Cas pratique — Portfolio personnel Full-Stack

1️⃣ Contexte & objectif du projet

Objectif principal

Créer un portfolio personnel moderne permettant de :

- présenter mon profil de développeuse web full-stack
- mettre en valeur mes projets (Wizdle, TripTogether, etc.)
- proposer un formulaire de contact fonctionnel
- apprendre à concevoir une API Express + MySQL
- comprendre les enjeux front / back / CORS / déploiement

2️⃣ Analyse fonctionnelle

Fonctionnalités principales

Frontend

- affichage des pages : Home, About, Skills, Projects, Contact
- formulaire de contact avec :
  - catégorie
  - nom
  - message
- confirmation visuelle après envoi

Backend (local)

- API REST développée avec Express
- routes CRUD pour les messages de contact
- connexion à une base MySQL
- tests avec Postman

3️⃣ Choix techniques

Frontend

React, TypeScript, Vite, useState, CSS, EmailJS

Backend (développé et testé en local)

Express, TypeScript, MySQL, mysql2, Postman

4️⃣ Architecture backend

Structure du projet

```
server/
├──── src/
│     ├──── app.ts
│     ├──── main.ts
│     ├──── router.ts
│     ├──── database/
│     │     └──── client.ts
│     └──── modules/
│           └──── contact/
│                 ├──── contactRepository.ts
│                 ├──── contactController.ts
│                 └──── contactRouter.ts
```

👉 Séparation claire des responsabilités

Couche	Rôle
Repository	Accès à la base (SQL uniquement)
Controller	Logique HTTP
Router	Définition des routes
App	Configuration Express
Main	Démarrage du serveur

5️⃣ API Express – Exemple

Routes exposées (local)

Méthode	Route	Rôle
GET	/api/contact	lire tous les messages
GET	/api/contact/:id	lire un message
POST	/api/contact	ajouter un message
PUT	/api/contact/:id	modifier un message
PUT	/api/contact/:id/read	marquer comme lu

Modèle de données MySQL

```
CREATE TABLE contact_message (
  id INT PRIMARY KEY AUTO_INCREMENT,
  category ENUM('wizdle', 'portfolio', 'general', 'cv') NOT NULL,
  name VARCHAR(100) NOT NULL,
  commentaires TEXT NOT NULL,
  is_read BOOLEAN DEFAULT FALSE,
  created_at DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

6️⃣ Tests backend

Tests réalisés avec Postman

- validation des routes GET / POST / PUT
- vérification des statuts HTTP (200, 201, 400, 404)
- simulation des erreurs
- validation des données envoyées

👉 Objectif : s'assurer que l'API fonctionne indépendamment du frontend

---

7️⃣ Problématique rencontrée en production

Problèmes

- CORS bloqué en production
- backend déployé séparément
- complexité inutile pour un simple portfolio

Analyse

- le backend était pertinent pour l'apprentissage
- mais surdimensionné pour un site vitrine
- dépendance au déploiement + CORS + variables serveur

8️⃣ Solution adoptée (choix final)

EmailJS côté frontend

Pourquoi EmailJS ?

Pas de backend requis

Fonctionne en prod immédiatement

Pas de CORS

Sécurisé via clés publiques

👉 Le backend reste documenté, testé en local, présenté comme cas pratique mais non utilisé en production.

9️⃣ Bilan & apprentissages

Compétences acquises

- conception d'une API REST
- architecture Express propre
- communication front ↔️ back
- gestion CORS
- déploiement et limites
- prise de décision technique

Conclusion

Ce projet m'a permis de comprendre quand utiliser un backend,
quand ne pas en utiliser, et surtout comment justifier un choix technique.