UNIVERSITÉ DE TECHNOLOGIE DE COMPIÈGNE

ÉTUDE EXPÉRIMENTALE

Analyse de données pédagogiques dans le contexte Faq2Sciences

Étudiant :
Minh Tri LÊ

Suivi par :
Stéphane CROZAT
Thibaut ARRIBE

2 juillet 2017

Table des matières

Table des figures

Liste des tableaux

Introduction

La motivation de cette TX est d'étudier quelle analyse de données il est possible d'effectuer à partir de données pédagogiques, dans un contexte Scenari. Nous utiliserons des données de tests de positionnement de physique et de biologie de l'Université de Lorraine. Les objectifs sont d'une part de faire une analyse du contenu et d'autre part d'en rendre une interprétation sous la forme d'un tableau de bord à destination des professeurs et étudiants.

Les données proviennent de la plateforme en ligne Faq2Sciences qui propose des tests de positionnement dans les matières scientifiques à l'université. Faq2Sciences peut être utilisée par de nouveaux étudiants au niveau Bac+1 ou par certaines universités. [4]

Faq2Sciences héberge des documents Scenari. Scenari est un logiciel de conception, création et publication de documents multimédia (diaporama, site web, PDF, ...). [3]

Les utilisateurs peuvent accéder aux documents Scenari, et donc aux tests par l'intermédiaire d'une API interne de LMS (Learning Management System) [5] à Faq2Sciences. Les documents envoient des *événements* à Faq2Sciences. L'analyse utilisera ces événements comme données. Un important aspect de cette étude sera de tenter de retrouver les *états* entre chaque événement.

Les données sont stockées sur un serveur ElasticSearch qui fonctionne en tant que moteur de recherche et permet donc d'effectuer des requêtes textuelles de manière très efficace. [6] Nous l'utiliserons donc pour accéder aux données nécessaires à l'analyse. Ces données sont traitées au préalable par Logstash qui gère les fichiers de log générés par les événements. Logstash permet donc d'unifier les données d'input dans un format de données commun et prêt à l'emploi pour ElasticSearch. [7]

Un diagramme UML détaillant l'architecture du système est en annexe A.

Le tableau de bord sera successivement développé avec Kibana ([22]) puis en JavaScript avec le framework de visualisation C3.js ([27]) et enfin Plotly ([28]).

Le projet de TX a démarré la semaine du 13/03/2017, en collaboration avec l'éditeur de la suite logicielle Scenari : *Kelis*. [8]

1 Description des données

1.1 Généralités et structure de données

Les données comportent des résultats de tests de physique et de biologie pour près de 660 étudiants de L1 de l'université de Lorraine durant la mi-septembre 2016. Les étudiants ont répondu à 10 questions de physique et 50 de biologie par tirage aléatoire (annexe B) parmi 31 questions de physique et 211 de biologie.
Les questionnaires sont organisés par sous-questionnaire (annexe C).

Les données sont au format JSON. En annexe D se trouve un exemple du contenu d'un enregistrement. Tous les attributs ne sont pas utilisés pour l'analyse. L'annexe E décrit les attributs (clés) et valeurs possibles d'un enregistrement ainsi que le détail de ceux considérés pour cette étude.
Nous utiliserons les attributs suivants : *_index*, *_type*, *_source.response*, *depot_path*, *verb*, *score_scaled*, *question_id*, *clt_ts*, *user et timestamp.*
L'user correspond au même utilisateur entre deux questionnaires.

1.2 Logique d'événements et états

Les événements générés par les documents questionnaires suivent la logique XApi [10]. De nombreux événements sont générés par l'activité de l'apprenant sur les documents Scenari.

Ces évènements (*verb*) sont :

— "*scored*" ⇔ "*responded*" : L'utilisateur a répondu à la question/(sous-)questionnaire et a donc généré un score numérique. Au début d'un questionnaire, un événement *scored* est initialisé à 0. Cela est automatiquement généré pour toutes les questions/(sous-)questionnaires tirés pour l'utilisateur.
De plus, chaque réponse et donc changement de réponse génère un événement *scored*.

— "*attempted*" : L'utilisateur a vu la question/le questionnaire. Au début d'un questionnaire, un événement *attempted* est automatiquement généré pour toutes les questions/(sous-)questionnaires tirées pour l'utilisateur.

— "*completed*" : Un (sous-)questionnaire a été complété par un utilisateur.

Nous disposons ainsi de **tous** les événements générés par les utilisateurs. Les données permettent donc de reconstituer la chronologie des actions d'un utilisateur. De plus, nous pouvons déduire *l'état* d'un utilisateur entre les événements (Changement de réponse, hésitation, réussite, difficulté).
Nous tenterons dans cette étude de déduire les états à partir des événements.

1.3 Jeux de données considérés

Pour la suite de cette analyse, nous utiliserons quatre jeux de données :

— Résultat au questionnaire de physique :
  — *par utilisateur*
  — *par question*
— Résultat au questionnaire de biologie :
  — *par utilisateur*
  — *par question*

Les événements générées par les (sous-)groupes de questionnaire ([C]) sont exclus car ils contiennent des informations qui résument les questions. Cela ajoute

donc des données redondantes (score cumulé du test après chaque réponse) ou non nécessaires à l'analyse (données générées automatiquement à l'initialisation du test).

De plus, nous excluons tous les enregistrements n'ayant pas le champ "response" car cela ne correspond aucunement à l'action d'un utilisateur (i.e. : un choix de réponse a été formulé). Cela permet d'exclure tous les événements générés par défaut à l'initialisation d'un test, qui ne sont pas pertinents pour l'analyse.
Ainsi, pour les données considérées (voir les requêtes du code), le nombre d'enregistrements est d'environ 60 000 pour la biologie et 13 500 pour la physique.

2 Analyse des données et réalisations techniques

Trois axes principaux ont été dégagés durant cette analyse.
Pour chacun des axes d'analyse, nous rendrons compte des idées et des résultats ayant pu être implémentés ou non, ainsi que ceux effectivement livrés à la fin du projet.
Le tableau de bord doit avoir un bon compromis entre exhaustivité et concision. Plusieurs références bibliographiques ([12][13][11]) ont aidé à apporter des idées pour les graphiques. Une idée commune aux graphiques et implémentée est l'ajout de 3 droites horizontales colorées représentant la moyenne μ, la moyenne $\pm$ l'écart-type $s*$.

2.1 Date de réponse

Comme les tests ont été joués à date et heure prédéfinie par l'université, la date ou l'heure de réponse par les étudiants n'est pas pertinente. À titre d'exemple, des graphiques effectués avec Kibana représentant le nombre d'événements "completed" et/ou "scored" en fonction du temps, se trouve en annexe F.

2.2 Premier axe : Analyse *par étudiant*

L'analyse des données par étudiant a pour but d'étudier des indicateurs par rapport à un étudiant ou groupe d'étudiant donné. Les données de tous les étudiants pour la physique et la biologie (environ 660) sont utilisées. Les jeux de données utilisés sont donc les résultats de physique ou biologie au questionnaire par utilisateur.

2.2.1 Paramètre du score (*score_scaled*)

2.2.1.1 Réalisation aboutie

Il est intéressant de voir le score moyen d'un étudiant sur l'ensemble d'un questionnaire.

Un premier graphique ayant été fait avec Kibana est fourni en annexe G. C'est un diagramme qui représente le paramètre du score en fonction des étudiants. À l'issue de ce premier rendu, il est plus intéressant de voir une distribution décroissante par rapport au score, ce que Kibana ne permet pas de faire. De plus, il y a des données indésirables que Kibana ne peut filtrer.

L'implémentation des graphiques finals pour l'analyse de ce paramètre fait partie du livrable et est en annexe H. Ces graphiques sont ordonnés de manière décroissante selon la valeur du score moyen.

2.2.1.2 Réalisation non aboutie

Il est aussi pertinent d'avoir plus de détail sur les valeurs extrêmes (maximum ou minimum). Une idée a été de faire un histogramme non homogène (intervalle de tailles variables), avec une plus grande discrétisation des intervalles aux extrêmes. Il faut donc trouver des paramètres (déciles, quartiles, ...) pour les intervalles de l'histogramme en début et en fin de l'axe des ordonnées pour avoir une représentation plus fine des extrémités.
Cette fonctionnalité n'est pas supportée par le framework Plotly.js mais fait l'objet d'une open request [14].

À cela, il faudrait ajouter la possibilité de voir dynamiquement la liste des 10 meilleurs/plus mauvais résultats sur clic souris d'une extrémité de l'histogramme.

Aussi, une autre idée est un diagramme en bâton avec les proportions (1^{er}, 3^{me} quartile, médiane) du score relativement superposé sur *une barre* pour chaque utilisateur. Ce graphique a été tenté, mais n'a pas abouti à cause d'un problème technique rencontré avec le framework. (Échec de l'utilisation du barmode *'relative'* malgré application de la documentation). [15]

Enfin, une dernière idée inspirée de la bibliographie ([12] [13]) est le gauge chart (Annexe J) qui permet de rapidement visualiser la performance des étudiants. Les étudiants sont alors répartis en 3 (ou plus) groupes en fonction de leurs résultats, de moins bien à très bien. Les paramètres de répartition peuvent être les quartiles et la médiane.

2.2.2 Paramètre du nombre de changements de réponse

Il est aussi intéressant de voir le nombre de changements de réponse d'un étudiant durant un questionnaire. Le nombre de changements de réponse d'un étudiant correspond aux nombres d'événements *scored* générés par un utilisateur par rapport à une question.

Pour cela, nous avons considéré d'une part la moyenne du nombre de changements d'un utilisateur sur l'ensemble du questionnaire. D'autre part, nous avons aussi les proportions (quartile, médiane) du nombre de changements à une question pour chaque utilisateur sur un graphique séparé.
Pour ces deux graphiques, les droites horizontales (moyenne, écart-type) sont ajoutées, de la même manière que pour le paramètre du score.

