Introduction to Software Architecture

- Date : 11 mai 2017
- Durée : 08h00 – 11h00 (3 heures)
- Aucun document autorisés; barème donné à titre indicatif.

Lisez tout le sujet (4 pages) avant de commencer à répondre aux questions ; Les questions sont identifiées en gras dans le texte ; On compte habituellement dix mots pour une ligne ; Les différentes parties sont indépendantes.

Vous devez répondre aux questions posées dans le sujet sur la copie d'examen fournie, avec d'éventuelles feuilles intercalaires. Vous ne pouvez pas sortir de la salle d'examen durant la première heure (avant 09h00), ni durant le dernier quart d'heure (après 10h45). **Toute fraude identifiée sera systématiquement transmise au conseil de discipline de l'UNS**.

Partie #1 : Question de recul /3 (0h15)

Durant votre projet « Carte d'infidélité », vous avez été confronté au développement concret d'une architecture 3-tiers. En utilisant les connaissances acquises durant votre formation dans le département (vous pouvez/devez faire référence à d'autres cours qu'ISA) et votre expérience en projet ISA, **répondez à la question donnée en annexe #1**. Votre réponse doit être synthétique (~150 mots, 200 max).

Partie #2 : Analyse de texte /5 (0h45)

L'annexe #2 contient un article anglophone sur les architectures 3-tiers. Sur la base de votre expérience en conception et mise en œuvre d'architecture 3-tiers acquise durant le semestre, **proposez une analyse critique** et synthétique de ce texte mettant en avant votre opinion argumentée (~150 mots, 200 max). L'identification de critères pertinents utilisés dans votre analyse fait partie de la question.

Partie #3 : Étude de cas /12 (2h00)

L'annexe #3 contient un *briefing client* du projet obtenu par votre société. Sur la base des informations contenue dans ce briefing, vous devez proposer une architecture 3-tiers y répondant. Les points suivants vous permettront de cadrer votre étude de cas :

1. Identifiez les objets métiers au sein de votre architecture ;
   - Décrivez ces objets sous la forme d'un diagramme de classes ;
   - Expliquez le *mapping* Objet-Relationnel quand c'est pertinent.
2. Identifiez les différents composants à mettre en jeu dans votre architecture ;
   - **Décrivez l'assemblage** sous la forme d'un diagramme de composants ;
   - **Décrivez les interfaces** de chaque composant (diagramme de classes).
3. **Justifiez vos choix de conception** pour cette architecture, en identifiant quels éléments appartiennent à quel tiers de l'application, et pourquoi ils sont nécessaires dans le système. Soyez synthétique : ~150 mots, 200 max ;
4. Expliquez **comment cette architecture s'implémente** dans l'écosystème J2E. Soyez synthétique : ~100 mots, 150 max.

Annexes

Annexe 1 : Thème pour la question de recul

« Quel est le rôle des stubs dans une architecture logicielle ? »

Annexe 2 : Why You Should NOT Implement Layered Architectures[1]

Abstraction layers in software are what architecture astronauts tell you to do. Instead, however, half of all applications out there would be so easy, fun, and most importantly: productive to implement if you just got rid of all those layers.

Frankly, what do you really need? You need these two:

- Some data access
- Some UI

Because that's the two things that you inevitably have in most systems. Users, and data. Here's Kyle Boon's opinion on possible choices that you may have:

"Really enjoying #ratpack and #jooq" (@kyleboon)

Very nice choice, Kyle.

Ratpack and jOOQ[2].

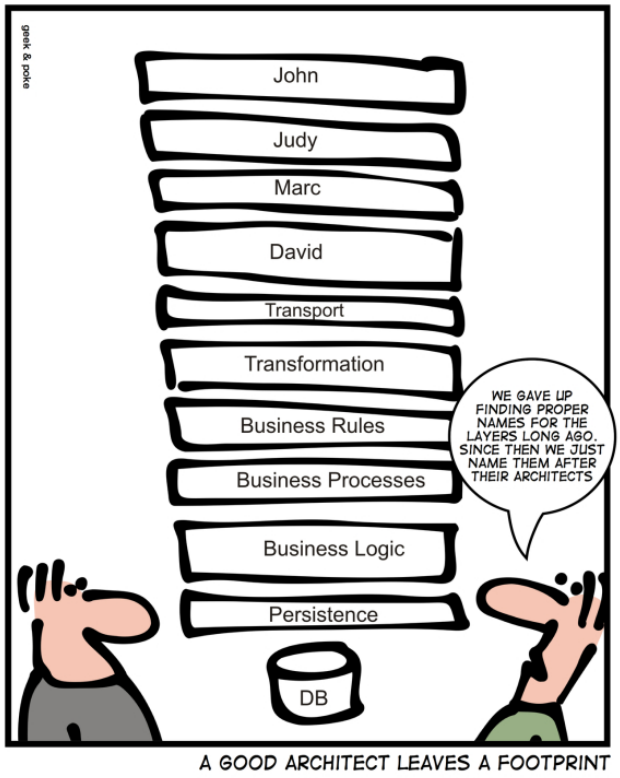

Fig 1. Geek and Poke's Footprints – Licensed CC-BY 2.0

You could choose any other APIs, of course. You could even choose to write JDBC directly in JSP. Why not. As long as you don't go pile up 13 layers of abstraction (Fig 1).

