DJANGO 6.0.5

La Révolution Web Python de Nouvelle Génération : Tâches d'arrière-plan natives, Sécurité CSP renforcée, Partials HTML & Vues Async.

MASTERCLASS EXPERT DEVOPS

01. Fondations & Setup

Entrez dans l'ère de la performance avec les nouveautés de base de Django 6.0.5.

PYTHON 3.12+ & ÉCOSYSTÈME

Configuration de Rupture

Django 6.0.5 s'affranchit du passé en exigeant strictement **Python 3.12, 3.13 ou 3.14**.

⚠ Fin officielle du support de Python 3.10 et 3.11 pour de meilleures performances CPU.

Bases de Données Rehaussées

Support complet de MariaDB 10.6+ (support 10.5 arrêté) et de PostgreSQL 16+.

Le champ par défaut (DEFAULT_AUTO_FIELD) est désormais nativement initialisé à **BigAutoField**.

POURQUOI DJANGO 6.0.5 ?

Tâches Natives

Un framework asynchrone intégré au cœur de Django, éliminant les dépendances complexes à Celery pour les besoins standards.

Sécurité Ultime

Intégration d'un middleware natif Content Security Policy (CSP) robuste contre les failles d'injection XSS de bout en bout.

HTML Modulaire

Les partials de templates évitent d'exploser vos structures de répertoires en encapsulant des composants au même endroit.

DJANGO 5.X VS DJANGO 6.0.5

Fonctionnalité	Django 5.x (Ancien)	Django 6.0.5 (Actuel)	Bénéfice Technique
Version Python	Supporte Python 3.10+	**Exige Python 3.12+**	Gains de mémoire substantiels et rapidité
Background Tasks	Bibliothèque tierce requise	**Framework natif core**	Maintenance facilitée et zero-dépendance
Sécurité CSP	Via packages tiers instables	**Middleware natif CSP**	Protection stricte par nonces automatisés
Rendu de Partials	Template entier obligatoire	**Rendu isolé de fragments**	Combinaison parfaite avec HTMX & Alpine