Enfin, ces deux graphiques font partie du livrable final et sont en annexe I

2.2.3 Paramètre du temps de réponse

Il est aussi pertinent de voir le temps de réponse d'un utilisateur.

Le graphique global pour chaque utilisateur n'a pas été réalisé avec succès car, les données nécessaires et le formatage sont complexes. Voir la partie technique à ce sujet 3.3.1.

Pour ces graphiques, la même méthodologie de visualisation que le paramètre précédent a été tentée.

2.2.4 Conclusion partielle

Le paramètre du score donne un rapide aperçu de la performance générale des étudiants.
Le nombre de changements et le temps de réponse d'un utilisateur apportent une mesure de l'hésitation très pertinente pour l'analyse.

Enfin, l'axe d'analyse par étudiant permet d'étudier la tendance générale d'un groupe utilisateur d'une période à une autre (en particulier par semestre ou année).

2.3 Second axe : Analyse *par questionnaire*

Nous étudions ici l'analyse des résultats aux tests **par question**. Les mêmes paramètres et mêmes méthodes de visualisation que le premier axe (2.2) sont utilisés donc nous ne mentionnerons que les différences dans l'analyse et/ou remarques importantes.

2.3.1 Paramètre du score (*score_scaled*)

L'implémentation des graphiques finals pour l'analyse de ce paramètre fait partie du livrable et est en annexe K. Ces graphiques sont ordonnés de manière décroissante selon la valeur du score moyen. Ils suivent la même logique que le paramètre du score pour l'axe d'analyse par étudiant.

On précise qu'il n'est pas intéressant d'utiliser des proportions (quartiles, médian) car les valeurs possibles sont binaires, donc la valeur d'une proportion ne sera que 0 ou 1.

2.3.2 Paramètre du nombre de changements de réponse

Même remarque que précédemment (Annexe L)

2.3.3 Paramètre du temps de réponse

Non implémenté pour les mêmes raisons qu'en 2.2.3.

2.3.4 Combinaison de paramètres

On peut afficher les 3 paramètres précédents sur un même graphique pour un questionnaire sous la forme d'un line chart. O En ordonnée il faut mettre à l'échelle le score, le nombre de changements et le temps de réponse à une question. Cela permet d'avoir un aperçu complet pour l'analyse du questionnaire.
Nous pouvons alors analyser cette combinaison pour les résultats de tous les étudiants au test ou uniquement un étudiant. Une analyse plus détaillée est en annexe O.
Ce graphique n'a pas été implémenté.

2.3.5 Paramètre du nombre d'événements

En plus des paramètres précédents, le rapport du nombre d'événements *"attempted"* sur le nombre d'événements *"scored"* est intéressant pour avoir une mesure de l'hésitation.
Le graphique représentant ce paramètre a été tenté sur Kibana (M) mais n'a pas été gardé.

2.3.6 Conclusion partielle

L'étude par question permet d'évaluer la difficulté ou la facilité d'un test en particulier. Si les indicateurs sont favorables (resp. défavorables) alors il y a un décalage positif (resp. négatif) entre le niveau attendu et le niveau effectif des étudiants. Cela amène donc à revoir la difficulté du test ou de certaines questions.

2.4 Troisième axe : Analyse *par groupe de questionnaires*

Nous comparons ici les différences de résultats entre les différents questionnaires.

Pour comparer le score aux tests de physique et de biologie et plus généralement entre plusieurs matières, un histogramme superposé ("*overlay*") a été testé (Annexe N) mais pas intégré au livrable.
Dans le cas de 2 questionnaires, il est encore possible de correctement visualiser la superposition des résultats sur l'histogramme.

Cependant, dans le cas de plusieurs questionnaires (donc autant de jeux de données différents), le graphique devient difficilement visualisable. Dans ce cas, on préférera un histogramme *"groupé"*, à la manière de ce bar chart : [16], mais avec des intervalles (type histogramme) en abscisse et donc une couleur par questionnaire. Le framework Plotly ne propose pas ce genre d'histogramme.

Cet axe permet donc de comparer la performance des étudiants entre plusieurs matières.

3 Éléments techniques

Cette section présente quelques détails techniques essentiels ainsi que les difficultés rencontrées et pertinentes pour une étude postérieure à celle-ci.

3.1 ElasticSearch : Requête et mapping

On rappelle qu'ElasticSearch est un moteur de recherche très performant pour rechercher du texte, moins pour faire du calcul. Pour traiter les données, on privilégie donc le traitement hors ElasticSearch, dans un langage supportant le format JSON.

Pour ElasticSearch, l'index est similaire à une base de données, et un type a une table de cette base.

Les principales manières de requêter ElasticSearch sont :

— en utilisant cURL

— par API REST : ce qui a été utilisé ici (requête *POST* principalement) [17]

L'application multiplateforme Postman [18] a été utilisée pour envoyer des requêtes POST vers le serveur pour tester les requêtes.

3.1.1 Le mapping

Le mapping permet de définir le format de chaque attribut (1) (texte, date, ...). De plus, on peut rendre le champ *analysé* ou *non analysé*. ElasticSearch étant un moteur de recherche, il peut donc *analyser* le texte et chercher des termes dérivés. Cela est coûteux en temps et en ressource donc on préférera requêter sur des champs non analysés si possible.
Pour savoir quel mapping est utilisé pour un index en particulier, il faut concaténer "*/_mapping*" sur le lien de l'index et regarder le mapping. Voir exemple en annexe P, un mapping réel d'un index Faq2sciences [26] et la documentation [19].

3.1.2 Query DSL

Pour faire des requêtes sur le serveur, il faut construire un objet JSON suivant la syntaxe query DSL d'ElasticSearch puis envoyer cet objet par requête *POST*. Voir la

documentation très complète à ce sujet [21]

3.2 Kibana : Limitation

Kibana est l'outil de visualisation de l'Elastic Stack. Il fonctionne *au-dessus* d'ElasticSearch et permet de rapidement visualiser les données sous forme de divers graphiques prêts à l'emploi. [22]
Après avoir fait quelques visualisations sur Kibana, des limitations sont apparues. Par exemple, sur certaines requêtes, il n'y a pas de moyen de trier l'axe des ordonnées par ordre croissant ou décroissant.
De plus, il est difficile de filtrer les données d'initialisation au test ainsi de ne garder que le dernier événement *scored* d'une réponse à une question. Cela correspond à la dernière réponse choisie par un utilisateur et donc au vrai score correspondant.

Kibana comprend donc des limitations qui laisse du bruit dans les données et est difficilement compatible avec la logique d'événements (type xApi).

3.3 JavaScript

Dû à ces limitations, le choix de remplacement s'est porté sur JavaScript en récupérant les données par des requêtes AJAX. (Voir chapitre sur AJAX : [23]).
Après réception des données, JavaScript et ses nombreux choix de framework de visualisation permettent une grande liberté pour le formatage, le calcul et l'affichage des données.

L'accès au serveur ElasticSearch se fait par basic authentification. Il faut donc rajouter des éléments dans la requête permettant l'authentification (Q).

Il faut aussi autoriser les requêtes CORS (Cross-Origin Ressource Sharing) [24]. Cela a été fait du côté serveur chez Kelis. Côté client, il faut modifier le header (R).

Enfin, nous avons organisé le code sous la forme d'une library JavaScript pour rendre le code réutilisable et pour ne pas laisser de variables globales.

3.3.1 Récupération des données

On récupère les quatre jeux de données par agrégation. [20]
On regroupe les données par étudiant puis par question (2.2) ou par question puis par étudiant (2.3).
Les quatre requêtes correspondantes sont dans le code final et en ligne sur un repository GitLab. [25]

Une difficulté pour la récupération de données est la gestion de l'asynchronisme. Il n'est pas possible de savoir quand le serveur ElasticSearch va recevoir la requête puis renvoyer une réponse côté client.
Pour gérer ce problème, il faut appeler les procédures de post-traitement de données (*callback*) sur changements d'événements des objets Xhr (Annexe S).

Pour calculer le temps de réponse à une question, il est possible de faire une approximation en faisant la différence Δ de temps entre le premier événement *scored* et dernier événement *scored*. Il faut de plus vérifier que le champ *response.choice* existe pour ces événements car cela correspond a un choix fait par l'utilisateur. (Annexe V)

3.3.2 Traitement des données

Un exemple de données reçu est en annexe T.

Il faut récupérer le dernier événement *scored* correspondant au vrai score de l'étudiant ainsi que le nombre de changements de réponse. Une explication plus technique et détaillée est en annexe T.
Nous avons ajouté des fonctions annexes pour le traitement des données :

— Calcul de moyenne sur un tableau simple (resp. matrice), retourne une moyenne (resp. tableau de moyenne)
— Calcul de l'écart-type corrigé sur un tableau simple (resp. matrice), retourne l'écart-type (resp. tableau d'écart-type)
— Calcul de proportion (quartiles) sur un objet, retourne la valeur du quartile.

On récupère ces différentes données dans des tableaux pour les transmettre au framework de visualisation.

3.3.3 Framework de visualisation

Une fois les données formatées, nous pouvons enfin les afficher.

Le premier choix de framework s'est porté sur C3.js [27] car il est libre, open source, basé sur D3.js mais sans sa complexité. Il est très simple à prendre en main mais a montré des limitations pour personnaliser le layout des graphiques (en particulier les titres de graphique). De plus, il n'y a pas indépendance entre layout et données. (Voir annexe U)

Le choix final du framework s'est porté sur Plotly.js qui est open source (license MIT), basé sur D3.js et Stack.gl, utilisable dans d'autres langages (R, Python, MatLab, R), simple à utiliser, et dispose d'outil dynamique natif comme le zoom. Il propose aussi un plus grand choix de graphiques, du plus simple au plus avancé (3D), un layout plus personnalisable et une documentation plus fournie ([28]).

