

Refactoring d'application web suivant le modèle des microservices

Etudiantes :

Lamyae KHAIRON
Meryam RHADI
Awa SECK

Encadrants:

Vincent BERRY
Chouki TIBERMACHINE

Reporteur:

Abdelhak-Djamel SERIAI



PLAN

I. Introduction

II. Contexte :

1. l'application *ShellOnYou*
2. Problème de l'application actuelle
3. Solutions proposées

III. Etat de l'art sur la migration vers les MS

IV. Gestion de projet

V. La migration d'un monolithe vers une architecture microservices

1. Création du microservice exercice
2. Communication entre monolithe et microservice

VI. Découpage en clusters

1. Construction et pré-traitement des données
2. Clustering
3. Inférence de nom du micro-service

VII. Conclusion et perspective d'amélioration



I. INTRODUCTION



Existe depuis 4 ans



307 utilisateurs



878 énoncés d'exercices



**1851 réponses
d'étudiants ont été
traitées**



13 nationalités d'étudiants

II. CONTEXTE

1. l'application *ShellOnYou*

- Adopte l'architecture MVC (Modèle-Vue-Contrôleur)
- Programée en Node.js et le framework Express, et adossée à une base de données Postgres
- Les vues de ShellOnYou sont écrites sous la forme de templates EJS

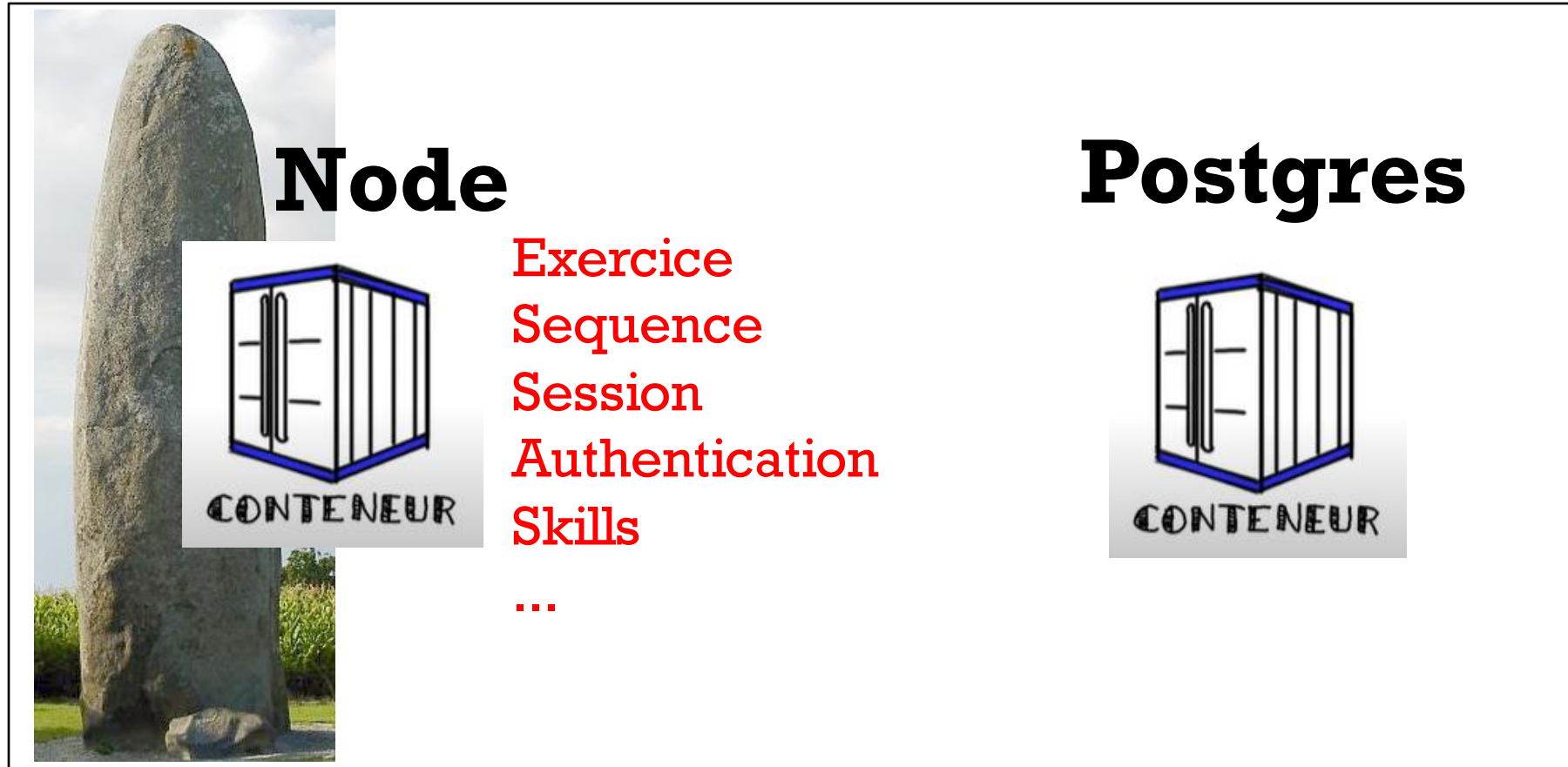


II. CONTEXTE

2. Problème de l'application *ShellOnYou*

- Couplage fort
- Passage à l'échelle limité
- Complexité de gestion de l'application
- Architecture monolithique

➤ Architecture monolithique

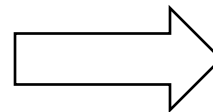
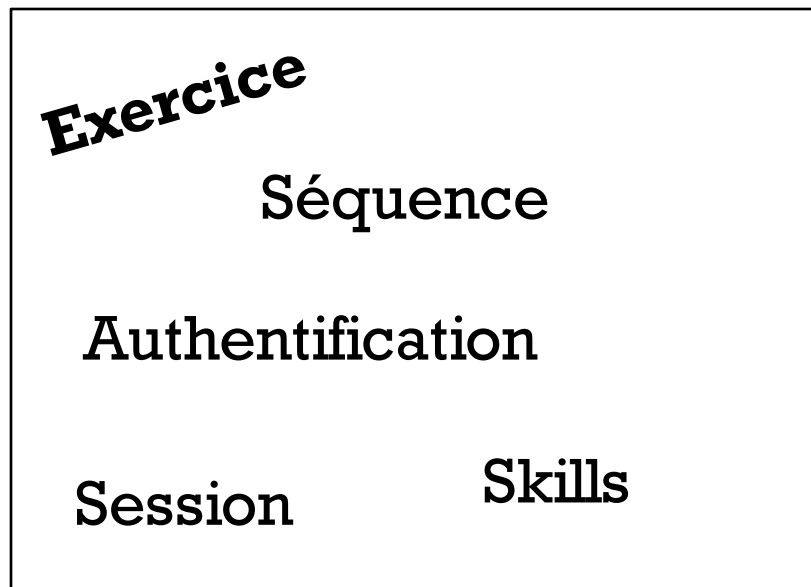


