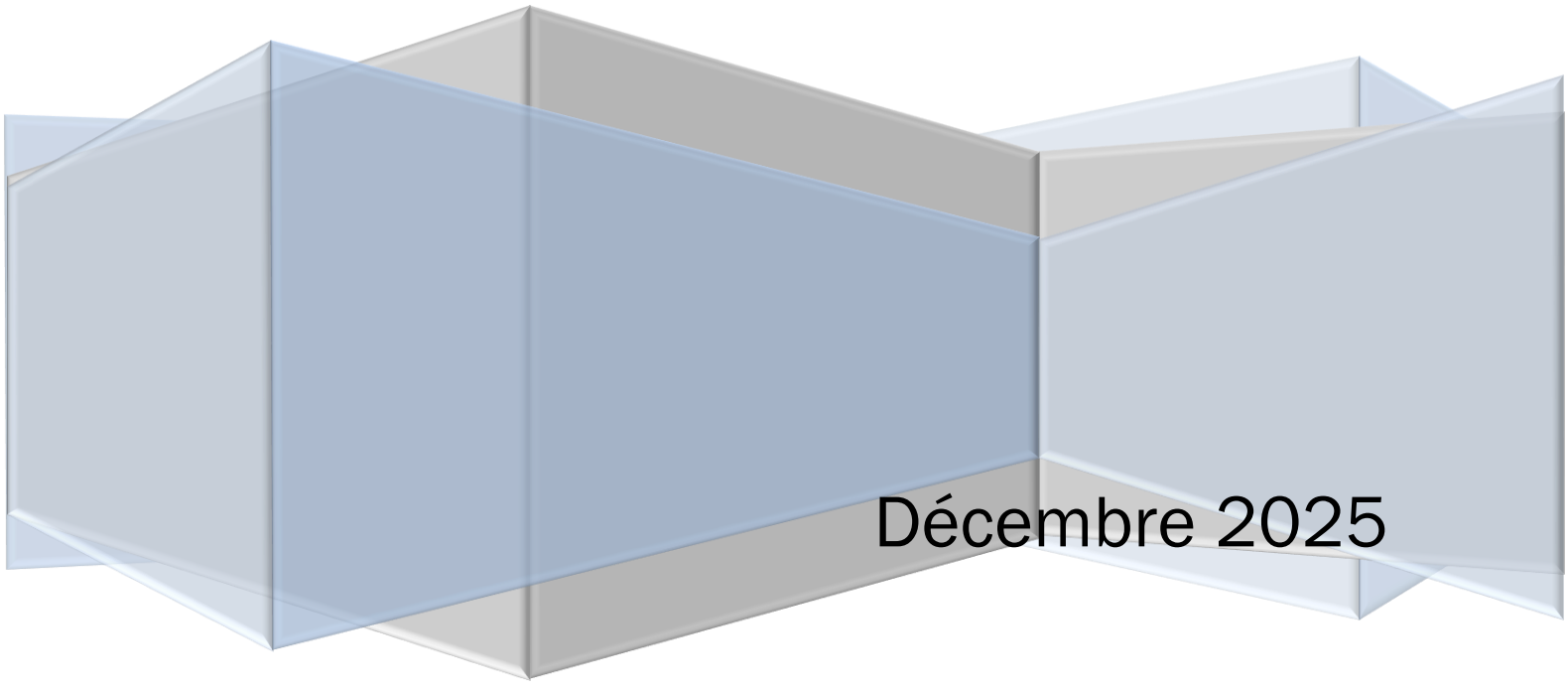
M2i Formation

Dossier Professionnel

Concepteur Développeur d'Applications

Ruben Trottein



Décembre 2025

Table des matières

Contenu

1. Introduction

Présentez-vous : parcours, expériences, motivations pour devenir CDA.