Dues au caractère asynchrone de la réception des données, les fonctions d'affichage des données sont appelées dans la fonction de callback, après traitement des données. Tous les paramètres, titres, légende, doivent donc être fournis à la fonction de callback pour qu'elle puisse à son tour passer ces paramètres à la fonction d'affichage. Pour cela, **on construit un objet** JavaScript auquel on rajoute des attributs contenant les paramètres nécessaires à l'affichage.

Enfin, encore dû à l'asynchronisme, il faut pour chaque jeu de données (donc un appel serveur) faire l'affichage de tous les graphiques pour ce jeu de données par une fonction inclue dans le callback.

Une difficulté rencontrée a été d'éviter la duplication de fonction d'affichage pour un résultat très similaire. Ce qui est en effet le cas avec les graphiques de nos quatre jeux de données. Nous avons donc utilisé une désignation communes des attributs pour les objets passés à la fonction de callback.
Le résultat est que nous avons 1 fonction d'affichage commune pour le score, et le nombre de changements de réponse (avec les proportions : quartiles, médiane) pour les quatre jeux de données.
L'autre fonction d'affichage commune aux jeux de données concerne l'affichage du nombre de changements de réponse (avec la moyenne).

3.4 Version Control System : GitLab UTC

Nous avons utilisé le logiciel open source *GitLab* ([29]) et en particulier celui interne à l'UTC. Cela permet la gestion du code, ainsi que la collaboration et le partage du projet. Le repository est en accès public donc il est visible par les utilisateurs qui

n'ont pas d'identifiant sur le GitLab de l'UTC mais est joignable par ceux qui en possèdent.

Pour déployer le code en ligne sur le serveur [1], il suffit de récupérer les derniers changements effectués sur le repository en se connectant au serveur.

Conclusion

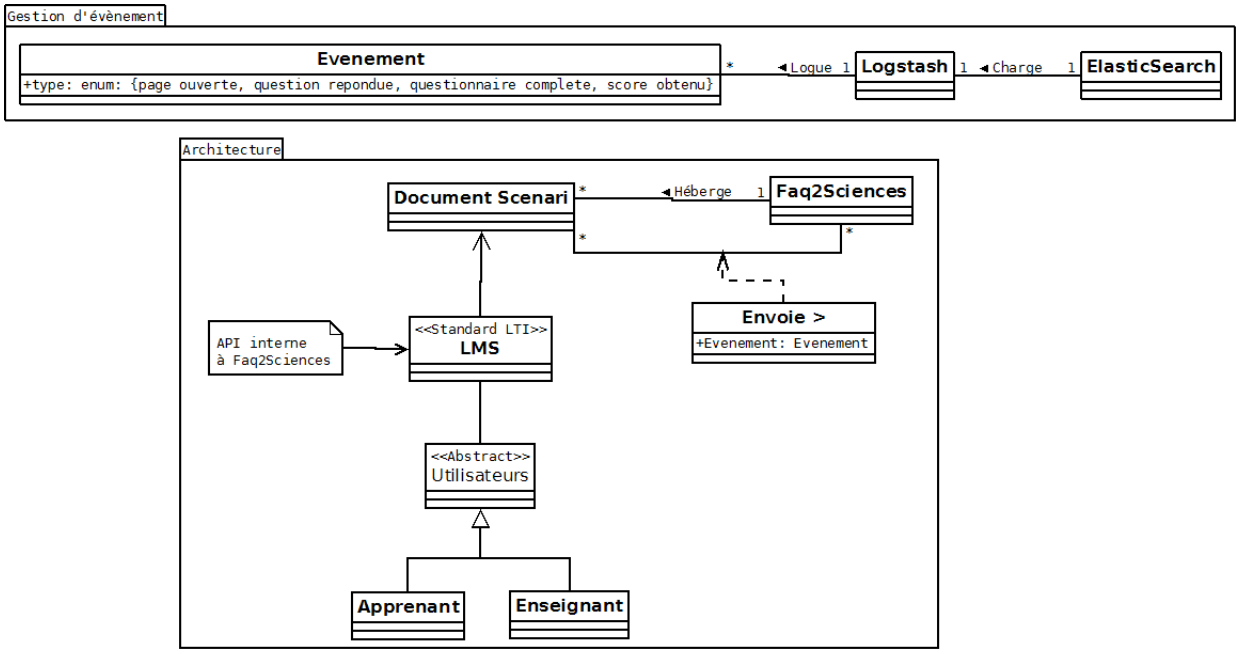
Cette étude expérimentale a permis de dégager trois axes d'analyse sur les données pédagogiques.

La mise en application de cette analyse a écarté les technologies qui auraient restreint les possibilités d'implémentation.

L'implémentation finale a principalement couvert les deux premiers axes par un tableau de bord développé en JavaScript avec le framework Plotly.js. Le code du livrable final se trouve à cette adresse [2] et est déployé sur ce site : [1].

Pour finir, l'évaluation finale de cette étude est disponible ici [30]. Il contient des pistes d'amélioration et d'évolution sur l'analyse et l'implémentation.

A Architecture du système



B Tirage aléatoire des étudiants aux questions des tests

Pour le questionnaire de physique (10 questions) :
— 3 questions parmi le groupe Phy01
— 3 questions parmi le groupe Phy02
— 1 question parmi le groupe Phy03
— 3 questions parmi le groupe Phy04

Pour le questionnaire de biologie (50 questions) :
— 10 questions parmi le groupe Bio01
— 3 questions parmi le groupe Bio02
— 13 questions parmi le groupe Bio03
— 14 questions parmi le groupe Bio04
— 5 questions parmi le groupe Bio05
— 5 questions parmi le groupe Bio06

C Structure des groupes de questions pour la physique

La structure des groupes de questions est identique pour la biologie.
Le fichier complet pour la biologie est disponible *ici* et *ici* pour la physique.

```
{
    "children": [
        {
            "id": "udFJDEATISe7oQfqnd9Kki", // id du groupe du
                questionnaire
            "title": "Quiz",
            "resultLink": "co/02-Main_1.html",
            "children":
            [
                {
```

```
                    "id": "qVTVybVQlxckqCv1EI54h",  // id du sous-
                        questionnaire 1
                    "title": "Phy01",
                    "children": [
                        {
                            "id": "mYYSeLORyOiHbhEbskfSOf", //id de la
                                question du sous-questionnaire
                            "title": "Question 9729"
                        },
                        {
                            "id": "CRbAfIUGOVgiJ4e3etjkKd",
                            "title": "Question 9735"
                        },
                        ...
                     ]
                },
                {
                    "id": "Dvj5AqVq4RiIbIFWaJuSud", // id du sous-
                        questionnaire 2
                    "title": "Phy02",
                    "children": [
                        {
                            "id": "kOM3Z9g8Fo3rQMssFEa9d",
                            "title": "Question 9495"
                        },
                        ...
                    ]
                 },
                 {
                  "id": "qgMx9nWg3feUknCfwecgli"
                 }
    ]
}
```

D Exemple du contenu d'un enregistrement

```
      {
        "_index": "faq2sciencesdistrib-2016.09.13",
        "_type": "XContentApi",
        "_id": "AVcjTPAYxCGFIcuyDQD1",
        "_score": null,
        "_source": {
          "evt": "distrib:logMsg",
          "type": "XContentApi",
          "response": {
            "choice": 2
          },
          "depot_path": "/Partenaires/UL/UL-Bio",
          "verb": "scored",
          "score_scaled": 1,
          "question_id": "Jg91o1JTiLhZQIH3PG2lBh",
          "clt_ts": 1473765983643,
          "score_raw": 1,
          "score_min": 0,
          "score_max": 1,
          "project_id": "2v",
          "participant_id": "5cy",
          "userNckNme": "Violette",
          "ltiTC": "moodle-UL",
          "ltiCtx": "15820",
```

```
          "user": "4924",
          "@version": "1",
          "@timestamp": "2016-09-13T11:27:01.503Z",
          "path": "/data/logs/f2s/webapps/sc/prl-distrib/actlog/
              act_current.log",
          "host": "depot-01.kelis.fr",
          "tags": [
            "faq2sciences-distrib",
            "distrib",
            "_grokparsefailure"
          ],
          "timestamp": 1473766021503
        }
```

E Description des attributs d'un enregistrement considérés et leurs valeurs

TABLE 1 – Description des attributs d'un enregistrement considérés et leurs valeurs

Attribut (clé)	Description	Valeur
_index	Nom de l'index ElasticSearch : Un index par jour	"faq2sciencesdistrib-2016.09.13", "faq2s__".09.14, .09.15, .09.16, .09.19, .09.20, .09.21, .09.22, .09.24
_type	Nom du type ElasticSearch	"XContentApi"
_source.response	Sous clé de _source Indique un choix de réponse de l'utilisateur	
_source.response.choice	Sous-clé de _source.response Valeur du choix de la réponse de l'utilisateur	Entier N
depot_path	Chemin de la ressource jouée depuis la plateforme	"/Partenaires/UL/UL-Bio", "/Partenaires/UL/UL-Phy01"
verb	Événement généré	"responded", "attempted", "scored", "completed"
score_scaled	Score obtenu ramené à 1 pour un étudiant. Cela peut aussi bien concerner une question qu'un questionnaire	Réel entre [0;1]
question_id	Identifiant de la question ou du questionnaire	Voir annexe C
clt_ts	Timestamp de l'évènement généré côté client	Heure POSIX (en milliseconde)
score_raw	Score brut obtenu sur la question/le questionnaire (ou nombre de réponses correctes) par un étudiant	1 pour une question Entier entre [0;50] pour la biologie Entier entre [0;10] pour la physique
score_min	Score minimum possible de la question/questionnaire	0
score_max	Score maximum possible de la question/questionnaire	Variable pour une question 50 pour la biologie 10 pour la physique
project_id	Contexte de diffusion : un id par ressource	
participant_id	Identifiant interne du participant, plus ou moins équivalent à user (un user peut avoir plusieurs participant_id)	
userNckNme	Surnom de l'utilisateur, généralement son prénom	
user	Identifiant interne de l'utilisateur, généré automatiquement	Entier >= 4913
timestamp	Timestamp côté serveur	Heure POSIX (en milliseconde)