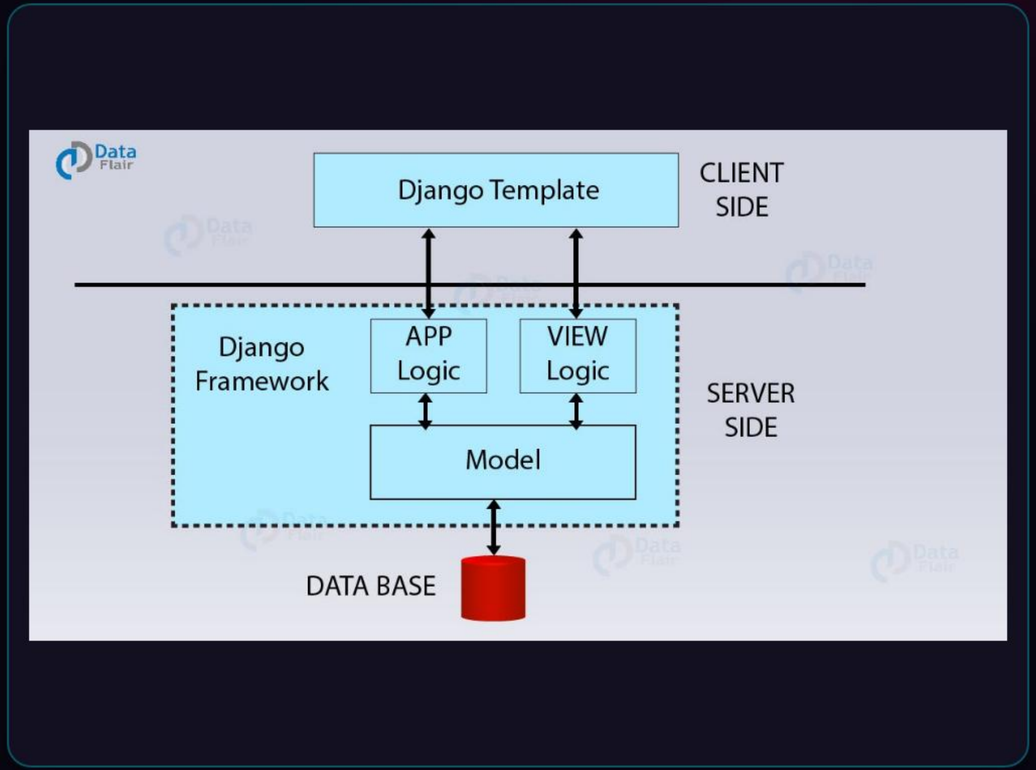
PREREQUIS & INITIALISATION

- **Machine Linux / MacOS :** Python 3.12 ou supérieur doit être installé par défaut (`python3 --version`).
- **Environnement Virtuel :** Isoler le système en créant un conteneur d'exécution local (`python -m venv .venv && source .venv/bin/activate`).
- **Installation :** Mise à jour de pip et installation directe (`pip install --upgrade pip && pip install django==6.0.5`).
- **django-admin :** La commande crée automatiquement les répertoires cibles s'ils n'existent pas (`django-admin startproject core_app .`).

L'ARCHITECTURE MTV INTEMPORELLE

Django s'appuie sur le pattern **Model-Template-View (MTV)**, une variation élégante du MVC classique de l'ingénierie logicielle :

- **Model :** Accès unifié et orienté objet à la donnée de votre application.
- **Template :** Présentation dynamique de l'information (HTML, XML, JSON).
- **View :** La logique d'aiguillage liant les requêtes réseau à la donnée SQL.



02. L'ORM de Django 6.0

Optimisations de performance, expressions de base de données avancées et modèles intelligents.

LA GÉNÉRATION DYNAMIQUE SQL

GeneratedField & Expression Defaults

Django 6.0 pousse la logique métier au plus près du moteur de base de données.

Avec **GeneratedField**, la base de données calcule elle-même les colonnes de manière native en s'appuyant sur les expressions logiques de vos modèles.

⚡ Zéro calcul superflu côté CPU Python lors des lectures !

CHAMPS DE MODÈLES DJANGO 6

L'utilisation combinée de `GeneratedField` et de `db_default` permet de garantir une intégrité parfaite des données au niveau SQL :

- `db_default` applique une valeur par défaut SQL.
- `GeneratedField` calcule des données concaténées.

```
from django.db import models from django.db.models.functions import Concat class Product(models.Model): title
```

MIGRATIONS ROBUSTES DE PRODUCTION

- **Squashed Migrations :** Django 6.0 permet désormais d'écraser des migrations qui ont elles-mêmes déjà été écrasées.
- **Support ZoneInfo :** Sérialisation et écriture native des instances `zoneinfo.ZoneInfo` dans l'historique des modifications.
- **Contraintes SQL dynamiques :** Système d'introspection amélioré limitant le risque de verrous lourds lors du déploiement de clés étrangères.
- **Migration continue :** Automatisation recommandée via pipeline CI/CD : `python manage.py migrate --noinput`.

TYPES DE CHAMPS & OPTIONS SQL

Option ORM	Type SQL généré	Comportement en DB	Bénéfice de performance
db_default=Now()	`DEFAULT CURRENT_TIMESTAMP`	La DB assigne la date à l'insertion	Aucun appel système Python de date
BigAutoField	`BIGINT AUTO_INCREMENT`	Clé primaire auto-incrémentée (64 bits)	Évite la saturation des IDs à grande échelle
GeneratedField	`GENERATED ALWAYS AS...`	Calcul en base lors des écritures	Lectures ultra-rapides sans reconstruction
db_index=True	`CREATE INDEX...`	Indexation de colonne ciblée	Optimise les requêtes de filtrage complexes