06 59 18 51 56
ruben.trottein@gmail.com
Bobigny 93000
Permis B, Véhiculé
mistertea.fr
linkedin.com/in/ruben-trottein/

COMPÉTENCES

- Conception de Site (Merise, MCD)
- BP dev (W3C, SEO, OWASP)
- Responsive (Mobile-first, app mobile)
- Agilité (SCRUM, Kanban, xp)
- Test Driven Developpement (CI/CD)

QUALIFICATIONS

Autres formations

- POEI Java Activ Consult'Ing (nv Bac +3)
- IFOCOP Formation DIW (Développeur Intégrateur Web RNCPII)

Certifications

- UiPath(rpa)
- OpenClassrooms HTML CSS JS

TECHNOLOGIES

Main Stack

- HTML, CSS, JavaScript, Php, Java
- Symfony, Angular, Spring,
- node.js, mongoDB, SQL

Autres technologies

- C#, Wordpress, React, ASP .net, UiPath(rpa)

Bureautique

- MS Office, Adobe

LOISIRS

MUSIQUE

CEM (C3) Piano, claviers, Orgue Jazz
Conduite de projet, sound design
Interprétation, composition

AUTRES

Natation, VTT, Basket,
Cinéma de Genre, Jeux vidéos
Mythologie, Littérature, Philosophie

LANGUES

Anglais courant
Italien conversationnel

Ruben TROTTEIN

DEVELOPPEUR JUNIOR JS / SYMFONY

Disponible dès le 13/12/2024

Après 6 ans de freelance en formation et développement Je souhaite trouver un poste à la sortie de la formation Concepteur Développeur d'Application que je termine en Décembre 2024

EXPÉRIENCES PROFESSIONNELLES

DÉVELOPPEUR SYMFONY
BIYN learning
2024

- Développement d'un LMS (Learning Management Software)
- Elaboration d'un environnement de test
- Mise en place du back-end et conception du modèle de données
- Développement de l'environnement utilsateurs

FORMATEUR EN DÉVELOPPEMENT FRONT-END
Auto-Entreprenariat
2020 - 2023

- Présentation et animation de cours et conférences
- Préparation de cours et supports
- Evaluation et correction du travail des apprenants
- Elaboration de programmes et exercices
- Développement d'environnements et applications d'entraînement

INGENIEUR EN DEVELOPPEMENT JAVA - .NET
POEI Chez Activ'consult'Ing puis poste chez CGI, La défense
2019 - 2020

- Formation intensive Web et JAVA /JEE
- Application de référentiel du personnel de l'entreprise
- Montée en compétence en RPA(UiPath) et .net (c#)
- Développement d'un Chatbot .net en équipe avec les sections IA et Data Analysis

DEVELOPPEUR WORDPRESS - WOOCOMMERCE
Stage de fin d'études Développeur Web
2018-2019

PROJETS

- Site Pro Domiciliation
- Site Formation
- Jeux et Exercices Pédagogique
- Sites E-Commerce
- Sites Vitrine
- Présentations PPT, JS

FORMATIONS

TITRE RNCP NIV6 CDA - M2I FORMATION
Concepteur Développeur d'applications
2024, en cours

M2 COMC - PARIS VIII
Organisation & Change Management
2011

Dernière version du CV ⬆

2. Description des projets réalisés

Chaque projet doit être décrit en détail. Incluez :

Titre et description du projet :
Nature du projet (application web, application mobile, etc.).
Contexte (professionnel, formation, stage, freelance, etc.).
Objectif (résoudre un problème spécifique, répondre à un besoin client).

Analyse préalable :
Expression des besoins du client ou de l'utilisateur.
Cahier des charges ou spécifications fonctionnelles.

Conception et organisation :
Diagrammes UML (use case, classes, séquence).
Modèle conceptuel de données (MCD).
Architecture choisie (MVC, microservices, etc.).
Outils ou méthodologies utilisés (Agile, Scrum, etc.).

Réalisation :
Technologies utilisées (langages, frameworks, bases de données).
Description des fonctionnalités développées.
Extraits de code significatifs (commentés et expliqués).
Gestion des versions (Git, GitHub/GitLab).

Problèmes rencontrés et solutions :
Décrivez les obstacles techniques ou organisationnels et comment vous les avez surmontés.

Tests et validation :
Types de tests réalisés (unitaires, fonctionnels, utilisateurs).
Résultats des tests.

