

Table des matières

I. projetmeteo utilisant API en stockant dans la BDD.....	2
A. Configuration.....	2
1. Configuration Initiale	2
2. Configuration Variables d'Environnements	3
B. Scripts de Configuration.....	4
C. Gestion des Logs	5
D. Script Principal : meteoCommande.js	6
E. Insertion data dans ELK (v5.5.3) dans le projetmeteo.....	8
1. Installation ElasticSearch en package (v13.3.1)	8
2. Mise a jour du script principale meteoCommande.js	8
II. projetmeteov2 -> ELK avec authentification (v8.17.1) avec NodeJS (v22.13.0)	9
A. Prérequis	9
B. Fichier Test de la connexion à ELK	10
C. Script projetmeteov2	11
III. Explication d'un index dans ELK	13
✦ A. Création automatique de l'index	13
✦ B. Détection automatique des types de propriétés	13
✦ C. Forcer un mapping personnalisé (optionnel)	14
✦ D. Vérifier le mapping d'un index dans ELK	14
✦ E. Rappel des termes employés	15
IV. Exécution du Projet.....	16
Ressources :	17

I. projetmeteo utilisant API en stockant dans la BDD

Application Node.js qui récupère et stocke la température moyenne d'un lieu via l'API Meteo Concept, avec gestion des logs et base MySQL.

A. Configuration

1. Configuration Initiale

1. Créer un répertoire pour le projet :

```
mkdir projet-meteo
```

```
cd projet-meteo
```

```
PS C:\Users\joan.perrey-prato\Documents\cours_nodejs\projetMeteo>
```

2. Initialiser un projet Node.js (créer un fichier package.json) :

```
npm init
```

3. Installer les dépendances nécessaires :

```
npm install dotenv mysql2 axios yargs winston moment-timezone
```

4. Créer l'arborescence du projet :

```
▼ cours_nodejs
  ▼ projetMeteo
    ▼ logs
      ≡ app.log
    > node_modules
    ⚙ .env
    $ .env.local
    ⚙ .gitignore
    JS logger.js
    JS meteoBdd.js
    JS meteoCommande.js
    {} package-lock.json
    {} package.json
```

2. Configuration Variables d'Environnements

1) Créer le fichier .env qui sera avec des valeurs par défaut :

```
cours_nodejs > projetMeteo > .env
1  # Database Configuration
2
3  # DB_HOST=ChangeMe!
4  # DB_USER=ChangeMe!
5  # DB_PASSWORD=ChangeMe!
6  # DB_NAME=ChangeMe!
7
8  # API Configuration
9
10 # API_TOKEN=ChangeMe!
11 # API_URL_FORECAST=https://api.meteo-concept.com/api/forecast/daily
12 # API_URL_CITIES=https://api.meteo-concept.com/api/location/cities
```

2) Créer le fichier .env.local qui sera avec les valeurs adaptées à votre environnement :

```
cours_nodejs > projetMeteo > .env.local
1  # Database Configuration
2  DB_HOST=localhost
3  DB_USER=root
4  DB_PASSWORD=
5  DB_NAME=stage_esi_joan_meteo
6
7  # API Configuration
8  API_TOKEN=8b23441798d4a66f4abff12039dc75c0cadd9c5e11df9e4e954252e945dbb832
9  API_URL_FORECAST=https://api.meteo-concept.com/api/forecast/daily
10 API_URL_CITIES=https://api.meteo-concept.com/api/location/cities
11
```

3) Ajouter au fichier .gitignore (ne fera pas de commit avec les fichiers suivant) :

```
cours_nodejs > projetMeteo > .gitignore
1  .env.local
2  logs/
```

