
# Plan de Travail

Équipe RawFinance Pro

Hackathon Rawbank

## BrainSoft

**Équipe :**

Debuze
Simbi
Strategie
Moza
Isambo

**Durée : 48 heures**

### Composition de l'Équipe :

— Frontend (React) : [Nom du développeur]
— Backend (Node.js) : Debuze
— IA/ML (Python) : [Nom du développeur]

2026

# Table des matières

# Stratégie de Communication

## Outils de Collaboration

### GitHub Repository (Obligatoire)

- **Repository unique** : `rawfinance-pro-hackathon`
- **Branches** :
  - `main` : Code de production
  - `develop` : Branche de développement principale
  - `feature/frontend-*` : Features frontend
  - `feature/backend-*` : Features backend
  - `feature/ml-*` : Modèles IA
- **Commits fréquents** : Au moins toutes les 2-3 heures
- **Pull Requests** : Pour chaque feature majeure avant merge

### Discord/Slack/WhatsApp (Communication Temps Réel)

- **Canal général** : Discussions quotidiennes
- **Canal #frontend** : Questions spécifiques frontend
- **Canal #backend** : Questions spécifiques backend
- **Canal #ml** : Questions spécifiques IA
- **Canal #blockers** : Blocages urgents
- **Stand-up quotidien** : 9h et 18h (15 min chacun)

### Notion/Trello (Gestion de Tâches)

- **Board Kanban** avec colonnes :
  - Backlog
  - En cours
  - Terminé
  - Bloqué
- **Tâches assignées** avec deadlines
- **Dépendances** clairement marquées

### Google Drive/Notion (Documentation)

- **API Documentation** : Endpoints backend documentés
- **Modèles de Données** : Schémas Supabase
- **Design Mockups** : Maquettes UI/UX
- **Rapport Final** : Documentation du projet

### Supabase Dashboard (Base de Données)

— **Accès partagé** : Tous les 3 membres ont accès
— **Migrations** : Documentées dans le repo
— **Seed Data** : Scripts partagés

## Protocole de Communication

### Communication Quotidienne

— **9h00** : Stand-up matin (15 min)
  — Ce que j'ai fait hier
  — Ce que je fais aujourd'hui
  — Blocages éventuels
— **13h00** : Point mi-journée (10 min)
  — Avancement
  — Aide nécessaire
— **18h00** : Stand-up soir (15 min)
  — Bilan de la journée
  — Plan pour demain
  — Blocages résolus

### Communication Asynchrone

— **Messages Discord** : Réponse sous 30 min pendant heures de travail
— **Issues GitHub** : Pour bugs et features
— **Pull Requests** : Review obligatoire avant merge

### Gestion des Blocages

1. **Niveau 1** : Blocage simple → Discord #blockers
2. **Niveau 2** : Blocage moyen → Appel rapide (15 min)
3. **Niveau 3** : Blocage critique → Réunion d'urgence (30 min)

# Plan Frontend (React)

## ÉTAPE 1 : Setup et Configuration (2-3 heures)

### Sous-étapes

1. **Initialisation du projet**
  — Créer projet React avec Vite : `npm create vite@latest frontend - -template react`
  — Installer dépendances : React Router, Axios, Tailwind CSS
  — Configurer ESLint et Prettier

2. **Structure des dossiers**

```
frontend/
|-- src/
|   |-- components/      # Composants reutilisables
|   |-- pages/           # Pages principales
|   |-- hooks/           # Custom hooks
|   |-- services/        # Services API
|   |-- contexts/        # Context API (auth, etc.)
|   |-- utils/           # Utilitaires
|   \-- assets/          # Images, styles
|-- public/
\-- package.json
```

3. **Configuration environnement**
   - Créer `.env` avec variables :

```
VITE_API_URL=http://localhost:3001
VITE_SUPABASE_URL=votre_url_supabase
VITE_SUPABASE_ANON_KEY=votre_anon_key
```

4. **Git Setup**
   - Initialiser repo Git
   - Créer branche `feature/frontend-setup`
   - Premier commit

