

# DevCraft AI - Prompt de Développement Cursor

## RÈGLE ABSOLUE

JAMAIS modifier, toucher ou altérer le fichier `main.py` - Il est VERROUILLÉ et parfaitement fonctionnel.

## Contexte du Projet

### Architecture Existante (V17) - INTOUCHABLE

- **main.py** : Pipeline CrewAI complet avec 5 agents (Architecte → Développeur → QA Code → QA Fonctionnel → DevOps)
- **Pipeline validé** : Architecture → Développement → Review → Tests fonctionnels → Git versioning
- **Stack** : CrewAI + LangChain (Claude Opus 4, Gemini 1.5 Pro, GPT-4o-mini)
- **Fonctionnel** : Génère des apps React+Express complètes, testées et versionnées

### Interface Web Créée

- **DevCraft AI** : Interface conversationnelle style Claude.ai
- **Workflow** : Description projet → Sélection features → Analyse IA → Génération
- **Intelligence contextuelle** : Détection automatique du type de projet selon mots-clés

## Mission : Créer l'Écosystème Complet

### 1. API Backend Flask/FastAPI (PRIORITÉ 1)

Fichier : `api/server.py`

#### Objectifs :

- Recevoir les données de l'interface web (projectData JSON)
- Déclencher main.py avec les paramètres enrichis
- Streaming des logs CrewAI vers le frontend
- Gestion d'états (en cours, terminé, erreur)

#### Spécifications :

python

*# Endpoints requis*

POST /api/analyze *# Analyse du projet utilisateur*

POST /api/generate *# Lance la génération CrewAI*

GET /api/status/{job\_id} *# Status temps réel*

GET /api/logs/{job\_id} *# Stream des logs*

GET /api/download/{job\_id} *# Téléchargement projet ZIP*

### Intégration main.py :

- Subprocess pour exécuter main.py avec arguments
- Parsing des outputs CrewAI
- Gestion des erreurs et timeouts

## 2. Interface Web Production (PRIORITÉ 2)

Dossier : `frontend/`

### Améliorations :

- Intégration API backend
- Progress bar temps réel avec WebSockets
- Téléchargement automatique du projet généré
- Gestion d'erreurs UX
- Analytics simples (projets créés, types populaires)

## 3. Templates Enrichis (PRIORITÉ 3)

Fichier : `templates/enhanced_templates.py`

### Extension du ProjectTemplateManager :

- Templates spécialisés selon l'analyse IA :
  - `fishing-platform` : Géolocalisation + couleurs océan
  - `ecommerce-stripe` : Paiements + inventory
  - `blog-seo` : CMS + optimisation SEO
  - `realtime-chat` : Socket.io + authentification

## 4. Déploiement Automatisé (PRIORITÉ 4)

Fichiers : `deploy/` + `docker/`

### Stack de déploiement :

- Railway.app integration (Procfile, railway.json)
- Docker containerization
- CI/CD GitHub Actions
- Variables d'environnement sécurisées

## 🎨 Spécifications Techniques

### Qualité Code (Top 1%)

- **Type hints** partout en Python
- **Error handling** robuste avec logging
- **Tests unitaires** pour chaque endpoint
- **Documentation** inline et README détaillé

### Architecture

```
devcraft-ai/
├── main.py                # ❌ INTOUCHABLE
├── api/
│   ├── server.py         # Backend API
│   ├── models.py         # Modèles Pydantic
│   └── utils.py          # Helpers
├── frontend/
│   ├── index.html        # Interface améliorée
│   └── assets/           # CSS/JS/Images
├── templates/
│   ├── enhanced_templates.py # Templates enrichis
│   └── generators/       # Générateurs spécialisés
├── deploy/
│   ├── Dockerfile
│   ├── railway.json
│   └── docker-compose.yml
└── tests/                # Tests complets
```

### Standards de Développement

- **FastAPI** avec async/await pour performance
- **Pydantic** pour validation des données
- **WebSockets** pour streaming temps réel
- **Logging structuré** avec rotation
- **Rate limiting** pour éviter les abus

## 💡 Fonctionnalités Innovantes

## Intelligence Contextuelle

- **Analyse sémantique** des descriptions projet
- **Suggestions automatiques** de features manquantes
- **Optimisations** selon le domaine détecté







## UX Avancée

- **Prévisualisation** du code avant génération
- **Historique** des projets générés
- **Templates favoris** personnalisables
- **Export** multi-formats (ZIP, Git repo, Docker)

## Prochaines Étapes Recommandées

1. **Créer l'API backend** qui wrappe main.py
2. **Connecter l'interface** web à l'API
3. **Ajouter les templates** enrichis selon domaines
4. **Déployer** sur Railway avec CI/CD

## Critères de Succès

-  main.py reste intact et fonctionnel
-  Interface web connectée et reactive
-  Génération temps réel avec feedback
-  Projets téléchargeables immédiatement
-  Templates intelligents selon contexte
-  Déployé et accessible en production

---

**Note :** Respecter la philosophie du projet - qualité professionnelle, workflow fluide, résultats immédiatement exploitables.