Verbale esterno del 2024-02-12

2024-02-16 — v1.1

overture.unipd@gmail.com

Destinatari	Prof. Tullio Vardanega Prof. Riccardo Cardin *Zextras* Gruppo *Overture*
Responsabile	Eleonora Amadori
Redattori	Eleonora Amadori Francesco Costantino Bulychov
Verificatori	Riccardo Fabbian Francesco Furno

Registro delle modifiche

Versione	Data	Autori	Verificatori	Dettaglio
1.1	2024-02-16	Eleonora Amadori	Riccardo Fabbian	Correzioni al verbale
1.0	2024-02-13	Francesco Costantino Bulychov	Francesco Furno	Stesura del verbale
0.1	2024-02-12	Francesco Costantino Bulychov	Francesco Furno	Prima bozza con risposte alle domande

Indice

1) Contenuti del verbale

1.1) Informazioni sulla riunione

- **Luogo**: Chiamata sulla piattaforma dell'azienda *Zextras*;
- **Ora di inizio**: 14:30;
- **Ora di fine**: 15:20;
- **Partecipanti**: Eleonora Amadori, Michele Bettin, Riccardo Bonavigo, Francesco Costantino Bulychov, Riccardo Fabbian, Francesco Furno, Alex Vedovato;
- **Partecipanti esterni**: Federico Rispo.

1.2) Ordine del giorno

- Allineamento tecnico con l'azienda proponente;
- Chiedere consigli utili riguardanti la fase di progettazione (design) del prodotto;
- Risolvere dei dubbi riguardanti gli stress test.

2) Sintesi dell'incontro

Nella fase iniziale dell'incontro abbiamo informato l'azienda di aver completato con successo la prima revisione del prodotto. Successivamente abbiamo chiarito alcuni dubbi riguardanti il design del prodotto e gli stress test.
Per quanto riguarda il primo punto, dopo aver mostrato lo stato attuale del PoC, il quale è stato esteso, abbiamo discusso i pattern individuati fino ad ora e Federico Rispo ne ha suggeriti degli altri.
Per gli stress test invece, è stato suggerito l'approccio da adottare: è bene che siano degli "scenari", ognuno dei quali non sarà altro che un flusso di operazioni plausibile per gli utenti.

3) Domande e risposte

3.1) Che tipo di architettura consigliate? Monolite o a microservizi?

Viene suggerito l'utilizzo dell'architettura a microservizi.
Nonostante sia generalmente più complicata, favorisce l'utilizzo di un'architettura esagonale: ponendo al centro la business logic e utilizzando le interfacce, siamo sicuri di ridurre le dipendenze.

3.2) Che consigli avete per la fase di progettazione?

In generale è importante utilizzare molte interfacce, soprattutto quando il codice presenta diverse classi concrete simili tra loro. In questo modo il codice sarà più mantenibile e facilmente estendibile.

3.3) Che design pattern ha senso usare?

Il prodotto che stiamo sviluppando ha per sua natura poca business logic. Questo implica che i pattern che ha senso implementare sono tendenzialmente meno che in altri progetti.

Tuttavia, ce ne sono diversi che possiamo adottare. Innanzitutto, è sensato fare uso del "builder pattern": risulta più agevole di avere un costruttore con tutte le variabili per ogni classe.
È buona norma usare anche la "dependency injection"; l'azienda consiglia `Guice`, il quale non è una framework completo ma solo una libreria per la dependency injection, la quale si può paragonare all'autowiring di `Spring`.
Lo "stategy pattern" è inseribile nella fase di dispatch delle `methodCall` di JMAP: invece di un controllo sul tipo concreto e successivo cast per la chiamata `execute`, conviene definire una interfaccia e classi che la implementano.

Per quanto rigurda la parte di persistenza, l'azienda consiglia di valutare l'utilizzo di un "adapter pattern", che consente la sostituzione della libreria utilizzata per l'accesso al database.

3.4) Possiamo utilizzare il builder di `lombok` come "builder pattern"?

In linea di massima sì: non fa differenza pratica che sia scritto direttamente da voi o meno.
È bene chiedere al Prof. Riccardo Cardin se questo possa essere comunque considerato come "design pattern adottato".

3.5) Che consigli avete per i benchmark?

Come già suggerito in passato, il consiglio è di utilizzare Locust, un framework di testing basato su python. Esso consente di decidere il numero di sessioni che si vogliono usare e quante richieste fare da ognuna di esse.
L'azienda consiglia di creare 5/10 account, differenziando il più possibile la tipologia degli utenti; per esempio un utente può inviare una mail ogni 10 secondi, mentre un altro ogni 10 minuti. Può essere utile anche combinare le caratteristiche di più profili.
Per le richieste al server, si possono utilizzare le funzionalità di comunicazione di rete fornite dal framework, senza dover ricorrere a librerie esterne. Anche per le richieste JMAP, il suggerimento è di costruirli ad-hoc a partire direttamente da dei file JSON (senza quindi passare per una libreria); probabilmente costa meno tempo.

3.6) Vanno testati i singoli metodi o gruppi di operazioni?

Per rendere il test più simile alla realtà è necessario creare un flusso di operazioni, denominato "scenario".
Un esempio di scenario è il seguente: login (recupero dell'oggetto di sessione), visualizzazione email, composizione email, invio email.
L'azienda è interessata a capire se tutto il flusso scala, più che richieste con singoli metodi JMAP. Pertanto si aspetta non più di 5 scenari da parte nostra.

4) Decisioni prese

- Valutare se utilizzare un'architettura a microservizi;
- Preferire, ove possibile, l'uso di interfacce rispetto a classi concrete;
- Chiedere al Prof. Riccardo Cardin se `lombok` potrà essere utilizzato o meno;
- Definire gli scenari su cui eseguire i benchmark.