**Livrables** : Projet React fonctionnel, structure créée, environnement configuré
**Communication** : Informer l'équipe que le setup est prêt, partager les variables d'environnement nécessaires

## ÉTAPE 2 : Authentification et Layout de Base (4-5 heures)

### Sous-étapes

1. **Page de connexion/inscription**
   - Composant Login avec email/phone
   - Composant Register multi-étapes
   - Intégration Supabase Auth
   - Gestion des erreurs
2. **Layout principal**
   - Header avec navigation
   - Sidebar (si nécessaire)
   - Footer
   - Responsive design
3. **Context d'authentification**
   - Créer `AuthContext.jsx`
   - Gestion session utilisateur
   - Protection des routes

4. **Routes principales**
   - `/` : Page d'accueil
   - `/login` : Connexion
   - `/register` : Inscription
   - `/dashboard` : Tableau de bord (protégé)
   - `/profile` : Profil utilisateur
5. **Services API de base**
   - Créer `api.js` avec Axios
   - Configuration intercepteurs
   - Gestion tokens

**Livrables** : Authentification fonctionnelle, layout de base, routes configurées
**Communication** :

- Demander à Backend : endpoints d'authentification disponibles
- Informer : structure des routes pour intégration backend

**Dépendances** : Backend doit avoir les endpoints `/api/auth/login` et `/api/auth/register` prêts

## ÉTAPE 3 : Pages Principales - Particuliers (6-7 heures)

**Sous-étapes**

1. **Dashboard Particulier**
   - Affichage score actuel (0-1000)
   - Graphique évolution du score
   - Montant de crédit éligible
   - Badges et achievements
2. **Page de scoring**
   - Affichage des facteurs de score
   - Barres de progression par facteur :
     - Mobile Money (40%)
     - Téléphone (20%)
     - Social (20%)
     - Transactionnel (15%)
     - Communautaire (5%)
   - Conseils pour améliorer le score
3. **Connexion Mobile Money**
   - Formulaire de connexion
   - Upload de fichier (CSV/PDF) pour historique
   - Affichage historique mocké (en attendant vraies données)
   - Visualisation des transactions
4. **Connexion données téléphoniques**
   - Formulaire déclaratif (fréquence recharges, etc.)

   - Ou upload SMS si possible
   - Affichage statistiques
5. **Connexion réseaux sociaux**
   - Boutons OAuth (Facebook, LinkedIn)
   - Affichage données sociales connectées
   - Gestion consentement

**Livrables** : Pages particuliers complètes, intégration données mockées
**Communication** :

- Demander à Backend : endpoints pour récupérer score, historique Mobile Money
- Demander à IA : format des données de score à afficher
- Informer : structure des composants pour intégration

**Dépendances** : Backend doit avoir endpoints `/api/users/score`, `/api/mobile-money/transactio`

## ÉTAPE 4 : Pages Principales - Entrepreneurs (6-7 heures)

**Sous-étapes**

1. **Dashboard Entrepreneur**
   - Score business (0-1000)
   - Revenus moyens mensuels
   - Graphique ventes (6-12 mois)
   - Prédiction revenus futurs (si IA prête)
   - Détection saisonnalité
2. **Page profil entreprise**
   - Formulaire enregistrement entreprise
   - Secteur d'activité
   - Localisation
   - Historique business
3. **Analyse transactions business**
   - Graphique ventes vs achats
   - Analyse de croissance
   - Détection patterns
   - Catégorisation automatique
4. **Types de crédit business**
   - Crédit trésorerie
   - Crédit investissement
   - Crédit saisonnier
   - Ligne de crédit renouvelable
5. **Simulation de crédit**
   - Formulaire demande crédit
   - Calcul mensualités
   - Affichage conditions

