

# SI40 Rapport : Partie 2



Matthieu DIEBOLT, Pierre GUEROUT, Joshua PLOUZENNEC, Mateo CHARTIER, Dave JONATHAN SUAREZ

## Table des matières

Introduction .....	3
Les technologies utilisées et l'architecture générale .....	4
La description des fonctionnalités développées .....	6
Gestions des documents (images et PDF) .....	6
Les devoirs .....	6
Les forums .....	7
Le journal de logs .....	7
Les autres fonctionnalités .....	8
Les choix de modélisation de la base .....	9
Gestions des documents (images et PDF) .....	11
Les devoirs .....	12
Les forums .....	13
Le journal de logs .....	15
L'API REST .....	19
PostgreSQL .....	19
MongoDB .....	20
Le serveur .....	21
Les captures .....	21
Les devoirs .....	21
Les forums .....	22
Le journal de logs : .....	23
Gestion des images et PDF : .....	25
MongoDB Analytics Implementation .....	25
Conclusion .....	27

# Introduction

Dans le cadre du projet collaboratif entre les unités d'enseignement SI40 et WE4B, nous avons poursuivi l'implémentation d'une plateforme pédagogique inspirée de Moodle. Après avoir réalisé une première version avec Symfony dans le cadre de WE4A, l'objectif de cette seconde partie consistait à redévelopper la partie web en utilisant Angular pour WE4B. Ce rapport se concentre sur la partie SI40, axée sur la conception et la gestion de la base de données du projet.

L'objectif principal de cette seconde partie, du point de vue de la base de données, était de reprendre la base de données SQL utilisée lors de la première phase et d'y ajouter une base de données NoSQL orientée documents, afin d'intégrer de nouvelles fonctionnalités à notre projet. Pour cela, nous avons utilisé MongoDB, une solution particulièrement adaptée à la gestion flexible de données complexes au format JSON.

Dans ce rapport, nous présentons l'architecture technique retenue, les fonctionnalités développées ainsi que les choix de modélisation effectués pour répondre aux besoins du projet. Ce projet nous a permis de mettre en pratique et d'approfondir les notions vues en cours et en travaux pratiques tout en développant une solution adaptée à un contexte concret.

## Les technologies utilisées et l'architecture générale

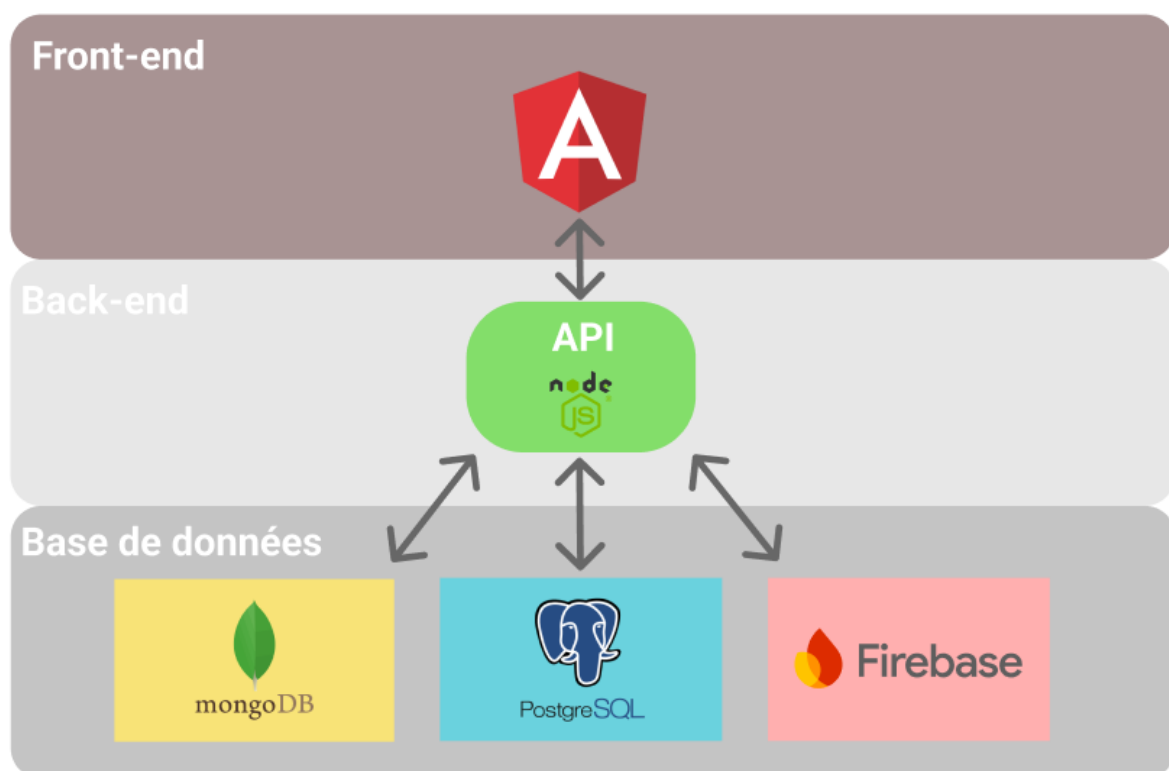
Pour la réalisation de ce projet, nous avons opté pour une architecture couramment utilisée dans le développement web, une architecture à trois niveaux : un front-end, un back-end et une partie base de données.

Du côté du front-end (partie WE4B), nous avons utilisé Angular, un framework web permettant de réaliser une interface utilisateur moderne, dynamique et réactive.

Le back-end, lui, a été réalisé avec AngularJS, qui a servi à la création de l'API REST permettant la communication entre le front-end et la base de données.