II. CONTEXTE

3. Solution proposée

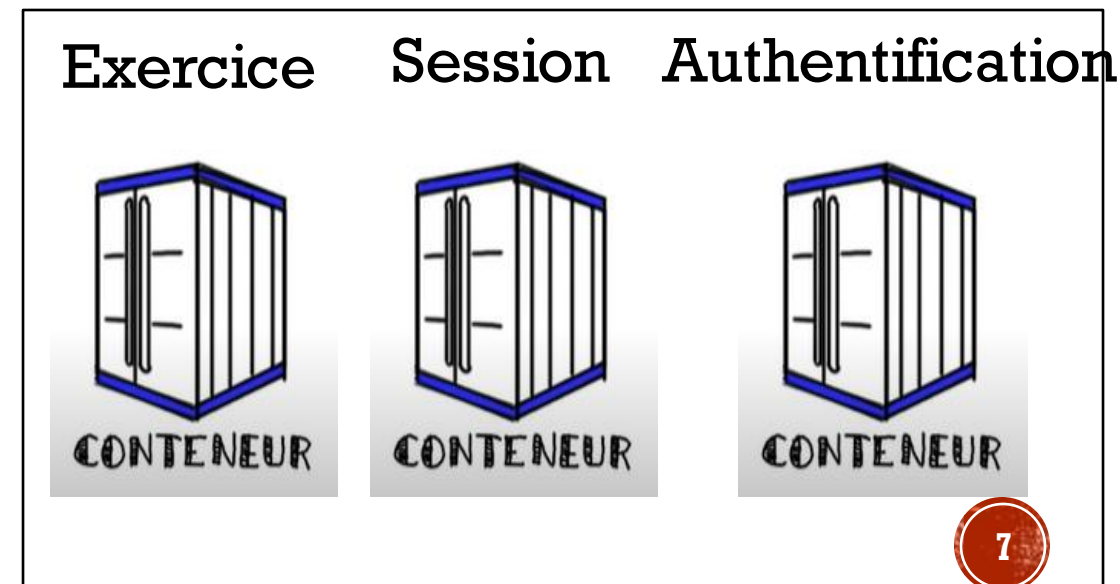
Architecture monolithique

Node



Solution : arch. à MS

Microservices

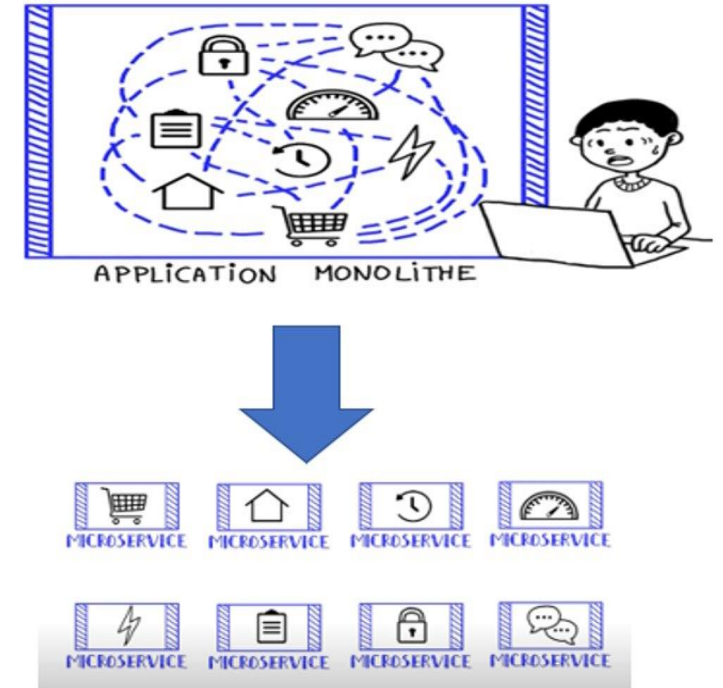


MISSION

Migration de l'application monolithique en une application sous forme de micro-services.

Deux approches :

- Refactorisation manuelle de la partie la plus utilisable du monolithe (gestion des exercices)
- Application d'une méthode d'identification des micro-services par une méthode automatisée de clustering (exploitant le schéma de la base de données)



IV. ETAT DE L'ART SUR LA MIGRATION VERS MS

Articles lus:

- From Monolith to Microservices: A Classification of Refactoring Approaches , auteur : *Jonas Fritsch, Justus Bogner Alfred Zimmer, mannStefan Wagner*, **année de publication:** 2018
- From monolithic systems to Microservices : An assessment framework, auteur : *Florian Auer, Valentina Lenarduzzi, Michael Felderer, Davide Taibi*, **année de publication:** 2021
- Towards à Technique for Extracting Microservices from Monolithic Enterprise Systems, auteur : *Alessandra Levcovitz, Ricardo Terra, Marco Tulio Valente*, **année de publication:** 2016
- Towards Migrating Legacy Software Systems to Microservice-based Architectures: a Data-Centric Process for Microservice Identification, auteur : *Yamina Romani, Okba Tibermacine, Chouki Tibermacine*, **année de publication:** 2022