— Soumission demande

**Livrables** : Pages entrepreneurs complètes, intégration scoring transactionnel
**Communication** :

— Demander à Backend : endpoints scoring transactionnel, prédictions revenus

— Demander à IA : format données prédictions, saisonnalité

— Informer : UI prête pour intégration modèles IA

**Dépendances** : Backend endpoints `/api/business/score`, `/api/business/revenue-prediction`

## ÉTAPE 5 : Demande de Crédit et Finalisation (5-6 heures)

**Sous-étapes**

1. **Workflow demande de crédit**
   — Page choix type de crédit
   — Formulaire adaptatif (particulier vs entrepreneur)
   — Simulation en temps réel
   — Soumission avec confirmation
2. **Suivi crédit actif**
   — Affichage crédit en cours
   — Plan de remboursement
   — Historique paiements
   — Prochaines échéances
3. **Paiement et remboursement**
   — Intégration Mobile Money pour paiement
   — Confirmation paiement
   — Mise à jour automatique score
4. **Notifications**
   — Alertes SMS/USSD (simulation)
   — Notifications in-app
   — Rappels remboursement
5. **Tests et polish**
   — Tests sur différents navigateurs
   — Responsive mobile/tablette
   — Gestion erreurs complète
   — Loading states
   — Animations légères

**Livrables** : Application frontend complète et fonctionnelle
**Communication** :

— Demander à Backend : endpoints finaux, tests d'intégration

— Informer : Application prête pour démo finale

**Dépendances** : Tous les endpoints backend doivent être disponibles

# Plan Backend (Node.js) - Debuze

## ÉTAPE 1 : Setup et Configuration (2-3 heures)

**Sous-étapes**

1. **Initialisation du projet**
   - Créer projet Node.js : `npm init -y`
   - Installer Express, CORS, dotenv
   - Installer Supabase client : `@supabase/supabase-js`
   - Structure dossiers :

```
backend/
|-- routes/           # Routes API
|-- controllers/      # Logique metier
|-- services/         # Services (ML, etc.)
|-- middleware/       # Middlewares
|-- models/           # Modeles de donnees
|-- utils/            # Utilitaires
\-- server.js         # Point d'entree
```

2. **Configuration Supabase**
   - Connexion à Supabase
   - Créer fichier `.env` :

```
PORT=3001
SUPABASE_URL=votre_url
SUPABASE_SERVICE_KEY=votre_service_key
SUPABASE_ANON_KEY=votre_anon_key
```

3. **Configuration Express**
   - Setup serveur Express
   - Middleware CORS
   - Middleware JSON parser
   - Gestion erreurs globale
4. **Git Setup**
   - Initialiser repo
   - Créer branche `feature/backend-setup`
   - Premier commit

**Livrables** : Serveur Express fonctionnel, connexion Supabase établie
**Communication** :

- Partager variables d'environnement avec équipe
- Informer : Structure API prévue
- Demander à Frontend : Format données attendu

## ÉTAPE 2 : Authentification et Gestion Utilisateurs (4-5 heures)

**Sous-étapes**

1. **Routes d'authentification**
   - `POST /api/auth/register` : Inscription
   - `POST /api/auth/login` : Connexion
   - `POST /api/auth/logout` : Déconnexion
   - `GET /api/auth/me` : Profil utilisateur actuel
   - Intégration Supabase Auth
2. **Middleware d'authentification**
   - Vérification token JWT
   - Extraction user ID
   - Protection routes
3. **Gestion profils utilisateurs**
   - `GET /api/users/:id` : Récupérer profil
   - `PUT /api/users/:id` : Mettre à jour profil
   - `POST /api/users/:id/profile-type` : Définir type (particulier/entrepreneur)
   - Stockage dans Supabase
4. **Gestion documents**
   - `POST /api/users/:id/documents` : Upload CNI, selfie
   - Intégration OCR (Tesseract.js ou API)
   - Vérification identité
5. **Tests endpoints**
   - Tester avec Postman/Thunder Client
   - Vérifier intégration Supabase

**Livrables** : Authentification complète, gestion utilisateurs fonctionnelle
**Communication** :

- Partager documentation API avec Frontend
- Informer : Endpoints disponibles pour tests
- Demander à Frontend : Format données inscription/connexion

**Dépendances** : Supabase configuré avec tables `users`, `user_profiles`

## ÉTAPE 3 : APIs Mobile Money et Données Téléphoniques (5-6 heures)

**Sous-étapes**

1. **Routes Mobile Money**
   - `GET /api/mobile-money/accounts/:phoneNumber` : Récupérer comptes
   - `GET /api/mobile-money/transactions/:phoneNumber` : Historique transactions
   - `GET /api/mobile-money/balance/:phoneNumber` : Solde actuel

- `POST /api/mobile-money/connect` : Connecter compte (simulation)
- Calcul statistiques (moyennes, fréquences, etc.)

