# Jobify Documentation

***Release 2.4.2***

**Carballo Matias, Guzzardi Gonzalo y Marshall Juan Patricio**

**Dec 06, 2016**

CONTENTS

Contents:

CHAPTER

ONE

# MANUAL DE PROYECTO

En este manual, vamos a entrar en detalle cómo fue la organización del proyecto dentro del equipo de trabajo. Desde como fuimos decidiendo, que problemáticas tuvimos, hasta que cosas cambiaríamos en la realización de un proyecto futuro.

## 1.1 Ambiente de trabajo

El ambiente de trabajo dentro del equipo siempre se caracterizó por 3 cosas: cooperación, buena onda entre los integrantes y bastante libre dentro de todo. Fue cooperativo principalmente porque frente a cada dificultad que se le presentaba a algún integrante del equipo, siempre el resto ofrecía ayudarlo y analizar con él el problema en cuestión. La parte propia de un ambiente con buenas relaciones, se debió más a que todos los individuos del equipo nos conocíamos previamente y hasta habíamos realizado el trabajo práctico de Taller 1 en conjunto. Más que un equipo de trabajo, somos un grupo de amigos trabajando juntos. Esto puede ser tanto algo muy bueno, como tener sus contras. Luego, hablamos de un ambiente bastante libre porque, más que con las fechas de los checkpoint y la entrega, no teníamos fechas de finalización propias o internas dentro del equipo. A su vez, cada uno fue realizando partes de su tarea, o épica, sin presión de tiempo internamente.

## 1.2 Herramientas utilizadas

Como herramientas de coordinación y comunicación dentro del equipo, utilizamos principalmente medios bastante informales. Desde un grupo de whatsapp, a cadenas de mails, reuniones en persona, llamadas de skype, google hangouts, etc. Mientras que como herramientas para el desarrollo en sí del trabajo, utilizamos todas las herramientas que estaban definidas en el enunciado del trabajo y decidimos por nosotros las que no estaban definidas. Trabajamos con computadoras con sistemas operativos Linux, (distribución Ubuntu y Mint principalmente). Para la parte de código C++, utilizamos la IDE codeblocks. Para el app de android utilizamos Android Studio.

## 1.3 Integrantes del equipo

El equipo estuvo formado por Baliña Federico, Carballo Matías, Guzzardi Gonzalo y Marshall Juan Patricio. Lamentablemente, el intergante Baliña Federico, decidió dejar la materia porque honestamente veía que con su trabajo, resto de las materias que cursaba y responsabilidades personales, le iba a ser muy difícil mantener al día su cuota de trabajo para el proyecto, y no se sentía a gusto aprobando una materia sin dar su mayor esfuerzo. Igualmente, nos avisó con el tiempo suficiente para poder reorganizar las tareas dentro del equipo. Este movimiento de personas en equipos, es muy común en proyectos informáticos. Por eso, es muy importante a la hora de la organización y el desarrollo poder acostumbrarse a cambios dentro de las estructuras de trabajo y adaptarse correctamente.

## 1.4 Planificación

Originalmente, partimos desde que leímos el enunciado al principio, con un listado de features asignados a cada checkpoint del trabajo. A medida que fueron pasando las semanas, con una serie de fechas de parciales y entregas de otras materias bastante desafortunadas, estuvimos pasando features que veíamos imposibles resolver para el checkpoint definido, posponiéndolo al siguiente checkpoint y en consecuencia a la parte final del trabajo. Tuvimos que hacer una replanificación de qué tema general le tocaba a cada integrante en consecuencia de la baja de un miembro del equipo. Por suerte, esta refactorización no presentó serios problemas, ya que hasta ayudó al pasaje de cosas el integrante que se fue. Como principal error dentro de la planificación general del proyecto, sentimos que fue una mala decisión dejar tanta funcionalidad en la última parte del desarrollo en el plan inicial. Tendríamos que haber previsto que en cada iteración del desarrollo, algún remanente sin implementar se iría agregando a las funcionalidades por hacer de la siguiente iteración, llevando a una última etapa a las corridas. Esto se agravió terriblemente con la decisión errónea de terminar antes trabajos de otras materias con fechas más cercanas y con el poco alcance que le dabamos a de fechas de parciales/recuperatorios de cada integrante del equipo.

