

# Eel: An E-learning Platform

Taha EL Khiraoui

5 janvier 2023

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Analyse technique</b>	<b>4</b>
2.1	Backend . . . . .	4
2.1.1	NodeJS . . . . .	5
2.1.2	ExpressJS . . . . .	5
2.1.3	MongoDB . . . . .	6
2.1.4	EJS . . . . .	6
2.1.5	Multer . . . . .	6
2.1.6	JWT . . . . .	7
2.2	Frontend . . . . .	7
2.2.1	Bootstrap . . . . .	7
2.2.2	JQuery . . . . .	8
2.2.3	Katex . . . . .	8
2.2.4	Markdown . . . . .	8
2.3	Routing . . . . .	9
2.4	Divers . . . . .	9
<b>3</b>	<b>Conception de l'application</b>	<b>10</b>
3.1	Introduction . . . . .	10
3.2	UML . . . . .	10
3.2.1	Diagramme de cas d'utilisation . . . . .	10
3.2.2	Diagramme de classe . . . . .	11
3.2.3	Diagramme d'activité . . . . .	12
3.3	MVC . . . . .	14
3.3.1	Rôle du Modèle . . . . .	14
3.3.2	Rôle de la Vue . . . . .	14
3.3.3	Rôle du Contrôleur . . . . .	15
3.4	ODM . . . . .	16
3.5	Fonctionnalités . . . . .	17
3.6	Conclusion . . . . .	17

<b>4</b>	<b>Réalisation du projet</b>	<b>18</b>
4.1	Introduction . . . . .	18
4.2	Interface étudiant . . . . .	18
4.3	Interface enseignant . . . . .	19
	4.3.1 Ajouter un cours . . . . .	19
	4.3.2 Ajouter des ressources . . . . .	20
4.4	CONCLUSION . . . . .	21
<b>5</b>	<b>Conclusion et Perspective</b>	<b>22</b>
5.1	Les problèmes rencontrés . . . . .	22
5.2	Les avancements possibles . . . . .	23

# Chapitre 1

## Introduction

Devant l'apparition du **Covid 19**, la transition vers l'enseignement à distance était nécessaire pour garantir à tout étudiant la continuité d'apprentissage. À but de faciliter ce changement pour notre établissement, la tâche de créer une plate-forme personnalisée nous à été confiée.

Cet environnement apporte plusieurs avantages, notamment :

- [Suivre](#) son avancement.
- [Progresser](#) à son rythme.
- [Accéder](#) facilement au contenu.

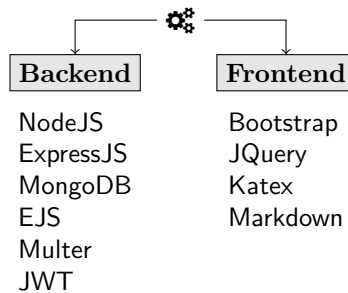
Mais à aussi des inconconvénients :

- [Longues](#) périodes devant l'écran.
- [Manque](#) d'interactions avec les professeurs.

## Chapitre 2

# Analyse technique

Pour atteindre le but dans une durée limitée, on s'est mis à chercher les différents **outils** et **langages** qui nous seront utiles, facilement adaptable à nos besoins.



### 2.1 Backend

C'est la **partie cachée** de l'application qui gouverne tout son fonctionnement, communication avec la base de données et gestion de fichiers.

/view/ : <u>id</u>	Router	viewController( <u>id</u> )
/user/ : <u>id</u>		userController( <u>id</u> )
Frontend		Backend

```
1 function viewController();
2 function userController();
3 router.get('/:courseId', viewController);
4 router.get('/:userId', userController);
```

### 2.1.1 NodeJS

NodeJS sert à exécuter Javascript côté serveur par l'intermédiaire du moteur V8. Il peut accéder au disque (lecture/écriture) et la base de données de manière non-bloquante, ce qui rend le traitement de plusieurs requêtes simultanément possible.