2. **Routes activité téléphonique**
   - `GET /api/phone-activity/topup-history/:phoneNumber` : Historique recharges
   - `GET /api/phone-activity/stats/:phoneNumber` : Statistiques complètes
   - Calcul régularité, fréquence, ancienneté
3. **Services de calcul**
   - Fonction calcul score Mobile Money
   - Fonction calcul score téléphone
   - Fonction agrégation statistiques
   - Normalisation données
4. **Intégration données mockées**
   - Lire depuis Supabase (tables mockées)
   - Format réponse standardisé
   - Gestion cas données manquantes
5. **Documentation API**
   - Documenter tous les endpoints
   - Exemples requêtes/réponses
   - Codes d'erreur

**Livrables** : APIs Mobile Money et téléphone fonctionnelles
**Communication** :

- Partager documentation API mise à jour
- Informer : Format données disponibles
- Demander à IA : Features nécessaires pour scoring

**Dépendances** : Données mockées dans Supabase (tables `mobile_money_transactions`, `phone_topup_history`)

## ÉTAPE 4 : Intégration IA et Scoring (6-7 heures)

**Sous-étapes**

1. **Service détection profil**
   - Créer `services/ml/profileDetection.js`
   - Intégrer modèle IA (ONNX.js ou API Python)
   - Endpoint `POST /api/ml/detect-profile`
   - Retourner : `{profile: 'particulier'|'entrepreneur', confidence: 0-1}`
2. **Service scoring alternatif**
   - Créer `services/ml/alternativeScoring.js`
   - Intégrer modèle scoring particuliers
   - Endpoint `POST /api/ml/calculate-alternative-score`
   - Retourner score 0-1000 + facteurs détaillés
3. **Service scoring transactionnel**

- Créer `services/ml/transactionalScoring.js`
- Intégrer modèle scoring entrepreneurs
- Endpoint `POST /api/ml/calculate-transactional-score`
- Retourner score + analyse business

4. **Service prédiction revenus**
   - Créer `services/ml/revenuePrediction.js`
   - Intégrer modèle LSTM (TensorFlow.js ou API)
   - Endpoint `POST /api/ml/predict-revenue`
   - Retourner prédictions 3-6 mois futurs
5. **Service détection fraude**
   - Créer `services/ml/fraudDetection.js`
   - Intégrer modèle détection anomalies
   - Endpoint `POST /api/ml/detect-fraud`
   - Retourner score de risque 0-100
6. **Routes scoring utilisateur**
   - `GET /api/users/:id/score` : Récupérer score actuel
   - `POST /api/users/:id/calculate-score` : Recalculer score
   - `GET /api/users/:id/score-factors` : Détails facteurs score
   - Stockage dans Supabase (`credit_scores`)

**Livrables** : Intégration IA complète, scoring fonctionnel
**Communication** :

- **CRITIQUE** : Coordonner avec développeur IA
  - Demander : Modèles prêts ? Format d'entrée/sortie ?
  - Informer : Structure données disponibles
  - Tester : Intégration modèles ensemble
- Partager endpoints ML avec Frontend
- Documenter : Format données pour IA

**Dépendances** :

- **CRITIQUE** : Modèles IA doivent être prêts (format JavaScript ou API Python)
- Tables Supabase : `credit_scores`, `score_factors`

## ÉTAPE 5 : APIs Crédit et Finalisation (6-7 heures)

**Sous-étapes**

1. **Routes demande de crédit**
   - `POST /api/credits/apply` : Soumettre demande
   - Validation données
   - Appel modèles IA (scoring + fraude)
   - Décision automatique/manuelle
   - Stockage dans `credit_applications`
2. **Routes gestion crédits**