## 1.5 Temas que quedaron pendientes

Dentro de los temas que quedaron pendientes, destacamos una mejor documentación en general. Dejamos de lado frente al desarrollo, todo lo que era parte de la documentación del proyecto. Nos quedaron todos los escritos de documentación, acumulados para el final del trabajo. Además, no llegamos a implementar el chat entre usuarios, el login de facebook ni realizar búsquedas complejas desde el cliente android (sí en el app server).

## 1.6 Elementos que subestimamos y resultaron complejas

Lograr una documentación robusta y completa. Implementar las llamadas http en el cliente android. Como hay que hacerlas fuera del UI thread de android, terminaron siendo más complicadas de lo que inicialmente habíamos estimado. Llegar a implementar un conjunto de tests completos, tanto de integración como tests unitarios, al mismo tiempo que al ir avanzando con el desarrollo.

## 1.7 Conclusión

Al desarrollar esta aplicación nos encontramos con nuevos desafíos que anteriormente no habíamos enfrentado. Primeramente el hecho de manejarnos en múltiples lenguajes y platamormas. Luego el manejo de un proyecto a través de git con su sistema de branches y el uso de issues. Por ultimo, el requerimiento de mantener un codigo de alta calidad, con extenso testeo para mantener una gran estabilidad y seguridad en el proyecto.

Debido a esto se necesitó desde el principio un claro diseño y una organización clara de las tareas a seguir. Una vez que nos adaptamos al funcionamiento de git esta herramienta nos resultó de gran ayuda con esto.

CHAPTER

TWO

# DOCUMENTACIÓN TÉCNICA SERVIDOR

## 2.1 Ambiente de desarrollo

### 2.1.1 Servidor

Para el desarrollo del servidor, se utilizó Codeblocks .

Sin embargo, más allá de su uso para facilitar el desarrollo, la compilación se hizo siempre a partir de cmake.

## 2.2 Librerías

Para manejar distintos recursos en el proceso de desarrollo se emplearon librerías open source.

### 2.2.1 Servidor http

Para realizar las operaciones para manejar el servidor http se empleó mongoose, el cual facilitó en gran medida el manejo de http requests.

### 2.2.2 Base de datos

Para el manejo de la base de datos por parte del servidor, se utilizó una base de datos no relacional del tipo *clave/valor* llamada leveldb.