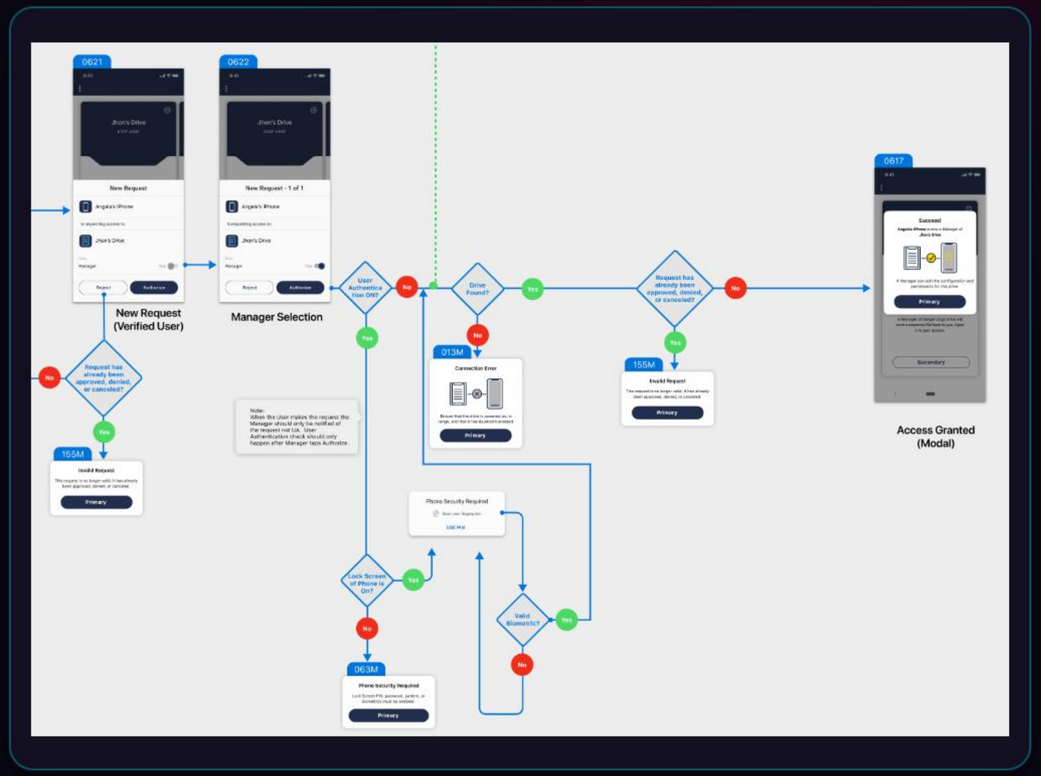
03. Le Routage URL

La porte d'entrée des requêtes HTTP. Gérer des URLs claires, performantes et sécurisées.

LE FLUX DE RÉSOLUTION D'URLS

Django utilise un moteur de routage basé sur des expressions régulières compilées pour garantir des performances optimales :

- **Aiguillage :** Chaque requête est soumise de haut en bas à la liste de patterns.
- **Converters :** Extraction automatique et conversion de type (ex: `int`, `uuid`, `slug`).
- **Secured Paths :** Élimination des chemins malformés avant d'attaquer la vue logicielle.



LES URLS ESSENTIELLES DJANGO

Voici la structure clé de routage d'un projet Django 6 de production (`urls.py`) :

```
from django.contrib import admin from django.urls import path, include urlpatterns = [ # 1. Interface d'administration native sé
```

ROUTAGE AVANCÉ & CONVERTERS

Convertisseurs Personnalisés

Au-delà des types natifs (`int`, `str`, `uuid`, `slug`, `path`), Django permet de déclarer des formats métier stricts via Regex.

Un convertisseur personnalisé intercepte les requêtes malicieuses avant que le

```
# Configuration des chemins d'application (shop/urls.py) from django.urls import path from . import views urlpatt
```

04. Vues & Templates

Générez des interfaces interactives d'une fluidité extrême grâce à la révolution des partials HTML.

FBVS VS CBVS VS ASYNC

Function-Based (FBVs)

Simples et directes. Idéales pour les algorithmes spécifiques, le prototypage rapide et l'intégration de micro-services HTMX.

Class-Based (CBVs)

Orientées objet et réutilisables. Parfaites pour industrialiser des CRUD complexes (ListView, DetailView, CreateView).

Vues Asynchrones

Exécution non-bloquante avec support des requêtes de base de données asynchrones (AsyncPaginator, etc.) sous serveur ASGI.

LA RÉVOLUTION DES PARTIALS

Intégration d'un Fragment HTML

Historiquement, pour renvoyer un fragment HTML d'une page (nécessaire avec HTMX/Alpine), il fallait fragmenter vos templates en dizaines de fichiers.

Django 6.0 introduit les balises **partialdef** et **partial** directement au sein du fichier global.

Rendu Direct par Code

Le développeur peut appeler directement et de manière isolée un partial nommé depuis le code Python d'une vue.

Économise des dizaines d'heures de maintenance de structures de fichiers complexes.