— `GET /api/credits/:id` : Détails crédit
— `GET /api/credits/user/:userId` : Crédits d'un utilisateur
— `GET /api/credits/:id/repayment-plan` : Plan remboursement
— `POST /api/credits/:id/repay` : Enregistrer paiement

3. **Service recommandation crédit**
   — Calcul montant optimal selon score
   — Suggestion durée et taux
   — Endpoint `GET /api/credits/recommendations/:userId`
4. **Notifications et alertes**
   — Service SMS/USSD (simulation)
   — Notifications remboursement
   — Rappels échéances
   — Webhooks pour intégration
5. **Tests finaux et optimisation**
   — Tests tous les endpoints
   — Gestion erreurs complète
   — Validation données
   — Performance (cache si nécessaire)
   — Documentation API complète

**Livrables** : Backend complet et fonctionnel
**Communication** :

— Partager documentation API finale
— Informer : Tous endpoints disponibles
— Coordonner : Tests d'intégration avec Frontend
— Préparer : Démo backend pour jury

**Dépendances** : Tous les modèles IA doivent être intégrés

# Plan IA/ML (Python)

## ÉTAPE 1 : Setup et Extraction Données (3-4 heures)

### Sous-étapes

1. **Setup environnement Python**
   — Créer environnement virtuel : `python -m venv venv`
   — Installer dépendances :

```
pandas numpy scikit-learn tensorflow
jupyter notebook supabase
matplotlib seaborn
```

   — Créer structure :

```
ml/
|-- notebooks/              # Jupyter notebooks
|-- scripts/                # Scripts Python
|-- datasets/               # CSV datasets
|-- models/                 # Modeles sauvegardes
\-- requirements.txt
```

2. **Connexion Supabase**
   - Configurer client Supabase Python
   - Tester connexion
   - Créer fichier `.env` avec credentials
3. **Script extraction données**
   - Créer `scripts/extract_data.py`
   - Extraire données Mobile Money → CSV
   - Extraire données téléphone → CSV
   - Extraire données utilisateurs → CSV
   - Sauvegarder dans `datasets/`
4. **Analyse exploratoire**
   - Notebook `01_data_exploration.ipynb`
   - Visualiser distributions
   - Identifier patterns
   - Détecter outliers
   - Statistiques descriptives
5. **Préparation datasets**
   - Nettoyage données
   - Feature engineering
   - Création labels (si nécessaire)
   - Split train/validation/test

**Livrables** : Datasets propres, prêts pour entraînement
**Communication** :

- Informer équipe : Structure données extraites
- Demander à Backend : Format données attendu
- Partager : Statistiques exploratoires intéressantes

**Dépendances** : Données mockées dans Supabase (créées par Backend ou script séparé)

## ÉTAPE 2 : Modèle Détection Profil (4-5 heures)

### Sous-étapes

1. **Préparation dataset**
   - Charger `profile_detection_dataset.csv`
   - Sélectionner features pertinentes

- Encoder labels (particulier=0, entrepreneur=1)
- Split train/test (80/20)

2. **Entraînement modèle**
   - Notebook `02_profile_detection.ipynb`
   - Tester plusieurs algorithmes :
     - Random Forest
     - Gradient Boosting
     - SVM
   - Hyperparameter tuning
   - Sélection meilleur modèle
3. **Évaluation**
   - Métriques : Accuracy, Precision, Recall, F1
   - Matrice de confusion
   - Feature importance
   - Validation croisée
4. **Export modèle**
   - Sauvegarder meilleur modèle
   - Format : `.pkl` (scikit-learn) ou `.h5` (TensorFlow)
   - Convertir en format JavaScript si possible :
     - ONNX pour scikit-learn
     - TensorFlow.js pour TensorFlow
   - Sauvegarder métadonnées (features, normalisation)
5. **Documentation**
   - Performance du modèle
   - Features utilisées
   - Format entrée/sortie
   - Instructions intégration

**Livrables** : Modèle détection profil prêt, documentation
**Communication** :

- **CRITIQUE** : Coordonner avec Backend
  - Informer : Format modèle (ONNX, TensorFlow.js, ou API Python)
  - Partager : Format entrée/sortie attendu
  - Tester : Intégration avec Backend
- Partager : Performance modèle, features importantes