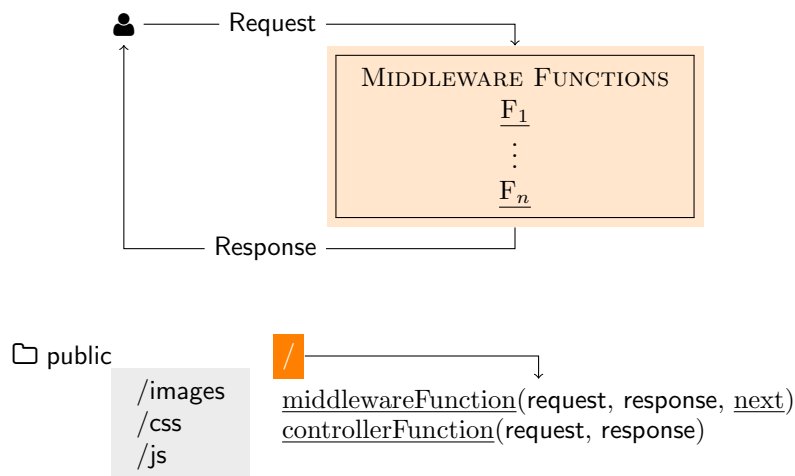
```
Thread0 fs.appendFile(file, content, callbackFunction)
Thread1 Course.findById('61c9c81fe085ee7b95a3e4f1', callbackFunction)
```

|  
t = 0

```
npm init -y
npm install %package%
npm start
```

### 2.1.2 ExpressJS

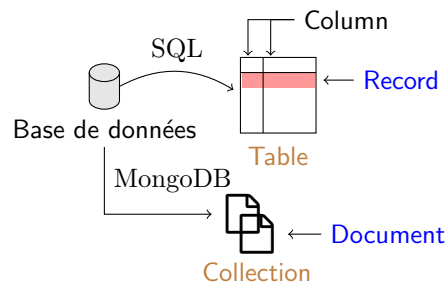
ExpressJS est un framework minimale qui facilite l'organisation d'**applications web** en **routes** qui servent des fichiers statiques ou exécutent un ensemble de fonctions.



```
1 const
2   express = require('express'),
3   path = require('path'),
4   app = express();
5 // ...
6 app.static('/', express.static(path.join(__dirname, 'public')));
7 app.get('/', middlewareFunction, controllerFunction);
8 app.listen(3000, () => console.log('Running on port 3000'));
```

### 2.1.3 MongoDB

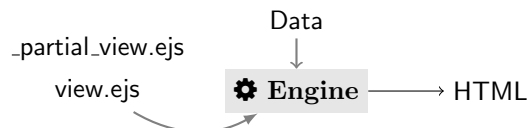
**MongoDB** est une **base de données** NoSQL orientée document, utilisée pour le stockage massive des données. Elle contient des **collections** composées d'un ensemble de **documents**, qui comportent un nombre variable de **champs**.



```
1 const mongoose = require('mongoose');
2 const uri = 'mongodb://localhost:27017/platformDB';
3 mongoose.connect(uri);
```

### 2.1.4 EJS

EJS est un **moteur de template** qui crée des vues dynamiques rapidement avec injection de variables quand un client accède à une route. La réutilisation des vues partielles est encouragée dans les vues.



```
1 app.set('view engine', 'ejs');
2 app.set('views', path.join(__dirname, 'views'));
3 app.get('/', (_, response) => response.render('home'));
```

### 2.1.5 Multer

Multer reçoit les fichiers transmis dans un répertoire spécifié au configuration, et filtre selon l'**extension** et la **taille**. Il s'intègre avec ExpressJS en créant des **intergiciels**.

```
1 const
2   multer = require('multer'),
3   timeout = require('connect-timeout');
4
5 const upload = multer({
6   dest: 'uploads/images/',
```

```

7     fileFilter: function(_, file, callback) {
8         const extension = path.extname(file.originalname);
9         const allowed = Array('.jpg', '.jpeg', '.png', '.gif');
10        return callback(null, allowed.includes(extension));
11    },
12    limits:
13    {
14        fileSize: 8 * 1024 * 1024 // Maximum: 8MB
15    }
16 });
17
18 app.post(
19     '/upload', // Handle POST requests to /upload.
20     timeout('3m'), // Leave connection open for 3 minutes.
21     upload.single('image'), // Put the image on disk.
22     uploadController // Process it.
23 );

```

### 2.1.6 JWT

JSON Web Token encode les données comme un **badge électronique** vérifiable avec une signature numérique calculée par l'algorithme HMAC. Il comporte une date d'expiration et peut être utilisé comme mécanisme d'authentification.

$$\text{DATA} \oplus \mathcal{Q} = \text{SIGNATURE}$$

Ce qui différencie cette technique est la **zone de stockage** du contenu des sessions, c'est complètement au côté client par l'usage des **cookies**.

$$\text{TOKEN} \xrightarrow{\text{expClaim, idData}} \text{SIGNATURE} \xrightarrow{\text{jwt.verify()}}$$

```

var token = jwt.sign(
    { email }, // Data
    process.env.SECRET_KEY, // Key
    { expiresIn: 3600 * 1000 } // Expiration date
);
console.log(jwt.verify(token, process.env.SECRET_KEY));

```

## 2.2 Frontend

Le contenu d'un site auquel l'utilisateur **peut accéder**, un ensemble de fichiers HTML, CSS et JS avec lesquelles il peut interagir directement.