B. Scripts de Configuration

Fichier meteoBdd.js, script qui crée la base de données et la table MySQL :

```
cours_nodejs > projetMeteo > JS meteoBdd.js > ...
1  const mysql = require('mysql2');
2
3  // Configuration de la connexion
4  const connection = mysql.createConnection({
5    host: 'localhost',
6    user: 'root',
7    password: '',
8    multipleStatements: true,
9  });
10
11 // Création de la base de données et de la table
12 const dbName = 'stage_esi_joan_meteo';
13 const createDatabaseAndTable = `
14   CREATE DATABASE IF NOT EXISTS ${dbName};
15   USE ${dbName};
16
17   CREATE TABLE IF NOT EXISTS ProjetMeteoApi (
18     id INT AUTO_INCREMENT PRIMARY KEY,
19     date DATE NOT NULL,
20     temperature FLOAT NOT NULL,
21     lieu VARCHAR(100) NOT NULL,
22     codeP INT NOT NULL,
23     api VARCHAR(100) NOT NULL,
24     UNIQUE(date, lieu)
25   );
26 `;
27
28 connection.connect((err) => {
29   if (err) {
30     console.error('Erreur de connexion :', err);
31     return;
32   }
33   console.log('Connexion réussie à MySQL');
34
35   connection.query(createDatabaseAndTable, (err) => {
36     if (err) {
37       console.error('Erreur lors de la création de la base de données ou des tables :', err);
38       return;
39     }
40     console.log('Base de données et table créées avec succès');
41     connection.end();
42   });
43 });
```

C. Gestion des Logs

Fichier logger.js , écrit dans les logs :

```
cours_nodejs > projetMeteo > JS logger.js > logger > transports > filename
1  const { createLogger, transports, format } = require('winston');
2  const moment = require('moment-timezone');
3
4  // Formatter personnalisé pour inclure le fuseau horaire
5  const customTimestamp = format((info) => {
6    info.timestamp = moment().tz('Europe/Paris').format('YYYY-MM-DD HH:mm:ss');
7    return info;
8  });
9
10 // Configuration du logger
11 const logger = createLogger({
12   level: 'info',
13   format: format.combine(
14     customTimestamp(),
15     format.printf(({ timestamp, level, message }) => {
16       return `${timestamp} [${level.toUpperCase()}] ${message}`;
17     })
18   ),
19   transports: [
20     new transports.Console(),
21     new transports.File({ filename: 'logs/app.log', level: 'info' }), // Écrit les logs dans app.log
22   ],
23 });
24
25 module.exports = logger;
```

D. Script Principal : meteoCommande.js

```

projetMeteo > JS meteoCommande.js > ...
1  require('dotenv').config(); // on charge le .env
2  require('dotenv').config({ path: '.env.local' }); // et le .env.local
3  const mysql = require('mysql2/promise');
4  const axios = require('axios');
5  const yargs = require('yargs');
6  const logger = require('./logger');
7
8  // Configuration de la connexion MySQL
9  const dbConfig = {
10   host: process.env.DB_HOST,
11   user: process.env.DB_USER,
12   password: process.env.DB_PASSWORD,
13   database: process.env.DB_NAME,
14 };
15
16 // Token et URL de l'API Meteo Concept
17 const API_TOKEN = process.env.API_TOKEN;
18 const API_URL_FORECAST = process.env.API_URL_FORECAST;
19 const API_URL_CITIES = process.env.API_URL_CITIES;
20
21 // Fonction pour calculer la différence en jours
22 const getDayDifference = (targetDate) => {
23   const today = new Date();
24   const target = new Date(targetDate);
25   const diffTime = target.getTime() - today.setHours(0, 0, 0, 0);
26   return Math.round(diffTime / (1000 * 60 * 60 * 24));
27 };
28
29 (async () => {
30   const argv = yargs
31     .option('date', { alias: 'd', type: 'string', demandOption: true })
32     .option('lieu', { alias: 'l', type: 'string', demandOption: true })
33     .option('codePostal', { alias: 'c', type: 'int', demandOption: true })
34     .argv;
35
36   const { date, lieu, codePostal } = argv;
37
38   // Validation du code postal
39   if (!/^\d{5}$/.test(codePostal.toString())) {
40     logger.error('Le code postal doit être composé de 5 chiffres.');