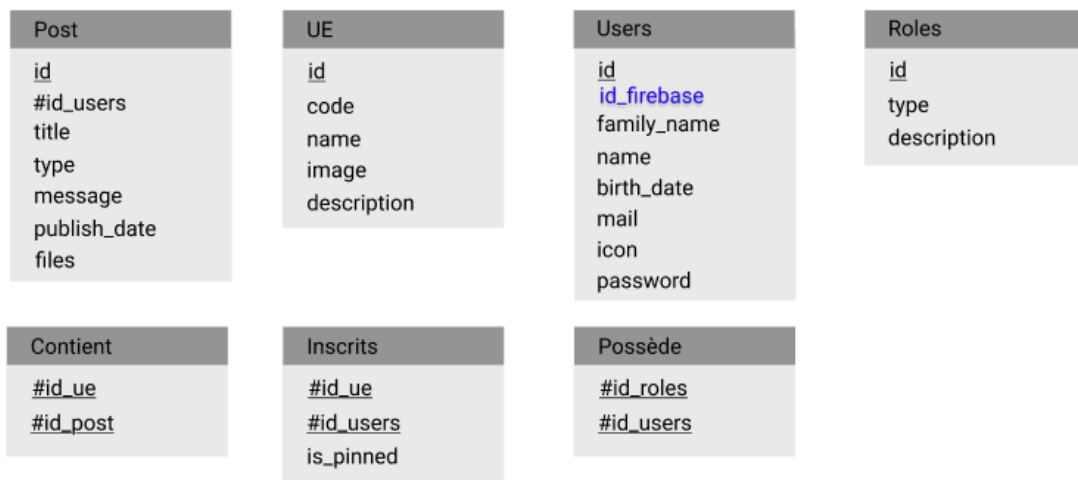
Enfin, pour le stockage et la gestion des données, nous avons opté pour la combinaison de MongoDB, PgAdmin et Firebase.

Voici un schéma de l'organisation des technologies utilisées :



Nous avons choisi de réutiliser PgAdmin pour reprendre le schéma de la base de données ainsi que les données que nous avons utilisées lors de la 1ère partie. Cela nous permet de ne pas perdre de temps à recréer le schéma dans un autre format et de travailler sur des données déjà créées pour la 1ère partie. PgAdmin est un outil d'administration pour les bases de données relationnelles PostgreSQL. Ainsi, nous utilisons exactement le même schéma que la 1ère partie à une modification près : nous avons ajouté un `id_firebase` dans `users` pour faire le lien entre Firebase et PgAdmin.

Rappel du schéma SQL :



MongoDB est une base de données NoSQL orientée documents, particulièrement adaptée à la gestion flexible de données complexes et évolutives au format JSON. Nous l'avons utilisée pour stocker les images et documents de l'application ainsi que pour implémenter les nouvelles fonctionnalités de notre application web en créant des collections et documents NoSQL. Nous stockons les images grâce à GridFS (un système de stockage de fichiers volumineux dans MongoDB)

Enfin, Firebase est une plateforme de services cloud développée par Google. Nous avons choisi d'utiliser cette base de données afin de gérer l'authentification, car Firebase propose des fonctionnalités intéressantes à cet effet comme une mise en place rapide et une sécurité renforcée. Il a l'avantage de nécessiter aucune manipulation de démarrage comme la base de données reste actif en web.

## La description des fonctionnalités développées

Nous avons grâce à l'ajout de MongoDB intégrer plusieurs nouvelles fonctionnalités, en plus de celle déjà implémenté dans la 1ère partie, parmi elles, la gestion des documents, les devoirs, les forums et un journal de logs.

### Gestions des documents (images et PDF)

Dans la 1ère partie du projets, nous stockions les images et les documents dans un dossiers local dans le code du projet. Cette méthode n'est pas adaptée à un projet en production, où il est mieux de stocker les fichiers dans une base de données afin d'assurer une meilleure sécurité, accessibilité et gestion des données.

Ainsi, dans cette version du projet, nous avons ajouté la possibilité de téléverser des documents PDF pour les posts et les devoirs, ainsi que la gestion des images des cours avec la base de données MongoDB

### Les devoirs

Pour les devoirs, nous avons repris la base des postes en ajoutant quelques modifications. La plus important étant la base de données, en effet cette fois ci, nous utilisons MongoDB pour assurer la gestion des devoirs.

Pour la création, cela fonctionne comme un poste de type de texte ou dépôt. A l'exception que lors de la création, une date de rendu est demandée au créateur. Et bien sûr, nous ne demandons pas de fichier pour la création.

Pour pouvoir rendre un devoir, une page spéciale a été créé pour les étudiants. Ils peuvent y déposer leur fichier PDF ainsi que consulter leur note et voir si le professeur a mis un commentaire.

Du côté professeur, une autre page a été mise en place pour pouvoir récupérer les fichiers de tous les élèves ayant soumis un fichier. Cela se présente sous forme d'un tableau dans lequel le professeur peut mettre une note, un commentaire et consulter le fichier PDF de chaque élève.

## Les forums

Notre plateforme intègre un système de forums permettant aux utilisateurs (étudiants et enseignants) d'échanger dans le cadre d'un cours autour d'un sujet choisi.

Pour chaque cours, un espace « forum » est défini où l'on peut voir les sujets de discussion proposés pour ce cours. Les utilisateurs peuvent ajouter leurs sujets et les enseignants peuvent supprimer les sujets s'ils le souhaitent.

Les utilisateurs peuvent entrer dans un sujet de discussion pour échanger à son propos. Tous les utilisateurs peuvent poster un message dans la discussion. Au niveau de la modération, les enseignants peuvent supprimer tous les messages qu'ils souhaitent et, pour les étudiants, ils ne peuvent supprimer que les messages postés par eux-mêmes.

## Le journal de logs

Dans la 1ère partie du projet, nous avons défini une table SQL de logs qui utilisait un trigger pour la compléter.

Dans cette seconde partie, nous avons redéfini les logs que nous voulions générer pour les activités des utilisateurs et utilisons mongoDB. Ainsi les logs ici ne sont pas générés avec la base de données mais sont lancés dans le frontend en fonction des actions des utilisateurs.

De plus, dans la 1ère partie, nous n'utilisions pas les logs dans l'application web, l'utilisateur n'y avait pas accès. Cette fois-ci, uniquement les professeurs ont accès aux logs pour voir les activités des étudiants présents dans leurs cours. Ils y ont accès en passant par la liste des utilisateurs inscrits dans le cours, en cliquant sur un utilisateur.