Livrables :
Présentez ce qui a été livré (code source, documentation, déploiement).
Page notion :
https://mistertea.notion.site/Index-des-projets-CDA-65e4bd8d7cec45178ff38c3cbcfcc07e

A. Projets CDA

APIs

Node.js noSql MongoDB (API / NoSql);

- API Zoologique d'identification d'Ours : SQL → CRUD;
- Movies / Netflix : SQL → CRUD;
- Foodtrucks : Node.js noSql MongoDB → CRUD ;

- Users : Node.js Sql + MongoDB ⟶ CRUD ;
- API Utilisateurs Securisée: Node.js noSql MongoDB ⟶ CRUD + JWT;

Applications

Symfony

- PartageToo : Site de troc/commerce style LeBonCoin
- Actunews : Blog d'actualités
- ForumNews : Réseau social de réaction / commentaire sur des articles

Php

- Surf : Blog en Php procédural, (Sql, Bootstrap)
- Marmitard : Blog culinaire Php procédural orienté Objet

Java Spring (Web):

- PayMe : Site de paiements style Lydia (Java, Spring, Thymeleaf, Sql)
- PayMe Hacking BackEnd : Site de paiements avec backdoor pirate (Java, Spring, Thymeleaf, Sql)
- Festival version Web : Site de gestion evenementielle. Ajout de groupes avec leurs scène a un ou plusieurs festivals à venir. Version Thymeleaf / Spring / SQL
- FirstApp : application avec rendu sans templating (Spring uniquement)

React

- Pokedex-React : React+ interrogation API

NodeJS / Symfony (BDDs Relationnelles) :

- festival (Java + Sql)
- blog (Sql via MCD)
- Actunews (Sql via Doctrine)

JavaScript (Full Front-end)

- Histoire interactive (HTML/CSS)
- Intégrations psd to html : recipe-page, Vitrines (Assurance / Boutique)
    - SunnySide (Défi FrontEnd Mentor)

      https://sunnyside-challenge-chi.vercel.app/

    - W-Challenge (FrontEnd Mentor)

      https://rubentrottein.github.io/w-challenge/

- To-Do List Full JS
- Quiz Full JS

Java (Console)

- Calculatrice
- Zelda
- Garage
- Festival version console
- Chifumi
- Potager
- Médiathèque
- Appli Notes

Kotlin (Mobile)

- Calculatrice
- To-Do list (Array)
- Simon
- UserList (ArrayList)

Ionic (Mobile)

- Liste Triable des directives Ionic
- Tableau Front-end de liste d'utilisateurs
- To-do list étendue avec API de sauvegarde
- Application de gestion utilisateurs (Ionic-storage / Local storage)

API

FoodTruck.js :

API Restauration rapide: MongoDB→ CRUD :

Nature du projet *: API RESTful*
Contexte *: Formation*
Objectif *: Gérer une liste de FoodTrucks avec leur menu consultable*
Analyse préalable *:*

On cherche à avoir de quoi identifier chaque Foodtruck et son menu , on les renvoie en JSON, et on peut créer de nouvelles fiches ou les modifier.

Cahier des charges ou spécifications fonctionnelles *:*

Création d'une application node.js en environnement de développement, récupération des informations avec postman.

Conception et organisation :

Diagrammes UML (use case, classes, séquence).

Modèle conceptuel de données (MCD)

.

Architecture choisie *: MVC avec le rôle de la vue simplifié (retour JSON)*
Outils ou méthodologies utilisés : Travaux dirigés

Réalisation :
Technologies utilisées : node.js, express.js, SQL, Postman
Description des fonctionnalités développées : CRUD simple

Extraits de code significatifs (commentés et expliqués).
Modèle :

Notre modèle contient un array pour les menu. Il est au format des schémas Mongoose

```
const mongoose = require('mongoose');
const Schema = mongoose.Schema;

const FoodTruckSchema = new Schema({
    name: {
        type: String,
        required: true
    },
    type: {
        type: String,
        required: true
    },
    menu: [{
        dish: {
            type: String,
            required: true
        },
        drink: {
            type: String,
            required: true
        },
        price: {
            type: String,
            required: true
        }
    }]
})

const FoodTruck = mongoose.model('FoodTruck', FoodTruckSchema)

module.exports = FoodTruck;
```

Routes :