### 2.2.1 Bootstrap

Bootstrap est un framework qui accélère le développement de pages web statiques, compatibles avec tous les navigateurs et s'adaptent à tout type d'écran.



```

1 <a href="#" class="btn btn-primary">Button</a>
2 <div class="row">
3 <div class="col-md-6">Column 0</div>
4 <div class="col-md-6">Column 1</div>
5 </div>

```

## 2.2.2 JQuery

JQuery vise réduire le **code requis** pour joindre du contenu dynamique à une page web, simplifier les API des navigateurs et offrir de nouvelles fonctionnalités utiles.

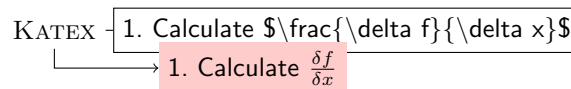
```

1 $(window).load(loadFunction);
2 $(document).on('submit', '#formId', submitFunction);

```

## 2.2.3 Katex

Katex est une bibliothèque qui d'afficher les symboles mathématiques dans une page web, caractérisée par sa rapidité et facilité d'utilisation.



```

1 const formula = document.getElementById('formulaId');
2 const params = new Object();
3 params['throwOnError'] = false;
4 params['delimiters'] = [
5   { left: '$$', right: '$$', display: true },
6   { left: '$', right: '$', display: false },
7   { left: '\\(', right: '\\)', display: false },
8   { left: '\\[', right: '\\]', display: true }
9 ];
10 formula.innerHTML = 'Formula: $e^{ix} = \\cos{x} + i\\sin{x}$';
11 renderMathInElement(formula, params);

```

## 2.2.4 Markdown

Markdown est un langage de structuration de données extensible et réutilisable, transformable en du **HTML** qui est en suite filtré par le module DOMPurify.

### Hello \$w\_{orl}^d\$  $\longrightarrow$  Hello  $w_{orl}^d$

```

1 const
2   jsdom = require('jsdom'),
3   katex = require('katex'),
4   markit = require('markdown-it'),
5   texmath = require('markdown-it-texmath'),

```

```

6     purifier = require('dompurify');
7     const
8         tm = texmath.use(katex),
9         md = markit({ html: true }).use(tm, { delimiters: 'dollars' }),
10        window = new jsdom.JSDOM('').window,
11        purify = purifier(window);
12    //
13    console.log(purify.sanitize(md.render('### Hello $w_orl^d$!')));

```


## 2.3 Routing

C'est un mécanisme d'association entre une demande d'un utilisateur et une série de fonctions à **exécuter** pour répondre à cette requête.

```

POST /user/create → createForm()
GET  /user/:userId → viewUser()

```


**Route parameter**

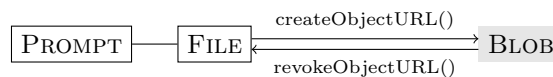
```

1     const
2         router = express.Router(),
3         userController = require('../controllers/userController');
4     router.route('/user/create')
5         .get(userController.createForm)
6         .post(userController.createUser);
7     router.get('/user/:userId', userController.viewUser);

```

## 2.4 Divers

Le flux d'une vidéo sélectionné par l'utilisateur est accessible directement de sa machine avec un **Blob** extractible à partir de l'interface **File** du navigateur.



La suggestion des mots de passe enregistrés est une fonctionnalité qui peut être **désactivée** par une page qui demande à l'utilisateur de faire entrer un **nouveau**.

\*\*\*\*\*

type="password"  
autocomplete="new-password"

## Chapitre 3

# Conception de l'application

### 3.1 Introduction

Dans ce chapitre on vise obtenir une idée générale sur le fonctionnement de notre application, décider son architecture et sa conception. Ce qui va nous permettre d'accélérer le procès de développement ainsi que d'identifier au préalable les problèmes qu'on pourra rencontrer.

### 3.2 UML

C'est une notation standardisée qui décrit précisément un système d'information sous forme de diagrammes qui expliquent son fonctionnement et repèrent tous ses entités et acteurs, il est fréquemment possible de décomposer toute la structure en un ensemble de sous-systèmes.