Pour chaque étudiant présent dans un cours, l'enseignant peut consulter :

- Les informations générales de l'élève
  - o Nom, prénom, ...
- Les activités générales de l'élève sur la plateforme
  - o Date et heure de la dernière connexion
  - o Date et heure de la dernière déconnexion

(Ces informations sont liées à l'ensemble de la plateforme, et non à un cours dans lesquelles on se situe)

- Les activités liées au cours dans le quelle les logs sont consultés

- Date de la dernière consultation de ce cours
- Nombre total de consultations du cours
- Nombre de messages postés dans les forums de ce cours
- Progression dans le cours
- Le journal d'activité détaillé dans ce cours
  - Liste chronologique des actions réalisées par l'étudiant dans ce cours (consultation, publication d'un message, création d'un forum, validation d'un post, ...)

De plus, grâce au système de logs, nous avons mis en place un suivi de progression permettant à l'étudiant de marquer comme lus ou achevés les posts publiés par les enseignants. Ainsi, depuis le tableau de bord de ses cours, l'étudiant peut visualiser le pourcentage de posts qu'il a complétés dans chaque cours.

## Authentification

La partie « login » de l'application permet aux utilisateurs de se connecter de façon sécurisée grâce à un formulaire Angular en Reactive Forms qui vérifie localement que l'email et le mot de passe sont valides avant envoi. Pour l'authentification, l'application utilise Firebase Authentication qui gère le stockage sécurisé des mots de passe et la vérification des identifiants, renvoyant un UID unique en cas de succès.

Ce UID permet ensuite de récupérer depuis notre propre API Express les informations et rôles associés à l'utilisateur dans notre base SQL. Firebase est particulièrement pratique : il simplifie la mise en œuvre en externalisant la partie la plus sensible (gestion des mots de passe), tout en restant facile à intégrer avec Angular et nos services backend.

L'implémentation actuelle offre ainsi un formulaire réactif et ergonomique, la gestion des erreurs côté client et serveur (messages « email requis », « format invalide », ou « identifiants incorrects »), ainsi qu'une passerelle sécurisée entre Firebase et notre système de gestion des comptes.

## Les autres fonctionnalités

Les autres fonctionnalités étaient déjà, pour la plupart, présentes dans la première version du projet. Voici une énumération non exhaustive des pages que nous avons implémentées :

- Page d'administration (catalogue et gestion utilisateurs et UE)



- Page de choix des UE (accessible après login, affichage **carte** ou **liste**, affichage activité)
- Page de contenu UE (posts visibles)
- Page de création/modification de posts (pour les profs)
- Page des inscrits à une UE (consultation et filtrables)
- Page de soumission de devoir (pour les élèves)
- Page de visualisation des devoirs rendus (pour les profs)
- Page des visualisations de forums d'un cours
- Page d'échange de message dans un forums
- Page de détails des logs des participant (pour les profs (accessible en cliquant sur un participant dans la liste de participants))
- Page de modification de l'image de l'ue (pour les profs)

Une liste encore plus détaillée est à retrouver dans le README du projet.

## Les choix de modélisation de la base

Dans le cadre de ce projet, nous avons donc choisi une architecture à 3 bases de données distinctes. Cela nous permet de combiner les avantages de chacune, les avantages des modèles relationnels (PgSQL), des bases orientées documents (MongoDB), et d'un service cloud spécialisé dans l'authentification (Firebase). Ce choix nous garantit une flexibilité, une évolutivité et une robustesse des données.

Nous utilisons PostgreSQL comme base principale pour stocker les informations dans un modèle relationnel. Cette base est héritée de la première partie et nous l'utilisons toujours pour stocker les mêmes tables et pour les mêmes fonctionnalités que la 1ère version.

MongoDB, elle, est utilisée pour les données non structurées ou semi-structurées et nous permet de garantir une souplesse et une évolutivité des schémas. Ce qui est particulièrement utile dans les nouvelles fonctionnalités ajoutées dans cette seconde version (la gestion des images et PDF, les devoirs, les forums, et un journal de logs).

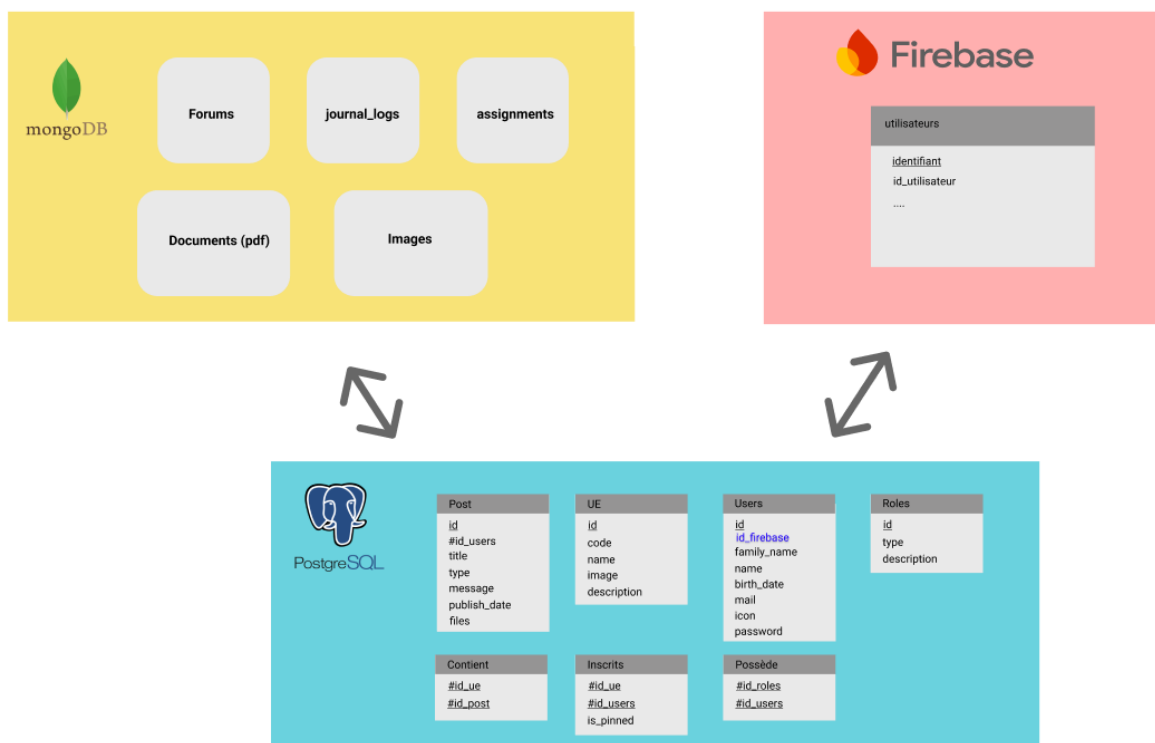
Enfin, nous utilisons Firebase uniquement pour la gestion de l'authentification des utilisateurs. Cette solution cloud propose un système d'authentification robuste, sécurisé et rapide à mettre en place.

