PYTHON & DATA : API, INTELLIGENCE ARTIFICIELLE

Créer une API REST avec Python et FastAPI

Développer des APIs modernes et performantes avec FastAPI. Typage, validation, documentation automatique, base de données : le framework Python qui monte en puissance.

DURÉE	TARIF HT	NIVEAU	LANGUE	GROUPE	FORMAT
5j (35h)	**1790.00 €**	**Intermédiaire**	**Français**	**2-8**	**Formation**

1 PUBLIC VISÉ

- Développeurs Python maîtrisant la POO souhaitant créer des APIs backend modernes et performantes.
- Personnes ayant suivi la formation Python : programmation orientée objet et traitement de données ou disposant d'une expérience Python équivalente.
- Développeurs back-end voulant adopter FastAPI comme alternative moderne à Flask ou Django REST Framework.
- Data scientists ou ingénieurs ML souhaitant exposer leurs modèles via une API REST professionnelle.

2 PRÉREQUIS

- Bonne maîtrise de Python et de la programmation orientée objet : classes, héritage, décorateurs, type hints.
- La formation Python : programmation orientée objet et traitement de données est recommandée.
- Notions de SQL et de base de données relationnelle.
- Savoir utiliser le terminal et VS Code.

3 OBJECTIFS PÉDAGOGIQUES

- Comprendre l'architecture de FastAPI et ses avantages par rapport à Flask et Django
- Créer des endpoints REST typés avec gestion des paramètres de route, de requête et du body
- Valider et sérialiser les données avec Pydantic v2
- Connecter une base de données PostgreSQL avec SQLAlchemy et gérer les migrations avec Alembic
- Implémenter une authentification JWT sécurisée avec gestion des rôles
- Configurer les middlewares et le CORS pour une API consommée par un frontend
- Exploiter la documentation automatique Swagger UI et ReDoc
- Tester une API FastAPI avec pytest et httpx
- Déployer une API FastAPI avec Uvicorn et Docker en production

4 PROGRAMME DÉTAILLÉ

Jour 1 (7h) – Fondamentaux FastAPI et premiers endpoints

Module 1 – Découverte de FastAPI (2h)

Pourquoi FastAPI ? (1h)

- FastAPI vs Flask vs Django REST Framework : positionnement et cas d'usage
- Les atouts de FastAPI : performances ASGI, typage natif, documentation automatique
- ASGI vs WSGI : comprendre le modèle asynchrone de FastAPI
- L'écosystème FastAPI : Uvicorn, Starlette, Pydantic
- FastAPI dans le monde réel : Netflix, Uber, Microsoft

Mise en place de l'environnement (1h)

- Créer un environnement virtuel et installer FastAPI avec Uvicorn
- Structure d'un projet FastAPI professionnel : app/, routers/, models/, schemas/, services/
- Lancer l'application avec uvicorn et le rechargement automatique
- La documentation automatique : Swagger UI sur /docs et ReDoc sur /redoc
- Cas pratique : premier endpoint Hello World avec documentation interactive

Module 2 – Routes, paramètres et réponses (3h)

Créer des endpoints REST (1h30)

- Les décorateurs de route : @app.get, @app.post, @app.put, @app.patch, @app.delete
- Paramètres de chemin : /items/{item_id} avec typage automatique
- Paramètres de requête : filtres, tri et pagination depuis l'URL
- Le body de la requête : recevoir et valider du JSON
- Les codes de statut HTTP : status_code et HTTPException
- Response model : contrôler les champs retournés dans la réponse
- Cas pratique : CRUD complet sur une ressource avec les bons statuts HTTP

Fonctionnalités avancées des routes (1h30)

- Les tags : organiser les endpoints dans Swagger UI
- Les dépendances avec Depends : factoriser la logique commune entre routes
- Les paramètres de formulaire et l'upload de fichiers avec UploadFile
- Les headers et les cookies : lire et écrire
- Les réponses personnalisées : JSONResponse, FileResponse, StreamingResponse
- Cas pratique : endpoint d'upload de fichier avec validation du type MIME

Module 3 – Pydantic v2 : validation et sérialisation (2h)

Les bases de Pydantic v2 (1h)