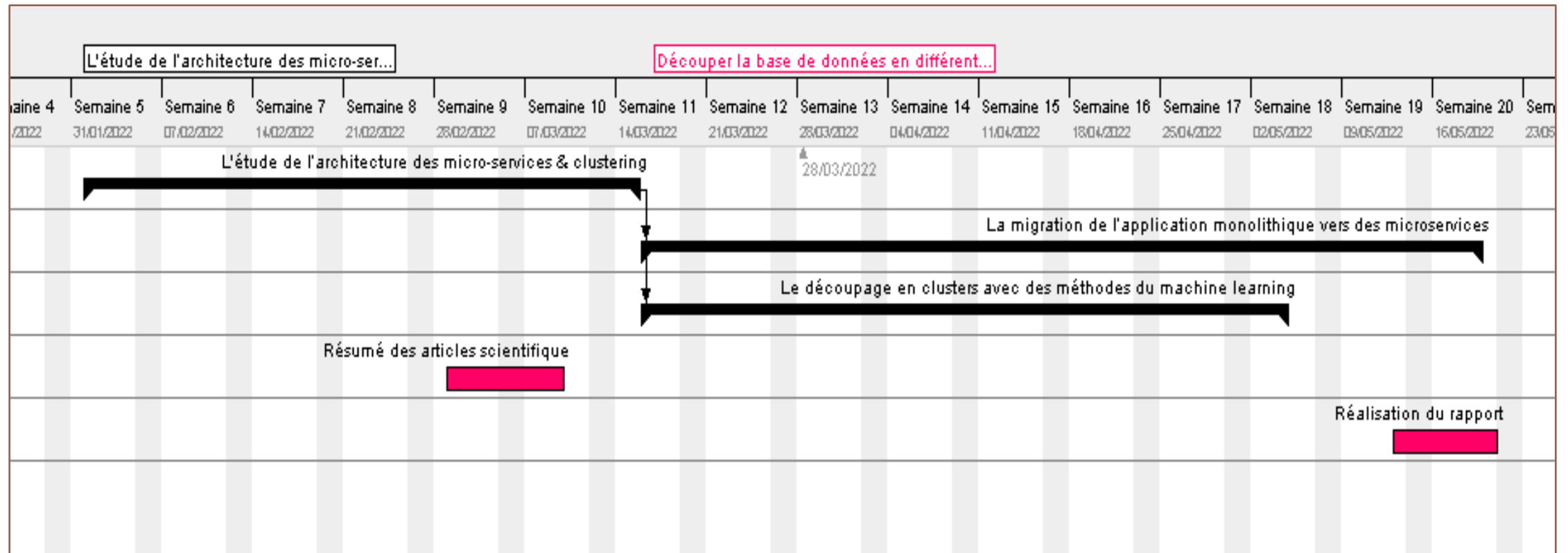
IV. GESTION DE PROJET

1. Outils de communication:



IV. GESTION DE PROJET

2. Diagramme de Gantt:



V. Migration du monolithe vers les microservices

1. Création du microservice exercice



**EXTRACTION DU
CODE DE
"GESTION DES
EXERCICES"
COMME MS**



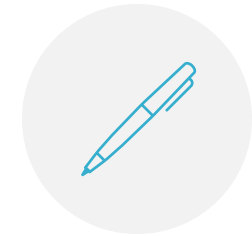
**CRÉATION D'UN SERVER
NODEJS**



**CRÉATION ET
INSTALLATION DES
DÉPENDANCES NODE
DE MS**

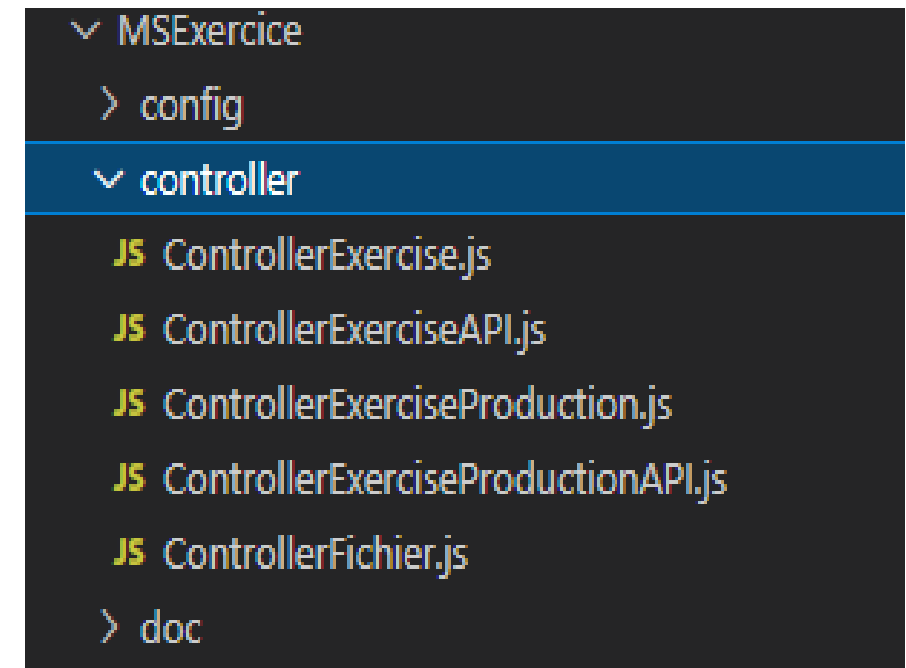
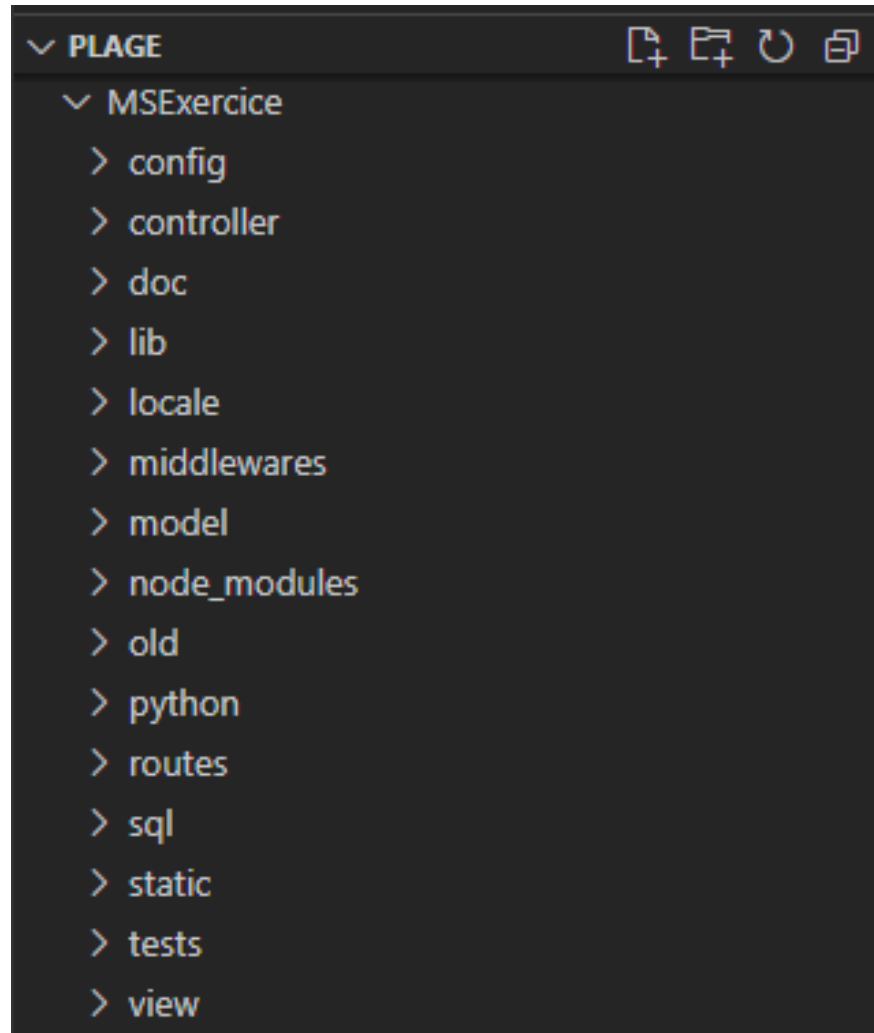


**DÉFINITION ET
CONSTRUCTION DES
IMAGES DOCKER DE MS**



**DÉCLARATION
DE MS DANS
L'ORCHESTRATION
DOCKER**

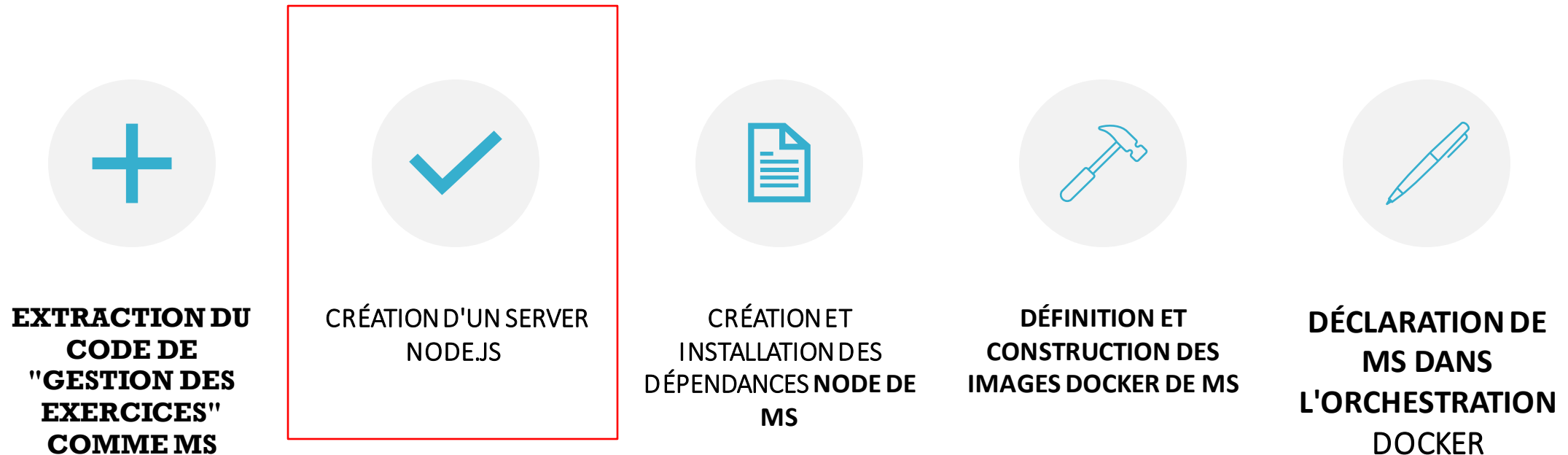
a. Extraction du code de "Gestion des exercices" comme un microservice



```
module.exports.readAll = async function (req, res) {  
  debug("Reading all");  
  const user = req.session;  
  
  const userList2= await axios.get('http://node:5001/api/users')  
  debug("userslist: "+userList2.data);  
  
  const tab = await ModelExercise.readAll();  
  
  if (tab) {  
    res.render("exercise/list", {  
      locale: req.i18n.getLocale(),  
      pageTitle: req.i18n.__("Exercises list"),  
      tab: tab,  
      user: user,  
      users: userList2.data,  
    });  
  } else {  
    res.status(500).end();  
  }  
};
```

V. Migration du monolithe vers les microservices

1. Création du microservice exercice



b. Création d'un server Node.js

```
JS index.js
MSExercice > JS index.js > ...
81
82 // ----- Exercice routes -----
83 app.use(require("../routes/exercises"));
84
85 // ----- Exercice Productions routes -----
86 app.use(require("../routes/exerciseProduction"));
87
88
89 //start services
90 app.listen(5021, ()=>{
91   |   | console.log('MSExercice listen in port 5021');
92   |   |
93   |   | })
94
95
```

V. Migration du monolithe vers les microservices

1. Création du microservice exercice



**EXTRACTION DU
CODE DE
"GESTION DES
EXERCICES"
COMME MS**



**CRÉATION D'UN SERVER
NODE.JS**



**CRÉATION ET
INSTALLATION DES
DÉPENDANCES NODE DE
MS**



**DÉFINITION ET
CONSTRUCTION DES
IMAGES DOCKER DE MS**



**DÉCLARATION DE
MS DANS
L'ORCHESTRATION
DOCKER**

c. Création et installation des dépendances node de ms

```
{ } package.json X
MSExercice > { } package.json > { } devDependencies
1  {
2    "name": "plage",
3    "version": "0.0.0",
4    "description": "A simple application for Shell Week @ IG Polytech Mtp",
5    "main": "index.js",
6    "repository": "",
7    "author": "...",
8    "scripts": {
9      "start": "node index.js",
10     "start-watch": "nodemon index.js",
11     "lint": "eslint index.js server.js controller --ext .js ",
12     "lint-fix": "eslint index.js server.js controller --ext .js --fix",
13     "test": "NODE_ENV=test APP_ENV=test jest",
14     "test-watch": "NODE_ENV=testjest --watch",
15     "ejs-lint": "./node_modules/.bin/ejslint view"
16   },
17   "dependencies": {
18     "archiver": "^5.3.0",
19     "async": "^3.2.3",
20     "axios": "^0.21.4",
21     "body-parser": "^1.19.0",
22     "child_process": "~1.0.2",
23     "connect-pg-simple": "^6.1.0",
24     "cookie-parser": "^1.4.4",
25     "crypto": "^1.0.1",
26     "debug": "^4.1.1",
27     "ejs": "^3.1.8",
28     "ejs-lint": "^1.2.1",
29     "engine": "^1.0.0",
30     "express": "^4.18.1",
31     "express-fileupload": "^1.2.1",
```