That's all bollocks, you're saying? We *need* layers to abstract away the underlying implementation so we can change it? OK, let's give this some serious thought. How often do you *really* change the implementation? Some examples:

- ***SQL***. You hardly change the implementation from Oracle to DB2
- ***DBMS***. You hardly change the model from relational to flat or XML or JSON
- ***JPA***. You hardly switch from Hibernate to EclipseLink
- ***UI***. You simply don't replace HTML with Swing
- ***Transport***. You just don't switch from HTTP to SOAP

[1] https://blog.jooq.org/2014/09/12/why-you-should-not-implement-layered-architecture/

[2] *Ratpack* & *jOOQ* sont deux frameworks Java (*jOOQ* pour écrire du SQL en Java et servir d'ORM, *Ratpack* pour des applications HTTP-based). L'auteur de cet article est aussi l'auteur de *jOOQ*.

- ***Transaction layer***. You just don't substitute JavaEE with Spring, or JDBC transactions

Nope.

Your architecture is *probably* set in stone.

And if – by the incredible influence of entropy and fate – you happen to have made the wrong decision in one aspect, about 3 years ago, well you're in for a major refactoring anyway. If SQL was the wrong choice, well good luck to you migrating everything to MongoDB (which is per se the wrong choice again, so prepare for migrating back). If HTML was the wrong choice, well even more tough luck to you. Likelihood of your layers not really helping you when a concrete incident happens: 95% (because you missed an important detail)

Layers = Insurance

If you're still thinking about implementing an extremely nice layered architecture, ready to deal with pretty much every situation where you *simply switch* a complete stack with another, then what you're really doing is filing a dozen insurance policies. Think about it this way. You can get:

- Legal insurance
- Third party insurance
- Reinsurance
- Business interruption insurance
- Business overhead expense disability insurance
- Key person insurance
- Shipping insurance
- War risk insurance
- Payment protection insurance
- … pick a random category

You can pay and pay and pay in advance for things that *probably* won't ever happen to you. Will they? Yeah, they might. But if you buy all that insurance, you pay heavily up front. And let me tell you a secret. *IF* any incident ever happens, chances are that you:

- Didn't buy that particular insurance
- Aren't covered appropriately
- Didn't read the policy
- Got screwed

And you're doing exactly that in *every* application that would otherwise already be finished and would already be adding value to your customer, while you're still debating if on layer 37 between the business rules and transformation layers, you actually need another abstraction because the rule engine could be switched any time.

Stop doing that

You get the point. If you have infinite amounts of time and money, implement an awesome, huge architecture up front. Your competitor's time to market (and fun, on the way) is better than yours. But for a short period of time, you were *that* close to the perfect, layered architecture!

Annexe 3 : Étude de cas *PolyPark*

La gestion des parkings sur le campus SophiaTech est un sujet sensible, à bien des niveaux.

La direction du campus a contacté votre société pour la mise en place d'un système de gestion des parkings universitaire, qui pourrait être déployé sur plusieurs campus (Sophia Antipolis servirait de site pilote).

- Le campus dispose de 6 parkings (P1 à P6), de 5 barrières (B1 à B5), et de 3 portails (P1 à P3). La capacité totale est d'environ 500 places (pour 1200 usagers).
- Les parkings disposent de contrôle d'accès :
  - Le parking P2 est un parking public.
  - Les parkings P1 et P3 sont réservés aux étudiants du campus, respectivement UNS et EURECOM.
  - Les parkings P4, P5 et P6 sont réservés aux personnels (laboratoires, enseignants et administratifs)
- Les barrières sont équipées de lecteurs de badges compatibles avec les cartes étudiants et les cartes professionnelles des usagers du campus. Chaque carte contient un identifiant unique qui lui est propre.
- Les portails sont ouverts de 06h30 à 22h30 tous les jours sauf le dimanche. Ils ne peuvent être ouverts que par des personnes préalablement autorisées en dehors de ces heures. Chaque portail dispose d'un numéro de téléphone, qu'un usager autorisé peut appeler avec son téléphone portable, déclenchant l'ouverture.
- Les étudiants covoiturant peuvent candidater pour obtenir une place sur un parking personnel, en indiquant la liste des étudiants faisant partie de la voiture. Il y a un quota max de places mises à disposition, et chaque chef de département d'enseignement attribue à discrétion les places disponibles pour sa formation. Les lauréats ne jouant pas le jeu (des contrôles peuvent avoir lieu à l'entrée le matin) peuvent être révoqués après 2 avertissement, et leur place remise en jeu.
- Les agents régisseurs du campus disposent sur leur téléphone de fonction d'une application mobile permettant d'identifier les plaques d'immatriculation des voitures des usagers (personnels et/ou étudiants) contrevenants (pas censé avoir accès à un parking donné, garé en dehors des places, garé sur une place handicapé)
- Des statistiques d'usages sont collectées pour connaitre l'occupation des parkings, en temps réel mais aussi à des fins d'historisation (pour affiner les quotas)