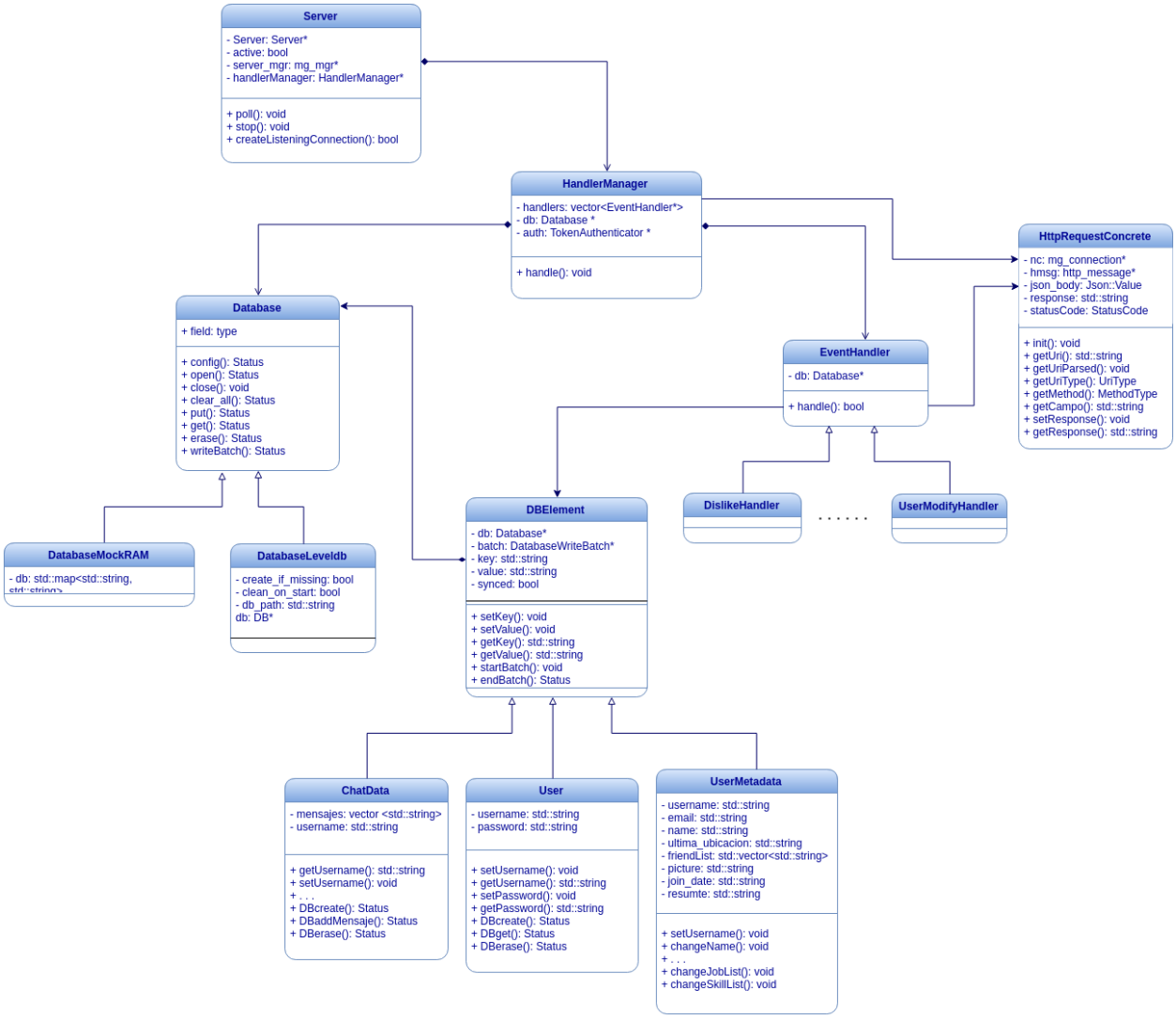
### 2.2.3 Datos json

Para la manipulación de datos json se hizo uso de la librerías jsoncpp. Los json se usaron tanto para el envío de datos a través de los mensajes http como en la persistencia de datos del lado del servidor.

## 2.3 Diseño

Para poder visualizar la arquitectura del proyecto se presenta aquí un diagrama UML que muestra las clases más importantes del programa.

Aquí se puede ver como Server es quien se encarga de mantener la conexión, pero HandlerManager es la clase que maneja todas las httprequests, seleccionando cada handler acorde a la acción que deba hacer. A su vez, cada handler


Server
- Server: Server*
- active: bool
- server_mgr: mg_mgr*
- handlerManager: HandlerManager*
+ poll(): void
+ stop(): void
+ createListeningConnection(): bool
HandlerManager
- handlers: vector<EventHandler*>
- db: Database *
- auth: TokenAuthenticator *
+ handle(): void
HttpRequestConcrete
- nc: mg_connection*
- hmsg: http_message*
- json_body: Json::Value
- response: std::string
- statusCode: StatusCode
+ init(): void
+ getUri(): std::string
+ getUriParsed(): void
+ getUriType(): UriType
+ getMethod(): MethodType
+ getCampo(): std::string
+ setResponse(): void
+ getResponse(): std::string
Database
+ field: type
+ config(): Status
+ open(): Status
+ close(): void
+ clear_all(): Status
+ put(): Status
+ get(): Status
+ erase(): Status
+ writeBatch(): Status
EventHandler
- db: Database*
+ handle(): bool
DatabaseMockRAM
- db: std::map<std::string,
DatabaseLeveldb
- create_if_missing: bool
- clean_on_start: bool
- db_path: std::string
db: DB*
DBElement
- db: Database*
- batch: DatabaseWriteBatch*
- key: std::string
- value: std::string
- synced: bool
+ setKey(): void
+ setValue(): void
+ getKey(): std::string
+ getValue(): std::string
+ startBatch(): void
+ endBatch(): Status
DislikeHandler
. . . . . .
UserModifyHandler
ChatData
- mensajes: vector <std::string>
- username: std::string
+ getUsername(): std::string
+ setUsername(): void
+ . . .
+ DBcreate(): Status
+ DBaddMensaje(): Status
+ DBerase(): Status
User
- username: std::string
- password: std::string
+ setUsername(): void
+ getUsername(): std::string
+ setPassword(): void
+ getPassword(): std::string
+ DBcreate(): Status
+ DBget(): Status
+ DBerase(): Status
UserMetadata
- username: std::string
- email: std::string
- name: std::string
- ultima_ubicacion: std::string
- friendList: std::vector<std::string>
- picture: std::string
- join_date: std::string
- resumte: std::string
+ setUsername(): void
+ changeName(): void
+ . . .
+ changeJobList(): void
+ changeSkillList(): void