**Dépendances** : Dataset `profile_detection_dataset.csv` prêt

## ÉTAPE 3 : Modèles Scoring (6-7 heures)

**Sous-étapes**

1. **Scoring alternatif (particuliers)**
   - Notebook `03_alternative_scoring.ipynb`

- Dataset : `alternative_scoring_dataset.csv`
- Features : Mobile Money, téléphone, social, transactionnel, communautaire
- Modèle : Gradient Boosting Regressor (score 0-1000)
- Entraînement et évaluation (MAE, $R^2$)
- Export modèle + métadonnées

2. **Scoring transactionnel (entrepreneurs)**
   - Notebook `04_transactional_scoring.ipynb`
   - Dataset : `transactional_scoring_dataset.csv`
   - Features : Ventes, croissance, stabilité, saisonnalité, achats
   - Modèle : Gradient Boosting Regressor
   - Entraînement et évaluation
   - Export modèle + métadonnées
3. **Calcul facteurs de score**
   - Fonction pour expliquer le score
   - Contribution de chaque facteur
   - Sauvegarder dans format lisible (JSON)
4. **Tests modèles**
   - Tester sur données de validation
   - Vérifier cohérence scores
   - Ajuster si nécessaire

**Livrables** : Modèles scoring alternatif et transactionnel prêts
**Communication** :

- **CRITIQUE** : Coordonner avec Backend
  - Partager : Modèles et format intégration
  - Informer : Format données entrée
  - Tester : Appels API/import modèles
- Partager : Performance modèles, facteurs importants

**Dépendances** : Datasets scoring prêts

## ÉTAPE 4 : Prédiction Revenus et Détection Fraude (5-6 heures)

**Sous-étapes**

1. **Prédiction revenus (LSTM)**
   - Notebook `05_revenue_prediction.ipynb`
   - Préparer séries temporelles (12-24 mois par utilisateur)
   - Créer séquences (6 mois → prédire 7ème)
   - Modèle LSTM avec TensorFlow/Keras
   - Entraînement et évaluation (MAE, MAPE)
   - Export TensorFlow.js ou API Python
2. **Détection saisonnalité**
   - Analyse time series decomposition

- Détection cycles saisonniers
- Identification mois de pic
- Fonction pour calculer facteur saisonnier

3. **Détection fraude**
   - Notebook `06_fraud_detection.ipynb`
   - Dataset : `fraud_detection_dataset.csv`
   - Features : Incohérences, patterns suspects, etc.
   - Modèle : Isolation Forest ou Autoencoder
   - Entraînement et évaluation
   - Export modèle
4. **Service API Python (optionnel)**
   - Si modèles complexes à convertir :
     - Créer Flask/FastAPI service
     - Endpoints pour chaque modèle
     - Documentation API
     - Déployer localement ou sur serveur

**Livrables** : Modèles prédiction et fraude prêts, service API si nécessaire
**Communication** :

- **CRITIQUE** : Coordonner avec Backend
  - Décider : Intégration directe (JavaScript) ou API Python ?
  - Si API Python : Partager URL et endpoints
  - Tester : Intégration complète
- Informer : Performance modèles, limitations

**Dépendances** : Datasets prédiction et fraude prêts

## ÉTAPE 5 : Intégration et Optimisation (4-5 heures)

**Sous-étapes**

1. **Tests d'intégration**
   - Tester tous les modèles avec Backend
   - Vérifier format données
   - Valider résultats
   - Corriger bugs
2. **Optimisation modèles**
   - Réduire taille si nécessaire
   - Optimiser performance
   - Cache prédictions si possible
3. **Documentation complète**
   - Documentation chaque modèle
   - Format entrée/sortie
   - Instructions déploiement

   - Performance et limitations

4. **Préparation démo**
   - Exemples de prédictions
   - Visualisations intéressantes
   - Explication modèles pour jury

5. **Backup et versioning**
   - Sauvegarder tous les modèles
   - Versionner dans Git
   - Documenter versions

**Livrables** : Tous modèles intégrés et fonctionnels
**Communication** :

- Finaliser : Intégration avec Backend
- Partager : Documentation complète
- Préparer : Présentation IA pour jury
- Coordonner : Tests finaux ensemble