SYNTAXE & RENDU DE PARTIALS

Exemple de Template (`books.html`) :

```
 Nos Livres {% partialdef book-info %}  {{ book.title }} - {{ book.author }}  {% endpartialdef %}
{{ book.title }} - {{ book.author }}
```

Rendu de fragment i

```
from django.shortcuts
```

MISES À JOUR DU TEMPLATE ENGINE

Nouveaux Outils des Templates

Finies les manipulations complexes pour connaître la taille d'un ensemble de données.

La nouvelle variable locale **forloop.length** permet d'accéder instantanément à la taille totale du tableau d'itération en cours de parcours.

Le tag `querystring` a également été revu pour préfixer nativement et proprement vos URLs avec le point d'interrogation ?.

05. Révolution Asynchrone

Optimisez l'utilisation des threads et servez des milliers de clients de manière non-bloquante.

LA FIN DU BLOCAGE I/O

L'ORM Devient Asynchrone

L'une des limites majeures des versions précédentes de Django était son ORM purement synchrone.

⚡ Django 6.0.5 introduit **AsyncPaginator** et **AsyncPage** pour paginer des bases SQL géantes de manière asynchrone.

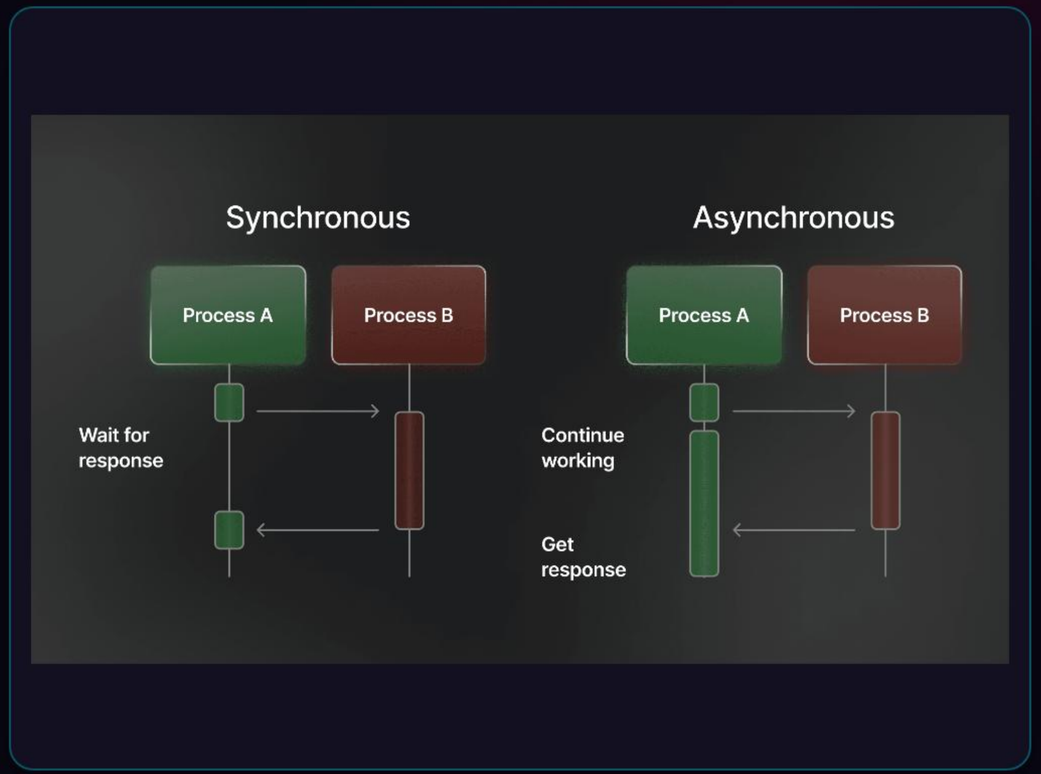
Support des Drivers Async

Intégration directe des adaptateurs PostgreSQL de type `psycopg` version async, permettant de libérer l'éventail de threads applicatifs.

L'EXÉCUTION ASGI NON-BLOQUANTE

Contrairement au traditionnel WSGI (un thread par requête), ASGI traite de multiples requêtes de manière simultanée sur un seul et même processus système :

- **Attente I/O :** Lorsqu'une requête attend le retour de la DB, le thread traite une autre demande.
- **Async Views :** Utilisation naturelle de la syntaxe de base Python : `async def` et `await`.



