principal, luego de haber realizado el instructivo de construcción, deberá dejar la aplicación de NodeJS escuchando en el puerto 5000 del servidor localhost. Asi como tambien la base de datos escuchando en el puerto 5432 del servidor localhost. Listas para ser utilizadas, exponiendo una web de administración descrita en el Manual del Administrador Shared.

Estructura

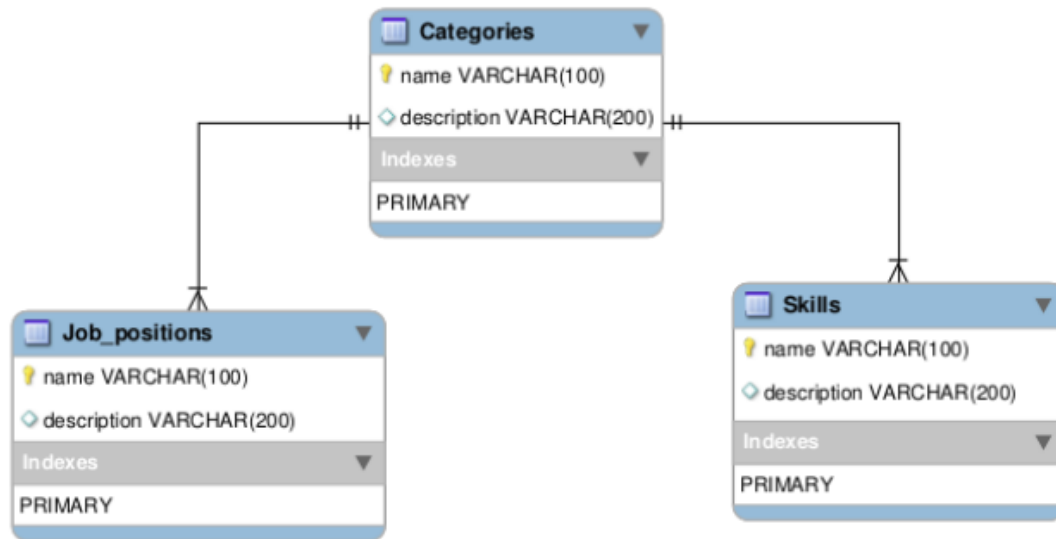
Las consultas a este servidor están expuestas en el index.js en el directorio raíz del proyecto y son realizadas a través la API [1], resumidamente posee los siguientes servicios:

```
app.get('/job_positions', db.getAllJobPositions);
app.get('/job_positions/categories/:name', db.getAllJobPositionsByCategory);
app.get('/categories', db.getAllCategories);
app.get('/skills', db.getAllSkills);
app.post('/skills/categories/:category', db.postskill);
app.post('/job_positions/categories/:category', db.postJobPosition);
app.post('/categories/', db.postCategory);
app.put('/categories/:name', db.putCategory);
app.put('/job_positions/categories/:category/:name', db.putJobPosition);
app.put('/skills/categories/:category/:name', db.putSkill);
app.delete('/job_positions/categories/:category/:name', db.deleteJobPosition);
app.delete('/skills/categories/:category/:name', db.deleteSkill);
app.delete('/categories/:name', db.deleteCategory);
```

Los cuales se encuentran implementados bajo los métodos que expone el módulo *queries.js*, donde se encuentran los accesos a la base de datos pertinentes en cada servicio. Todas las respuestas de dicho servicio son en formato JSON. En cuanto a la web de administración esta se encuentra en el directorio *view/pages/* donde estan los archivos principales de la web y los controladores separados por skills, jobPosition y category.

Base de datos

Este servidor posee una base de datos *PostgreSQL* donde se almacenan los *jobPositions*, los *skills* y las *categories*. La estructura de la misma puede ser representada a través del siguiente diagrama:



Para una mejor comprensión puede recurrir al archivo `sharedserver.sql` el cual contendrá la creación de las 3 tablas (`category`, `job_position` y `skill`) con sus respectivos atributos y claves.

Administracion web

El Shared Server posee una web de administración online en la plataforma Heroku [9]. Esta provee de las acciones denominadas alta, baja, modificación y borrado de `jobPositions`, `skills` y `categories` donde se puede ver este tema más detallado en el Manual del Administrador Shared. Dicha web se encuentra en la url : <https://guarded-sands-84788.herokuapp.com/#/> Cabe destacar que este dominio en Heroku posee las mismas características que el dominio local en el cual se trato en este documento. API El Shared Server respeta la API-REST[1] para la interacción con el App Server y posteriormente la aplicación Android.

Contacto

Por cualquier inquietud, duda, consulta, usted puede enviar un email a la casilla de correos linkedunservices@gmail.com y se le responderá a la brevedad. También puede contribuir al proyecto, reportar bugs, etc en la página de github del proyecto.

Referencias

- [1] API Restful Shared Server : <http://rebilly.github.io/ReDoc/?url=https://gist.githubusercontent.com/NickCis/d6a8132a228440c41889b4e0003efc3b/raw/jobify-shared-api.yaml>
- [2]pg : <https://www.npmjs.com/package/pg>
- [3]Express : <https://www.npmjs.com/package/express>
- [4]Winston : <https://www.npmjs.com/package/winston>
- [5]pg-promise: <https://www.npmjs.com/package/pg-promise>
- [6]bluebird : <https://www.npmjs.com/package/bluebird>
- [7]body-parser: <https://www.npmjs.com/package/body-parser>
- [8]ejs : <https://www.npmjs.com/package/ejs>
- [9]Heroku : <https://www.heroku.com/>