empleará los DBElements que necesite par subir, modificar o extraer datos de la clase Database, que es quien se encarga de manejar la base de datos.

## 2.4 Documentación de REST API

### 2.4.1 Endpoints validos

- POST /users/ sign up
- POST /sessions/ login
- DELETE /sessions/ logout
- DELETE /users/ borrar usuario
- GET /users/'username' traer info del usuario
- PUT /users/'username' modificar perfil de tal usuario
- PUT /like/'username'/'username2' like del 1er usuario al 2do
- DELETE /like/'username'/'username2' dislike del 1er usuario al 2do
- POST /users/'username'/addFriend/'username2' manda solicitud de amistad username a username2
- POST /users/'username'/acceptFriend/'username2' username acepta la solicitud de amistad de username2
- GET /top_skill/ devuelve el skill con mayor ranking
- GET /top_job/ devuelve el job con mayor ranking
- GET /users/'username'/profile/ trae el perfil del usuario username
- GET /chat/'username'/'username2' trae el chat entre el username y el username2
- POST /chat/'username'/'username2' postea un mensaje al chat

### 2.4.2 Otros

- cualquier otro request es invalido, y se devuelve error

CHAPTER

THREE

# ADMINISTRACIÓN

## 3.1 Manual de instalación

La aplicación Jobify está pensada para correr en linux 64 bits. A su vez se requieren tener ciertas librerías y programas instalados. Por esto se pueden proseguir dos métodos distintos con la intención de ejecutar el servidor. Si se desea instalar las librerías y programas sobre el equipo y luego correr el servidor sobre estas instalaciones se puede seguir la guía de instalación sobre el equipo.

## 3.2 Instalación en Equipo

La aplicación Jobify está pensada para correr en linux 64 bits. A su vez se requieren tener ciertas librerías y programas instalados. Esta guía de instalación detalla como conseguir esto, orientando las consignas para la versión Ubuntu 14.04. Sin embargo esto debería poder funcionar con otras versiones de Ubuntu y ser análoga a otras versiones de Linux.

### 3.2.1 CMake

En caso de no tener cmake instalado:

```
$ sudo apt-get update
$ sudo apt-get install cmake
```

### 3.2.2 Clonando el github

Primero se debe clonar nuestro github, usando el siguiente comando:

```
$ git clone https://github.com/Taller-7552-II/Jobify
```

En el caso de no tener git instalado:

```
$ sudo apt-get install git
```

### 3.2.3 Compilando el server

Para compilar el servidor hay que situarse en la carpeta /server/ y correr el script build.sh

```
$ cd server/
$ ./build.sh
```

### 3.2.4 Corriendo el server

Para correr el servidor hay que situarse en la carpeta /server/build/ y ejecutar ./Server. Si se quiere cambiar el directorio donde se va a crear la base de datos, -Ddb_path,/home/mi_path. Tambien se puede cambiar el puerto de la siguiente manera, -Dport,:8000. A continuacion se mostrará un ejemplo de un server abierto en el puerto 8080 con la base de datos en /home/new_path

```
$ cd server/
$ ./build.sh -Ddb_path,/home/new_path -Dport,:8080
```

### 3.2.5 Correr tests de python

Situado en la carpeta /server/pythonintegrationtests, correr el script test_server_func.py. Esto ejecutara los test de integracion de python. Se debe tener instalado python 2.7

```
$ python server_test.py
```

