Manual del proyecto

1.0

Arancibia Luis
Mussi Tomás
Dufau Ezequiel
Ackermann Alfredo

dic. 01, 2016

Índice general

CAPÍTULO 1

Objetivo

Documentar el proceso de trabajo, la organización, el entorno de trabajo, herramientas, trato con los compañeros.

CAPÍTULO 2

Organización

Como paso inicial, para desarrollar el proyecto se hizo un análisis general del trabajo a realizar, intentando instalar la mayor parte de las tecnologías a utilizar y poder tener el entorno de desarrollo operativo.

Se asignaron tres responsables a cada una de las tres aplicaciones diferentes a desarrollar, aunque esos responsables no desarrollaron únicamente en su aplicación designada, sino que la idea fue que hubiera un referente a quién recurrir en caso de cualquier problema.

El cuarto responsable fue el que realizó la integración de tecnologías que quedaron postergadas en el principio del desarrollo, por ejemplo, una vez integrado mongoose y jsoncpp, en el app server ya se podía incluir funcionalidad de la aplicación, por más que faltara la integración con la base de datos o una biblioteca de peticiones http para comunicarse con el shared server.

No hubo un líder formal desde el principio, lo cual fue una mala decisión, porque la comunicación era una inundación de información a los otros y que tal vez agregaban ruido a la conversación. Hacia la segunda mitad del cuatrimestre, no hubo una designación formal, pero dos integrantes hicieron una revisión del alcance pedido del trabajo, realizaron un control sobre el avance y se empezaron a crear tareas con los puntos pendientes en un excel y a asignar responsables para el cumplimiento de dichas tareas. Esta organización resultó más eficiente y se redujeron un poco los tiempos ociosos que tal vez tuviera algún integrante y que no sabía en qué otra cosa podía contribuir.

También hubo inconvenientes en los canales de comunicación elegidos. Dado que el curso utiliza slack, se propuso realizar un slack propio del grupo de trabajo, pero uno de los integrantes directamente ni lo usaba, y sólo un integrante realmente se mantenía al día con las conversaciones. Esto también dio lugar a desmotivaciones y desencuentro entre los integrantes. También se disponía de un grupo de Whatsapp, que tal vez no es lo más apropiado para la comunicación de un proyecto, pero terminó siendo la herramienta utilizada junto con emails.

Se subestimó la configuración del entorno del app server, en el cual se desperdició mucho tiempo y estábamos empecinados a compilar de forma estática, cosa que finalmente nos dimos cuenta que era mucho más sencillo instalar las bibliotecas necesarias en cada computadora que se fuera a utilizar para desarrollar y linkear dinámicamente, porque el principal problema se tuvo con cmake y lograr entender como funciona y como realizar la compilación de la forma en que se buscaba.

CAPÍTULO 3

Entorno de trabajo

Para el desarrollo de la aplicación, primero se realizó un consenso sobre qué sistema operativo utilizar. La decisión de utilizar Linux fue unánime, y dio la casualidad que todos los integrantes disponían de la misma versión (Ubuntu 14.04). La idea era tener listo el ambiente de trabajo dentro de la primera semana, luego de la publicación del enunciado del trabajo práctico. No se pudo cumplir con esto por dos motivos:

1. La instalación y compilación de bibliotecas de terceros fue ardua y difícil de integrar todas las bibliotecas a cmake para compilación estática. Luego se pasó a compilar dinámicamente para no desperdiciar más tiempo.
2. No se pudo prever desde el principio el alcance total del trabajo y, por ejemplo, luego de tener el app server funcionando se descubrió la necesidad de instalar un cliente para realizar peticiones http, por lo que la instalación de curlpp fue posterior a la primera semana de desarrollo.