F Analyse par la date des événements générés

Attention, ces graphiques ont été faits sur Kibana et comprennent donc des données indésirables qui n'ont pas pu être filtrées (initialisation du test, ensemble des changements de réponse). (Voir 3.2)

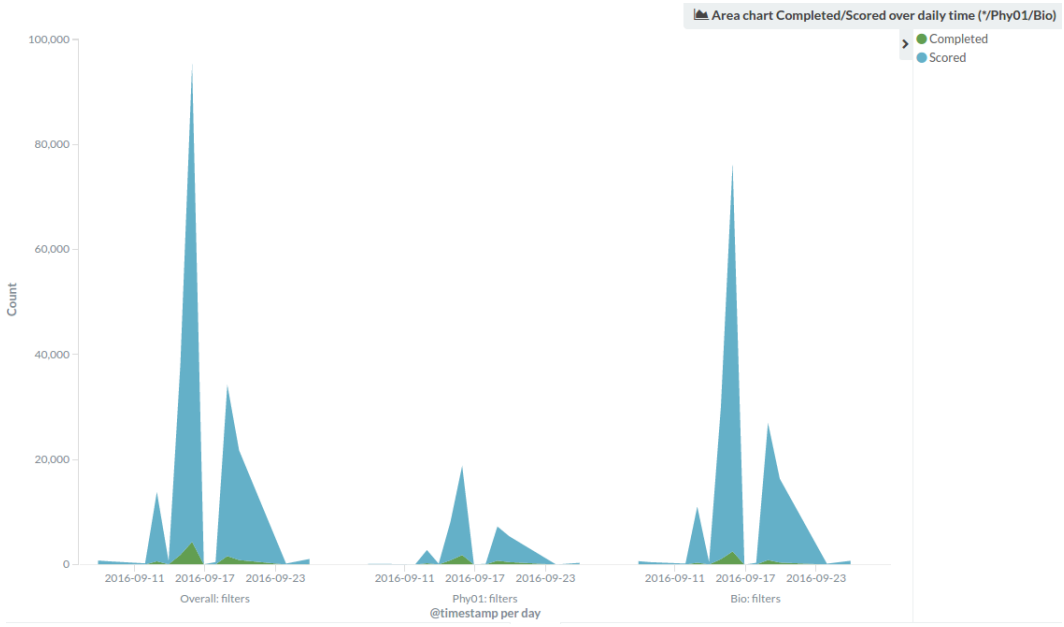


FIGURE 1 – Area chart : Nombre d'événements *"scored"* ou *"completed"* en fonction du temps pour la physique, la biologie et les deux questionnaires

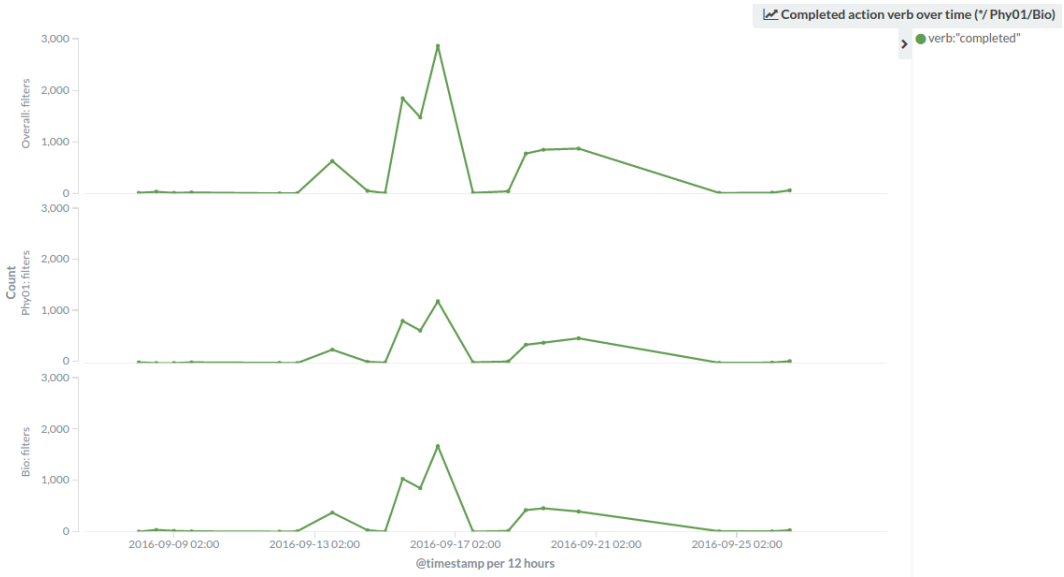


FIGURE 2 – Line chart : Nombre d'événements, *"completed"* en fonction du temps pour la physique, la biologie et les deux questionnaires

G Analyse par étudiant : Graphique du score sur Kibana

Attention, ces graphiques ont été faits sur Kibana et comprennent donc des données indésirables qui n'ont pas pu être filtrées (initialisation du test, ensemble des changements de réponse)

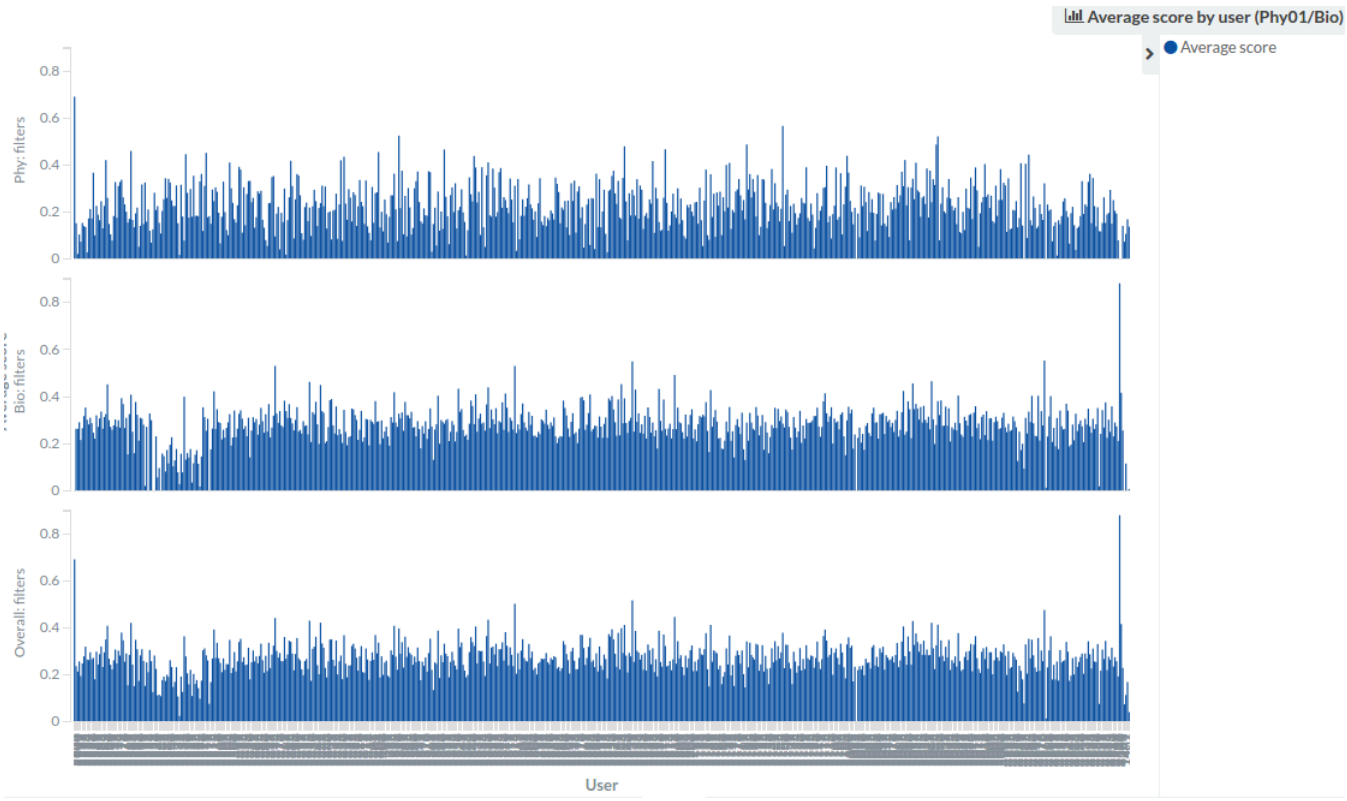


FIGURE 3 – Score moyen par utilisateur sur les questionnaires de physique, biologie et les deux questionnaires

H Analyse par étudiant : Graphique final du score

Ces deux graphiques font partie du livrable final. À noter que les utilisateurs ont répondu à 10 questions pour la physique contre 50 questions pour la biologie.