Notre routeur express nous permet une gestion RESTful des endpoint et gère l'accès aux données

```
const express = require("express");
const app = express();
const port = 3444;
const mongoose = require("mongoose");
main().catch(err => console.log(err));

Qodo Gen: Options | Test this function
async function main(){
    await mongoose.connect('mongodb://localhost:27017/FoodTruck_db');
    console.log("[DATABASE] MongoDb --FoodTrucks-- is connected");
}

app.use(express.json());
app.use(express.urlencoded({extended:false}))

app.get("/", (req, res) => {
    res.send("Bienvenue dans FoodTruckAPI 1.0");
})

app.use('/api/foodTruck', require('./routes/foodTruckRoute'));

app.listen(port, ()=> console.log(`[Server] is running on Port : ${port}`));
```

Gestion des versions : Git

Problèmes rencontrés et solutions :

Abstraction souhaitée sur l'application afin de récupérer la structure et la réutiliser.
Apprentissage de node et express
Pas de déploiement

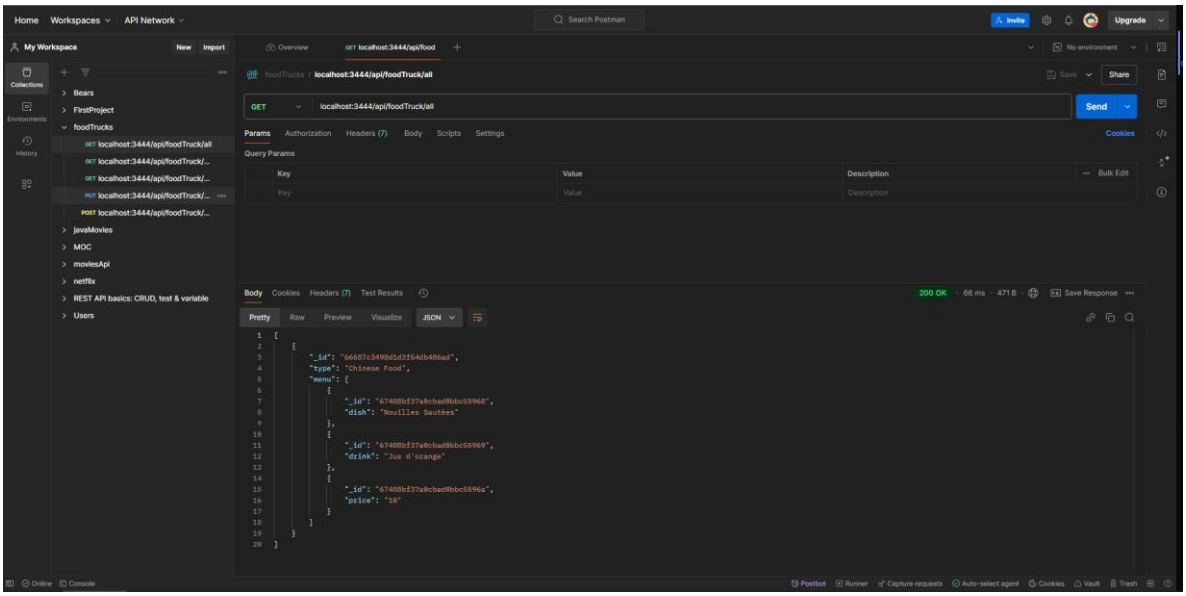
Tests et validation :

Types de tests réalisés ; unitaires
Résultats des tests : Vérification du fonctionnement de l'API et du retour JSON

Livrables :

Vue :

Retour JSON du GET All sous Postman



Bears.js :

API Zoologique d'identification d'Ours : SQL → CRUD :

Nature du projet *: API RESTful*
Contexte *: Formation*
Objectif *: Repertorier les espèces d'Ours et leurs caractéristiques*
Analyse préalable *:*

On cherche a avoir quelques informations pertinentes pour chacun de nos Ours, on les affiche, et on peut créer de nouvelles fiches ou les modifier.

Cahier des charges ou spécifications fonctionnelles *:*

Création d'une application node.js en environnement de développement, récupération des information avec js fetch ou postman.

Conception et organisation :

Diagrammes UML (use case, classes, séquence).