Diagramme de cas d'utilisation

Diagramme de classe

Diagramme d'activité

#### 3.2.1 Diagramme de cas d'utilisation

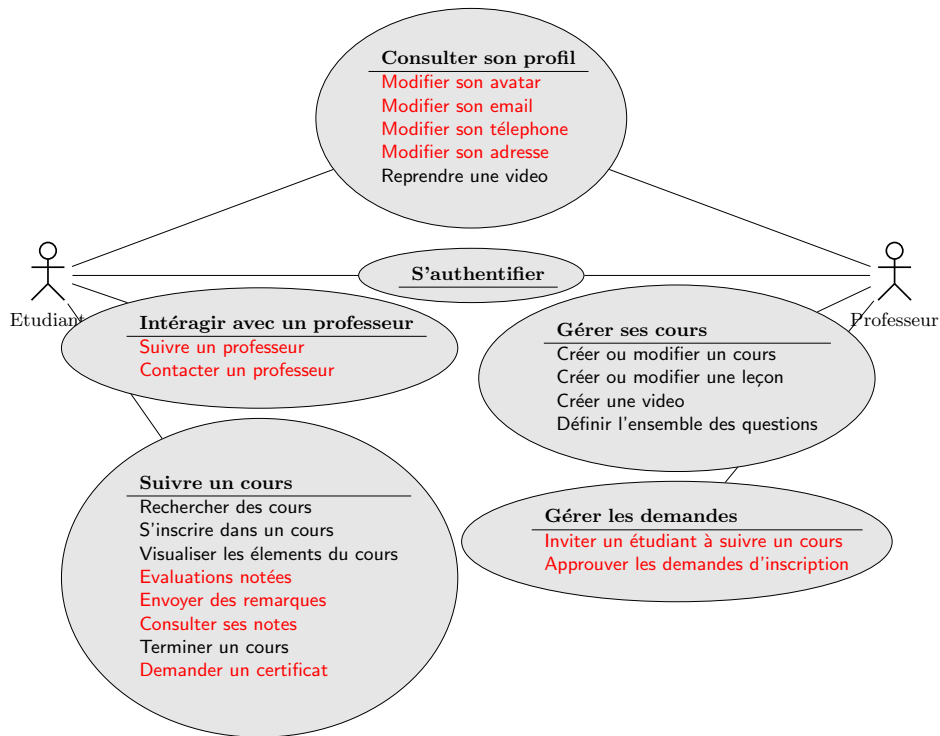
Ce diagramme sert à modéliser le comportement du système en identifiant les actions (cas d'utilisation) que les acteurs sont capable d'effectuer, et les étapes à suivre pour obtenir un résultat observable.

##### Identification des acteurs

- Visiteur
- Étudiant
- Professeur

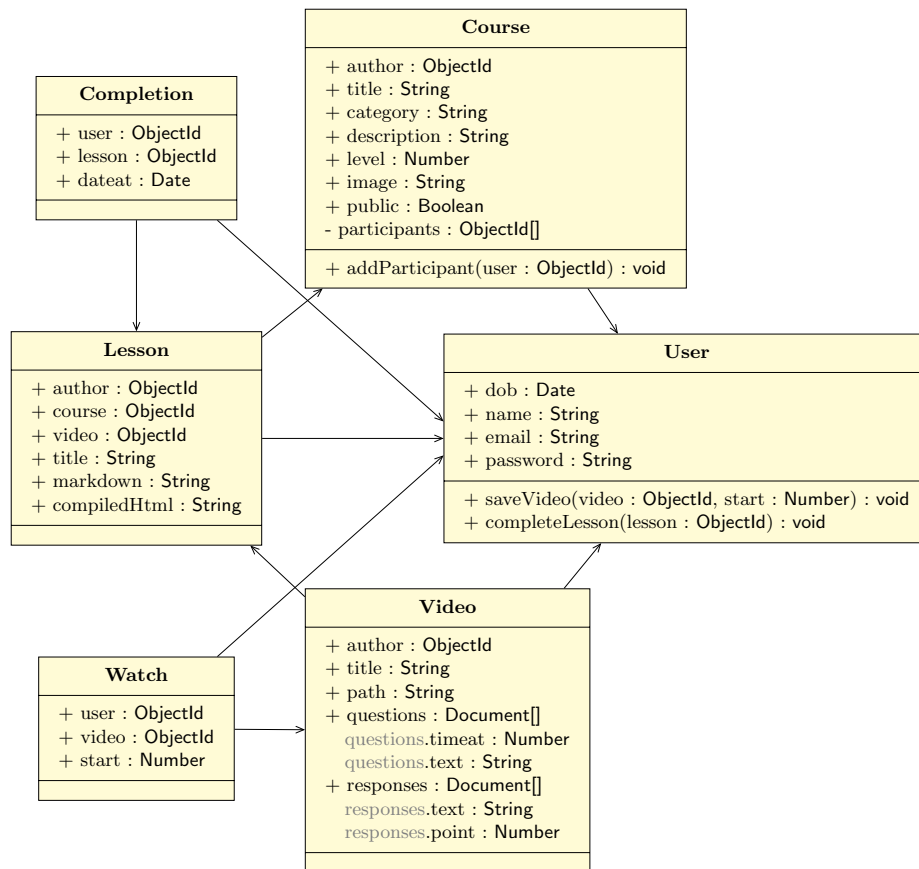
### Cas d'utilisation

C'est une fonction réalisable dans le système, directement liée par des **relations** au acteurs concernés.