Les 3 bases ont des liens entre elles puisque certains identifiants sont échangés et synchronisés entre les bases.

Par exemple, entre Firebase et PostgreSQL, lors de la création d'un utilisateur via Firebase, un id unique (identifiant) est généré. Cet id est systématiquement stocké dans la table users de PostgreSQL, dans un champ dédié (id\_firebase). Cela permet de faire le lien entre l'id utilisé pour l'authentification (Firebase) et les données stockées dans le modèle relationnel (nom, prénom, email, rôle, ...).

De même, il existe de nombreuses références croisées entre MongoDB et PostgreSQL. Par exemple, l'id d'un document image ou PDF stocké dans MongoDB est également enregistré dans un champ de la base PostgreSQL. Ainsi, pour les images associées à un cours, l'id de l'image (MongoDB) est référencé dans le champ image dans la table UE. De la même manière, pour les forums dans MongoDB, on stocke la référence à l'id d'un utilisateur ou l'id d'une UE PostgreSQL.

Voici un schéma expliquant quelle base de données gère quoi et les liens entre elles :



Ce modèle hybride, associant plusieurs bases spécialisées et synchronisées, nous permet de combiner les points forts de chaque solution pour répondre efficacement aux besoins du projet. Et ainsi nous offre de la scalabilité, de la performance, de l'évolutivité et de la sécurité.

## Gestions des documents (images et PDF)

Pour la gestion des documents (images et PDF), nous avons choisi d'utiliser GridFS, le système de gestion de fichiers proposé par MongoDB pour stocker des fichiers volumineux. Nous utilisons une collection dédiée pour les images et une autre pour les fichiers PDF. Cela nous permet d'avoir une séparation logique des documents pour simplifier la gestion et la recherche ou filtrage des documents.

Voici la structure des documents images et PDF enregistré :

```
Images.files {  
  _id: ObjectId,  
  length: Number,  
  chunkSize: Number,  
  uploadDate: Date,  
  filename: String,  
  contentType: String  
}
```

```
documentsPdf.files {  
  _id: ObjectId,  
  length: Number,  
  chunkSize: Number,  
  uploadDate: Date,  
  filename: String,  
  contentType: String  
}
```

Les deux collections possèdent les mêmes champs, le champ `length` permet de récupérer la taille du document, le champ `upload date` permet de récupérer la date d'enregistrement du document, `filename` pour le nom du document, et `contentType` pour le type de contenu par exemple pour une image png on aura "image/png" et pour un pdf on aura "application/pdf". Le champs `chunkSize` permet de à GridFS de gérer les chunks. Puisque GridFS divise automatiquement les fichiers volumineux en plusieurs parties appelées "chunks", qui sont stockées dans des collections dédiées (ici `images.chunks` et `documentsPdf.chunks`), afin de permettre le stockage et la récupération efficace même de fichiers dépassant la taille limite d'un document MongoDB.

Voici des exemples de documents :

```
{
  _id: ObjectId("6846d4a6ad53a47a7862c22b"),
  length: 3713,
  chunkSize: 261120,
  uploadDate: ISODate("2025-06-09T12:33:42.599Z"),
  filename: "plage.jpg",
  contentType: "image/jpeg"
}
```

```
{
  _id: ObjectId("68555063f5ffc06b696a5cee"),
  length: 107926,
  chunkSize: 261120,
  uploadDate: ISODate("2025-06-20T12:13:23.970Z"),
  filename: "Projet SI40_WE4B.pdf",
  contentType: "application/pdf"
}
```

Le choix de modélisation de la base de données de séparer les images et les PDF dans deux collections différentes nous permet d'optimiser les traitements et la récupération des documents mais également de laisser place à de futures évolutions comme par exemple ajouté des champs particuliers pour les PDF uniquement.

## Les devoirs

Concernant la gestion des devoirs, nous avons créé une collection assignments dans laquelle chaque document correspond à un devoir.

La structure d'un document assignment est la suivante :

```
Assignment {
  coursId: Number,
  title: String,
  authorId: Number,
  publishDate: Date,
  deadline: Date,
  type: String,
  sort_order: Number,
  messages: String,
  submit: [
    {
      userId: Number,
      fileId: Number,
      grade: Number,
      comment: String,
      state: String
    },
    ...
  ]
}
```

Chaque assignment possède son propre id (pas visible sur l'image) pour pouvoir l'identifier de façon unique. Il est également composé d'un champ coursId pour l'associer au cours dans lequel il est présent, authorId qui correspond à l'id de la personne ayant créé le devoir. Title, publishDate, deadline, type et messages sont des champs qui servent à décrire le devoir, on retrouve donc son titre, sa date de publication et de rendue, le type de message (information, annonce, important) ainsi qu'une description.

Il y a également un champ submit qui forme un sous document, il s'agit des travaux que les élèves rendront. Il est composé de userId pour identifier l'élève, fileId pour récupérer son fichier, grade pour lui attribuer une note ainsi que comment pour ajouter un commentaire et state pour afficher l'état du rendu.

Les champs sort\_order et state ne sont pas utilisés car il s'agit de fonctionnalité que nous aurions aimé implémenter mais qui n'ont jamais vu le jour. L'objectif à travers sort\_order était de réorganiser les différents postes au bon vouloir du prof. Pour state, il s'agit d'afficher si l'élève rend son travail en retard ou à l'heure.

## Les forums

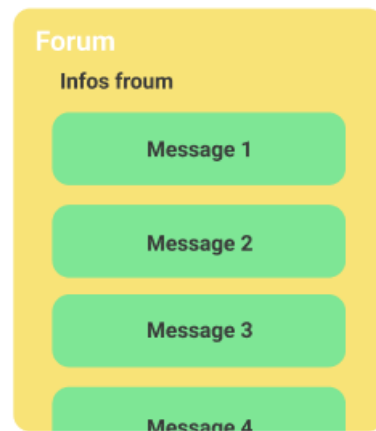
Pour la gestion des forums, nous avons choisi de créer une collection forums dans laquelle chaque document représente un forum lié à un cours spécifique.