V. Migration du monolithe vers les microservices

1. Création du microservice exercice



**EXTRACTION DU
CODE DE
"GESTION DES
EXERCICES"
COMME MS**



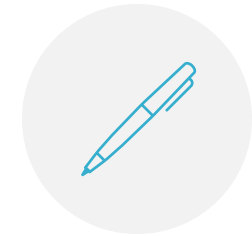
**CRÉATION D'UN SERVER
NODEJS**



**CRÉATION ET
INSTALLATION DES
DÉPENDANCES NODE DE
MS**



**DÉFINITION ET
CONSTRUCTION DES
IMAGES DOCKER DE MS**



**DÉCLARATION DE
MS DANS
L'ORCHESTRATION
DOCKER**

d. Définition et construction des images Docker de MS

```
Dockerfile X
MSExercise > Dockerfile
5
6 FROM node:10
7
8 # Installs pip and psycopg2-binary necessary for some python exercises
9 RUN apt-get update && apt-get install -y python-pip
10 RUN pip install psycopg2-binary
11
12 # Create app directory and default directory to put things
13 # (This way we do not have to type out full file paths but
14 #   can use relative paths based on the working directory. )
15 WORKDIR /usr/src/app
16
17 # Install app dependencies
18 # (this way we don't rebuild node modules each time we re-build the container ;
19 #   if package.json changes then node modules will be rebuilt )
20 # A wildcard is used to ensure both package.json AND package-lock.json are copied
21 # where available (npm@5+)
22 #
23 # NOT NECESSARY IN DEV VERSION ?
24 # COPY package*.json ./
25 # COPY package*.json /usr/src/app
26
27 COPY package.json package.json
28 COPY package-lock.json package-lock.json
29
30
31 RUN npm install
```

V. Migration du monolithe vers les microservices

1. Création du microservice exercice



**EXTRACTION DU
CODE DE
"GESTION DES
EXERCICES"
COMME MS**



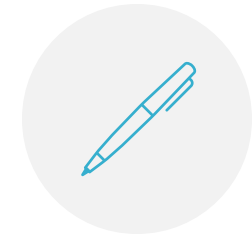
**CRÉATION D'UN SERVER
NODEJS**



**CRÉATION ET
INSTALLATION DES
DÉPENDANCES NODE DE
MS**



**DÉFINITION ET
CONSTRUCTION DES
IMAGES DOCKER DE MS**



**DÉCLARATION DE
MS DANS
L'ORCHESTRATION
DOCKER**

e. Déclaration de MS dans l'orchestration Docker

```
docker-compose.yml X
docker-compose.yml
20
27   MSExercice:
28     volumes:
29     | - ./MSExercice:/usr/src/app
30     build: ./MSExercice
31     image: "exercice/exercice:21"
32     container_name: MSExercice
33     depends_on:
34     | - postgres
35     ports:
36     | - 5021:5021
37     networks:
38     | - plagenet
39     env_file:
40     | - variables.env
41
```


Créer un nouvel exercice

Exercise name :

jouons avec ls

Énoncé de l'exercice :

Fichier

Editer

Voir

Format

Outils

↶

↷

Paragraphe

▼

B

I

≡

≡

≡

≡

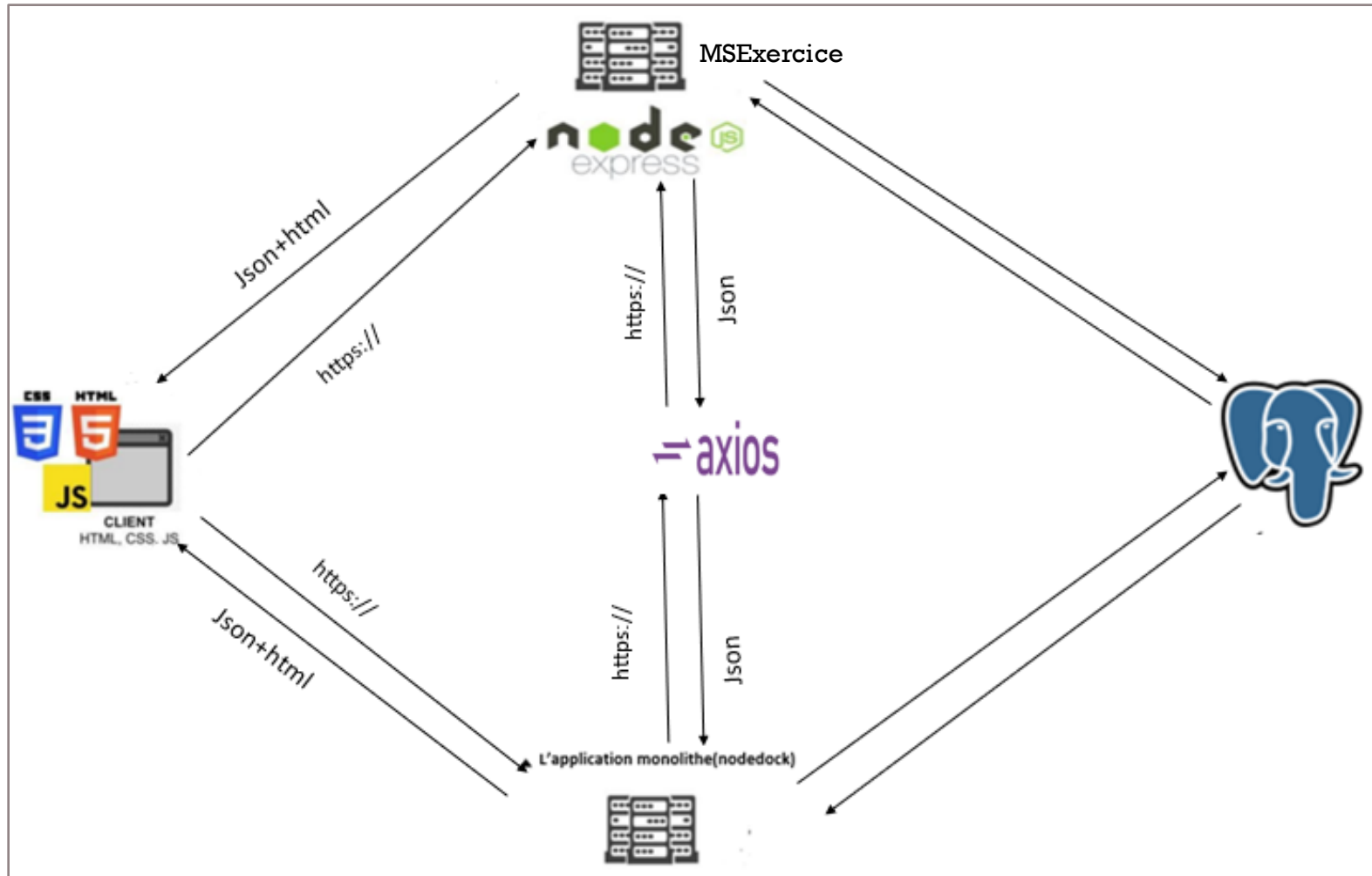
≡

≡

V. Migration du monolithe vers les microservices

2. Communication entre monolithe et microservice

a. Utilisation de la bibliothèque axios



Exemple

```
router.get('/exercise/:ex_id',connection, (req, res) =>{
  debug("Route monolithe vers MS exercise/:ex_id");
  const user = req.session;
  axios.get("http://MSExercice:5021/exercise/:ex_id")
    .then(async function (response) {
      res.statusCode=200;
      debug(response.data);
      debug(response.data.archivePath)
      res.render("exercise/detail", {
        author: response.data.author,
        exo: response.data.exo,
        FCAsize: response.data.FCAsize,
        FCAname: response.data.FCAname,
        urlArchiveFile: response.data.urlArchiveFile,
        FSCsize: response.data.FSCsize,
        FSCname: response.data.FSCsize,
        urlCreationScript: response.data.urlCreationScript,
        FMSsize: response.data.FMSsize,
        FMSname: response.data.FMSname,
        urlAnalysisScript: response.data.urlAnalysisScript,
        locale: req.i18n.getLocale(),
        pageTitle: req.i18n.__( "Exercise details"),
        user: user,
        archivePath : response.data.archivePath,
        creationScriptPath: response.data.creationScriptPath,
        analysisScriptPath: response.data.analysisScriptPath,
        urlAnalysisScript: response.data.urlAnalysisScript
      });
    })
    .catch(function (error) {
      res.statusCode=500
    })
})
```