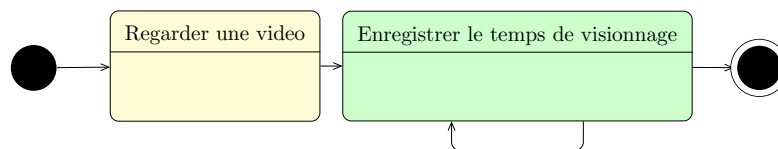
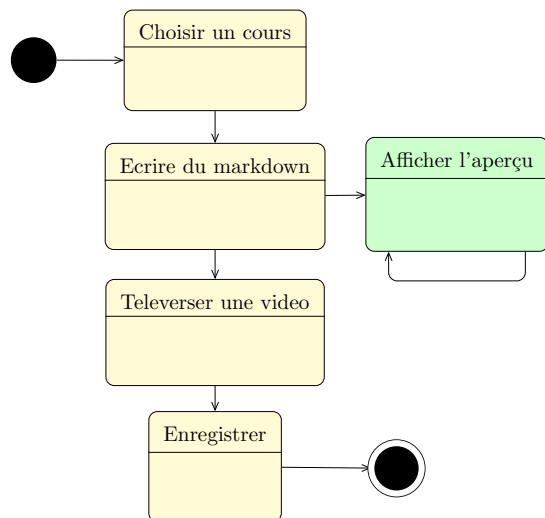
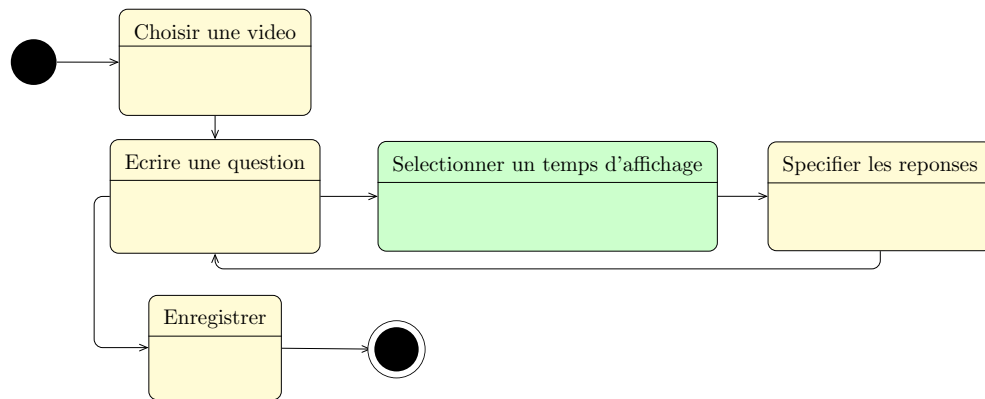
### 3.2.2 Diagramme de classe

Ce diagramme représente la totalité des **classes** du modèle ainsi que leur **propriétés** (types inclus (String, Number...)), méthodes, et les relations qui existent. Il contient l'ensemble des **entités** à prendre en considération dans la réalisation.



### 3.2.3 Diagramme d'activité

Ce diagramme décrit le **comportement du modèle**, les **étapes** à suivre pour réaliser une **action**, montre le chemin d'exécution qui les **relie**, son **départ** est marqué par un noeud initial, et sa **fin** est indiqué par un noeud final.

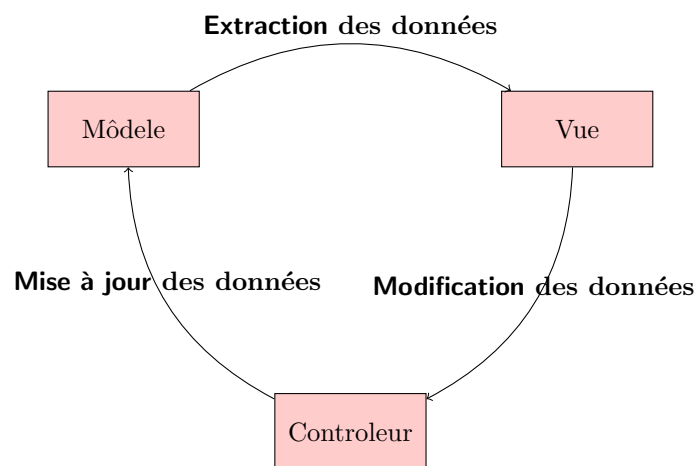


### 3.3 MVC

L'architecture MVC est une façon d'organiser les éléments d'un projet à but de faciliter sa gestion et modification. Elle consiste à distinguer ces trois couches :

- **Modèle**
- **Vue**
- **Contrôleur**

Chacun à un rôle précis et influence l'autre directement ou indirectement.