```

```
72
73   const dayDifference = getDayDifference(date);
74   if (dayDifference < 0 || dayDifference > 13) {
75     logger.error("Les prévisions sont disponibles uniquement pour les 14 jours suivants incluant aujourd'hui.");
76     await connection.end();
77     return;
78   }
79
80   const forecastResponse = await axios.get(API_URL_FORECAST, {
81     params: { token: API_TOKEN, insee: city.insee },
82   });
83
84   const forecast = forecastResponse.data.forecast.find((f) => f.day === dayDifference);
85   if (!forecast) {
86     logger.error('Données introuvables pour cette date. ');
87     await connection.end();
88     return;
89   }
90
91   const avgTemp = (forecast.tmin + forecast.tmax) / 2;
92   const codePostalApi = forecast.cp;
93
94   await connection.query(
95     'INSERT INTO projetmeteoapi (date, temperature, lieu, codeP, api) VALUES (?, ?, ?, ?, ?)',
96     [date, avgTemp, lieu, codePostalApi, 'Meteo Concept']
97   );
98
99   logger.info(`Température moyenne pour ${lieu} (${codePostalApi}) le ${date} : ${avgTemp}°C (API)`);
100   await connection.end();
101 } catch (error) {
102   logger.error(`Erreur : ${error.message}`);
103 }
104 }());
```

E. Insertion data dans ELK (v5.5.3) dans le projetmeteo

1. Installation ElasticSearch en package (v13.3.1)

npm install elasticsearch@13.3.1

2. Mise a jour du script principale meteoCommande.js

1 – Ajout package :

```
7  const { Client } = require('elasticsearch'); // Client Elasticsearch // MAJ2
8  process.noDeprecation = true; // MAJ2
```

2 – Config ELK :

```
19  // Configuration du client Elasticsearch // MAJ2
20  const esClient = new Client({
21    host: process.env.ES_NODE,
22    log: 'error',
23  });
```

3 – Insertion dans ELK :

```
113  // Insertion dans Elasticsearch // MAJ2
114  const esDocument = {
115    date,
116    temperature: avgTemp,
117    lieu,
118    codeP: codePostalApi,
119    api: 'Meteo Concept',
120  };
```

4 – Connexion a l'index et au document et gestion des erreurs :

```
122  try {
123    // MAJ2
124    await esClient.index({
125      index: 'projetmeteoapi',
126      type: 'forecast', // Type nécessaire pour Elasticsearch 5.5.3
127      body: esDocument,
128    });
129    // MAJ2
130    logger.info(`Données insérées dans Elasticsearch : ${JSON.stringify(esDocument)}`);
131  } catch (error) {
132    logger.error(`Erreur lors de l'insertion dans Elasticsearch : ${error.message}`);
133  }
134
135  await connection.end();
136  } catch (error) {
137    logger.error(`Erreur : ${error.message}`);
138  }
139  })();
```

✚ Remarque : A la ligne 126 le type est précisé. Mais depuis la v7.x.x d'ELK, on doit plus préciser ce type, mais seulement l'index et le document.

II. projetmeteov2 -> ELK avec authentication (v8.17.1) avec NodeJS (v22.13.0)

A. Prérequis

1. Dans une VM linux,

- 1.1 Mise en place de Docker et des conteneurs ELK
- 1.2 Configuration du réseau et de l'accès

2. Développement du projetmeteov2

- **2.1** Configuration initiale : voir -> [#I. 1. A. Configuration Initiale](#)
Pour les dépendances (3.), nous pourrions installer uniquement :
`npm install elasticsearch axios dotenv moment-timezone winston yargs`
- **2.2** Configuration de l'environnement : [#I. 1. B. Configuration Variables d'Environnements](#)

Pour le .env et .env.local remplacer par ceci :

```
cours_nodejs > projetMeteov2 > .env
1  # # API Meteo Concept
2  # API_TOKEN=VOTRE_TOKEN
3  # API_URL_FORECAST=https://api.meteo-concept.com/api/forecast/daily
4  # API_URL_CITIES=https://api.meteo-concept.com/api/location/cities
5
6  # # Elasticsearch
7  # ES_NODE=https://IP_VM:9200
```

(pour mieux sécurisé il faudra mettre les identifiants ELK dans .env)

3. Ajouter la gestion des logs

- Copier la gestion des logs : [#I. 3. Gestion des Logs](#)

B. Fichier Test de la connexion à ELK

1. Créé un fichier de test pour vérifier si la connexion à Elasticsearch fonctionne sans trop de complexité en désactivant la vérification TLS globalement :

```
cours_nodejs > projetMeteov2 > JS testElasticjs > ...
1 process.env.NODE_TLS_REJECT_UNAUTHORIZED = '0'; // Désactiver la vérification des certificats TLS
2 require('dotenv').config();
3 require('dotenv').config({ path: '.env.local' });
4
5 const { Client } = require('@elastic/elasticsearch');
6
7
8 const esClient = new Client({
9   node: process.env.ES_NODE,
10  auth: {
11    username: 'elastic',
12    password: 'nb3tzuxdVwovVpXRaDjBa',
13  },
14 });
15
16
17 (async () => {
18   try {
19
20     const response = await esClient.info();
21     console.log('Elasticsearch est accessible :', response);
22   } catch (error) {
23     console.error('Erreur de connexion à Elasticsearch :', error);
24   }
25 })();
```