Modèle conceptuel de données (MCD).
Architecture choisie *(MVC, microservices, etc.). : MVC avec le rôle de la vue simplifié (retour JSON)*
Outils ou méthodologies utilisés : Travaux dirigés

Réalisation :
Technologies utilisées : node.js, express.js, SQL, Postman, HTML, CSS, vanilla JS
Description des fonctionnalités développées : CRUD simple et affichage navigateur

Extraits de code significatifs (commentés et expliqués).
Contrôleur :

Dans nos route on définit les endpoint du crud et on gère les erreurs. Ces endpoint sont utilisables directement via PostMan

```
bearRouter.get('/all', async (req, res) =>{
    try {
        const bears = await Bear.find();
        res.json(bears);
    } catch (error) {
        res.json({message: 'Error fetching Bears', error});
    }
})
bearRouter.post('/new',async (req, res) =>{
    const newBear = new Bear(req.body);
    try {
        const saveBear= await newBear.save()
    } catch (error) {
        res.json({message:'Error in BEARing', error});
    }
    res.json({message:'New Bear Created \n', newBear});
})

bearRouter.get('/:id/show',async (req, res) =>{
    try {
        const bearToFind = await Bear.findOne({_id: req.params.id});
        res.json({message:'Found Bear : ', bear: bearToFind});
    } catch (error) {
        res.json({message:'Bear not found', error});
    }
})

bearRouter.put('/:id/edit',async (req, res) =>{
    try{
        const bearToUpdate = await Bear.findOneAndUpdate({_id: req.params.id }, req.body,
        {
            new: true,
            runValidators: true,
        })
        res.json({ message : 'Bear is now Updated', bear: bearToUpdate } );
    } catch (error){
        res.json({ message : 'No Update to the Bear ', error } );
    }
})
```

Routes :

Notre routeur express nous permet une gestion RESTful des endpoint et gère l'accès aux données

```
const express = require("express");
const app = express();
const port = 3456;
const mongoose = require("mongoose");

const cors = require('cors');

// Use CORS middleware
app.use(cors());


main().catch(err => console.log(err));
Qodo Gen: Options | Test this function
async function main(){
    await mongoose.connect('mongodb://localhost:27017/Ours_db');
    console.log("[DATABASE] MongoDb --Bears-- is connected");
}

app.use(express.json());
app.use(express.urlencoded({extended:false}))

app.get("/", (req, res) => {
    res.send("Bienvenue dans l'API des Ours");
})

app.use('/api/bears', require('./routes/bearRoute'));

app.listen(port, ()=> console.log(`[Server] is running on Port : ${port}`));
```

Gestion des versions : Git

Problèmes rencontrés et solutions :

Abstraction souhaitée sur l'application afin de récupérer la structure et la réutiliser.
Apprentissage de node et express
Pas de déploiement

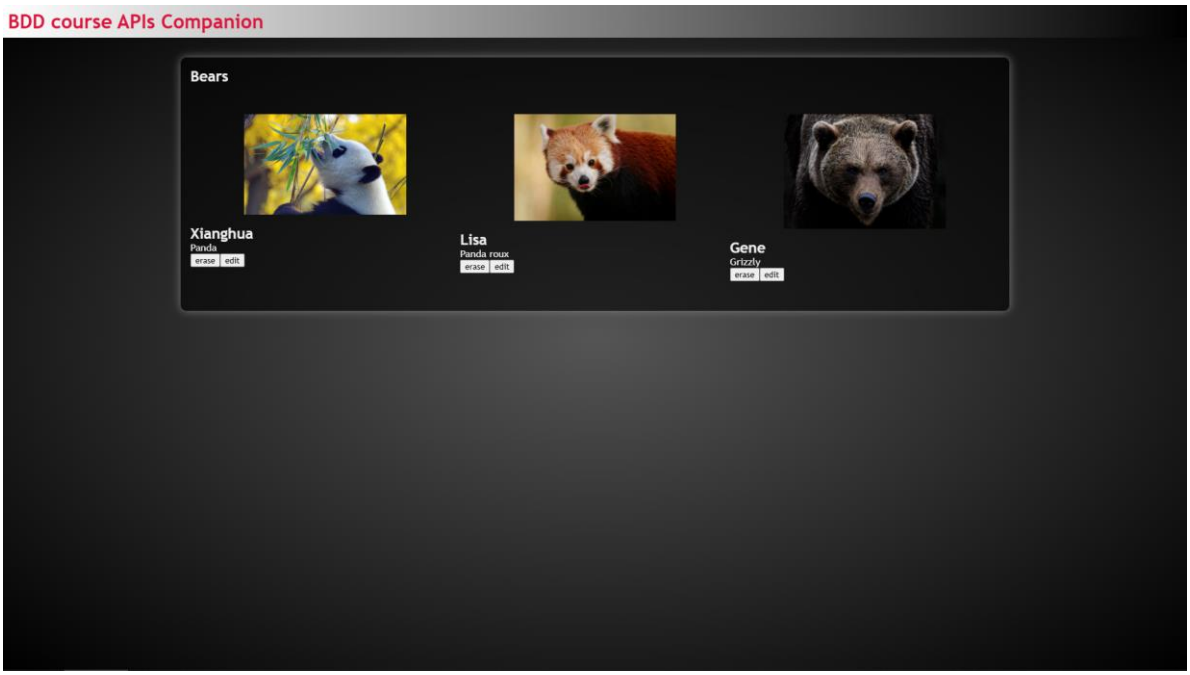
Tests et validation :

