Génération de projets - Modèle

Vous pouvez utiliser un générateur de projet pour commencer, qui réalisera pour vous la mise en place de bases côté architecture globale, sécurité, base de données et premières routes d'API.

Un générateur de projet fera toujours une mise en place très subjective que vous devriez modifier et adapter suivant vos besoins, mais cela reste un bon point de départ pour vos projets.

Full Stack FastAPI PostgreSQL

GitHub : https://github.com/tiangolo/full-stack-fastapi-postgresql

Full Stack FastAPI PostgreSQL - Fonctionnalités

- Intégration **Docker** complète (basée sur Docker).
- Déploiement Docker en mode Swarm
- Intégration **Docker Compose** et optimisation pour développement local.
- Serveur web Python **prêt au déploiement** utilisant Uvicorn et Gunicorn.
- Backend Python **FastAPI** :
    - **Rapide** : Très hautes performances, comparables à **NodeJS** ou **Go** (grâce à Starlette et Pydantic).
    - **Intuitif** : Excellent support des éditeurs. Complétion partout. Moins de temps passé à déboguer.
    - **Facile** : Fait pour être facile à utiliser et apprendre. Moins de temps passé à lire de la documentation.
    - **Concis** : Minimise la duplication de code. Plusieurs fonctionnalités à chaque déclaration de paramètre.
    - **Robuste** : Obtenez du code prêt pour être utilisé en production. Avec de la documentation automatique interactive.
    - **Basé sur des normes** : Basé sur (et totalement compatible avec) les normes ouvertes pour les APIs : OpenAPI et JSON Schema.
    - **Et bien d'autres fonctionnalités** comme la validation automatique, la sérialisation, l'authentification avec OAuth2 JWT tokens, etc.
- Hashage de **mots de passe sécurisé** par défaut.
- Authentification par **jetons JWT**.
- Modèles **SQLAlchemy** (indépendants des extensions Flask, afin qu'ils puissent être utilisés directement avec des *workers* Celery).
- Modèle de démarrages basiques pour les utilisateurs (à modifier et supprimer au besoin).
- Migrations **Alembic**.
- **CORS** (partage des ressources entre origines multiples, ou *Cross Origin Resource Sharing*).
- *Worker* **Celery** pouvant importer et utiliser les modèles et le code du reste du backend.
- Tests du backend REST basés sur **Pytest**, intégrés dans Docker, pour que vous puissiez tester toutes les interactions de l'API indépendamment de la base de données. Étant exécutés dans Docker, les tests peuvent utiliser un nouvel entrepôt de données créé de zéro à chaque fois (vous pouvez donc utiliser ElasticSearch, MongoDB, CouchDB, etc. et juste tester que l'API fonctionne).
- Intégration Python facile avec **Jupyter Kernels** pour le développement à distance ou intra-Docker avec des extensions comme Atom Hydrogen ou Visual Studio Code Jupyter.
- Frontend **Vue** :
    - Généré avec Vue CLI.
    - Gestion de l'**Authentification JWT**.
    - Page de connexion.
    - Après la connexion, page de tableau de bord principal.
    - Tableau de bord principal avec création et modification d'utilisateurs.

    - Modification de ses propres caractéristiques utilisateur.
    - **Vuex**.
    - **Vue-router**.
    - **Vuetify** pour de magnifiques composants *material design*.
    - **TypeScript**.
    - Serveur Docker basé sur **Nginx** (configuré pour être facilement manipulé avec Vue-router).
    - Utilisation de *Docker multi-stage building*, pour ne pas avoir besoin de sauvegarder ou *commit* du code compilé.
    - Tests frontend exécutés à la compilation (pouvant être désactivés).
    - Fait aussi modulable que possible, pour pouvoir fonctionner comme tel, tout en pouvant être utilisé qu'en partie grâce à Vue CLI.
- **PGAdmin** pour les bases de données PostgreSQL, facilement modifiable pour utiliser PHPMYAdmin ou MySQL.
- **Flower** pour la surveillance de tâches Celery.
- Équilibrage de charge entre le frontend et le backend avec **Traefik**, afin de pouvoir avoir les deux sur le même domaine, séparés par chemins, mais servis par différents conteneurs.
- Intégration Traefik, comprenant la génération automatique de certificat **HTTPS** Let's Encrypt.
- GitLab **CI** (intégration continue), comprenant des tests pour le frontend et le backend.

Full Stack FastAPI Couchbase

GitHub : https://github.com/tiangolo/full-stack-fastapi-couchbase

⚠ **ATTENTION** ⚠

Si vous démarrez un nouveau projet de zéro, allez voir les alternatives au début de cette page.

Par exemple, le générateur de projet Full Stack FastAPI PostgreSQL peut être une meilleure alternative, étant activement maintenu et utilisé et comprenant toutes les nouvelles fonctionnalités et améliorations.

Vous êtes toujours libre d'utiliser le générateur basé sur Couchbase si vous le voulez, cela devrait probablement fonctionner correctement, et si vous avez déjà un projet généré en utilisant ce dernier, cela devrait fonctionner aussi (et vous l'avez déjà probablement mis à jour suivant vos besoins).

Vous pouvez en apprendre plus dans la documentation du dépôt GithHub.

Full Stack FastAPI MongoDB

...viendra surement plus tard, suivant le temps que j'ai. 😅 🎉

Modèles d'apprentissage automatique avec spaCy et FastAPI

GitHub : https://github.com/microsoft/cookiecutter-spacy-fastapi

Modèles d'apprentissage automatique avec spaCy et FastAPI - Fonctionnalités

- Intégration d'un modèle NER **spaCy**.
- Formatage de requête pour **Azure Cognitive Search**.
- Serveur Python web **prêt à utiliser en production** utilisant Uvicorn et Gunicorn.
- Déploiement CI/CD Kubernetes pour **Azure DevOps** (AKS).
- **Multilangues**. Choisissez facilement l'une des langues intégrées à spaCy durant la mise en place du projet.

- **Facilement généralisable** à d'autres bibliothèques similaires (Pytorch, Tensorflow), et non juste spaCy.