

Guide Complet - Application Soakoja de A à Z

Table des Matières

1. [Vue d'ensemble de l'application](#)
 2. [Technologies utilisées](#)
 3. [Structure des fichiers](#)
 4. [Installation et démarrage](#)
 5. [Analyse détaillée du code](#)
 6. [Débogage étape par étape](#)
 7. [Problèmes courants et solutions](#)
 8. [Amélioration et maintenance](#)
-

Vue d'ensemble de l'application {#vue-densemble}

Qu'est-ce que Soakoja ?

Soakoja est une application web de gestion d'eau potable pour Madagascar. Elle permet de :

- Gérer les employés et leurs accès
- Planifier les activités de maintenance
- Suivre les ouvrages d'eau (forages, puits, sources)
- Gérer les points d'eau et abonnés
- Administrer des cartes sociales pour tarifs préférentiels

Architecture générale

Navigateur Web ←→ Serveur Node.js ←→ Base de données SQLite
... (Interface) (Logique métier) ... (Stockage données)

Technologies utilisées {#technologies}

1. Node.js

- **Qu'est-ce que c'est ?** Un environnement pour exécuter du JavaScript côté serveur
- **Pourquoi ?** Permet de créer des serveurs web avec JavaScript
- **Dans votre projet :** C'est le moteur qui fait tourner votre application

2. Express.js

- **Qu'est-ce que c'est ?** Un framework web pour Node.js
- **Pourquoi ?** Simplifie la création d'API et de routes web
- **Dans votre projet :** Gère les URLs (routes) et les requêtes HTTP

3. SQLite

- **Qu'est-ce que c'est ?** Une base de données légère stockée dans un fichier
- **Pourquoi ?** Facile à utiliser, pas besoin de serveur de base de données
- **Dans votre projet :** Stocke toutes les données (employés, plannings, etc.)

4. Sequelize

- **Qu'est-ce que c'est ?** Un ORM (Object-Relational Mapping)
- **Pourquoi ?** Permet de manipuler la base de données avec du JavaScript
- **Dans votre projet :** Interface entre votre code JavaScript et SQLite

5. EJS

- **Qu'est-ce que c'est ?** Un moteur de template
- **Pourquoi ?** Permet de créer des pages HTML dynamiques
- **Dans votre projet :** Génère les pages web avec les données

Structure des fichiers {#structure}

```
soakoja-app/
├── server.js..... # Fichier principal du serveur
├── setup.js..... # Script de configuration
├── package.json.... # Configuration du projet
├── soakoja_views.html # Templates HTML/EJS
├── database/..... # Dossier pour la base de données SQLite
├── views/..... # Templates EJS (créés par setup.js)
├── public/..... # Fichiers statiques (CSS, JS, images)
└── node_modules/... # Dépendances installées par npm
```

Rôle de chaque fichier :

server.js → Cœur de l'application **package.json** → Configuration et dépendances **setup.js** → Script d'initialisation **views/** → Pages web de l'interface utilisateur

Installation et démarrage {#installation}

Étape 1 : Prérequis

```
bash
```

```
# Vérifier si Node.js est installé  
node --version  
npm --version
```

Si pas installé, télécharger depuis : <https://nodejs.org/>

Étape 2 : Installation des dépendances

```
bash
```

```
# Dans le dossier de votre projet  
npm install
```

Que fait cette commande ?

- Lit le fichier `package.json`
- Télécharge toutes les dépendances listées
- Les place dans le dossier `node_modules/`

Étape 3 : Configuration initiale

```
bash
```

```
# Exécuter le script de setup  
node setup.js
```

Que fait ce script ?

- Crée la structure des dossiers
- Génère le fichier CSS
- Crée le fichier README

Étape 4 : Démarrage

```
bash
```

```
# Démarrer l'application  
npm start
```

Que se passe-t-il ?