Figure1: Route de l'affichage des détails d'un exercice dans le monolithe

```
// Details
router.get('/exercise/:ex_id', function (req, res) {
  ControllerExercise.read(req, res)
})
```

Figure2: Route de l'affichage des détails d'un exercice dans le MSExercice

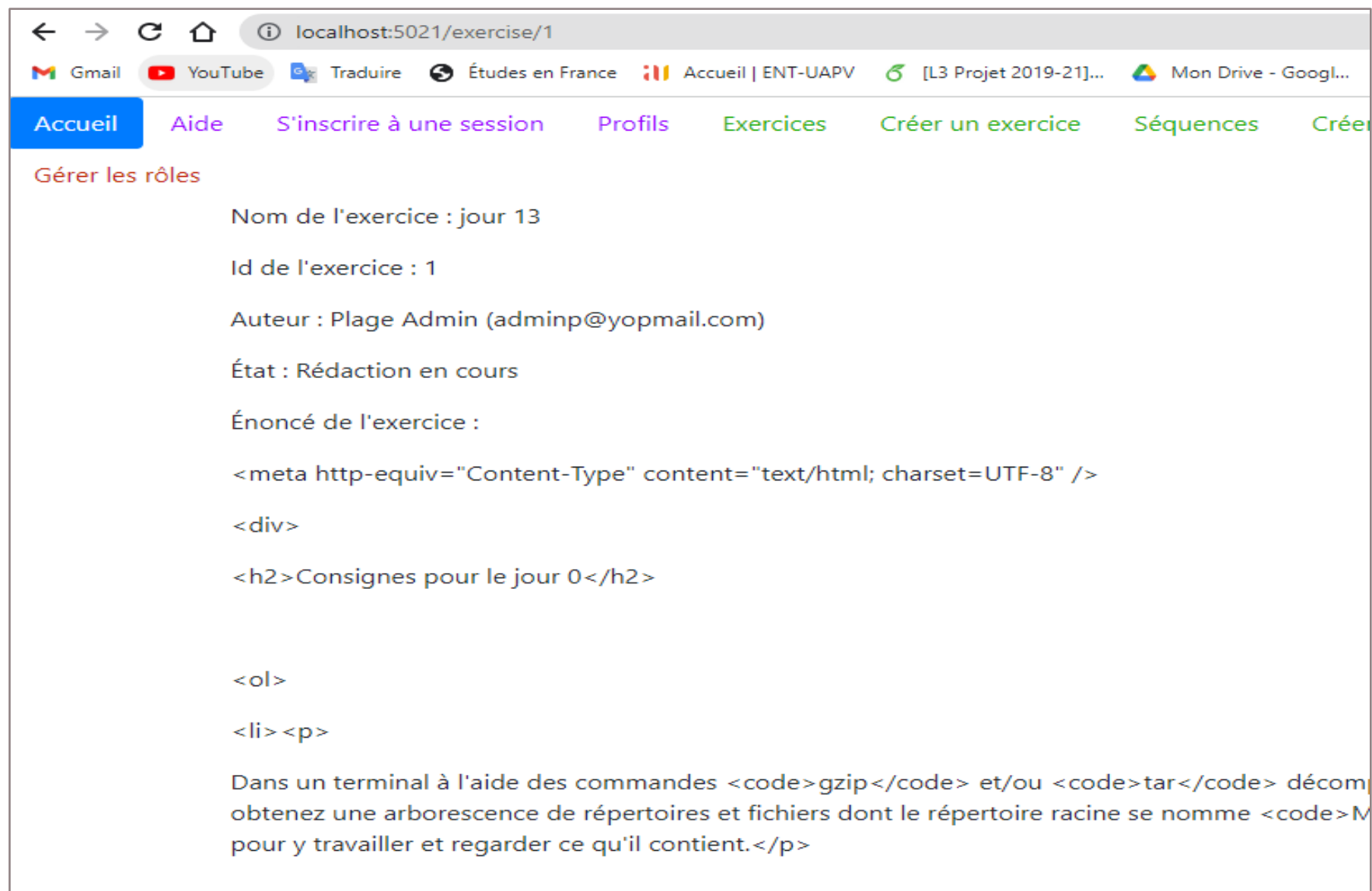
Exemple

```
module.exports.read = async function (req, res) {
  debug("Want to get an exercise with its files");
  //id de l'user qui esr connecter pour recuperer les exo qui a écrit
  const user = req.session;
  const exo = await ModelExercise.readById('1');
  if (exo) {
    const resExo = new ModelExercise(exo);
    let fCA, fSCS, fMS;
    let archivePath, urlArchiveFile;
    let creationScriptPath, urlCreationScript;
    let analysisScriptPath, urlAnalysisScript;
    const empty = { name: "Aucun fichier", size: 0, md5: 0, data: 0 };
    if (resExo.template_archive !== undefined) {
      fCA = new ModelFile(JSON.parse(resExo.template_archive));
      // new
      let fileName = fCA.name;
      let randomFolderName = crypto.randomBytes(20).toString('hex');
      archivePath = "static/files/"+randomFolderName+"/"
      fs.mkdir(archivePath, { recursive: true }, function (err) {
        if (!err) {
          archivePath += fileName
          fs.writeFile(archivePath, Buffer.from(fCA.data), function (err) {
            if (!err) {
              debug("archive file put in folder "+archivePath)
            }
          })
        }
      })
    }
  }
}
```

```
res.status(200).send(JSON.stringify({
  author: author,
  exo: resExo,
  FCAsize: fCA.size,
  FCAname: fCA.name,
  urlArchiveFile: urlArchiveFile,
  FSCsize: fSCS.size,
  FSCname: fSCS.name,
  urlCreationScript: urlCreationScript,
  FMSSize: fMS.size,
  FMSname: fMS.name,
  urlAnalysisScript: urlAnalysisScript,
  locale: req.i18n.getLocale(),
  pageTitle: req.i18n.__("Exercise details"),
  user: user,
  archivePath :archivePath,
  creationScriptPath: creationScriptPath,
  analysisScriptPath: analysisScriptPath,
  urlAnalysisScript: urlAnalysisScript
})));
} else {
  res.status(500).end();
}
```

Traitement et envoi de réponse dans le contrôleur de
MSExercice

Résultat



The screenshot shows a web browser window with the address bar at `localhost:5021/exercice/1`. The browser's bookmark bar includes links to Gmail, YouTube, Traduire, Études en France, Accueil | ENT-UAPV, [L3 Projet 2019-21]..., and Mon Drive - Googl... The application's navigation menu features links for Accueil (highlighted in blue), Aide, S'inscrire à une session, Profils, Exercices, Créer un exercice, Séquences, and Créer. Below the menu, a red link labeled "Gérer les rôles" is visible. The main content area displays the following details for an exercise:

- Nom de l'exercice : jour 13
- Id de l'exercice : 1
- Auteur : Plage Admin (adminp@yopmail.com)
- État : Rédaction en cours
- Énoncé de l'exercice :

```
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />

<div>

<h2>Consignes pour le jour 0</h2>

<ol>

<li><p>

Dans un terminal à l'aide des commandes gzip et/ou tar décom
obtenez une arborescence de répertoires et fichiers dont le répertoire racine se nomme M
pour y travailler et regarder ce qu'il contient.</p>
```