**Dépendances** : Backend prêt pour intégration

# Synchronisation et Intégration

## Timeline Synchronisée (48 heures)

**Heures 0-12 : Setup et Fondations**

- **Frontend** : Setup + Auth + Layout
- **Backend** : Setup + Auth + APIs de base
- **IA** : Setup + Extraction données + Exploration

**Point de synchronisation H12** :

- Vérifier : Authentification fonctionne end-to-end
- Vérifier : Connexion Supabase partout
- Décider : Format données standardisé

**Heures 12-24 : Développement Parallèle**

- **Frontend** : Pages particuliers
- **Backend** : APIs Mobile Money + Téléphone
- **IA** : Modèle détection profil + Scoring alternatif

**Point de synchronisation H24** :

- Tester : Intégration Frontend-Backend (particuliers)
- Tester : Modèle détection profil avec Backend
- Vérifier : Scoring alternatif fonctionne

**Heures 24-36 : Intégration IA**

- **Frontend** : Pages entrepreneurs
- **Backend** : Intégration modèles IA
- **IA** : Modèles scoring transactionnel + Prédictions

**Point de synchronisation H36** :

- **CRITIQUE** : Intégration IA complète
- Tester : Tous les modèles fonctionnent
- Vérifier : Scoring transactionnel opérationnel

**Heures 36-48 : Finalisation et Tests**

- **Frontend** : Finalisation + Tests
- **Backend** : APIs crédit + Tests finaux
- **IA** : Optimisation + Documentation

**Point de synchronisation H48** :

- Tests complets end-to-end
- Démo finale préparée
- Documentation complète

## Protocole d'Intégration

**Intégration Frontend-Backend**

- Backend expose endpoints → Frontend consomme
- Format JSON standardisé
- Gestion erreurs cohérente
- Tests avec Postman avant intégration Frontend

**Intégration Backend-IA**

- **Option A** : Modèles JavaScript (ONNX.js, TensorFlow.js)
  - IA exporte modèles → Backend importe directement
  - Plus rapide, pas de service séparé
- **Option B** : API Python séparée
  - IA déploie service Flask/FastAPI
  - Backend appelle API Python
  - Plus flexible pour modèles complexes

**Décision à prendre ensemble** selon complexité modèles

**Tests d'Intégration**

- Tests unitaires chaque composant
- Tests d'intégration Frontend-Backend
- Tests d'intégration Backend-IA
- Tests end-to-end complets

## Gestion des Conflits Git

### Workflow Recommandé

1. **Pull avant push** : Toujours `git pull` avant `git push`
2. **Branches séparées** : Chaque membre sur sa branche feature
3. **Merge fréquent** : Merge dans `develop` toutes les 4-6 heures
4. **Résolution conflits** : Appel rapide si conflit complexe

### En cas de Conflit

1. Ne pas paniquer
2. Communiquer sur Discord immédiatement
3. Décider qui résout (généralement celui qui a touché le fichier en dernier)
4. Tester après résolution

# Checklist Finale et Objectifs

## Checklist Finale Avant Démo

### Frontend

— Toutes les pages fonctionnent
— Responsive mobile/tablette
— Gestion erreurs complète
— Loading states partout
— Design cohérent

### Backend

— Tous les endpoints fonctionnent
— Authentification sécurisée
— Intégration IA complète
— Gestion erreurs robuste
— Documentation API à jour

### IA

— Tous les modèles fonctionnent
— Intégration Backend validée
— Performance acceptable
— Documentation complète
— Exemples de prédictions

### Intégration

- Tests end-to-end passent
- Données cohérentes
- Pas de bugs critiques
- Démo fluide

## Objectifs Finaux

### MVP Fonctionnel

- Inscription/Connexion
- Détection profil automatique
- Calcul score (alternatif ou transactionnel)
- Demande de crédit
- Approbation automatique
- Suivi crédit

### Bonus si Temps

- Prédictions revenus
- Détection saisonnalité
- Recommandations intelligentes
- Gamification
- Notifications