- BaseModel : déclarer un schéma avec des types Python
- Les validateurs de champ : Field, contraintes, valeurs par défaut
- Les types Pydantic : EmailStr, HttpUrl, UUID, Annotated
- Valider des données manuellement : model_validate et model_dump
- Cas pratique : schémas de création et de réponse pour une ressource utilisateur

Pydantic avancé (1h)

Les validateurs personnalisés avec @field_validator et @model_validator

- Schémas imbriqués : relations entre modèles Pydantic
- Les schémas de requête vs les schémas de réponse : éviter d'exposer les données sensibles
- model_config : configurer le comportement du modèle
- Cas pratique : schémas distincts pour la création, la mise à jour et la réponse

Jour 2 (7h) – Base de données avec SQLAlchemy et Alembic

Module 4 – SQLAlchemy : modèles et connexion (3h)

Configurer SQLAlchemy avec FastAPI (1h)

- Installer SQLAlchemy et psycopg2 pour PostgreSQL
- Configurer le moteur et la session SQLAlchemy
- La dépendance get_db : injecter la session dans les routes avec Depends
- Configurer la connexion via les variables d'environnement avec pydantic-settings
- Cas pratique : connexion à PostgreSQL avec pool de connexions

Déclarer les modèles SQLAlchemy (2h)

- DeclarativeBase : déclarer les tables comme classes Python
- Les colonnes : types, contraintes, valeurs par défaut
- Les relations : ForeignKey, relationship, back_populates
- Les relations ManyToMany avec une table de jointure
- Les colonnes calculées automatiquement : created_at, updated_at
- Cas pratique : modèles User, Post, Category avec toutes leurs relations

Module 5 – CRUD avec SQLAlchemy et Alembic (2h)

Opérations CRUD (1h)

- Créer un enregistrement : session.add et session.commit
- Lire des enregistrements : session.get, session.execute avec select
- Filtrer et paginer les résultats avec where, limit et offset
- Mettre à jour et supprimer des enregistrements
- Cas pratique : couche CRUD réutilisable pour chaque ressource

Migrations avec Alembic (1h)

- Installer et configurer Alembic dans un projet FastAPI
- Générer une migration automatique depuis les modèles SQLAlchemy
- Appliquer et annuler des migrations : upgrade et downgrade
- Gérer les migrations en production sans interruption de service
- Cas pratique : créer et appliquer les migrations du projet fil rouge

Module 6 – Architecture en couches (2h)

Séparer les responsabilités (1h)

- Le pattern Router : découper l'application en modules avec APIRouter
- La couche service : isoler la logique métier des routes
- La couche repository : abstraire l'accès aux données
- L'injection de dépendances FastAPI : relier les couches proprement

Configuration et paramètres (1h)

pydantic-settings : gérer la configuration avec validation et typage

- Environnements multiples : development, testing, production
- Le pattern Settings singleton avec lru_cache
- Cas pratique : refactoriser l'application en architecture Router/Service/Repository

Jour 3 (7h) – Authentification et sécurité

Module 7 – Authentification JWT (4h)

Mettre en place l'authentification (2h)

- Hacher les mots de passe avec passlib et bcrypt
- Générer et vérifier des tokens JWT avec python-jose
- L'endpoint /auth/register : créer un compte utilisateur
- L'endpoint /auth/login : vérifier les identifiants et retourner un JWT
- OAuth2PasswordBearer : le schéma d'authentification FastAPI
- Cas pratique : système de connexion complet avec génération de token

Protéger les routes et gérer les rôles (2h)

- La dépendance get_current_user : décoder le JWT et récupérer l'utilisateur
- Protéger un endpoint avec Depends(get_current_user)
- Les rôles utilisateur : implémenter ROLE_USER et ROLE_ADMIN
- Dépendances de rôle : require_admin, require_editor
- Les refresh tokens : renouveler l'accès sans redemander les identifiants
- Cas pratique : endpoints protégés avec contrôle d'accès par rôle

Module 8 – Middlewares et sécurité (3h)

Middlewares FastAPI (1h30)