#### 3.3.1 Rôle du Modèle

Le modèle contient les données réelles retirées d'une **source quelconque** (Base de données, Fichier JSON...), l'accès et modification de ses **variables** est possible, à condition que l'intégrité des données soit préservée.

```
models
  choice.js
  completion.js
  course.js
  lesson.js
  user.js
  video.js
  watch.js
```

#### 3.3.2 Rôle de la Vue

La vue constitue l'interface visible à l'utilisateur. Elle introduit les variables du modèle à la page comme des **paramètres dynamiques**, et renvoie les **actions**

de l'utilisateur vers le contrôleur.

```

└─ views
    └─ course
        _form.ejs
        create.ejs
        edit.ejs
        view.ejs
    └─ lesson
        _form.ejs
        create.ejs
        edit.ejs
        view.ejs
    └─ user
        create.ejs
        view.ejs
    └─ video
        create.ejs
        view.ejs
    create.ejs
    footer.ejs
    header.ejs
    home.ejs
    listing.ejs
    login.ejs
    navbar.ejs
```

### 3.3.3 Rôle du Contrôleur

Le contrôleur est le coeur de l'application, il se charge de la **synchronisation** du modèle et la vue. Il reçoit tous les événements de l'utilisateur et enclenche les actions à effectuer. Si une action nécessite un changement des données, le contrôleur demande la modification des données au modèle et ensuite avertit la vue que les données ont changé pour que celle-ci se mette à jour. Certains événements de l'utilisateur ne concerne pas les données mais la vue. Dans ce cas, le contrôleur demande à la vue de se modifier.

```

└─ controllers
    courseController.js
    lessonController.js
    userController.js
    videoController.js
```



## 3.4 ODM

Une couche d'abstraction qui relie entre un modèle virtuel et une base de données orientée document. Dans notre cas, on a utilisé `mongoose` pour communiquer avec MongoDB à partir de NodeJS. De créer un schéma (**structure**) et un modèle pour chaque collection est nécessaire.

```
1 const mongoose = require('mongoose');
2 const simpleSchema = mongoose.Schema(/* ... */);
3 const Simple = mongoose.model('Simple', simpleSchema);
4 module.exports = Simple;
```

### 3.5 Fonctionnalités



### 3.6 Conclusion

Ce chapitre à expliqué l'architecture de la plate-forme qu'on à construit, en présentant :

- les diagrammes **UML**.
- la structure **MVC**.
- la technique **ODM**.

La phase suivante est la réalisation.

## Chapitre 4

# Réalisation du projet

### 4.1 Introduction

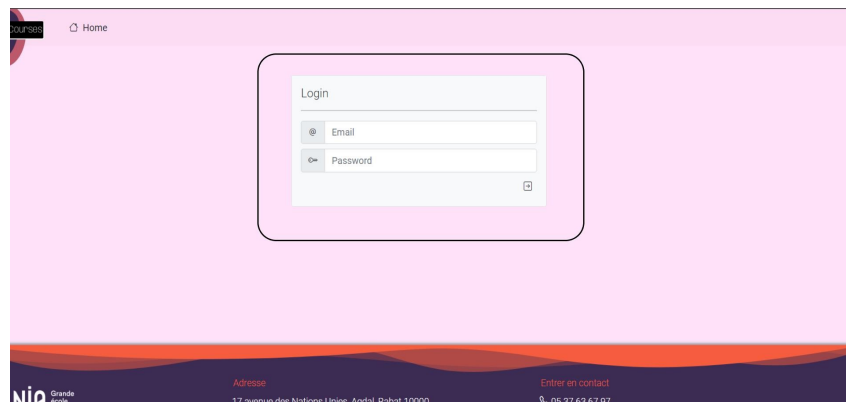
Cette partie a pour objectif majeur de présenter le produit final. C'est la phase de réalisation de cette plate-forme dynamique qui utilise des technologies spécifiques. C'est-à-dire les principales interfaces graphiques.