Para el desarrollo de las tres aplicaciones se dispuso la utilización de los siguientes entornos:

- **Aplicación Android:** la aplicación de Android se desarrolló utilizando el Android Studio. No hubo una metodología de testear contra un servidor que respondiera peticiones que se harían al server real porque se consideró desperdicio de tiempo que estaría mejor utilizado en diseñar las pantallas mientras se desarrollaba la funcionalidad del app server. Una vez disponible el server con los primeros servicios, se utilizó cmake para la compilación automatizada del mismo.
- **Application server:** el app server se desarrolló con el editor de textos Sublime y como herramienta de compilación cmake. Además de crear pruebas de integración con scripts realizados en Python, se utilizó la herramienta Postman para probar y debuggear cuando el funcionamiento del app server no era el esperado.
- **Shared server**: el shared server también se desarrolló utilizando el editor de textos Sublime, además de la plataforma Heroku para poder desplegar la aplicación. Esto fue muy útil, porque los integrantes que más desarrollaron el app server y necesitaban probar la interacción y la API no disponían de computadoras de 64 bits para poder utilizar una imagen de docker, además de que con la imagen era más difícil actualizar el código para verificar, por lo que tener la plataforma online de Heroku fue de gran ayuda.

CAPÍTULO 4

Calidad

Para garantizar la calidad del software desarrollado, se utilizaron los siguientes procesos:

Integración continua

Se utilizó la herramienta de integración continua Travis integrada con el repositorio Github donde se hosteo el código de la aplicación. A pesar que al principio no se disponían de tests unitarios o tests de integración para ejecutar, cada vez que se publicaba código en el repositorio al menos se garantiza que dicho código compile. Una vez integrado con Google Test, fue de gran utilidad tener los tests unitarios, principalmente porque uno a veces se olvidaba de ejecutar los tests y al publicar código llegaba un mail de Travis informando el estado de cada branch del repositorio.

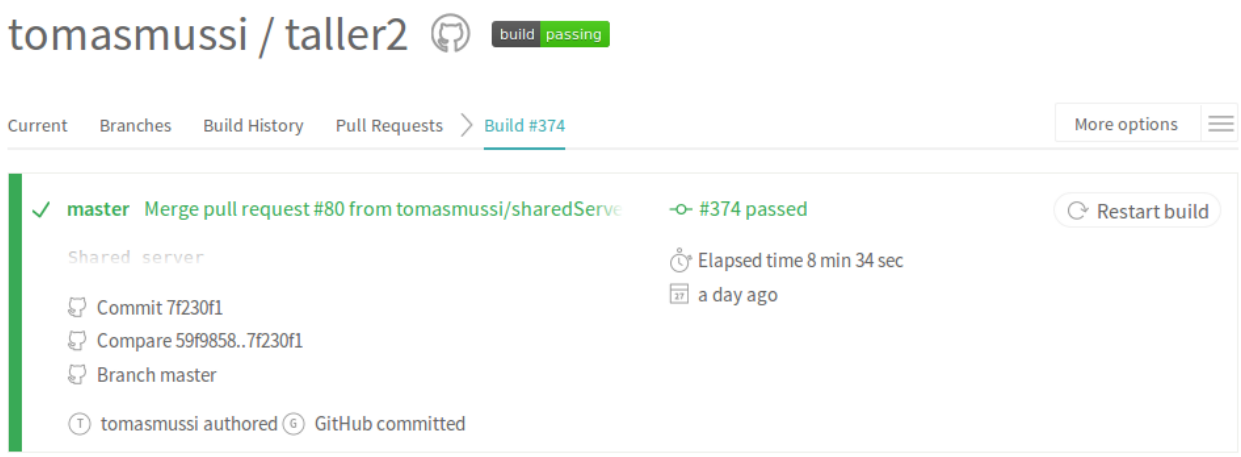

Imagen que muestra la utilización de travis en el proyecto

Tests unitarios

En el app server se realizaron tests unitarios utilizando el framework de Google Test. Los casos de prueba aseguran que el funcionamiento interno de las clases no cambien el comportamiento. Es esencial para los métodos que devuelven información del usuario, por ejemplo, para asegurar que los cambios no modificarán las respuestas tal como se especifican en la API-REST. En el shared server se implementaron tests sobre la API con un script realizado en Python. Luego el formato del mismo script se reutilizó en la API del app server.

```
[----------] Global test environment tear-down
[==========] 52 tests from 7 test cases ran. (18 ms total)
[  PASSED  ] 52 tests.


The command "./server/test/testing" exited with 0.

Done. Your build exited with 0.
```