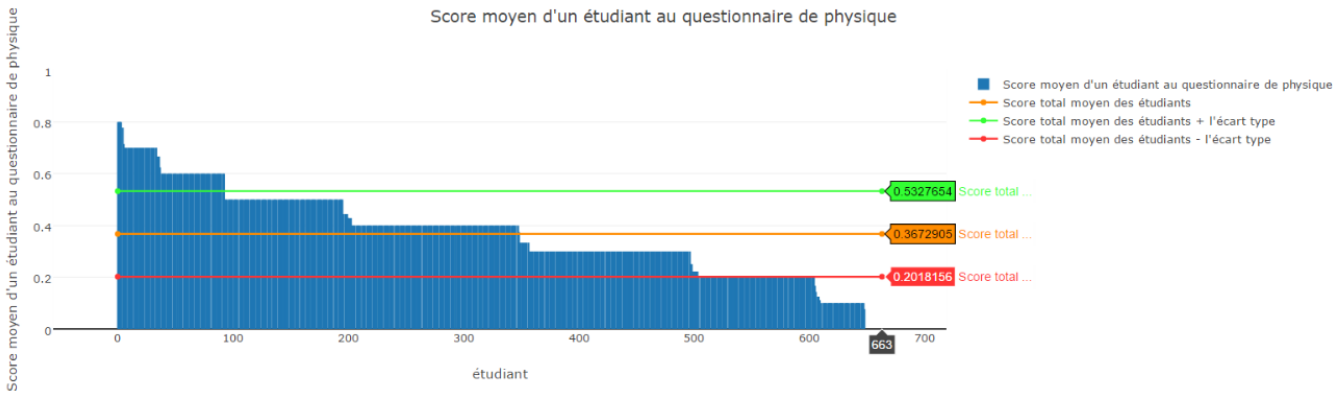


FIGURE 4 – Score moyen par utilisateur sur le questionnaire de physique

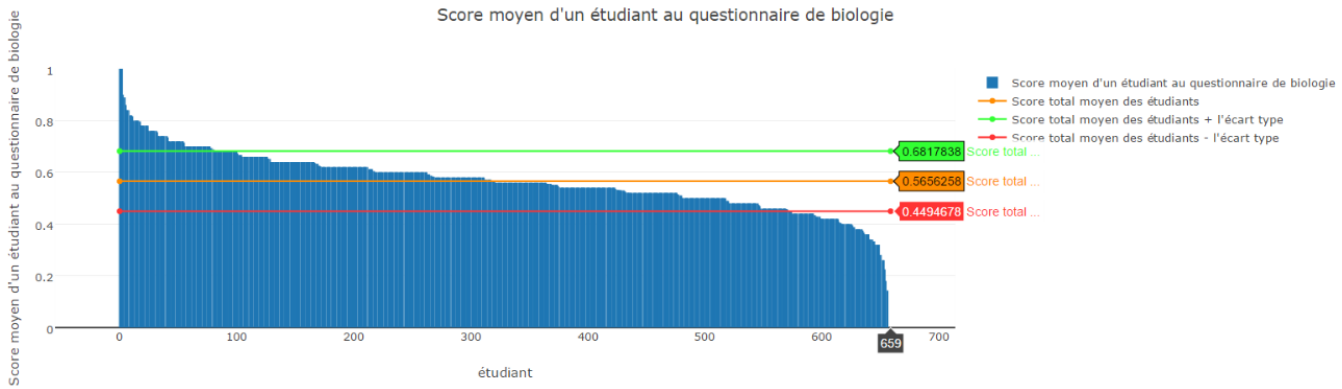


FIGURE 5 – Score moyen par utilisateur sur le questionnaire de biologie

I Analyse par étudiant : Graphique final du nombre de changements de réponse

Ces deux graphiques font partie du livrable final.

Les graphiques pour la biologie étant très similaires, ils ne seront pas présentés dans ce rapport (mais font évidemment partie du livrable final)

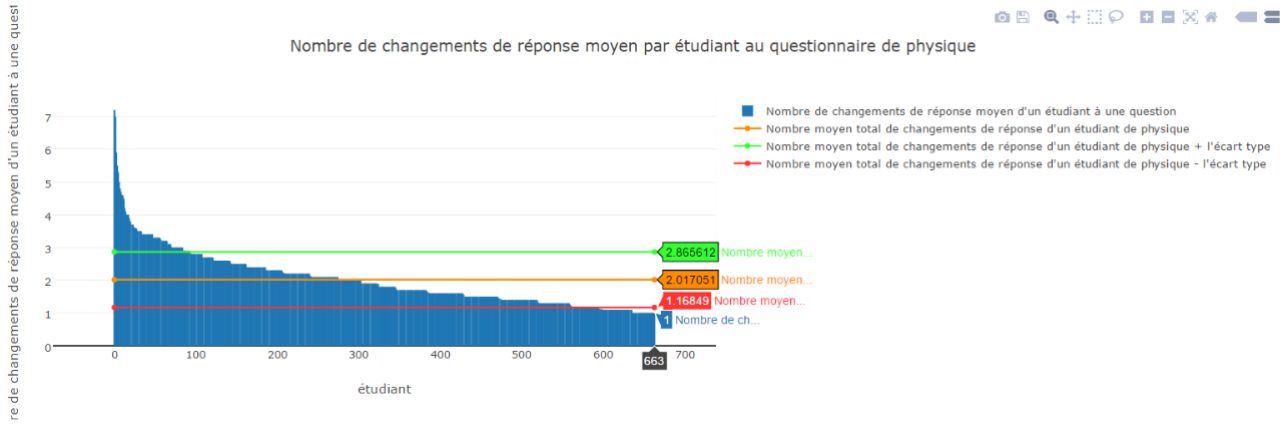


FIGURE 6 – Nombre de changements de réponse moyen par étudiant au questionnaire de physique

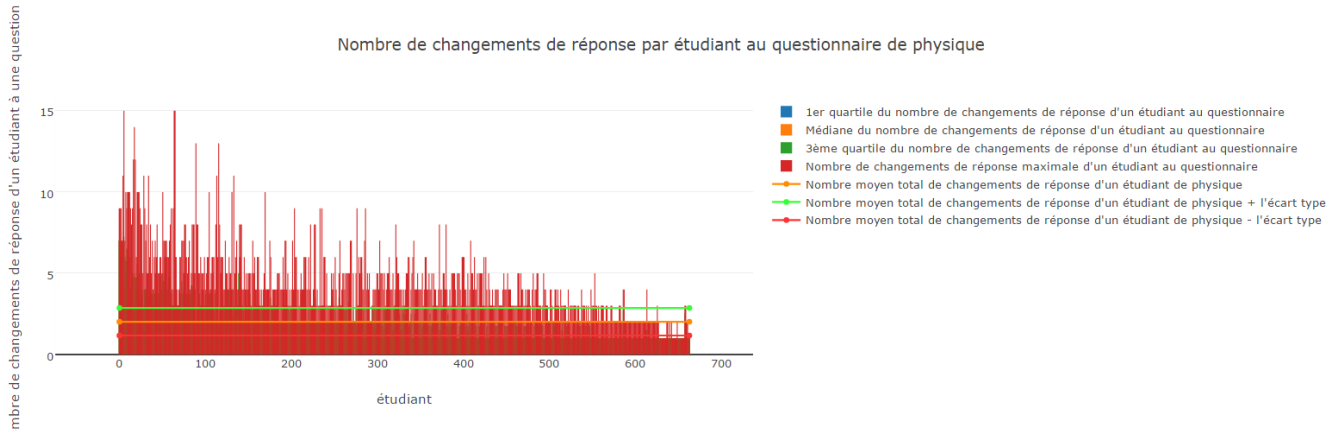


FIGURE 7 – Nombre de changements de réponse par étudiant au questionnaire de biologie (quartiles, médiane)

En zoomant la figure 7, on obtient :

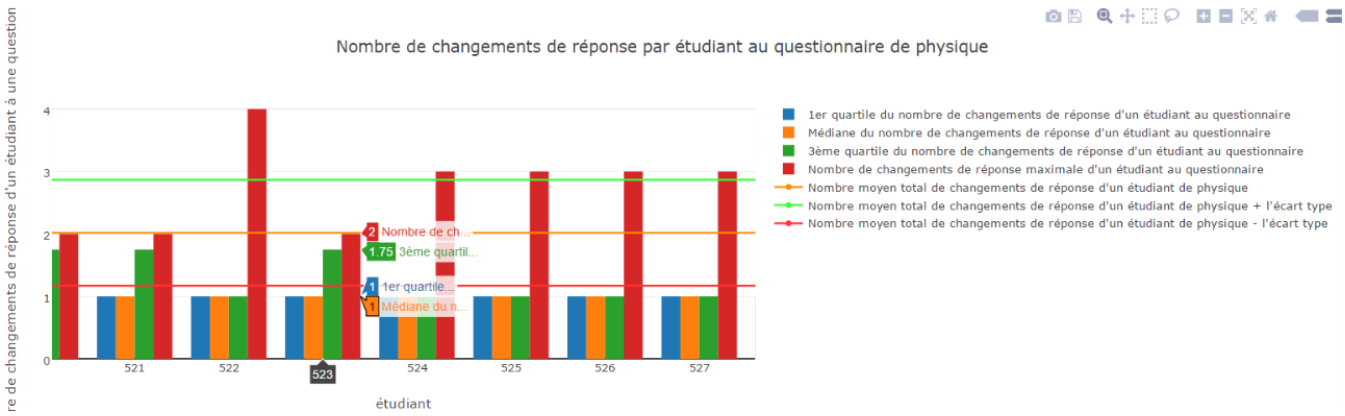


FIGURE 8 – Nombre de changements de réponse par étudiant au questionnaire de biologie (quartiles, médiane)

J Exemple de gauge chart

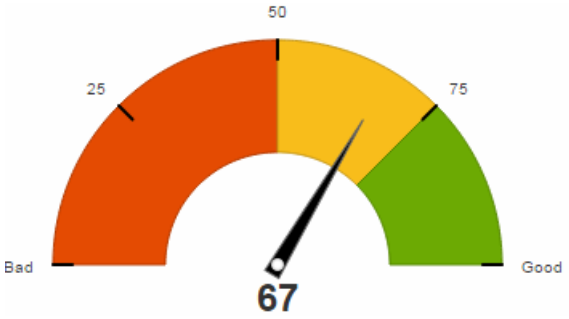


FIGURE 9 – Exemple de gauge chart

K Analyse par question : Graphique final du score

Il y a 31 questions différentes pour la physique et 211 pour la biologie.