V. Migration du monolithe vers les microservices

2. Communication entre monolithe et microservice

b. Parse du contenu des requêtes (body) dans index du monolithe

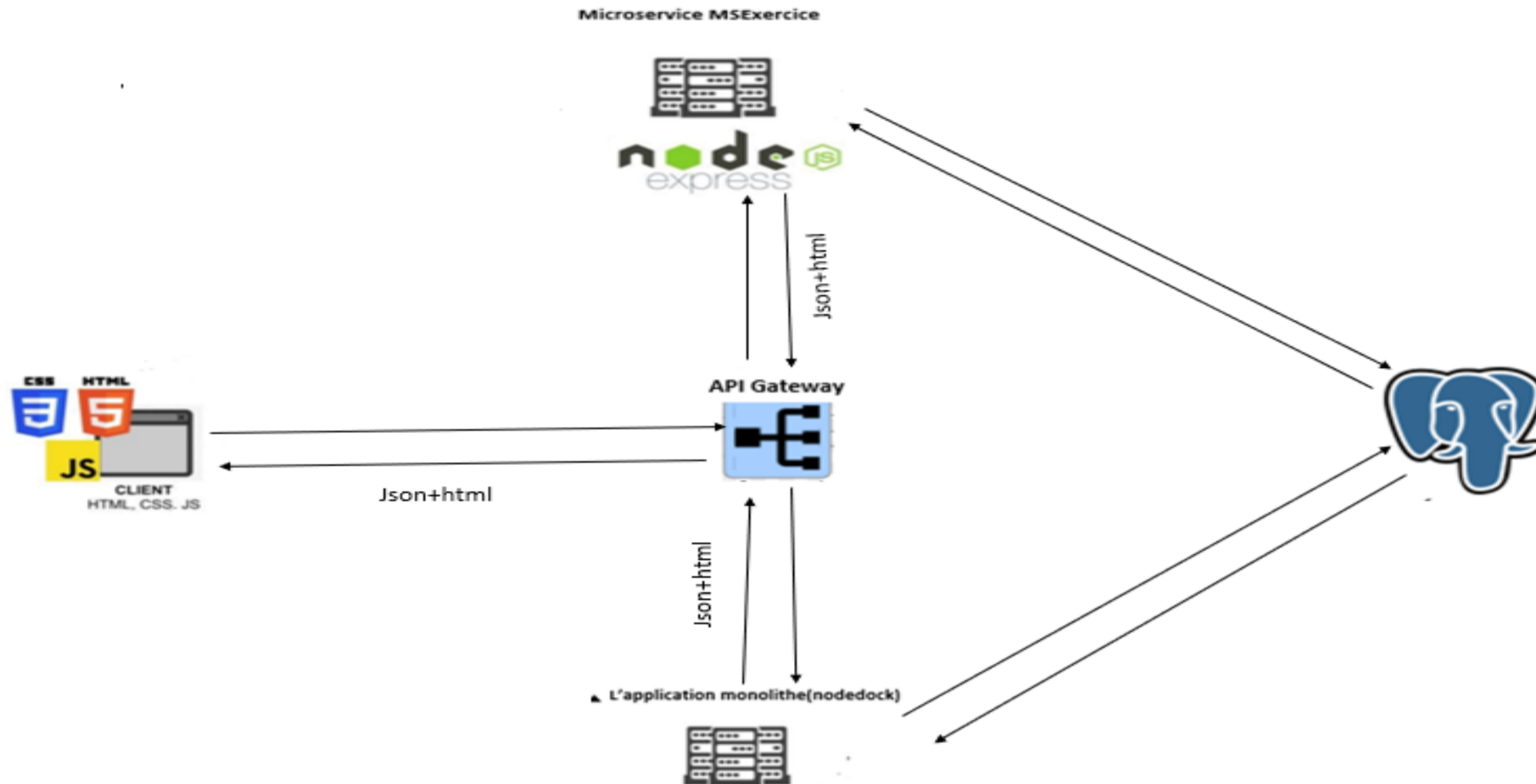
```
MSExercice      prefLocale: 'en' },  
MSExercice      body: {},  
MSExercice      _body: true,  
MSExercice      length: undefined,  
MSExercice      route:
```

```
JS index.js  X  
nodedock > JS index.js > ...  
73  app.use(express.json()); // <=== parse request body as JSON  
74  app.use(  
75    express.urlencoded({  
76      extended: true,  
77    })  
78  );  
79
```

V. Migration du monolithe vers les microservices

2. Communication entre monolithe et microservice

C. Création de API Gateway

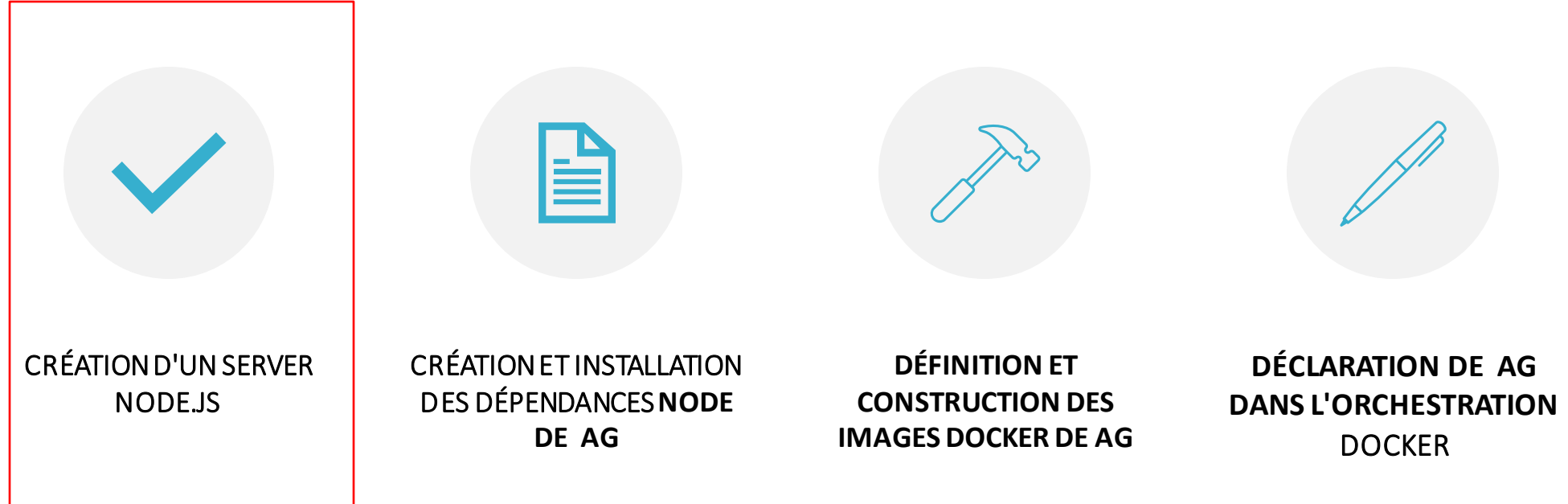


V. Migration du monolithe vers les microservices

2. Communication entre monolithe et microservice

C. Création de API Gateway

- Etapes



a. Création d'un server node.js

```
JS index.js
Gateway > JS index.js > ...
15
16 var httpProxy = require('http-proxy');
17 var apiProxy = httpProxy.createProxyServer();
18 //les routes de exercice sont envoyés à MSExercice
19 app.use("/Ms", function(req, res) {
20   apiProxy.web(req, res, { target: 'http://MSExercice:5021'})
21 }
22 });
23 // les autres routes sont envoyées au monolithe :
24 app.use("/", function(req, res) {
25
26   apiProxy.web(req, res, { target: 'http://node:5001'})
27
28 });
29 app.listen(5022,()=>{
30   console.log('ShellOnYou **API gateway** listening on port 5022');
31 });
32
```

V. Migration du monolithe vers les microservices

2. Communication entre monolithe et microservice

C. Création de API Gateway

- Etapes



CRÉATION D'UN SERVER
NODE.JS



CRÉATION ET INSTALLATION
DES DÉPENDANCES NODE
DE AG



DÉFINITION ET
CONSTRUCTION DES
IMAGES DOCKER DE AG



DÉCLARATION DE AG
DANS L'ORCHESTRATION
DOCKER

b. Création et installation des dépendances node de AG

```
package.json X
Gateway > {} package.json > {} dependencies
1  {
2    "name": "plage",
3    "version": "0.0.0",
4    "description": "A simple application for Shell Week @ IG Polytech Mtp",
5    "main": "index.js",
6    "repository": "",
7    "author": "...",
8    "scripts": {
9      "start": "node index.js",
10     "start-watch": "nodemon index.js",
11     "lint": "eslint index.js server.js controller --ext .js ",
12     "lint-fix": "eslint index.js server.js controller --ext .js --fix",
13     "test": "NODE_ENV=test APP_ENV=test jest",
14     "test-watch": "NODE_ENV=testjest --watch",
15     "ejs-lint": "../node_modules/.bin/ejslint view"
16   },
17   "dependencies": {
18     "archiver": "^5.3.0",
19     "async": "^3.2.3",
20     "axios": "^0.21.4",
21     "body-parser": "^1.19.0",
22     "child_process": "~1.0.2",
23     "connect-pg-simple": "^6.1.0",
24     "cookie-parser": "^1.4.4",
25     "crypto": "^1.0.1",
26     "debug": "^4.1.1",
27     "ejs": "^3.1.8",
28     "ejs-lint": "^1.2.1",
29     "express": "^4.17.1",
30     "express-fileupload": "^1.2.1",
31     "express-http-proxy": "^1.6.3",
```