- Configurer le CORS avec CORSMiddleware
- Middleware de logging : tracer toutes les requêtes entrantes
- Middleware de gestion du temps de réponse
- Middleware d'authentification global
- Cas pratique : stack de middlewares pour une API consommée par un frontend React

Sécurité et gestion des erreurs (1h30)

- Exception handlers globaux : formater toutes les erreurs de manière cohérente
- Valider toutes les entrées : ne jamais faire confiance aux données client
- Rate limiting avec slowapi : protéger l'API contre les abus
- Headers de sécurité HTTP : ajouter CSP, X-Frame-Options via middleware
- Cas pratique : handler global d'erreurs avec réponses JSON standardisées

Jour 4 (7h) – Fonctionnalités avancées et tests

Module 9 – Programmation asynchrone avec FastAPI (2h)

Async/await dans FastAPI (1h)

- Fonctions synchrones vs asynchrones dans FastAPI : quand utiliser async def
- Appels HTTP asynchrones avec httpx.AsyncClient
- Tâches en arrière-plan avec BackgroundTasks
- Cas pratique : endpoint qui envoie un email en arrière-plan sans bloquer la réponse

WebSockets et événements (1h)

- Créer un endpoint WebSocket avec FastAPI
- Les événements de démarrage et d'arrêt : lifespan
- Initialiser la connexion BDD et les ressources au démarrage
- Cas pratique : endpoint WebSocket de notifications en temps réel

Module 10 – Tests avec pytest et httpx (3h)

Tester une API FastAPI (1h30)

- Configurer pytest pour FastAPI : conftest.py et fixtures
- TestClient vs AsyncClient : tester les endpoints synchrones et asynchrones
- Créer une base de données de test isolée avec SQLite en mémoire
- Surcharger les dépendances dans les tests : mocker get_db et get_current_user
- Cas pratique : tests des endpoints publics avec vérification des codes HTTP et du JSON

Tests avancés (1h30)

- Tester les endpoints protégés : générer un token JWT de test
- @pytest.mark.parametrize : tester plusieurs cas avec un seul test
- Mesurer la couverture de code avec pytest-cov
- Tester les cas d'erreur : 400, 401, 403, 404, 422
- Cas pratique : suite de tests complète pour les endpoints CRUD et d'authentification

Module 11 – Documentation et versioning (2h)

Enrichir la documentation automatique (1h)

- Documenter les endpoints avec summary, description et response_description
- Documenter les schémas Pydantic avec Field et model_config
- Configurer les métadonnées de l'API : titre, version, contact, licence
- Ajouter l'authentification JWT dans Swagger UI
- Exporter le schéma OpenAPI en JSON pour les clients externes

Versioning de l'API (1h)

- Stratégies de versioning : URI (/api/v1), header, query param
- Implémenter le versioning par préfixe de router dans FastAPI
- Maintenir plusieurs versions simultanément sans duplication de code
- Déprécier un endpoint proprement avec deprecated=True
- Cas pratique : API versionnée avec v1 et v2 coexistants

Jour 5 (7h) – Déploiement et projet de synthèse

Module 12 – Déploiement en production (3h)

Packaging et configuration production (1h)

- Configurer Uvicorn pour la production : workers, timeout, access log
- Gunicorn avec des workers Uvicorn : haute disponibilité
- Variables d'environnement de production : pydantic-settings et secrets
- Configurer PostgreSQL en production et gérer les migrations Alembic

Conteneurisation avec Docker (2h)

- Écrire un Dockerfile optimisé pour FastAPI : image légère, multi-stage build

Docker Compose : orchestrer FastAPI, PostgreSQL et un reverse proxy Nginx

- Variables d'environnement dans Docker Compose
- Healthcheck : vérifier que l'API est opérationnelle
- Cas pratique : déploiement complet de l'API avec Docker Compose

Module 13 – Projet de synthèse (4h)