La structure d'un document forum est la suivante :

Schéma du document :

```
Forum {  
  coursId: Number,  
  title: String,  
  authorId: Number,  
  createdAt: Date,  
  messages: [  
    {  
      content: String,  
      authorId: Number,  
      createdAt: Date  
    },  
    ...  
  ]  
}
```

Schéma graphique du document :



Dans chaque documents forums, on retrouve un champ title, le titre du forum. Un champ course\_id qui fait référence à l'identifiant du cours (stocker dans pgSQL) auquel est rattaché le forum. Ce qui nous permet de lier facilement chaque forum à son cours respectif. Chaque forum possède également un champ authorId qui identifie l'utilisateur ayant créé le forum et la date de création du forum est conservée dans le champ createdAt. De plus les forums possèdent une liste des messages qui regroupe l'ensemble des messages échangés dans ce forum.

Chaque message forme un sous-document contenant un champ content, le contenu textuel du message, un champ createdAt pour stocker la date de soumission du message et un champ authorId qui fait référence à l'id d'un user dans la base pgSQL afin d'identifier l'auteur du message.

Nous avons donc fait le choix ici de l'embedding, pour deux raisons. Premièrement, car les messages et les forums sont fortement liés : l'un va avec l'autre et chaque message est unique à un forum. De plus, cela limite le nombre de requêtes nécessaires pour récupérer l'intégralité d'une discussion

Voici un exemple de documents :

```

{
  "_id": "684a7a12f13468bb564b4a99",
  "coursId": 3,
  "title": "Problème avec l'exercice 5",
  "createdAt": "2025-06-20T08:15:27.000+00:00",
  "messages": [
    {
      "_id": "684a7a18f13468bb564b4a9a",
      "content": "Bonjour, je n'arrive pas à comprendre la consigne de l'exercice 5. Est-ce que quelqu'un peut m'aider ?",
      "createdAt": "2025-06-20T08:16:05.000+00:00",
      "authorId": 12
    },
    {
      "_id": "684a7a25f13468bb564b4a9b",
      "content": "La consigne demande de démontrer la propriété vue en cours. Je peux t'envoyer mon brouillon si tu veux.",
      "createdAt": "2025-06-20T08:17:10.000+00:00",
      "authorId": 7
    }
  ],
  "__v": 2
}
]
}

```

Les choix de ce modèle nous permettent de répondre aux besoins de fonctionnement des forums. Il nous permet d'afficher efficacement les messages et de lier les messages et les forums avec des utilisateurs et des cours présents dans notre base de données pgSQL.

## Le journal de logs

Pour la gestion des historiques de connexion et d'activités des utilisateurs, nous avons choisi de créer une collection userLogs dans laquelle chaque document représente l'historique d'activité d'un utilisateur sur la plateforme. Ainsi chaque utilisateur de la plateforme possède son document de logs.

La structure d'un document forum est la suivante :

```
journal_logs {
  userId: Number,
  lastLogin: Date,
  lastLogout: Date,
  loginCount: Number,
  courses: [
    {
      courseId: Number,
      lastViewed: Date,
      viewsCount: Number,
      progressCount: Number,
      forumMsgCount: Number,
      checkedPosts: [Number],
      activity: [
        {
          type: String,
          date: Date,
          forumId: ObjectId,
          messageId: ObjectId,
          postId: Number
        },
        ...
      ]
    },
    ...
  ],
  ...
}
```

Schéma graphique du document :



Dans chaque document `journal_logs`, on retrouve un champ `userId`, qui fait référence à l'identifiant d'un utilisateur stocké dans `pgSQL`. On y stocke également la date de la dernière connexion (`lastLogin`), la date de la dernière déconnexion (`lastLogout`) ainsi qu'un compteur du nombre de connexions total à la plateforme (`loginCount`).

La structure possède également une liste de sous documents `courses`. Cette liste contient un document pour chaque cours que l'utilisateur a visité au moins une fois. Par défaut, cette liste est vide, puis elle s'enrichit automatiquement à chaque nouvelle visite de cours par l'utilisateur (ou met à jour les informations si le cours a déjà été visité). Chaque objet `course` contient les champs suivants : `courseId` (l'identifiant du cours (référence au id dans la base `pgSQL`)), `lastViewed` (la date de dernière visite du cours), `viewsCount` (le nombre total de visites de ce cours par l'utilisateur), `progressCount` (un compteur de progression des posts marquer comme terminer), `forumMsgCount` (le nombre de messages postés par l'utilisateur dans les forum de ce cours), `checkedPosts` (une liste des id des posts que l'utilisateur à marquer comme terminer (référence aux id des posts dans la base `pgSQL`). Enfin, chaque sous documents possède lui-même une liste de sous documents `activity`.



Les sous documents de la liste activity nous permet de décrire précisément chaque action effectuée par l'utilisateur et à quel moment dans ce cours. Ils possèdent chaque un type qui définit l'action réalisé. Dans notre application on a pour l'instant définit les types suivants : view (qui indique quand l'utilisateur a consulté le cours), forum-message (indique quand un utilisateur a posté un message), forum-message-delete (indique quand l'utilisateur a supprimé le message), create-forum (indique quand un utilisateur a créé un post), check-post (indique quand un utilisateur a validé un post). Pour chaque document on enregistre également la date ou l'action a été réaliser dans le champ date. De plus des paramètres optionnels sont présent pour chaque document, par exemple on peut retrouver les forumId, messageId et postId. Cela nous permet de préciser sur quelle message, forum ou post l'action a été réalisé.

Dans cette modélisation, nous avons fait le choix d'utiliser à la fois du embedding et du linking.

Nous avons choisi d'imbriquer les sous-documents courses et activity directement dans le document principal journal\_logs. Ce choix s'explique par le fait que les logs de cours et d'activité sont fortement liées au log principal de l'utilisateur. De plus cela permet de visualiser rapidement tous les logs d'un utilisateur directement en faisant un requête dans la base de données MongoDB sans avoir besoin de faire des jointures avec d'autres collections.