Types de tests réalisés ; unitaires
Résultats des tests : Vérification du fonctionnement de l'API et du retour JSON

Livrables :

Vue

Petite page web simple avec un fetch javascript pour afficher les Ours et les supprimer. Formulaire d'édition



MoviesApi

API repertoire de films : MongoDB → Application Web:

Nature du projet *: Application web Vanilla JS*
Contexte *: Formation*
Objectif *: Répertorier des films et leurs caractéristiques*
Analyse préalable *:*

On cherche à avoir quelques informations pertinentes pour chacun de nos films, on les affiche, et on peut créer de nouvelles fiches ou les modifier / supprimer.

Cahier des charges ou spécifications fonctionnelles *:*

Création d'une application node.js en environnement de développement, récupération des informations avec js fetch et affichage sur une page web.

Conception et organisation :

Diagrammes UML (use case, classes, séquence).

Modèle conceptuel de données (MCD).
Architecture choisie *: MVC*

Outils ou méthodologies utilisés : Travaux dirigés

Réalisation :
Technologies utilisées : node.js, express.js, MongoDB, Postman, HTML, CSS, vanilla JS
Description des fonctionnalités développées : CRUD simple et affichage navigateur

Extraits de code significatifs (commentés et expliqués).

Liste des endpoints ;
Chacun des endpoints permet une opération sur un des trois modèles disponibles (User, Movie ou Category)
La route / utilise uniquement le GET des films tandis que les routes login et logout identifient un utilisateur test actuellement présent en dur dans la BDD

Les Sample Data créent un jeu de données rapide au premier lancement de l'application pour peupler la vue d'accueil.

```
//General
router.get("/", welcome)

//Users
router.get("/login", login)
router.get("/logout", logout)

router.get("/users/all", getAllUsers)
router.get("/users/get/:id", getUser)
router.post("/users/add", createUser)
router.delete("/users/delete/:id", deleteUser)
router.put("/users/update/:id", updateUser)

//Videos
router.get("/videos/all", getAllMovies)
router.get("/videos/get/:id", getMovie)
router.post("/videos/add", createMovie)
router.delete("/videos/delete/:id", deleteMovie)
router.put("/videos/update/:id", updateMovie)

//Categories
router.get("/categories/all", getAllCategories)
router.get("/categories/get/:id", getCategory)
router.post("/categories/add", createCategory)
router.delete("/categories/delete/:id", deleteCategory)
router.put("/categories/update/:id", updateCategory)

//Sample Data
router.post("/sample/categories", sampleCategories)
router.get("/sample/movies", sampleMovies)
router.post("/sample/users", sampleUsers)

module.exports = router;
```

Gestion des versions : Git

Problèmes rencontrés et solutions :