1. Node.js exécute `server.js`
2. Le serveur se lance sur le port 3000

3. La base de données SQLite est créée/initialisée

4. Des données de test sont insérées

Étape 5 : Accès

Ouvrir dans le navigateur : <http://localhost:3000>

Comptes de test disponibles :

- **admin** / **admin123** (Administrateur niveau 4)
- **marie** / **marie123** (Caissière niveau 2)
- **jean** / **jean123** (Agent terrain niveau 1)

🔍 Analyse détaillée du code {#analyse}

1. Package.json - Configuration du projet

```
json

{
  "name": "soakoja-app",
  "version": "1.0.0",
  "description": "Application de gestion d'eau potable pour Soakoja Madagascar",
  "main": "server.js",
  "scripts": {
    "start": "node server.js",
    "dev": "nodemon server.js",
    "setup": "npm install && node setup.js"
  },
  "dependencies": {
    "express": "^4.18.2", // Framework web
    "sequelize": "^6.35.1", // ORM pour base de données
    "sqlite3": "^5.1.6", // Driver SQLite
    "bcrypt": "^5.1.1", // Cryptage mots de passe
    "express-session": "^1.17.3", // Gestion sessions
    "body-parser": "^1.20.2", // Parsing des requêtes
    "ejs": "^3.1.9" // Moteur de template
  }
}
```

Explication des dépendances :

- **express** : Le serveur web principal
- **sequelize + sqlite3** : Gestion de la base de données
- **bcrypt** : Sécurise les mots de passe

- **express-session** : Maintient les utilisateurs connectés
- **body-parser** : Lit les données des formulaires
- **ejs** : Crée les pages HTML dynamiques

2. Server.js - Analyse section par section

A. Importation des modules (lignes 1-6)

javascript

```
const express = require('express');
const session = require('express-session');
const bodyParser = require('body-parser');
const bcrypt = require('bcrypt');
const { Sequelize, DataTypes } = require('sequelize');
const path = require('path');
```

Que fait ce code ?

- Importe toutes les librairies nécessaires
- `require()` charge les modules installés via npm

B. Configuration de base (lignes 8-10)

javascript

```
const app = express();
const PORT = process.env.PORT || 3000;
```

Que fait ce code ?

- Crée une instance Express (notre serveur web)
- Définit le port (3000 par défaut, ou variable d'environnement)

C. Configuration de la base de données (lignes 12-17)

javascript

```
const sequelize = new Sequelize({
  dialect: 'sqlite',
  storage: 'database/soakoja.db',
  logging: false
});
```

Que fait ce code ?

- Configure Sequelize pour utiliser SQLite
- La base sera stockée dans `database/soakoja.db`
- `logging: false` désactive les logs SQL (pour plus de sécurité)

D. Configuration des sessions (lignes 19-25)

```
javascript

app.use(session({
  secret: 'soakoja-secret-key-2024',
  resave: false,
  saveUninitialized: false,
  cookie: { maxAge: 24 * 60 * 60 * 1000 }
}));
```

Que fait ce code ?

- Configure le système de sessions (pour rester connecté)
- `secret` : clé de cryptage des sessions
- `maxAge` : durée de validité (24 heures)

E. Middleware (lignes 27-31)

```
javascript

app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(express.static('public'));
app.set('view engine', 'ejs');
```

Que fait ce code ?

- `bodyParser` : lit les données des formulaires
- `express.static('public')` : sert les fichiers CSS/JS
- `view engine` : utilise EJS pour les templates

F. Modèles de base de données (lignes 33-95)

Modèle Employe (lignes 33-44) :

```
javascript
```

```

const Employe = sequelize.define('Employe', {
  id: { type: DataTypes.INTEGER, primaryKey: true, autoIncrement: true },
  nom: { type: DataTypes.STRING, allowNull: false },
  prenom: { type: DataTypes.STRING, allowNull: false },
  email: { type: DataTypes.STRING, unique: true, allowNull: false },
  mot_de_passe: { type: DataTypes.STRING, allowNull: false },
  niveau_acces: { type: DataTypes.INTEGER, defaultValue: 1 },
  // ... autres champs
});