PROGRAMMATION ASYNC NATIVE

Déclaration d'un contrôleur asynchrone sécurisé exécutant des requêtes SQL parallélisées :

```
from django.http import JsonResponse from .models import UserActivity # Vue nativement non-bloquante async def activity_feed(req
```

06. Background Tasks Natives

Le tout nouveau Tasks Framework introduit dans le Core de Django 6.0.5 (DEP 14).

DEP 14 : UN FRAMEWORK CORE

Déclaration Simple

Décorateur natif `@task` pour transformer n'importe quelle méthode Python en tâche pouvant être déportée.

Cas d'Usage Recommandé

Idéal pour l'envoi d'emails transactionnels, la génération de documents lourds ou le nettoyage d'éléments éphémères.

Intégration d'Exécuteurs

Une interface universelle sur laquelle s'appuient les bibliothèques d'infrastructure pour l'exécution réelle.

DILEMME : TASKS VS CELERY ?

Django Background Tasks

Natif au framework. API simple, robuste et unifiée qui évite de surcharger l'architecture système.

✓ Recommandé pour 90% des applications web classiques de production.

Celery & RabbitMQ

Solution externe lourde. Permet des workflows ultra-complexes (Chord, Canvas) et de la scalabilité matérielle poussée.

⚠ Idéal pour les architectures Big Data ou les millions de micro-jobs asynchrones complexes.

DÉCLARER & DÉCLENCHER UNE TÂCHE

Définition (`tasks.py`**) :**

```
from django.tasks import task @task def send_welcome_email(user_id): # Exécution en tâche de fond user = User.objects.get(id=user_id)
```

07. Sécurité & APIs

Protégez vos applications des pirates avec le support natif de la Content Security Policy (CSP).

VAINCRE LE CROSS-SITE SCRIPTING

CSP

NATIVE SUPPORT

ContentSecurityPolicyMiddleware

Historiquement, injecter une entête CSP de manière intelligente relevait du parcours du combattant.

Django 6.0 intègre un middleware complet de gestion de **Content Security Policy** avec support dynamique des nonces pour sécuriser vos scripts inline.

🔒 Les attaques d'injection JavaScript malicieuses sont purement et simplement bloquées par le navigateur.

CONFIGURATION CSP SOUS DJANGO 6

Pour activer la protection, intégrez le middleware et définissez vos règles d'accès au niveau des paramètres de base de l'application :

```
# core_app/settings.py MIDDLEWARE = [ 'django.middleware.security.SecurityMiddleware', 'django.middleware.common.CommonMiddlewar
```

LE NOUVEL EMAIL API

- **API de messagerie modernisée :** Django 6 abandonne ses anciens adaptateurs au profit de la bibliothèque email native et moderne de Python.
- **Excellente gestion Unicode :** Disparition définitive des bugs complexes d'encodage lors de l'envoi d'accents ou d'émojis complexes.
- **Pièces Jointes Inline :** Simplification drastique de la commande pour attacher des fichiers complexes ou des images directement dans le HTML.
- **Exigence de kwargs :** Les arguments positionnels dépréciés sont désormais interdits pour éviter toute faille logicielle de mauvais typage.