### 4.2 Interface étudiant

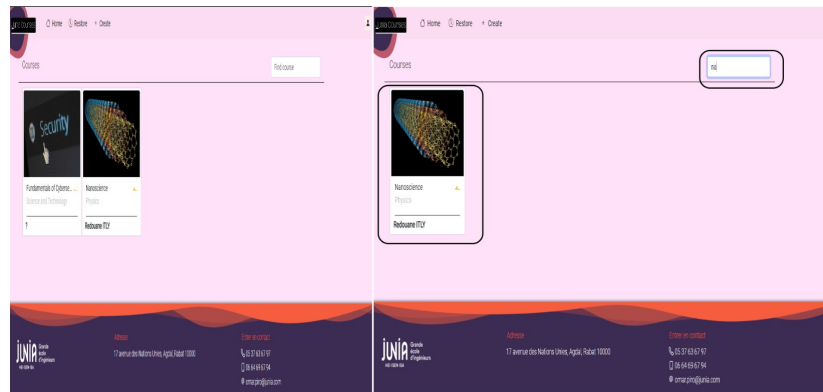
Dans les paragraphes qui suivent, nous allons exposer différentes situations que peut rencontrer l'étudiant.

**Connexion** : Cette page permet à l'étudiant de se connecter a base de données pour pouvoir, par la suite, suivre des cours. Les champs de formulaire de connexion sont :

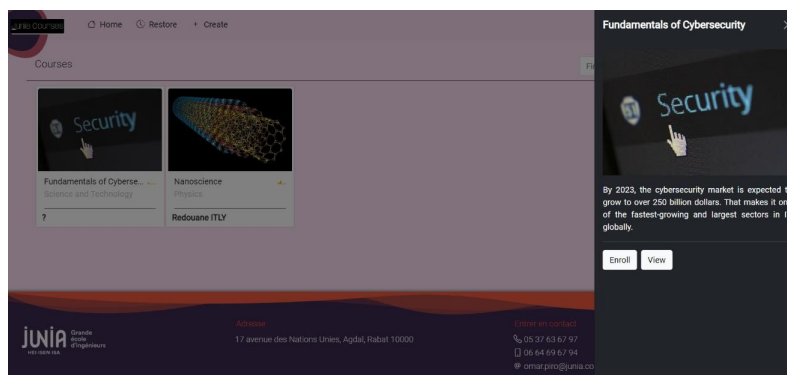
- o Adresse Mail
- o Mot de passe



**CATÉGORIES** : Cette rubrique permet à l'étudiant de consulter et rechercher les différents cours .



**COURS** : Cette page permet à l'étudiant de consulter ou de s'inscrire aux différents cours.



Contenue de cours : cette page contient toutes les ressources de chaque cours :

- o Les Vidéos
- o Les leçons

## 4.3 Interface enseignant

Dans les paragraphes qui suivent, nous allons exposer différentes situations que peut rencontrer l'enseignant. L'enseignant peut effectuer plusieurs tâches administratives tel que :

### 4.3.1 Ajouter un cours

Pour créer un cours E-learning, il faut indiquer le titre du cours, sa description, la catégorie ainsi que la difficulté tout en ajoutant une image du cours.

Junia Courses

Home Restore Create

Title Category

Description

Level Easy

Image Choose File No file chosen

☐ Public

Create

### 4.3.2 Ajouter des ressources

#### Création de leçons :

Cependant pour l'ajout d'une leçon on peut l'écrire directement sur la plateforme à l'aide du Markdown. Et aussi importer des vidéos aux leçons.