En revanche, dans les sous-documents activity, nous ne stockons que les id des entités externes (linking ) telles que les forums (forumId), les messages (messageId ) ou les posts (postId ) au lieu de dupliquer leur contenu dans ces documents. Nous avons fait ce choix afin de ne pas surcharger inutilement les documents de logs. En effet, dans l'état actuel de l'application, nous n'exploitons pas réellement ces identifiants, pour chaque activité on indique seulement le type d'action et la date. Les id sont donc là pour une évolution futur ou permettre a un administrateur de la base de données de retrouver les message, forums, ou post liée au cas où. Toutefois, si nous souhaitions à l'avenir affiché un historique plus précis des actions de l'utilisateur (par exemple, permettre de visualiser le contenu des messages supprimés), il serait alors plus pertinent d'envisager une approche par embedding.

Voici un exemple de documents :

```
{
  "_id": "6855ba0726a95e82f89b5c2b",
  "userId": 40,
  "loginCount": 0,
  "courses": [
    {
      "courseId": 10,
      "lastViewed": "2025-06-22T21:09:15.434Z",
      "viewsCount": 98,
      "progressCount": 4,
      "forumMsgCount": 0,
      "checkedPosts": [56, 55, 53, 54],
      "activity": [
        {
          "type": "view",
          "date": "2025-06-20T19:44:10.013Z",
          "_id": "6855ba0a26a95e82f89b5c3e"
        },
        {
          "type": "view",
          "date": "2025-06-20T19:44:14.448Z",
          "_id": "6855ba0e26a95e82f89b5c7f"
        },
        {
          "type": "check-post",
          "date": "2025-06-20T19:44:16.959Z",
          "postId": 56,
          "_id": "6855ba1026a95e82f89b5c95"
        }
        // ...
      ],
      "_id": "6855ba0a26a95e82f89b5c3d"
    },
    {
      "courseId": 9,
      "lastViewed": "2025-06-22T21:10:03.390Z",
      "viewsCount": 9,
      "progressCount": 2,
      "forumMsgCount": 0,
      "checkedPosts": [57, 58],
      "activity": [
        {
          "type": "view",
          "date": "2025-06-20T19:46:00.410Z",
          "_id": "6855ba7826a95e82f89b5fbd"
        },
        {
          "type": "check-post",
          "date": "2025-06-20T19:46:01.382Z",
          "postId": 57,
          "_id": "6855ba7926a95e82f89b5fd4"
        }
        // ...
      ],
      "_id": "6855ba7826a95e82f89b5fbc"
    }
    // ...
  ],
  "_v": 299
}
```

Ainsi cette structure nous permet de suivre précisément l'activité de chaque utilisateur sur la plateforme, en regroupant toutes ses actions et statistiques dans un seul et même document. De plus, on garde une structure évolutive, où il est facile d'enrichir l'historique des actions si nécessaire (par exemple, en passant à du embedding pour certaines entités s'il y a un besoin d'audit plus détaillé). Enfin, la définition du tableau checkedPosts et le compteur progressCount nous permet de réaliser un affichage dynamique de la progression des utilisateurs dans les cours.

# L'API REST

Une API REST permet de faire des requêtes HTTP (GET, POST, etc.) pour visualiser, ajouter, supprimer ou modifier des données dans une base de données. Dans notre projet, nous utilisons une API REST dans le back-end pour faire la liaison entre les différentes bases de données et le front-end. Pour cela, nous avons choisi Node.js, qui offre de bonnes performances pour gérer de nombreuses requêtes simultanées.

## PostgreSQL

Pour se connecter à la base de données PostgreSQL, on utilise la librairie pg et on configure un pool de connexion avec les informations d'accès (utilisateur, mot de passe, base, port).

Voici un exemple de pool que nous utilisons :

```
const { Pool } = require('pg');

const pool = new Pool({
  user: 'admin', // nom d'utilisateur postgresql
  host: 'localhost',
  database: 'moodle_tr', // nom de la base pgadmin
  password: 'admin', // mdp de l'utilisateur postgresql
  port: 5432 // port par défaut de postgresql
});
```

Pour exécuter une requête REST, on définit une route express qui, lors d'un appel HTTP (GET, POST, PATCH ...), utilise le pool pour envoyer une requête SQL à la base, puis retourne la réponse au client

Par exemple la route suivante :

```
router.get('/users', async (req, res) => {
  const { rows } = await pool.query('SELECT * FROM users');
  res.json(rows);
});
```

Cette route grâce à la requête SQL, `SELECT * FROM users`, va récupérer tous les utilisateurs présents dans la base PostgreSQL.

## MongoDB

Pour se connecter à MongoDB, on utilise la librairie `mongoose` et on lance la connexion vers l'URL du serveur MongoDB.

Voici un exemple de connexion :

```
const mongoose = require('mongoose');
await mongoose.connect('mongodb://localhost:27017/projet_moodle');
```

Pour faire une requête REST, on définit une route Express qui, lors d'un appel HTTP (GET, POST, PATCH ...), utilise les modèles Mongoose pour interroger ou modifier les documents de la base de données, puis envoie la réponse au client.

Par exemple la route suivante :

```
router.get('/cours/:coursId', async (req, res) => {
  const coursId = Number(req.params.coursId);
  const forums = await Forum.find({ coursId }).sort({ createdAt: -1 });
  res.json(forums);
});
```

Cette route GET, récupère tous les forums d'un cours donné grâce à son id et les affiche de façon décroissant sur la date.

## Firebase

Pour l'authentification, on utilise Firebase Authentication via la librairie officielle. La connexion se fait simplement en appelant `signInWithEmailAndPassword` avec l'email et le mot de passe. Firebase vérifie les identifiants et renvoie un UID unique. Cette approche évite de stocker soi-même les mots de passe, renforce la sécurité et simplifie le code backend, tout en restant facile à intégrer dans notre API Express pour récupérer les informations complémentaires de l'utilisateur.

## Le serveur

Enfin, on définit un serveur Express qui centralise toutes les routes REST. Il fait donc le lien entre le frontend et les bases de données, en recevant les requêtes http, en interrogeant ou modifiant les données selon les routes, puis en renvoyant les réponses au client. Voici comment est lancé le serveur :

```
app.listen(3000, () => {  
  console.log('API démarrée sur http://localhost:3000');  
});
```

## Les captures

Dans cette section, vous pourrez retrouver les captures d'écran des fonctionnalités implémentées avec MongoDB.