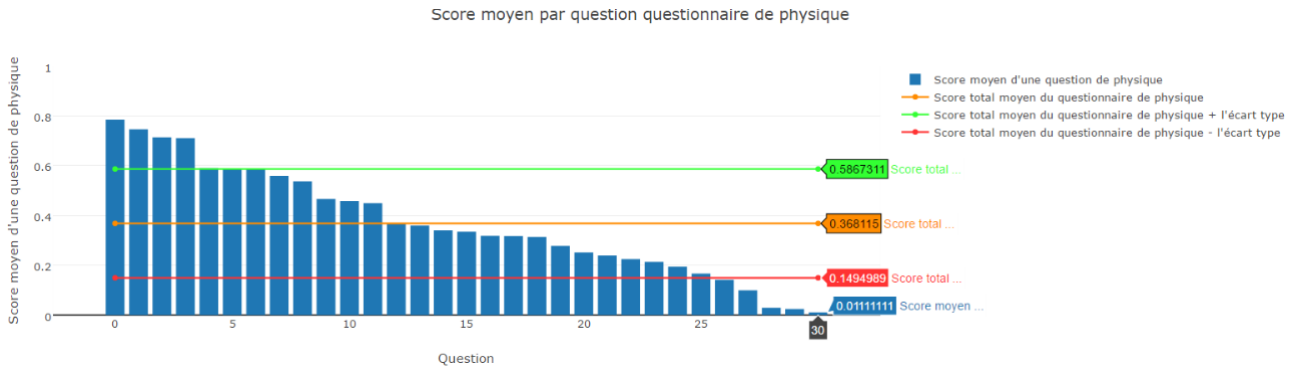


FIGURE 10 – Score moyen par question au questionnaire de physique

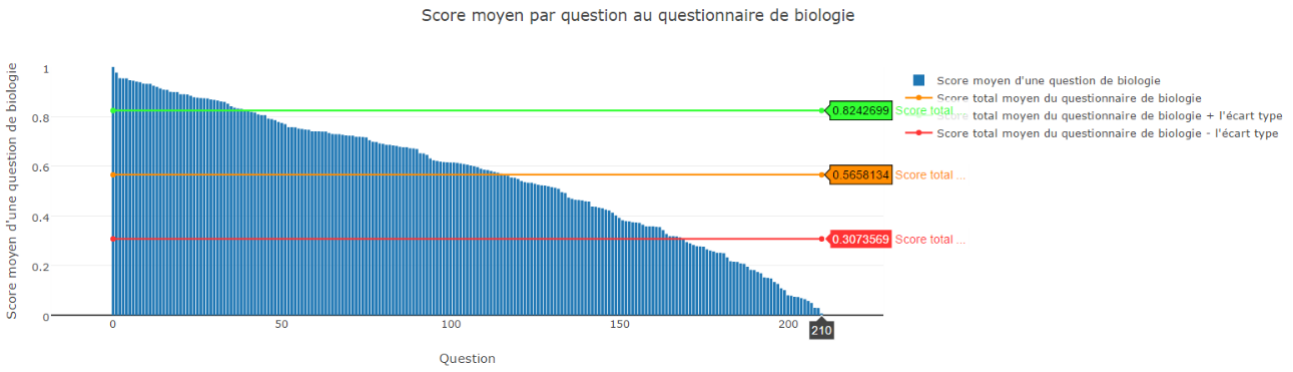


FIGURE 11 – Score moyen par question au questionnaire de biologie

L Analyse par question : Graphique final du nombre de changements de réponse

Il y a 31 questions différentes pour la physique et 211 pour la biologie.

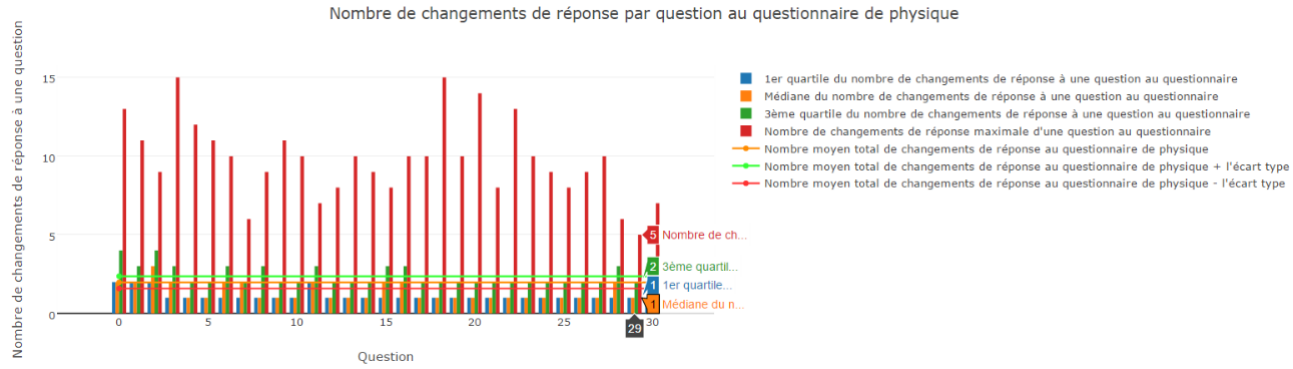


FIGURE 12 – Nombre de changements de réponse moyen par question au questionnaire de physique

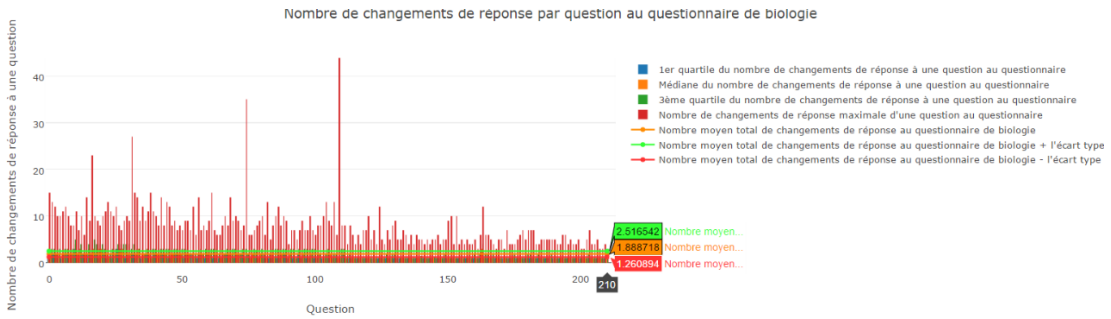


FIGURE 13 – Nombre de changements de réponse par question au questionnaire de biologie (quartiles, médiane)

En zoomant la figure 13, on obtient :

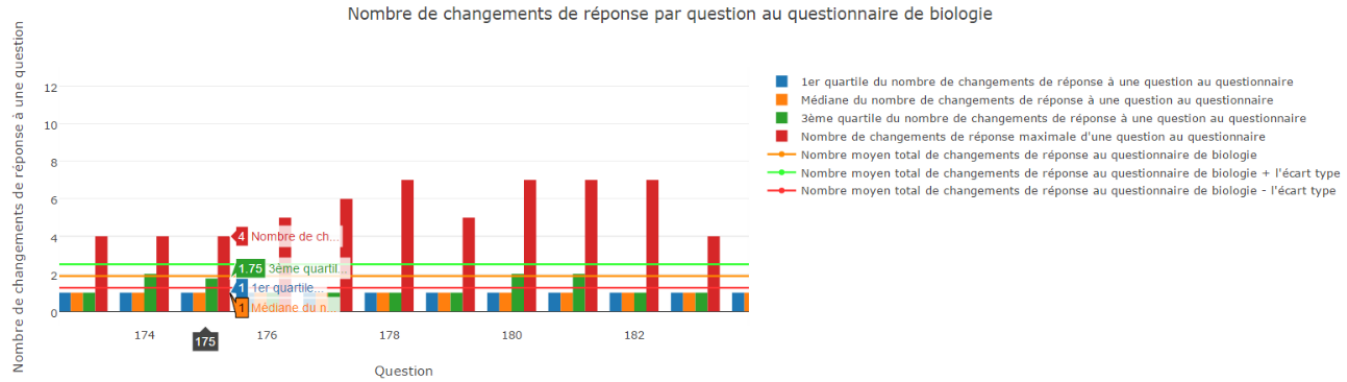


FIGURE 14 – Nombre de changements de réponse par question au questionnaire de biologie (quartiles, médiane)

M Analyse par question : Graphique du nombre d'événements *completed scored* et *attempted*

Il y a 31 questions différentes pour la physique et 211 pour la biologie.