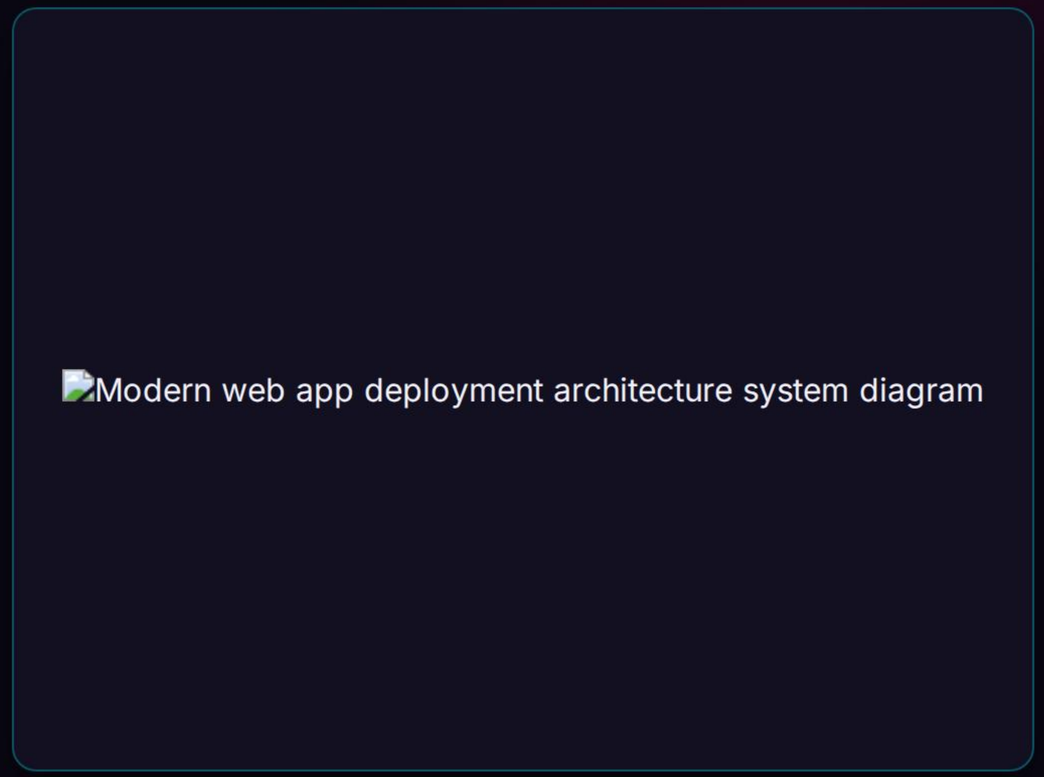
08. Déploiement & Prod

Mettre en production son application Django 6.0.5 de manière sécurisée sous architecture Linux stable.

L'ARCHITECTURE SERVEUR MODERNE

L'architecture optimale pour faire tourner votre application Django 6 de production s'appuie sur la complémentarité de multiples services Linux découplés :

- **Nginx :** Reverse-proxy gérant la sécurité SSL/TLS et servant les fichiers statiques.
- **Uvicorn / Gunicorn :** Serveurs d'application ASGI gérant le cycle d'exécution Python de votre code.
- **PostgreSQL :** Base relationnelle isolée et hautement disponible.



LA CHECKLIST DE SÉCURITÉ PROD

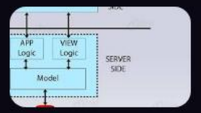
Paramètre de Configuration	Valeur de Production	Raison d'Ingénierie	Risque en cas d'erreur
`DEBUG`	**False**	Désactive l'affichage détaillé des erreurs Python	Fuite massive de vos variables et clés d'API
`ALLOWED_HOSTS`	`['mondomaine.com']`	Filtre les requêtes basées sur l'entête HTTP Host	Attaques par empoisonnement d'URLs tierces
`SECURE_SSL_REDIRECT`	**True**	Force toutes les connexions à passer en HTTPS sécurisé	Écoutes de réseau et vol d'identifiants utilisateurs
`SESSION_COOKIE_SECURE`	**True**	N'envoie les cookies que sur des canaux chiffrés	Détournement de session utilisateur (Hijacking)

SESSION Q & A

docs.djangoproject.com/en/6.0/

architecture@expert-django.fr

IMAGE SOURCES



https://miro.medium.com/v2/resize:fit:1400/0*r7ALuIxaXPSboehX.jpg

Source: medium.com

https://cdn.prod.website-files.com/687e8d1b96312cc631cafec7/68c493b80ceeb286edb4c08c_689b86e10695dc21ed3576bb_689a5c2670568f506a0444fb_AD_4nXcYt9Qy0oWNrBOvHIYg5qTAMhNjaR95sFmf0aR7-vh3nZ1TGNvbHKvaV1ddv0JSAu8Yhx0IQMR4ErkZVE-wwifsL8IvjkQI2b9gw4Yqeg8-Ze4jj-dV13AZ8kxBOm96BIJmYVM6.png

Source: webflow.com

https://framerusercontent.com/images/oyf1FaA8kyZa0Y22FvN9UPQyEk.png?width=3833&height=2156

Source: pieces.app

https://tech-stack.com/blog/content/images/2024/06/Modern_Web_App_Architecture__Trends__and_Best_Practices__for_2024.png

Source: tech-stack.com