Imagen que muestra la ejecución de los tests unitarios

```
======================================================================
======================================================================
Ejecutando test de API ...\n
Test de add token FCM ... ok
Test add contact not exist ... ok
Test add contact not exist ... ok
Test add contact ... ok
Test de location empty ... ok
Test de location ... ok
Test de lookup ... ok
Test add Message ... ok
Test Message empty ... ok
Test test message user contact not exist ... ok
Test view Message ... ok
Test de new User ... ok
Test de request en log_out_error ... ok
Test de request en fb_login sin user ... ok
Test de edit and view profile ... ok
Test de view profile user not exist ... ok
Test de vote a friend ... ok
Test de vote not friend ... ok
Test de vote user not exist ... ok

----------------------------------------------------------------------
Ran 19 tests in 0.664s

OK
```

Imagen que muestra la ejecución de los tests de la ApiREST

Code coverage

Para garantizar la calidad total de la aplicación, se realizaron principalmente tests sobre el application server, que es el punto crítico de interacción con la aplicación de android y las consultas realizadas al shared server. Como métrica se utilizó el code coverage del app server, es decir, medir cuánto código del app server está verificado por las pruebas

y cuantas porciones de código quedaron excluidas, además de verificar un poco de comportamiento de las bibliotecas de terceros como la base de datos clave-valor. Se estableció un mínimo de 75 % de cobertura de código sobre el total.

LCOV - code coverage report

			Hit	Total	Coverage
Current view:	top level - src	Lines:	1093	1339	81.6 %
Test:	coverage.info	Functions:	163	194	84.0 %
Date:	2016-12-01 01:09:59				

Filename	Line Coverage		Functions	
ApiJsonController.cpp	74.8 %	398 / 532	83.3 %	30 / 36
Chat.cpp	100.0 %	37 / 37	100.0 %	7 / 7
Chat.h	0.0 %	0 / 1	0.0 %	0 / 1
DatabaseHandler.cpp	80.8 %	21 / 26	88.9 %	8 / 9
HerokuService.cpp	2.5 %	1 / 40	33.3 %	2 / 6
Message.cpp	95.7 %	44 / 46	90.0 %	9 / 10
Message.h	0.0 %	0 / 1	0.0 %	0 / 1
Notificator.cpp	92.1 %	35 / 38	100.0 %	4 / 4
Notificator.h	100.0 %	1 / 1	100.0 %	1 / 1
TokenFCM.cpp	100.0 %	6 / 6	100.0 %	3 / 3
TokenFCM.h	0.0 %	0 / 1	0.0 %	0 / 1
TokenFCMHandler.cpp	86.7 %	13 / 15	71.4 %	5 / 7
User.cpp	95.1 %	174 / 183	92.5 %	37 / 40
User.h	0.0 %	0 / 1	0.0 %	0 / 2
UserHandler.cpp	86.2 %	156 / 181	96.0 %	24 / 25
UserHandler.h	50.0 %	2 / 4	50.0 %	1 / 2
UserList.cpp	100.0 %	29 / 29	100.0 %	6 / 6
UserList.h	0.0 %	0 / 1	0.0 %	0 / 1
log.cpp	58.6 %	17 / 29	55.6 %	5 / 9
md5.cpp	95.2 %	159 / 167	91.3 %	21 / 23

Imagen que demuestra el coverage obtenido en la aplicación

Code review

Además de realizar tests sobre el código, para asegurar la calidad del mismo se realizaron revisiones de código, utilizando como herramienta los pull requests que proporciona Github. El proceso consiste en que un integrante trabaja sobre un fix o nueva funcionalidad en un branch. Cuando opina que está en condiciones de integrar ese código, crea un pull request que será revisado por otro integrante del grupo. Con las revisiones, el revisor puede criticar y realizar preguntas sobre el funcionamiento, obligando al autor a explicar y exponer su punto de vista. Este sistema contribuyó a integrar errores de desarrollo y propuso la creación de tests unitarios que al autor original se le habían escapado.

Log

Se utilizó un log en el appServer para tener un registro de las operaciones que iba haciendo el appServer. Esto permitió una acelerar la busqueda de errores. Y en android se utilizo el log que viene por defecto.