Attention, ces graphiques ont été faits sur Kibana et comprennent donc des données indésirables qui n'ont pas pu être filtrées (initialisation du test, ensemble des changements de réponse)

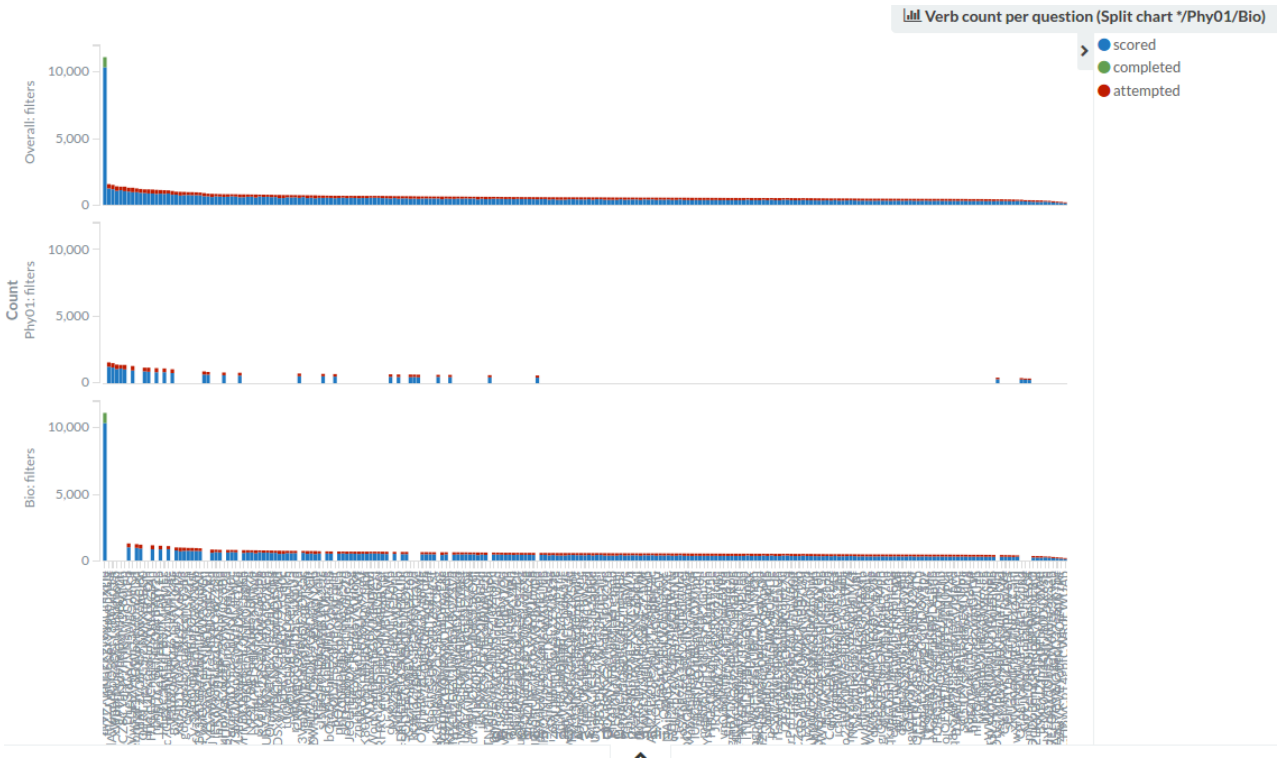


FIGURE 15 – Nombre d'événements *completed scored* et *attempted* par question au questionnaire de physique, biologie et aux deux questionnaires

N Analyse par groupe de questionnaires : Graphique du score par questionnaire

Attention, ces graphiques comprennent des données indésirables qui n'ont pas pu être filtrées (données d'initialisation du test)

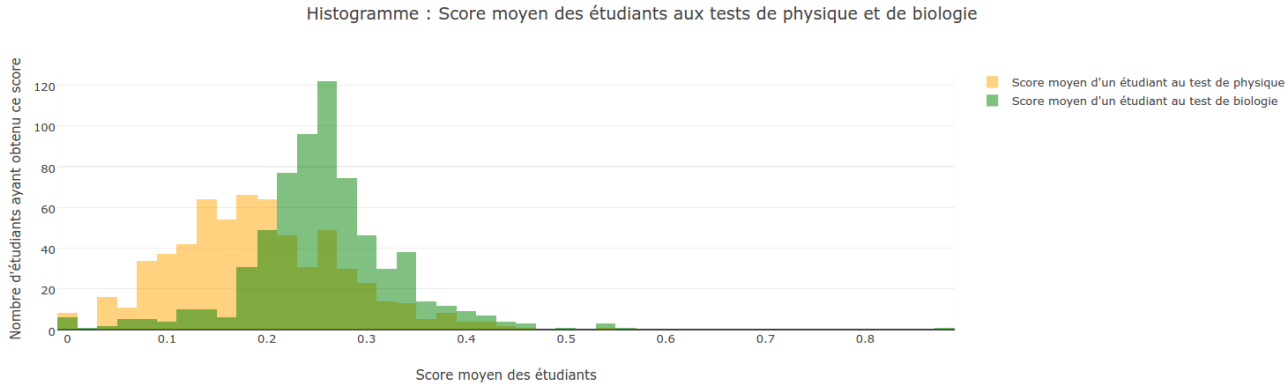


FIGURE 16 – Histogramme : Score moyen des étudiants aux tests de physique et de biologie

O Analyse par question : Combinaison de paramètre - Exemple de line chart

Voici un exemple de line chart superposé.

— En rouge, on pourrait avoir le score moyen au questionnaire en fonction d'une question
— En bleu, on pourrait avoir le nombre de changements moyens de réponse au questionnaire en fonction d'une question
— En vert, on pourrait avoir le temps moyen du temps de réponse au questionnaire en fonction d'une question

On peut considérer l'ensemble des étudiants sur un questionnaire ou aussi pour un seul étudiant en particulier. Dans ce cas-là, ce n'est pas la moyenne des valeurs d'un paramètre mais plutôt sa valeur effective (i.e. le score de l'étudiant, le temps de réponse et le nombre de changements à une question).

Si le temps de réponse est visuellement supérieur au nombre de changements de réponse alors on peut en déduire que l'utilisateur a bien pris le temps de réfléchir avant chaque changement de réponse. Cela peut aussi bien correspondre à une anomalie de données.

Au contraire, si c'est le nombre de changements de réponse qui est supérieur alors on peut en déduire que l'utilisateur ne connaît pas la réponse au choisi peut être aléatoirement ses/sa réponse-s.

Si le temps de réponse est visuellement proche du nombre de changements, on distingue alors deux cas :

— Ces valeurs sont hautes (**adjectif à qualifier et à formaliser : aucune recherche bibliographique n'a été faite à ce sujet*) : dans ce cas, on peut en déduire que l'utilisateur a beaucoup hésité, car il/elle a mis du temps à répondre et a beaucoup changé de réponse
— Ces valeurs sont basses : L'utilisateur n'a pas/peu hésité et connaît la réponse à la question.

Pour chaque situation, le score peut renforcer/contredire l'hypothèse.
En particulier, si le scénario est positif pour l'utilisateur alors un bon score renforce l'hypothèse faite, sinon cela la contredit et/ou est une anomalie.
Et inversement si le scénario est négatif pour l'utilisateur : un mauvais score renforce l'hypothèse, sinon cela la contredit.

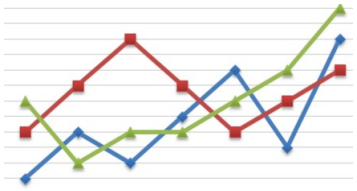


FIGURE 17 – Exemple de line chart superposé

P Extrait du mapping d'un index

Une requête sur un champ non analysé cherche le texte exact contrairement au champ analysé où il est possible de chercher les dérivations d'un mot.

```
1                    "userNckNme": {
2                        "type": "string", //userNckNme est une chaine de
                             caractere
```

```
                        "norms": {
                            "enabled": false
                        },
                        "fielddata": {
                            "format": "disabled"
                        },
                        "fields": {
                            "raw": {  // s'il y a un champ raw alors il
                                existe un champ non analyse pour cet
                                attribut, qui est a privilegier pour les
                                requetes
                                "type": "string",
                                "index": "not_analyzed",
                                "ignore_above": 256
                            }
                        }
                    },
                    "verb": {
                        "type": "string", //verb est une chaine de
                            caractere
                        "norms": {
                            "enabled": false
                        },
                        "fielddata": {
                            "format": "disabled"
                        },
                        "fields": {
                            "raw": {
                                "type": "string",
                                "index": "not_analyzed",
                                "ignore_above": 256
                            }
                        }
                    }
```

Q Basic authentification : modification du header

```
vXhr.setRequestHeader("Authorization", "Basic " + btoa("login:password"));
```

R CORS : modification du header

```
    vXhr.withCredentials = true;
```

S Gestion de l'asynchronisme en JavaScript

Dès que l'objet vXhr est reçu et qu'il n'y a pas de problème alors une fonction dite de *callback* peut alors effectuer le post-traitement des données (formatage, affichage).

```
  vXhr.onreadystatechange = function () {

        if (vXhr.readyState == 4 && (vXhr.status == 200 || vXhr.status==0)
            )
        {// Si pas d'erreur -> traitement

            console.log("ready");
            callback(JSON.parse(vXhr.responseText), graph_data);
            return vXhr.responseText;
    }
```

T Exemple d'enregistrement reçu avant traitement

Agrégation par utilisateur puis par question (biologie).

On expliquera le processus pour traiter les données. La démarche est très similaire à l'agrégation par question puis par utilisateur.

Les attributs qui nous intéressent pour le traitement de données sont (des objets vers les sous-objets imbriqués) :

— le premier "key" : correspond à l'id de l'utilisateur. Toutes les données concernant cet utilisateur sont comprises *dans* les sous-objets puisque l'on fait une agrégation par utilisateur, puis par question. On a donc un tableau de question dans le sous-objet *"questions"*.

— le deuxième "key" : correspond à l'id d'une question tirée pour l'utilisateur

— "doc_count" : le nombre d'enregistrements existants pour cette question *et* générés par cet utilisateur. C'est donc le nombre de changements de réponse à cette question pour cet utilisateur.

— "timestamp" (dans le sous-objet *"max_time"*)

— "score_scaled" : valeur du dernier événement score enregistré à cette question par l'utilisateur. C'est donc le vrai score de l'utilisateur à cette question.

On récupère les valeurs de ces attributs pour un utilisateur sur toutes les questions auxquelles il a répondu. On réitère ceci pour tous les utilisateurs.

Dans le cas d'une agrégation par question puis par utilisateur, la seule différence est que l'ordre des *"key"* est inversé : les utilisateurs deviennent les sous-objets imbriqués par rapport à une question.