C. Script projetmeteov2

```

cours_nodejs > projetMeteov2 > # meteoCommandeV2.js > ...
1 process.env.NODE_TLS_REJECT_UNAUTHORIZED = '0'; // Désactiver la vérification des certificats SSL globalement
2 process.removeAllListeners('warning'); // pour supprimer le warning inutile de désactivation de certificat
3
4 require('dotenv').config();
5 require('dotenv').config({ path: '.env.local' });
6 const axios = require('axios');
7 const yargs = require('yargs');
8 const { Client } = require('@elastic/elasticsearch');
9 const logger = require('./loggerV2');
10
11 // Configuration du client Elasticsearch
12 const esClient = new Client({
13   node: process.env.ES_NODE,
14   auth: { username: 'elastic', password: 'nb3tzuxdVmovVpXRaDjB' },
15   log: 'error',
16 });
17
18 // Token et URL de l'API Meteo Concept
19 const API_TOKEN = process.env.API_TOKEN;
20 const API_URL_FORECAST = process.env.API_URL_FORECAST;
21 const API_URL_CITIES = process.env.API_URL_CITIES;
22
23 // Fonction pour calculer la différence en jours
24 const getDayDifference = (targetDate) => {
25   const today = new Date();
26   const target = new Date(targetDate);
27   return Math.round((target.getTime() - today.setHours(0, 0, 0, 0)) / (1000 * 60 * 60 * 24));
28 };
29
30 // Fonction pour vérifier si la donnée existe déjà dans Elasticsearch
31 const checkIfDataExists = async (date, lieu, codePostal) => {
32   const { hits } = await esClient.search({
33     index: 'projetmeteov2',
34     body: {
35       query: {
36         bool: {
37           must: [
38             { term: { date } },
39             { match: { lieu } },
40             { term: { codePostal } },
41           ],
42         },
43       },
44     },
45   });
46   return hits.total.value > 0 ? hits.hits[0]._source : null;
47 };
48
49

```

```

50 (async () => {
51   const argv = yargs
52     .option('date', { alias: 'd', type: 'string', demandOption: true })
53     .option('lieu', { alias: 'l', type: 'string', demandOption: true })
54     .option('codePostal', { alias: 'c', type: 'int', demandOption: true })
55     .argv;
56
57   const { date, lieu, codePostal } = argv;
58
59   // Vérification si la donnée existe déjà
60   const existingData = await checkIfDataExists(date, lieu, codePostal);
61   if (existingData) {
62     logger.info('🔥 Données déjà existantes dans ELK : ${JSON.stringify(existingData)}');
63     return;
64   }
65
66   try {
67     // Recherche de la ville via l'API Meteo Concept
68     const locationResponse = await axios.get(API_URL_CITIES, {
69       params: { token: API_TOKEN, search: `${lieu} ${codePostal}` },
70     });
71
72     const matchingCities = locationResponse.data.cities.filter(
73       (city) => city.name.toLowerCase() === lieu.toLowerCase() && city.cp === codePostal
74     );
75
76     if (matchingCities.length === 0) {
77       logger.error('❌ Aucune correspondance entre la ville et le code postal.');

```

```
89
90 // Récupération des prévisions météo
91 const forecastResponse = await axios.get(API_URL_FORECAST, {
92   params: { token: API_TOKEN, insee: city.insee },
93 });
94
95 const forecast = forecastResponse.data.forecast.find((f) => f.day === dayDifference);
96 if (!forecast) {
97   logger.error('❌ Données introuvables pour cette date.');
```

return;

```
99 }
100
101 const avgTemp = (forecast.tmin + forecast.tmax) / 2;
102
103 // Insertion dans Elasticsearch
104 const esDocument = { date, temperature: avgTemp, lieu, codePostal, api: 'Meteo Concept' };
105
106 try {
107   await esClient.index({ index: 'projetmeteoapiv2', document: esDocument });
108   logger.info('✅ Données insérées dans Elasticsearch : ${JSON.stringify(esDocument)}');
```