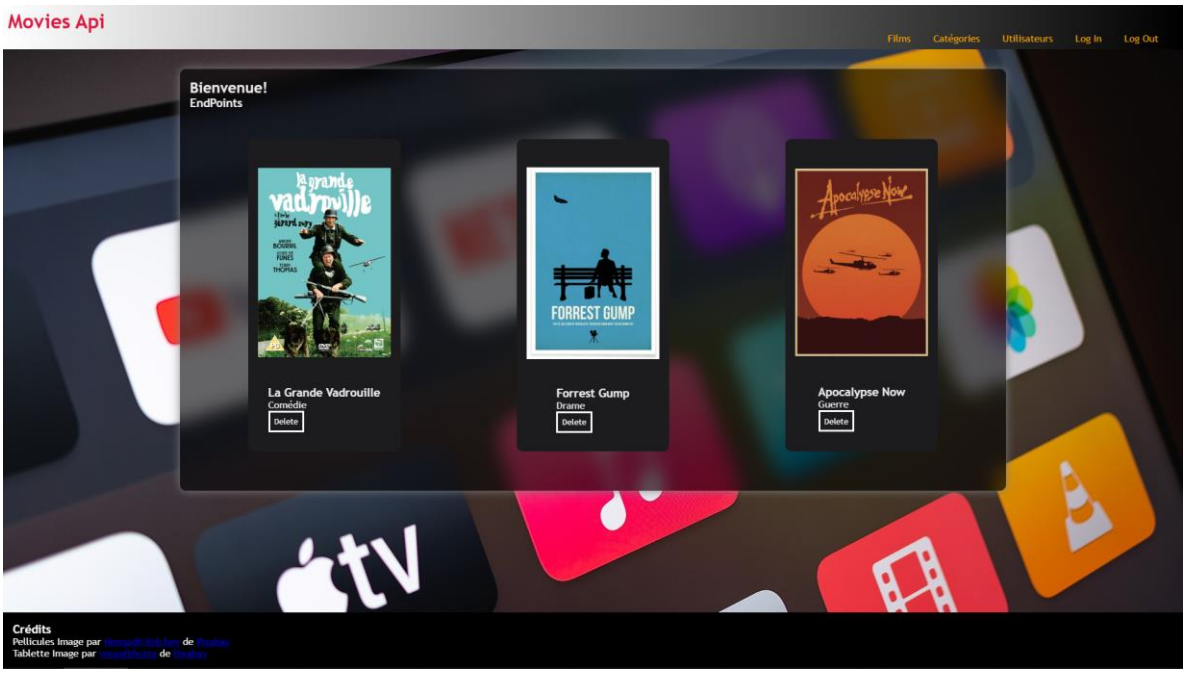
Abstraction souhaitée sur l'application afin de récupérer la structure et la réutiliser.
Utilisation d'un utilisateur autorisé
Pas de déploiement

Tests et validation :

Types de tests réalisés ; unitaires
Résultats des tests : Vérification du fonctionnement de l'API et du retour JSON

Livrables :

Vue permettant de manager la liste de films. Impossible de fonctionner sans utiliser la commande login de la nav



UsersAPI :

API liste d'utilisateurs : Mongo DB + MySQL → Application Web:

Nature du projet *: API + Application web Vanilla JS*
Contexte *: Formation*
Objectif *: Gerer une liste d'utilisateurs répartis sur deux SGBD*
Analyse préalable *:*

Liste très simple d'utilisateurs pouvant interroger et intervenir sur une BDD relationnelle et une BDD noSql simultanément

Cahier des charges ou spécifications fonctionnelles *:*

Création d'une application node.js hydratée à la fois par une BDD SQL et une BDD MongoDB en environnement de développement, récupération des information avec js fetch et affichage sur une page web.

Conception et organisation :

Diagrammes UML (**use case**, **classes**, **séquence**).

Modèle conceptuel de données (**MCD**).
Architecture choisie *: MVC*

Outils ou méthodologies utilisés : Travaux dirigés

Réalisation :
Technologies utilisées : node.js, express.js, MongoDB, SQL, Postman, HTML, CSS, vanilla JS
Description des fonctionnalités développées : Double connexion et affichage navigateur

Extraits de code significatifs (commentés et expliqués).