Dans ce cas, il faut récupérer les valeurs pour une question sur tous les utilisateurs ayant répondu à cette question. On réitère ceci pour toutes les questions.

```
"aggregations": {
        "users": {
            "doc_count_error_upper_bound": 0,
            "sum_other_doc_count": 0,
            "buckets": [
                {
                    "key": "5350",  //id de l'etudiant
                    "doc_count": 136, //nombre d'enregistrements existant
                        pour cet utilisateur
                    "questions": {
                        "doc_count_error_upper_bound": 0,
                        "sum_other_doc_count": 0,
                        "buckets": [
                            {
                                "key": "KVUKxs2xDclYMWxpBriF6", //id de la
                                    question
                                "doc_count": 11,  //nombre de changements
                                    de reponse de l'etudiant pour cette
                                    question
                                "max_time": { //derniere reponse
                                    enregistree par l'etudiant pour cette
                                    question
                                    "hits": {
                                        "total": 11,
                                        "max_score": null,
                                        "hits": [
                                            {
                                                "_index": "
                                                    faq2sciencesdistrib
                                                    -2016.09.16",
                                                "_type": "XContentApi",
```

```
                                    "_id": "
                                        AVczNiLCxCGFIcuyEtYv",
                                    "_score": null,
                                    "fields": {
                                        "timestamp": [
                                            1474032965617 //
                                                timestamp de la
                                                 derniere
                                                reponse
                                        ],
                                        "score_scaled": [
                                            0 //score de l'
                                                utilisateur a
                                                cette question
                                        ]
                                    },
                                    "sort": [
                                        1474032965617
                                    ]
                                }
                            ]
                        }
                    },
                    "min_time": {
                        "hits": {
                            "total": 11,
                            "max_score": null,
                            "hits": [
                                {
                                    "_index": "
                                        faq2sciencesdistrib
                                        -2016.09.16",
                                    "_type": "XContentApi",
                                    "_id": "
                                        AVczKCmfxCGFIcuyEo7W",
                                    "_score": null,
                                    "fields": {
                                        "timestamp": [
                                            1474032049434
                                        ]
                                    },
                                    "sort": [
                                        1474032049434
                                    ]
                                }
                            ]
                        }
                    }
                },
```

U Framework de visualisation : Limites de C3.js

La légende des données en abscisse et ordonnée doit être fournie dans le tableau de donnée à afficher. Il faut donc rajouter ces légendes en début de liste pour chaque données à afficher.

```
c3.generate({
    bindto: '#bar_avg_user',
    data: {
        columns: [
            ["label1",1 ,2,5,6],
            ["label2",10,3,4,2],

        ],
```

FIGURE 18 – Syntaxe de C3.js pour les structures de données : Non indépendance entre layout et données

V Extrait d'un exemple de la timeline d'un utilisateur durant un questionnaire pour le temps de réponse

Pour calculer le temps de réponse à une question, il faut calculer la différence de temps entre le premier événement *scored* (avec le champ response) et le dernier événement *scored* (avec le champ response). Il faut réitérer ceci pour chaque question et chaque utilisateur.

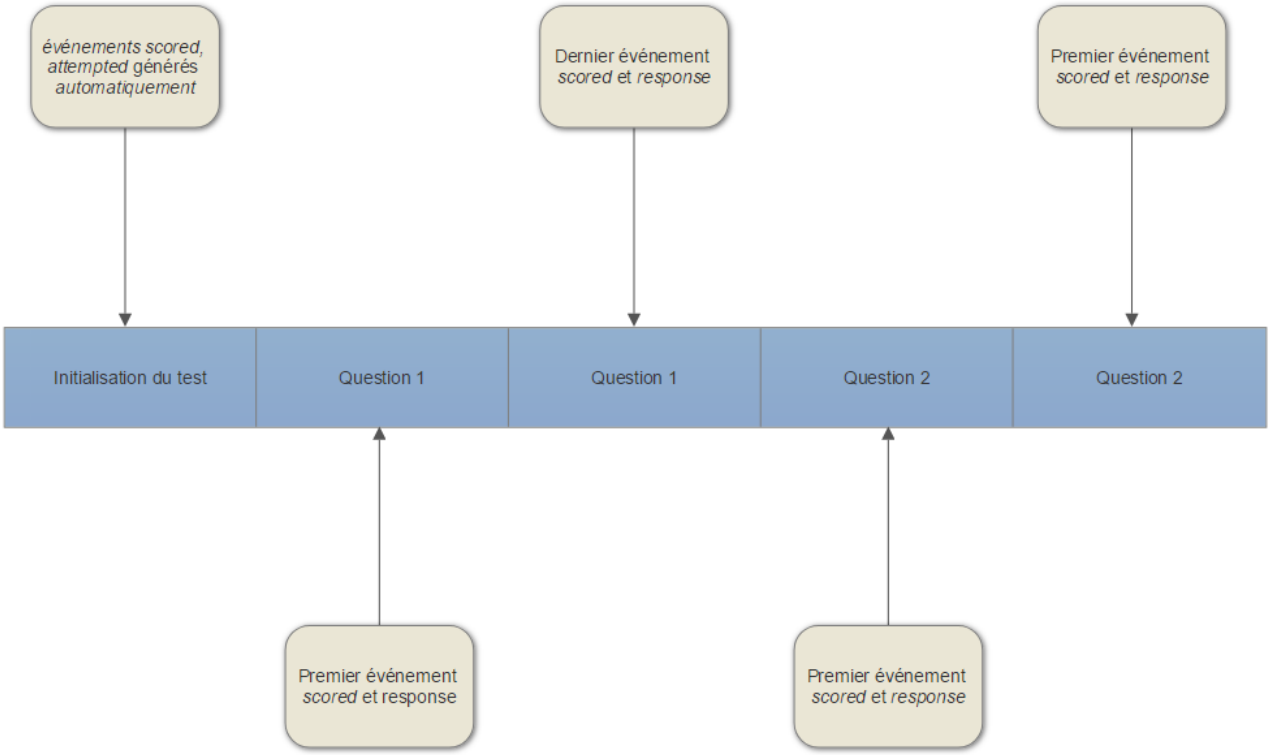


FIGURE 19 – Extrait de la timeline d'un utilisateur

Références

[1] Faq2Sciences - TX - Menu du tableau de bord http://pic.crzt.fr/txf2s/TX-faq2sciences/.

[2] Files · master · Minh Tri Lê / TX-faq2sciences · GitLab https://gitlab.utc.fr/leminhtr/TX-faq2sciences/tree/master.

[3] La chaîne éditoriale [Scenari.org] https://scenari.org/co/principes.html.

[4] Faq2Sciences, se tester pour bien préparer son entrée à la fac https://www.faq2sciences.fr/.

[5] Learning management system - EduTech Wiki http://edutechwiki.unige.ch/en/Learning_management_system.

[6] Elasticsearch : RESTful, Distributed Search & Analytics | Elastic https://www.elastic.co/products/elasticsearch.

[7] Logstash : Collect, Parse, Transform Logs | Elastic https://www.elastic.co/products/logstash.

[8] Kelis https://www.kelis.fr/co/kelis.html.

[9] Index vs. Type https://www.elastic.co/blog/index-vs-type.

[10] Overview - Tin Can API http://tincanapi.com/overview/.

[11] *Yeonjeong Park, Il-Hyun Jo*, Development of the Learning Analytics Dashboard to Support Students' Learning Performance http://www.jucs.org/jucs_21_1/development_of_the_learning/jucs_21_01_0110_0133_park.pdf.

[12] KlassData | Learning Analytics Moodle - SmartKlass http://klassdata.com/smartklass-learning-analytics-plugin/learning-analytics-moodle-smartklass/.

[13] Learning Locker - The Open Source Learning Record Store https://learninglocker.net/.

[14] Variable bin sizes for histograms #360 https://github.com/plotly/plotly.js/issues/360.

[15] Bar chart with relative barmode https://plot.ly/javascript/bar-charts/#bar-chart-with-relative-barmode.

[16] Grouped bar chart https://plot.ly/javascript/bar-charts/#grouped-bar-chart.

[17] REST API Quick Tips http://www.restapitutorial.com/lessons/restquicktips.html

[18] Postman | Supercharge your API workflow https://www.getpostman.com/

[19] Mapping | ElasticSearch Reference [5.4] | Elastic https://www.elastic.co/guide/en/elasticsearch/reference/current/mapping.html

[20] Aggregations | ElasticSearch Reference [5.4] | Elastic https://www.elastic.co/guide/en/elasticsearch/reference/5.4/search-aggregations.html

[21] Query DSL | ElasticSearch Reference [5.4] | Elastic https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html

[22] Kibana : Explore, Visualize, Discover Data | Elastic https://www.elastic.co/products/kibana

[23] Dynamisez vos sites web avec JavaScript ! https://openclassrooms.com/courses/dynamisez-vos-sites-web-avec-javascript

[24] DÉMYSTIFIER CORS (CROSS-ORIGIN RESOURCE SHARING) http://blog.inovia-conseil.fr/?p=202

[25] dashboard.js · master · Minh Tri Lê / TX-faq2sciences · GitLab https://gitlab.utc.fr/leminhtr/TX-faq2sciences/blob/master/dashboard.js#L42

[26] Mapping d'un index ElasticSearch de Faq2Sciences https://kibana4.kelis.fr/~~es/faq2sciencesdistrib-2016.09.26/_mapping

[27] C3.js | D3-based reusable chart library http://c3js.org/

[28] plotly.js chart attribute reference https://plot.ly/javascript/reference/

[29] Help . GitLab https://gitlab.utc.fr/help

[30] 2017P-tx-jury-faq2sciences | Etherpad https://pad.picasoft.net/p/2017P-tx-jury-faq2sciences