} catch (error) {

```
109   logger.error('❌ Erreur lors de l'insertion dans Elasticsearch : ${error.message}');
```

}

```
112 } catch (error) {
113   logger.error('❌ Erreur : ${error.message}');
```

}

```
114 }
115 })();
116
```

Explication générale du projetmeteo2 :

- 1 - Désactivation de la vérification SSL → Lignes 1-2
 - 2 - Chargement des variables d'environnement (dotenv) → Lignes 4-5
 - 3 - Importation des modules nécessaires (dépendance, package) → Lignes 6-9
 - 4 - Connexion à Elasticsearch → Lignes 12-16
 - 5 - Définition des URLs de l'API (token, api cities et forecast) → Lignes 19-21
 - 6 - Fonction pour calculer la différence en jours → Lignes 24-28
 - 7 - Vérification si la donnée existe dans Elasticsearch → Lignes 31-48
 - 8 - Récupération des arguments en ligne de commande → Lignes 51-57
 - 9 - Retour log si la donnée est déjà présente dans Elasticsearch → Lignes 60-64
 - 10 - Recherche de la ville via l'API Météo Concept → Lignes 68-81
 - 11 - Validation de la date demandée → Lignes 84-88
 - 12 - Récupération des prévisions météo via l'API → Lignes 91-99
 - 13 - Calcul de la température moyenne → Ligne 101
 - 14 - Insertion des données dans Elasticsearch → Lignes 104-107
- Pour la création de l'index (ligne 107), ne pas mettre de majuscule.*
- 15 - Gestion des logs et erreurs globales → Lignes 108-115

III. Explication d'un index dans ELK

A. Création automatique de l'index

Insertion d'un document avec `esClient.index()`, Elasticsearch :

- ✓ Vérifie si l'index (`projetmeteoapiv2`) existe.
- ✓ **S'il n'existe pas**, il le crée automatiquement.
- ✓ Insère le document dans cet index.

Exemple de l'insertion dans le code (ligne 107) :

```
107      await esClient.index({ index: 'projetmeteoapiv2', document: esDocument });
```

Ici, si **projetmeteoapiv2** n'existe pas, Elasticsearch le crée tout seul avant d'insérer `esDocument`.

B. Détection automatique des types de propriétés

Quand Elasticsearch crée un index **automatiquement**, il **devine** le type des champs du premier document qu'il reçoit (**mapping dynamique**).

Exemple avec le document inséré :

```
node meteoCommandeV2 -d 2025-02-08 -l Montels -c 30360
```

 Elasticsearch va interpréter les types ainsi :

Voir différence entre les type d'entier (long ,integer,...) : [Leçon types entiers](#)

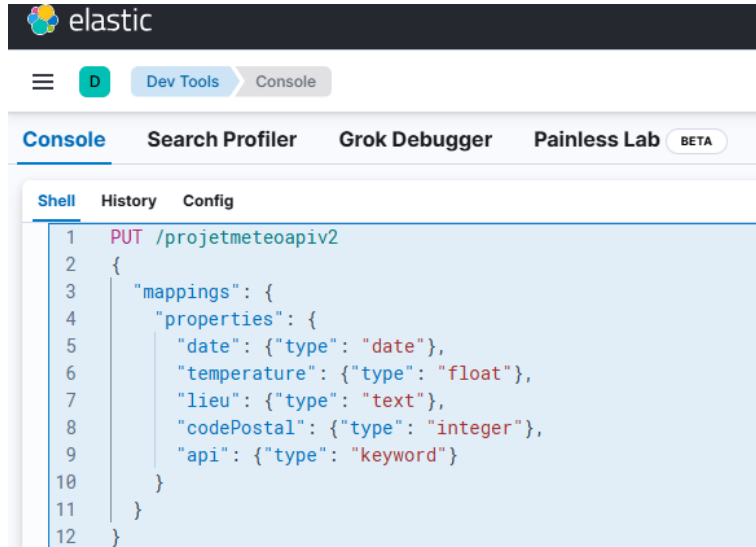
Propriétés (ou champs)	Type détecté (ou mapping)
date	date si bien formaté (sinon text)
temperature	long
lieu	text
codePostal	long
api	text

Problème potentiel :

- **date** risque d'être détecté comme text au lieu de date, ce qui peut poser problème pour les recherches par date.
- Pour éviter ce souci, il faut **prédéfinir** l'index avec un **mapping**.