### 3.2.6 Correr unit test del server

Se debe estar situado en la carpeta server. Para correr las unit test se debe haber compilado el server con ./build.sh. A su vez, se debe tener instalado lcov y coveralls-lcov. De no ser el caso, correr el script install-coveralls. Luego se podrá correr el script run_tests.sh el cual ejecuta todas las unit test.

```
$ ./build.sh
$ sudo ./install_coveralls
$ ./run_tests.sh
```

### 3.2.7 Correr coverage del server

Se debe estar situado en la carpeta server. Para correr el coverage se debe haber compilado el server con ./build.sh. A su vez, se debe tener instalado lcov y coveralls-lcov. De no ser el caso, correr el script install-coveralls como root. Luego se podrá correr el script run_coverage.sh el cual ejecuta todas las unit test y se realiza el coverage del server.

```
$ ./build.sh
$ sudo ./install_coveralls
$ ./run_coverage.sh
```

## 3.3 Ejecución en Equipo

## 3.4 Corriendo el server

Para correr el servidor hay que situarse en la carpeta /server/build/ y ejecutar ./Server. Si se quiere cambiar el directorio donde se va a crear la base de datos, -Ddb_path,/home/mi_path. Tambien se puede cambiar el puerto de la siguiente

manera, -Dport,:8000. A continuacion se mostrará un ejemplo de un server abierto en el puerto 8080 con la base de datos en /home/new_path

```
$ cd server/
$ ./build.sh -Ddb_path,/home/new_path -Dport,:8080
```

## 3.5 Otras opciones

### 3.5.1 Correr tests de python

Situado en la carpeta /server/pythonintegrationtests, correr el script test_server_func.py. Esto ejecutara los test de integracion de python. Se debe tener instalado python 2.7

```
$ python server_test.py
```

### 3.5.2 Correr unit test del server

Se debe estar situado en la carpeta server. Para correr las unit test se debe haber compilado el server con ./build.sh. A su vez, se debe tener instalado lcov y coveralls-lcov. De no ser el caso, correr el script install-coveralls. Luego se podrá correr el script run_tests.sh el cual ejecuta todas las unit test.

```
$ ./build.sh
$ sudo ./install_coveralls
$ ./run_tests.sh
```

### 3.5.3 Correr coverage del server

Se debe estar situado en la carpeta server. Para correr el coverage se debe haber compilado el server con ./build.sh. A su vez, se debe tener instalado lcov y coveralls-lcov. De no ser el caso, correr el script install-coveralls como root. Luego se podrá correr el script run_coverage.sh el cual ejecuta todas las unit test y se realiza el coverage del server.

```
$ ./build.sh
$ sudo ./install_coveralls
$ ./run_coverage.sh
```

## 3.6 Mantenimiento

**El mantenimiento del Server de Jobify es un aspecto de suma importancia para el correcto funcionamiento del sistema. Para est**

- La base de datos
- El Log

La base de datos es una parte fundamental en cuento al desarrollo del sistema ya que es la encargada de mantener todos los archivos y datos de los usuarios. Esta suele encontrarse en la carpeta /tmp/prod/ pero puede cambiarse su ubicación como fue indicado en la sección Instalación , más especificamente en “Corriento el Server”. Si se cambia de directorio base, los datos no se trasladan así que al hacer este cambio se debe tener en cuenta que se tiene una base de datos nueva.

Como se aclaró anteriormente, la base de datos se encarga de guardar usuarios y chats al mismo tiempo, además de las sesiones. La manera de identificar rapidamente con que tipo de dato estamos tratando es al ver la clave que poseen en la base de datos. Los elementos de la base de datos son:

- User : (clave : "$username") Guarda el nombre de usuario y su contraseña.
- User Metadata : (clave : "$username.usertoken") : Guarda todos los datos del usuario, desde el perfil hasta la cantidad de archivos que tiene y cuales puede acceder.
- Chat Data : (clave : "usernameusername2)") : Guarda una lista de mensajes del chat entre 2 usuarios en el orden que llegaron

Dentro de la carpeta *build* se encuentra un archivo llamado log.txt en el cual se encuentran todas las actividades importantes que debe realizar el servidor. A medida que se agregan funcionalidades o se actualiza el código es fundamental seguir logeando para poder detectar errores o fallas en el server.

## 3.7 Known Issues

Si el servidor se cae mientras corre el cliente, el cliente no se entera que el server murió