Réalisation d'une API REST FastAPI complète

- Cahier des charges : API de gestion de contenu ou de plateforme e-commerce
- Fonctionnalités : architecture Router/Service/Repository, modèles SQLAlchemy avec relations, migrations Alembic, authentification JWT avec rôles USER et ADMIN, schémas Pydantic v2 distincts par opération, gestion des erreurs globale, tests pytest couvrant les endpoints critiques, documentation Swagger enrichie, déploiement avec Docker Compose
- Étape 1 : modèles SQLAlchemy, schémas Pydantic et migrations
- Étape 2 : endpoints CRUD avec authentification et rôles
- Étape 3 : tests, documentation et configuration production
- Étape 4 : déploiement Docker Compose et recette finale
- Revue de code collective : architecture, sécurité, qualité des tests
- Retour formateur individualisé sur le projet rendu

5 COMPÉTENCES VISÉES

- Concevoir et développer une API REST complète et documentée avec FastAPI
- Modéliser et interroger une base de données PostgreSQL avec SQLAlchemy et Alembic
- Valider toutes les entrées et sorties avec Pydantic v2 sans duplication de code
- Sécuriser une API avec l'authentification JWT et la gestion des rôles
- Écrire des tests automatisés pour valider le comportement des endpoints
- Déployer et maintenir une API FastAPI en production via Docker

6 MODALITÉS PÉDAGOGIQUES

Formation délivrée en présentiel ou distanciel (visioconférence). Le formateur alterne entre méthode démonstrative (construction progressive d'une API complète tout au long de la semaine), méthode interrogative (analyse des choix d'architecture et des compromis de conception) et méthode active (exercices pratiques et projet de synthèse fil rouge). L'accent est mis sur les pratiques professionnelles réelles : typage strict, tests, sécurité et déploiement.

7 MOYENS PÉDAGOGIQUES ET TECHNIQUES

- Support de cours numérique mis à disposition des apprenants
- Dépôt GitHub de démonstration avec le projet fil rouge versionné par étape
- Environnement de développement : VS Code + Python 3.12+ + Docker Desktop
- Pour le distanciel : visioconférence (Zoom ou équivalent), partage d'écran, chat en direct
- Accès à la plateforme pédagogique LaPolaris (supports, ressources, émargement)

8 MODALITÉS D'ÉVALUATION

- En cours de formation : exercices pratiques corrigés à chaque module
- En fin de formation : réalisation d'une API REST FastAPI complète avec authentification, tests et déploiement Docker
- Questionnaire d'auto-évaluation des acquis en fin de parcours

9 CRITÈRES D'ÉVALUATION

- Architecture en couches respectée : Router, Service, Repository sans mélange des responsabilités
- Schémas Pydantic v2 distincts pour la création, la mise à jour et la réponse sans exposition de données sensibles
- Authentification JWT fonctionnelle avec protection des routes selon les rôles
- Tests pytest couvrant les cas nominaux et les cas d'erreur des endpoints critiques
- API déployée et accessible via Docker Compose avec base de données PostgreSQL

10 MODALITÉS DE VALIDATION

Attestation de fin de formation délivrée à l'issue du parcours, conditionnée à une assiduité d'au moins 80 % et à la réalisation du projet de synthèse. L'attestation précise les objectifs atteints et les compétences acquises.

11 SUIVI ET ACCOMPAGNEMENT

- Feuilles d'émargement signées par demi-journée (présentiel) ou émargement numérique (distanciel)
- Traçabilité des activités pédagogiques réalisées
- Attestation d'assiduité délivrée en fin de formation
- Suivi individuel via les exercices corrigés et le projet de synthèse

12 CONDITIONS D'ACCÈS

Formation accessible sur inscription directe, sans prérequis administratif particulier. Le financement peut être pris en charge par l'employeur dans le cadre d'un plan de développement des compétences, ou en autofinancement avec possibilité de paiement en plusieurs fois.

13 DÉLAIS D'ACCÈS

Inscription possible jusqu'à **5 jours ouvrés** avant le début de la session. Pour toute demande urgente, nous contacter directement.

ACCESSIBILITÉ · HANDICAP

Nos formations sont accessibles aux personnes en situation de handicap. Pour toute situation nécessitant un aménagement (matériel, temporel ou pédagogique), nous vous invitons à nous contacter avant l'inscription afin d'étudier les adaptations possibles. Référent handicap : contact@lapolaris.fr

LaPolaris

TÉL. +33762584798

EMAIL contact@lapolaris.fr

WEB lapolaris.fr