## Les devoirs

Voici un devoir visualisé par un étudiant :

### Exercice : Droits Linux

Sur un système Linux, affiche les permissions d'un fichier avec ls -l. Explique les droits utilisateur, groupe et autres.

Ajouter un travail

28/06/2025 18:09

### Exercice : Droits Linux

Ouvert le : 28/06/2025 18:09

A rendre avant le : 06/07/2025 18:11

#### Travaux rendus

Statut des travaux remis	Devoir rendu
Statut de l'évaluation	Non évalué
Temps restant	Travail remis
Commentaires	Pas de commentaire

Choisir un fichier    Auc...hoisi

Valider

Et voici un devoir visualisé par un professeur :

### Exercice : Droits Linux

Sur un système Linux, affiche les permissions d'un fichier avec ls -l. Explique les droits utilisateur, groupe et autres.

Rendu

28/06/2025 18:09

## Exercice : Droits Linux



Ouvert le : 28/06/2025 18:09  
A rendre avant le : 06/07/2025 18:11

Nom	Prenom	Note	Commentaire	Fichier
Richard	Chloe	<input type="text" value="10"/>	<input type="text" value="tres bien"/>	<a href="#">Télécharger le PDF</a>
Robert	Nathan	<input type="text" value="7"/>	<input type="text" value="pas mal"/>	<a href="#">Télécharger le PDF</a>
Dubois	Enzo	<input type="text" value="7"/>	<input type="text" value="super"/>	<a href="#">Télécharger le PDF</a>
Girard	Louise	<input type="text" value="8"/>	<input type="text" value="ok"/>	<a href="#">Télécharger le PDF</a>
exemple	etudiant	<input type="text"/>	<input type="text"/>	<a href="#">Télécharger le PDF</a>

Valider toutes les modifications

## Les forums

Voici la page de forums (ici pour un professeur car il peut supprimer les forums) :

Titre	Auteur	Dernier message	Nombre de messages	Supprimer
Programmation avancée	Thomas Gabriel 28/06/2025	Je conseille de commencer par la POO car elle est souvent enseignée en premier et utilisée dans beaucoup d'applications. Ensuite, découvrir la programmation fonctionnelle permettra d'élargir son horizon et écrire du code plus robuste. 28/06/2025 18:57	9	
Technologie	exemple prof 28/06/2025	Je ne suis pas d'accord. La programmation fonctionnelle (PF) offre une meilleure gestion des effets de bord grâce à l'immuabilité des données. Cela rend le code plus prévisible et facile à tester. 28/06/2025 18:55	7	
<input type="text" value="Titre du forum"/>		<div>Ajouter un forum</div>		

Et le contenu d'un forum (en tant qu'étudiant puisqu'il ne peut que supprimer ses message) :

**Sujet : algorithme tris bulle**

Retour

par 'Inconnu', 28/06/2025

**Robert Nathan**, 28/06/2025 18:20 :  
quelqu'un connait cette algo ?

**Morel Ethan**, 28/06/2025 18:29 :  
oui

**exemple etudiant**, 28/06/2025 21:38 :  
Moi, je ne connais pas. Quelqu'un peut m'expliquer ?

Votre message...

Envoyer

## Le journal de logs :

Voici un exemple des logs accessible en cliquant sur un participant dans la liste des participants d'un cours ( en tant que professeur) (affichage pas exacte car plusieurs screen) :

### Details Participants

[Retour](#)

#### Infos générales :

Prénom : **Morel**  
Nom : **Ethan**  
Mail : **ethan.morel@example.com**  
Role : **etudiant**

#### Activité générale :

Dernière connexion : **samedi 28/06/2025 à 18h29 (il y a 3h 11min 15s)**  
Dernière déconnexion : **samedi 28/06/2025 à 18h30 (il y a 3h 10min 11s)**  
Nombre de login : **4**

#### Activité dans ce cours :

Dernière consultation : **samedi 28/06/2025 à 18h30 (il y a 3h 10min 20s)**  
Nombre de consultation du cours : **2**  
Nombre de messages dans les forums : **1**  
Progression : **2**

## Journal d'activité dans ce cours :

- samedi 28/06/2025 à 18h30 (il y a 3h 10min 18s) - **a validé un post**
- samedi 28/06/2025 à 18h30 (il y a 3h 10min 20s) - **a consulté ce cours**
- samedi 28/06/2025 à 18h30 (il y a 3h 10min 22s) - **a posté un message**

Gestion de la progression ( ici un post validé et un autre non ) :

### VPN

Un VPN permet de créer un tunnel sécurisé entre deux points via Internet. Il chiffre le trafic et assure la confidentialité des données.

28/06/2025 18:08

Deja validé

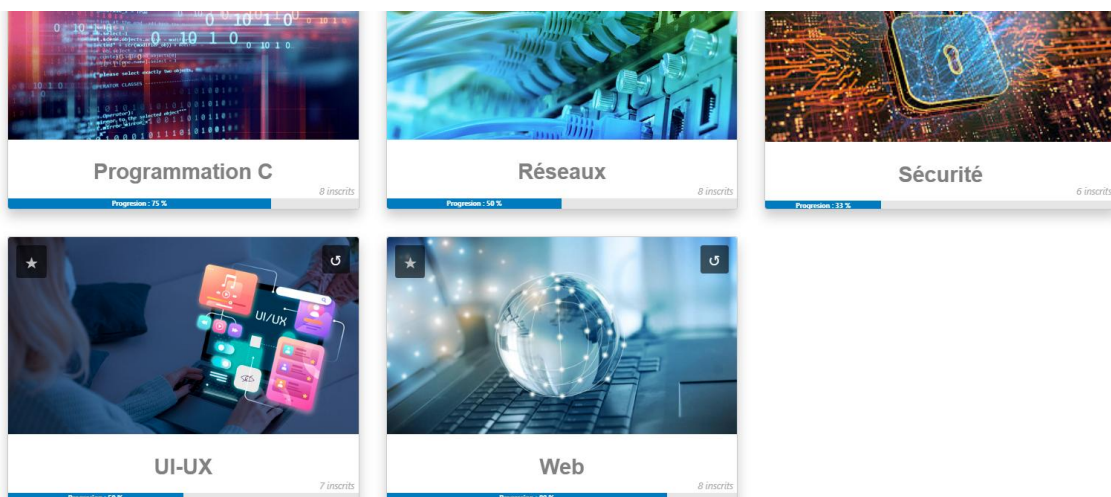
### Processus

Un processus est un programme en cours d'exécution. Il possède un espace mémoire propre et peut créer des sous-processus (fils).