V. Migration du monolithe vers les microservices

2. Communication entre monolithe et microservice

C. Création de API Gateway

- Etapes



CRÉATION D'UN SERVER
NODE.JS



CRÉATION ET INSTALLATION
DES DÉPENDANCES NODE
DE AG



DÉFINITION ET
CONSTRUCTION DES
IMAGES DOCKER DE AG



DÉCLARATION DE AG
DANS L'ORCHESTRATION
DOCKER

c. Définition et construction des images Docker de AG

```
Dockerfile X
Gateway > Dockerfile
5
6 FROM node:10
7
8 # Installs pip and psycopg2-binary necessary for some python exercises
9 RUN apt-get update && apt-get install -y python-pip
10 RUN pip install psycopg2-binary
11
12 # Create app directory and default directory to put things
13 # (This way we do not have to type out full file paths but
14 # can use relative paths based on the working directory. )
15 RUN mkdir -p /usr/src/app
16
17 WORKDIR /usr/src/app
18
19 # Install app dependencies
20 # (this way we don't rebuild node modules each time we re-build the container ;
21 # if package.json changes then node modules will be rebuilt )
22 # A wildcard is used to ensure both package.json AND package-lock.json are copied
23 # where available (npm@5+)
24 #
25 # NOT NECESSARY IN DEV VERSION ?
26 # COPY package*.json ./
27 # COPY package*.json /usr/src/app
28
29 COPY package.json package.json
30 COPY package-lock.json package-lock.json
31
32 COPY package.json /usr/src/app/
33
34 RUN npm install
35
```

V. Migration du monolithe vers les microservices

2. Communication entre monolithe et microservice

C. Création de API Gateway

- Etapes



CRÉATION D'UN SERVER
NODE.JS



CRÉATION ET INSTALLATION
DES DÉPENDANCES NODE
DE AG



DÉFINITION ET
CONSTRUCTION DES
IMAGES DOCKER DE AG



DÉCLARATION DE AG
DANS L'ORCHESTRATION
DOCKER


d. Déclaration de AG dans l'orchestration Docker

```
docker-compose.yml X
docker-compose.yml
44
45   Gateway:
46     volumes:
47       - ./Gateway:/usr/src/app
48     build: ./Gateway
49     image: "gateway/gateway:25"
50     container_name: gateway
51     depends_on:
52       - postgres
53     ports:
54       - 5022:5022
55     networks:
56       - plagenet
57     env_file:
58       - variables.env
59
```



DÉMONSTRATION

Back

Recording in Progress



00:00 / 05:00



You can hide the page during recording.

VI. DÉCOUPAGE EN CLUSTERS

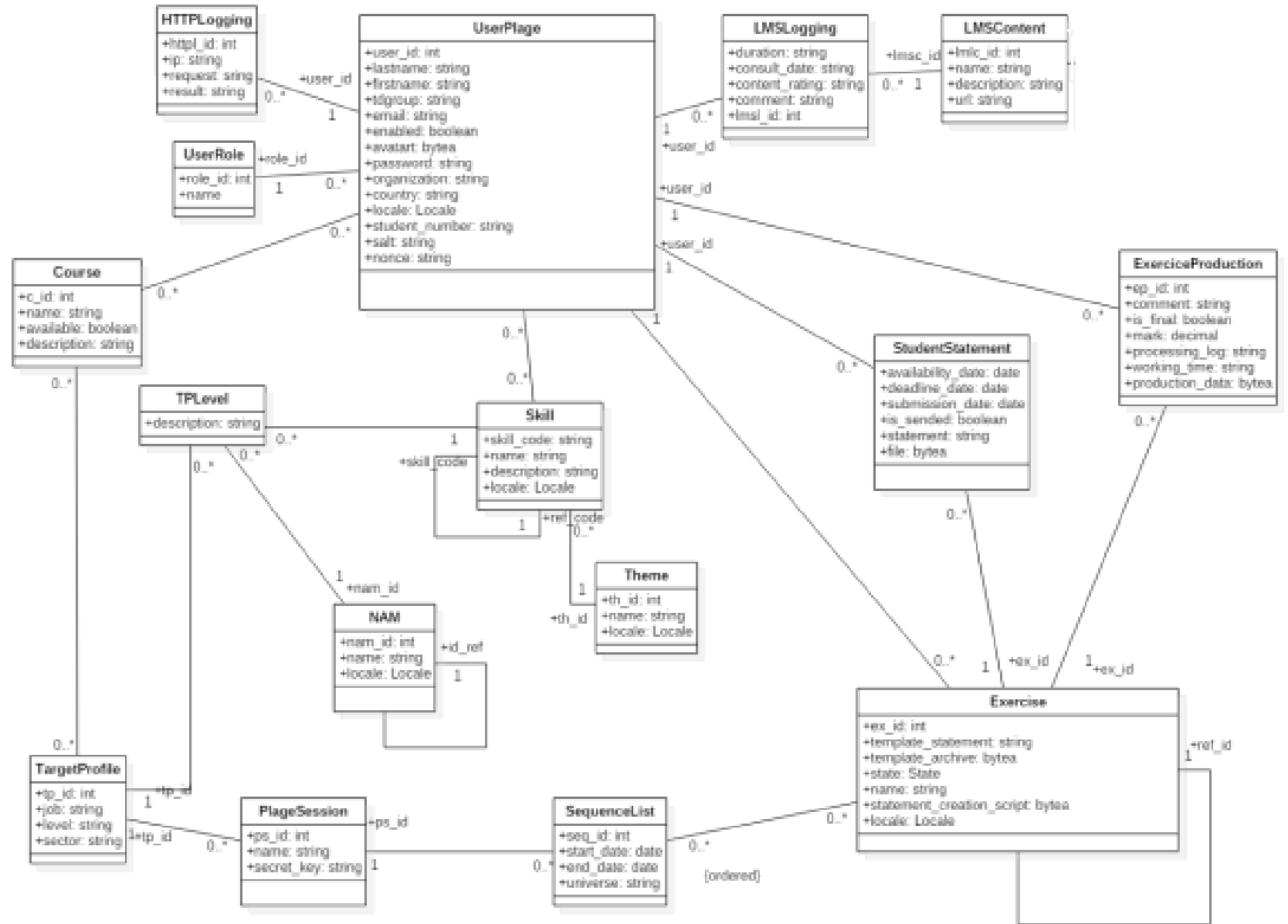
VI. DÉCOUPAGE EN CLUSTERS

- 1. Construction et pré-traitement des données**
- 2. Clustering**
- 3. Inférence de nom du micro-service**

VI. DÉCOUPAGE EN CLUSTERS

1. Construction et pré- traitement des données

❖ Construction:



VI. DÉCOUPAGE EN CLUSTERS

1. Construction et pré-traitement des données

❖ Construction:

- ✓ Le nom de la table est répété plusieurs fois dans le document.
- ✓ La clé étrangère et le nom de la table source sont multipliés dans la table cible.
- ✓ De même le nom de table cible est aussi multiplié dans la table source.

VI. DÉCOUPAGE EN CLUSTERS

1. Construction et pré-traitement des données

❖ Pré-traitement :

- ✓ ***Tokénisation***: chaque symbole de document est divisé en mots séparés (jetons).
- ✓ ***Lemmatisation***: transformation de chaque mot en son lemme (racine).