Junia Courses

Home Restore Create

Title Video

Content

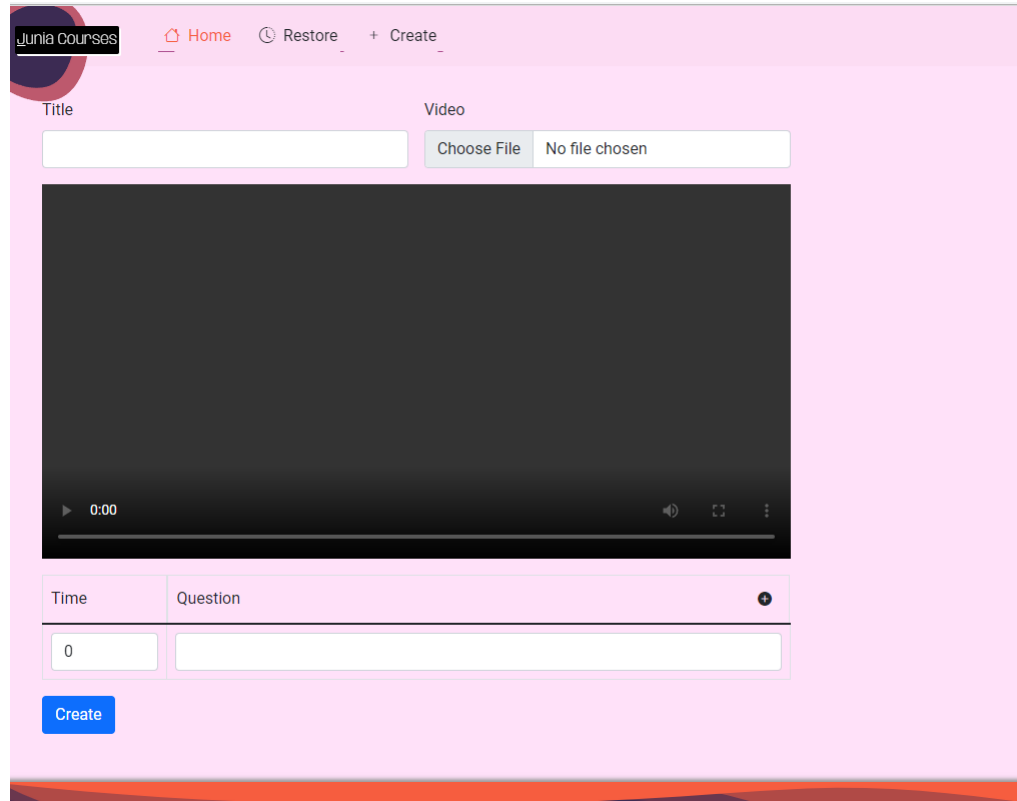
B I H " ≡ ≡ % 🖼️ 👁️ ✕ ⓘ

lines: 1 words: 0 1:1

Submit

### Importation de la vidéo :

Vous pouvez importer des vidéos sans limite. En revanche, plus une vidéo est lourde, plus elle mettra du temps à charger. Ainsi le professeur peut créer un questionnaire durant la visualisation de la vidéo qui servira à tester l'apprenant



The screenshot shows the 'Junia Courses' web interface. At the top, there is a navigation bar with 'Home', 'Restore', and '+ Create' links. Below this, the 'Video' section is active, featuring a 'Title' input field and a 'Video' section with a 'Choose File' button and 'No file chosen' text. A large video player is centered, showing a black screen with a play button and a 0:00 timestamp. Below the video player, there is a table with two columns: 'Time' and 'Question'. The 'Time' column has a value of '0'. The 'Question' column is empty. A 'Create' button is located at the bottom left of the form.

## 4.4 CONCLUSION

La partie de réalisation détermine une idée plus claire sur les tâches qui sont réalisées dans cette plate-forme par la présentation des interfaces graphiques. Enfin avec cette partie nous terminons la phase de développement de ce projet.

## Chapitre 5

# Conclusion et Perspective

Ce projet d'étude consiste à concevoir une plateforme qui permet de réaliser un système d'apprentissage en ligne dédié à notre grande école Junia . C'est une application presque finalisée et accompagnée de tous les documentations technique et conceptuelle nécessaire à sa bonne évolution. Pour concevoir ce travail nous avons présenté premièrement le cadre de ce projet. En second, nous avons montré la phase de conception. Finalement, nous avons traité toutes les phases nécessaires à la réalisation de cette application, et dans cette phase nous avons appris à mieux manipuler les langages Nodejs, ExpressJs, nous avons aussi approfondi nos connaissances sur le traitement des bases de données avec MongoDB. Par ailleurs, ce projet a totalement répondu à nos attentes. Des améliorations pourraient aussi être apportées à ce site par exemple dans le cas d'une réelle utilisation du site. Enfin, la réalisation de ce projet de travail en équipe sur une durée limitée est un bon entraînement.

### 5.1 Les problèmes rencontrés

Durant ce projet on a vraiment baver pour y'arriver à notre objectif celui de créer une plateforme e-learning pour notre établissement. On s'est confronté à plusieurs problèmes tout le long du projet que ça soit par rapport au temps qui était pas évident , ainsi que tout était nouveau pour nous que ça soit les langages utilisés , comment gérer une base de données ,ce qui fait on a consacré beaucoup de temps pour la recherche à fin de mieux comprendre les choses et savoir les outils convenables à notre projet . Quelques problèmes qu'on a rencontrés durant ce projet est qu'on a réussi à résoudre :

- Identifier les outils convenables pour ce projet.
- Conception de l'application.
- Mise en place des vidéos.
- Écriture des leçons.

## 5.2 Les avancements possibles

- **Collecte** de données.
- **Recommandations** automatiques.
- **Ajouter** des fonctionnalités administratives.