C. Forcer un mapping personnalisé (optionnel)

Pour que **Elasticsearch ne choisisse pas les types automatiquement**, il faut créer l'index avec un **mapping précis** AVANT d'insérer des données (soit par nodejs soit directement dans Kibana), exemple avec ELK :



```

1 PUT /projetmeteoapiv2
2 {
3   "mappings": {
4     "properties": {
5       "date": {"type": "date"},
6       "temperature": {"type": "float"},
7       "lieu": {"type": "text"},
8       "codePostal": {"type": "integer"},
9       "api": {"type": "keyword"}
10    }
11  }
12 }

```

D. Vérifier le mapping d'un index dans ELK

Pour consulter les types d'un index :	1 GET projetmeteoapiv2/
Extrait d'affichage :	<pre> { "projetmeteoapiv2": { "aliases": {}, "mappings": { "properties": { "api": { "type": "text", "fields": { "keyword": { "type": "keyword", "ignore_above": 256 } } }, "codePostal": { "type": "long" }, "date": { "type": "date" }, "lieu": { "type": "text", "fields": { "keyword": { "type": "keyword", "ignore_above": 256 } } }, "temperature": { "type": "long" } } } } } </pre>

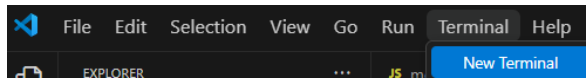
E. Rappel des termes employés

Voici comment on pourrait organiser les termes :

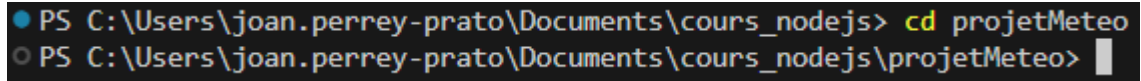
- **index** : projetmeteoapiv2
- **document** : chaque enregistrement JSON avec des propriétés comme date, temperature, etc.
- **propriétés** (ou champs) : date, temperature, lieu, codePostal, api
- **mapping** : description de chaque propriété et de son type

IV. Exécution du Projet

1. Ouvrir le Terminal de VS Code ou Ctrl+shift+ù, puis choisir le dossier du projet :



2. Bien vérifier a être dans le bon dossier où est le projet



3. Créer la base de données et la table :

`node meteobdd`

4. Récupérer la température d'un lieu pour une date donnée :

`node meteoCommande -d 2025-02-01 -l Montels -c 30360`

- 5a. Affichage des résultats grâce à la commande ci-dessus :

- Résultat n°1 :

```
2025-01-22 14:17:04 [INFO] Température moyenne pour Montels (30360) le 2025-02-01 : 5.5°C (API)
```

A la 1ere tentative, il va chercher les infos à l'api puis il stocke dans la bdd.

- Résultat n°2 :

```
2025-01-22 14:21:19 [INFO] Température moyenne pour Montels (30360) le 2025-02-01 : 5.5°C (BDD)
```

A la 2eme identique tentative, il va chercher les mêmes infos à la bdd.

- Vérifier les logs : Les logs seront écrits dans la console et dans le fichier `logs/app.log`.

OU

- 5b. Affichage des résultats grâce à la commande ci-dessus :

- ```
2025-01-28 16:59:16 [INFO] Données insérées dans Elasticsearch : {"date":"2025-02-04","temperature":4,"lieu":"Monte
```

Il va chercher dans l'api et insère dans ELK !

## **Ressources :**

**Listes des bibliothèques nécessaires ( mettre à jour le fichier package.json )**

npm install dotenv mysql2 axios yargs winston moment-timezone

Ressources :

**dotenv** : charge les variables d'environnement à partir des fichiers .env et .env.local.

**mysql2** : permet de se connecter et d'interagir avec une base de données MySQL.

**axios** : pour effectuer des requêtes HTTP à l'API météo.

**yargs** : facilite la gestion des arguments en ligne de commande.

**winston** : gestionnaire avancé de logs.

**moment-timezone** : gère les dates et fuseaux horaires (utilisé dans les logs).