Pré-traitement

```
▼ Preprocessing

✓ [21] # stop words
0 s stop_words = set(stopwords.words('english'))
# punctuation
punctuation = set(string.punctuation)
# lemmatization
lemmatization = WordNetLemmatizer()

✓ [22] # function clean
0 s def clean(documents):
    # split documents and remove stop words

    split_doc = " ".join([i for i in documents.lower().split() if i not in stop_words])

    # remove punctuation
    punc_doc = ''.join([j for j in split_doc if j not in punctuation])

    # normalize the text
    normalized = " ".join([lemmatization.lemmatize(word) for word in punc_doc.split()])

    return normalized
```

VI. DÉCOUPAGE EN CLUSTERS

2. Clustering

- *Mesure de similarité*: en utilisant la méthode **Jaccard Index** (ou **Jaccard Similarity**).

```
# Using the Jaccard index to get the word similarity
def jaccard_similarity(x,y):
    """ returns the jaccard similarity between two lists """
    intersection_cardinality = len(set.intersection(*[set(x), set(y)]))
    union_cardinality = len(set.union(*[set(x), set(y)]))
    return intersection_cardinality/float(union_cardinality)
```

VI. DÉCOUPAGE EN CLUSTERS

2. Clustering

- ***Vectorisation des documents:***

Nous avons utilisé la méthode ***TF-IDF*** pour vectoriser nos documents.

Il s'agit d'une mesure statistique utilisée pour mesurer l'importance d'un terme pour un document dans un ensemble de données.

VI. DÉCOUPAGE EN CLUSTERS

2. Clustering

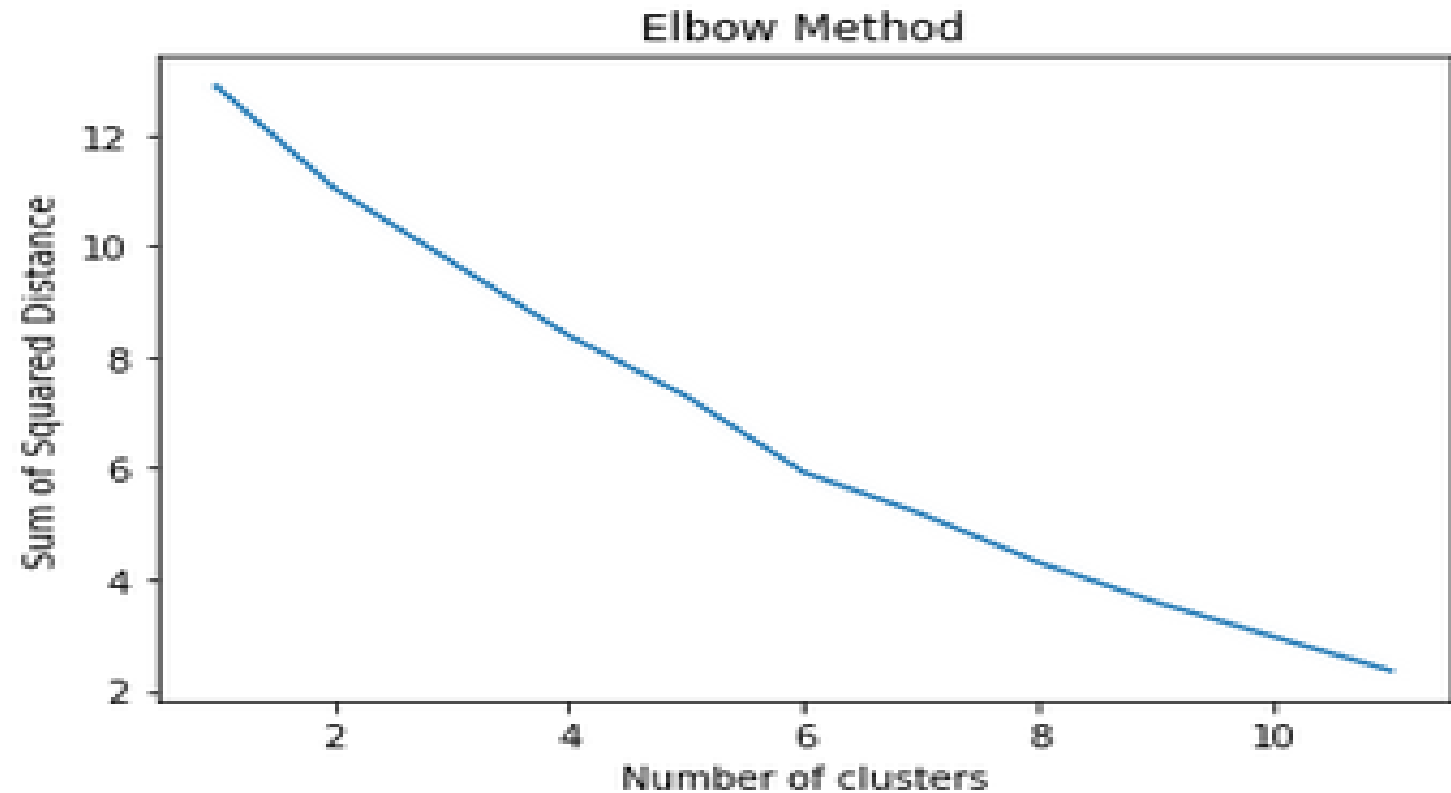
- ***Identification du nombre de clusters***

Afin d'identifier le nombre optimal de clusters, nous avons exploité une méthode largement utilisée nommée ***Elbow***.

Elbow calcule la fonction de coût générée à l'aide de plusieurs valeurs de k (nombre de clusters) par rapport à la somme des distance au carré entre les documents et leur assignation centroïdes de cluster.

Identification du nombre de clusters

Le nombre de clusters est choisi là où la courbe commence à former un coude (un elbow, d'où le nom).



VI. DÉCOUPAGE EN CLUSTERS

2. Clustering

▪ Méthode de clustering

L'algorithme de clustering K-means a été utilisé pour regrouper les documents représentés par les vecteurs générés.

K-means est une méthode simple pour regrouper une collection d'observation selon un nombre spécifié de clusters (k).

```
cluster 0 --> [0, 'acquiredskill', 'skill', 'theme']
cluster 1 --> [1, 'exercise', 'exerciseproduction', 'profile', 'sequencelist']
cluster 2 --> [2, 'userplage', 'userrole']
cluster 3 --> [3, 'plagesession', 'studentstatement', 'usersession']
cluster 4 --> [4, 'exerciselevel', 'nam', 'profilelevel']
cluster 5 --> [5, 'session']
```

VI. DÉCOUPAGE EN CLUSTERS

3. Inférence de nom du micro-service

Chaque micro-service potentiel sera nommé avec le symbole dominant dans le cluster associé.

VII. CONCLUSION ET PERSPECTIVES

CONCLUSION

- Bilan: deux approches :
 - Migration du monolithe vers MS
 - Clustering pour assister l'identification des MS
- Compétences acquises:
 - Développement avec une architecture à MS : Node, Docker, ...
 - Travail en groupe
- Difficultés rencontrées
 - Prise en main du code de l'application Shell On You au début
 - Les technologies utilisées dans ce projet nécessitent d'avoir un ordinateur puissant

PERSPECTIVES D'AMELIORATION

- ❑ Réalisation de plusieurs autres micro-services.
- ❑ Intégration des MS dans une même orchestration
- ❑ Inclusion de RabbitMQ qui est un système permettant de gérer des files de messages afin de permettre à différents MS de communiquer plus simplement

BIBLIOGRAPHIE:

- Jonas Fritzsche, Justus Bogner, Alfred Zimmermann, Stefan Wagner. From Monolith to Microservices : A Classification of Refactoring Approaches. Software Engineering. 2018. url: <https://arxiv.org/abs/1807.10059>
- Alessandra Levcovitz, Ricardo Terra, Marco Tulio Valente. Towards a Technique for Extracting Microservices from Monolithic Enterprise Systems. Research gate. 2016, url: https://www.researchgate.net/publication/302892908_Towards_a_Technique_for_Extracting_Microservices_from_Monolithic_Enterprise_Systems
- Florian Auer, Valentina Lenarduzzi, Michael Felderer, Davide Taibi. From monolithic systems to Microservices : An assessment framework. Science direct. 2021. url : <https://www.sciencedirect.com/science/article/pii/S0950584921000793>
- Yamina Romani, Okba Tibermacine, Chouki Tibermacine. Towards Migrating Legacy Software Systems to Microservice-based Architectures : a Data-Centric Process for Microservice Identification. Lirmm. 2022. url : https://www.lirmm.fr/~tibermacin/papers/2022/YRetAI_ICSA_NEI_2022.pdf
- Cookie connecté. Shéma de monolithe vs microservices. Cookie connecté. 2020. url: https://www.youtube.com/watch?v=ucHwpljUS2w&ab_channel=Cookieconnect%5C%C3%5C%A9