Double connexion MySQL / Mongo DB :

```
/**Connexion SQL via promise */

const mysql = require("mysql2/promise");
Qodo Gen: Options | Test this function
const dbPromise = () => {
  return mysql.createConnection({
    host: "127.0.0.1",
    user: "root",
    password: "",
    database: "user_db"
  });
};

dbPromise()
  .then((connection) => {
    console.log("Connected to MySQL");
    app.set("db", connection);
  })
  .catch((err) => {
    console.error("Error connecting to MySQL:", err);
});

/* Connexion MongoDB */
//mongoDb Connection
main().catch(err => console.log(err));
Qodo Gen: Options | Test this function
async function main(){
  await mongoose.connect('mongodb://localhost:27017/User_db');
  console.log("[DATABASE] MongoDb --Users-- is connected");
}
/**/

//Main route
app.get("/", (req, res) => {
    res.send("Bienvenue dans l'API des Utilisateurs");
})


app.use('/api/users', require('./routes/userRoute'));

app.listen(port, ()=> console.info(`[Server] is running on http://localhost:${port}`));
```

Liste des endpoints ;
Chacun des endpoints est doublé pour une utilisation SQL ou Mongo DB

```
/** Routes */
userRouter.get('/all', getAllUsers);
userRouter.get('/sql/all', getAllUsersMySql);
/**/
userRouter.post('/new', createUser);
userRouter.post('/sql/new', createUserMySql);

userRouter.get('/:id/show', getUserById);
userRouter.get('/sql/:id/show', getUserByIdMySql);

userRouter.delete('/:id/destroy', deleteUserById);
userRouter.delete('/sql/:id/destroy', deleteUserByIdMySql);

userRouter.put('/:id/edit',editUserById)
userRouter.put('/sql/:id/edit',editUserByIdMySql)

module.exports = userRouter;
```

Gestion des versions : Git

Problèmes rencontrés et solutions :

Abstraction souhaitée sur l'application afin de récupérer la structure et la réutiliser.
Utilisation de deux SGBD par la même API

Tests et validation :

Types de tests réalisés ; unitaires
Résultats des tests : Vérification du fonctionnement de l'API et du retour JSON

Livrables :

Vue récupérant les données de deux SGBD différent. Une liste MongoDb et une liste MySQL


Users API
Liste MongoDb
John
john.doe@mail.net
Leon
leon.kennedy@umbrella.org
Jane
jane.doe@mail.net
Liste SQL
Testarossa
testarossa@enzo.vroum
Squall
squall.leonhart@bgu.edu
Homer
homer.simpsons@burns-energy.org
Delete
delete@garbage.tv

B. Projets perso

3. Développement des compétences

Pour chaque activité-type du titre professionnel, illustrez les compétences développées. Le référentiel est organisé en deux activités principales :

Développer des composants d'interface utilisateur.
Exemples : création d'une page web avec React ou Symfony, intégration d'APIs.
Lien avec vos projets.

Développer la persistance des données et des composants back-end.
Exemples : développement d'une API REST, gestion des bases de données.
Lien avec vos projets.

Objectifs de formation (m2i)

A l'issue de cette formation, vous serez capable de :

Concevoir et développer des composants d'interface utilisateur en intégrant les recommandations de sécurité

- Maquetter une application
- Développer une interface utilisateur de type desktop
- Développer des composants d'accès aux données
- Développer la partie front-end d'une interface utilisateur Web
- Développer la partie back-end d'une interface utilisateur Web

Concevoir et développer une application multicouche répartie en intégrant les recommandations de sécurité

- Collaborer à la gestion d'un projet informatique et à l'organisation de l'environnement de développement
- Concevoir une application
- Développer des composants métier
- Construire une application organisée en couches
- Développer une application mobile
- Préparer et exécuter les plans de tests d'une application
- Préparer et exécuter le déploiement d'une application

Concevoir et développer la persistance des données en intégrant les recommandations de sécurité

- Concevoir une base de données
- Mettre en place une base de données
- Développer des composants dans le langage d'une base de données

4. Résultats et bilan

Auto-évaluation :

Points forts et axes d'amélioration.

Compétences acquises pendant la formation ou le projet.

Retour client ou utilisateur :

Si possible, incluez des témoignages ou retours des clients.

Perspectives :

Objectifs professionnels futurs.

Domaines dans lesquels vous souhaitez évoluer (web, mobile, IA, DevOps, etc.).

5. Annexes

Documentation technique des projets.
Diagrammes (MCD, UML).
Extraits de cahiers des charges.
Captures d'écran des applications ou démonstrations vidéo (liens vers un dépôt GitHub ou un portfolio en ligne si applicable).