28/06/2025 18:09

Marquer comme validé

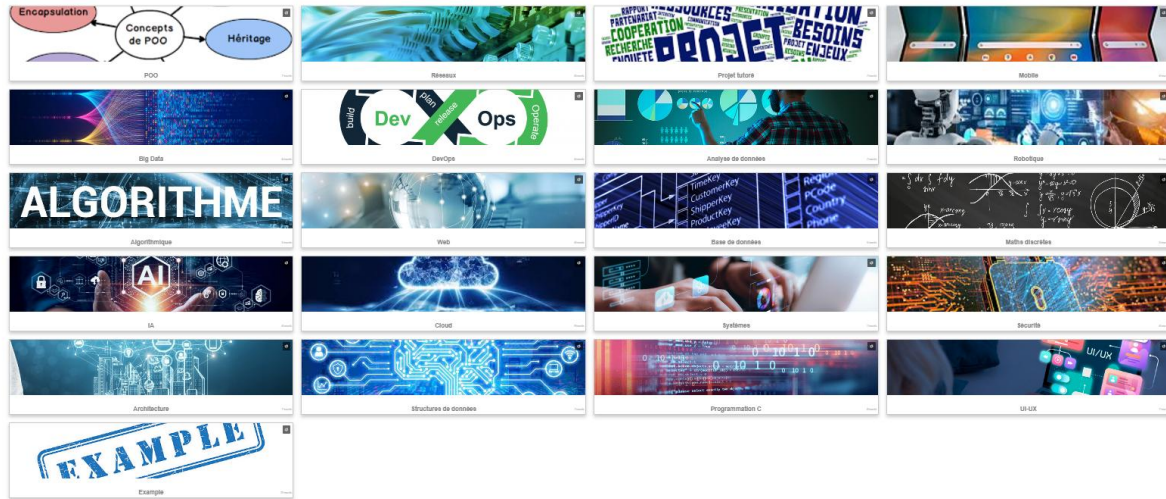
Affichage de la progression dans le tableau de bord des UE :



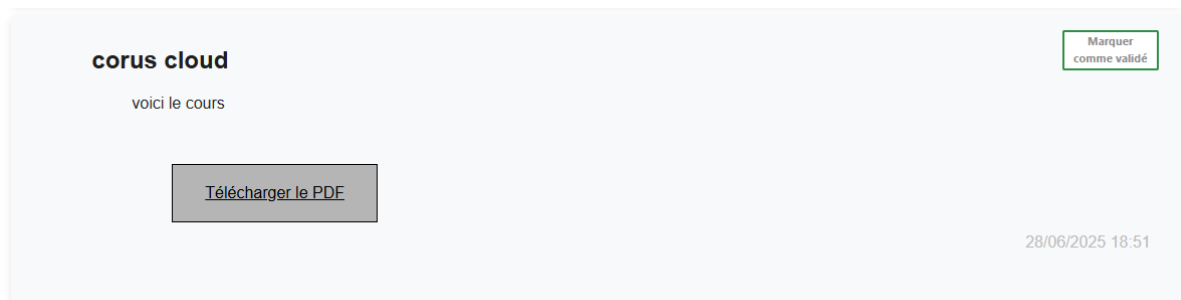


## Gestion des images et PDF :

Images :



PDF :



## MongoDB Analytics Implementation

In addition to the requested topics, we have created queries and aggregations in order to extract data from the MongoDB database.

The queries and additional explanations can be found in the mongo\_analytics folder.

Our MongoDB implementation enables powerful learning analytics through three key approaches:

1. Student Engagement Tracking
2. Forum Activity Analysis
3. Assignment Performance

These are the key benefits for Educators:

Early Intervention : Identify struggling students through activity patterns

Content Optimization: See which materials generate most engagement

Time Management: Understand when students are most active

Progress Tracking: Visualize class-wide completion rates

Implementation Notes:

- Used MongoDB's document structure for natural data nesting
- Created indexes on frequently queried fields (userId, courseId)
- Scheduled weekly aggregation jobs during low-traffic periods
- Integrated results with Angular dashboard components

This implementation provides teachers with actionable insights while maintaining system performance, demonstrating MongoDB's effectiveness for educational analytics.

We made multiple queries and mapreduce functions to achieve this. (documents listed in mongodb folder)

# Conclusion

Cette seconde partie du projet nous a permis de mettre en pratique les notions sur les bases de données NoSQL vues en cours et en travaux pratiques de SI40, plus particulièrement la modélisation et la gestion de bases de données MongoDB.

Cela nous a également permis de découvrir de nouvelles méthodologies sur les bases de données, comme le stockage de documents avec MongoDB via GridFS, mais aussi de nouvelles technologies telles que Firebase.

De plus, l'un des aspects les plus enrichissants de ce projet a été la gestion et l'intégration de bases de données dans un contexte hybride mêlant SQL, NoSQL et Firebase. Cela nous a permis de réfléchir à une architecture de bases de données qui communiquent entre elles.

La structuration du projet en deux parties, l'une avec uniquement du SQL puis une seconde liant SQL et NoSQL, nous a permis de comprendre les avantages et les inconvénients de chaque approche.

Nous avons également pu écrire des requêtes pour interroger et agréger des données de MongoDB afin d'obtenir des informations et étudier les données.

Enfin, nous avons pu découvrir le fonctionnement des API REST, notamment en l'occurrence ici une API Node.js. Cela nous a permis de comprendre plus précisément, en pratiquant, comment fonctionne un projet frontend/backend.

En conclusion, ce projet a été très enrichissant, car il a permis de mettre en pratique les notions vues en cours tout en découvrant de nouveaux concepts, et a également favorisé le développement de notre communication et de notre travail d'équipe.