```

Explication des types de données :

- `DataTypes.INTEGER` : Nombre entier
- `DataTypes.STRING` : Texte
- `DataTypes.DATE` : Date
- `DataTypes.ENUM` : Liste de valeurs prédéfinies
- `allowNull: false` : Champ obligatoire
- `unique: true` : Valeur unique dans la table
- `autoIncrement: true` : S'incrémente automatiquement

Les autres modèles suivent la même logique :

- **Ouvrage** : Infrastructure d'eau (forages, puits, etc.)
- **Planning** : Activités planifiées
- **PointEau** : Points de distribution
- **CarteSociale** : Cartes à tarif préférentiel

G. Relations entre tables (lignes 97-100)

javascript

```

Planning.belongsTo(Employe, { foreignKey: 'employe_id' });
Planning.belongsTo(Ouvrage, { foreignKey: 'ouvrage_id' });
PointEau.belongsTo(Ouvrage, { foreignKey: 'ouvrage_id' });

```

Que fait ce code ?

- Définit les liens entre les tables
- Un Planning appartient à un Employé et un Ouvrage
- Un Point d'eau appartient à un Ouvrage

H. Middleware d'authentification (lignes 102-115)

javascript

```
function requireAuth(req, res, next) {
... if (req.session && req.session.user) {
... return next();
...
}
res.redirect('/login');
}

function requireLevel(level) {
... return (req, res, next) => {
... if (req.session && req.session.user && req.session.user.niveau_acces >= level) {
... return next();
...
}
... res.status(403).send('Accès refusé');
...
};
}
```

Que fait ce code ?

- `requireAuth` : Vérifie si l'utilisateur est connecté
- `requireLevel` : Vérifie le niveau d'autorisation
- Si non autorisé → redirection ou erreur 403

I. Routes de l'application (lignes 117-225)

Route de connexion (GET /login) :

javascript

```
app.get('/login', (req, res) => {
... res.render('login', { error: null });
});
```

- Affiche la page de connexion
- `res.render()` utilise le template EJS

Route de connexion (POST /login) :

javascript

```

app.post('/login', async (req, res) => {
  try {
    const { email, password } = req.body;
    const user = await Employe.findOne({ where: { email, statut: 'actif' } });

    if (user && await bcrypt.compare(password, user.mot_de_passe)) {
      req.session.user = /* données utilisateur */;
      res.redirect('/dashboard');
    } else {
      res.render('login', { error: 'Email ou mot de passe incorrect' });
    }
  } catch (error) {
    res.render('login', { error: 'Erreur de connexion' });
  }
});

```

Que fait ce code ?

1. Récupère email/password du formulaire
2. Cherche l'utilisateur dans la base
3. Vérifie le mot de passe avec bcrypt
4. Si correct → sauvegarde la session et redirige
5. Si incorrect → affiche erreur

Autres routes importantes :

- `GET /dashboard` : Page d'accueil après connexion
- `GET /plannings` : Liste des plannings
- `POST /plannings` : Création d'un nouveau planning
- `GET /admin` : Zone d'administration (niveau 4 requis)

J. Initialisation de la base de données (lignes 227-298)

javascript

```

async function initializeDatabase() {
  try {
    await sequelize.sync({ force: true }); // Recrée toutes les tables
    console.log('✓ Base de données initialisée');

    // Création des employés de test
    const employees = [
      {
        nom: 'Admin',
        email: 'admin',
        mot_de_passe: await bcrypt.hash('admin123', 10),
        niveau_acces: 4
      },
      // ... autres employés
    ];
    ...

    for (const emp of employees) {
      await Employe.create(emp);
    }
    // ... création des autres données de test
  } catch (error) {
    console.error('✗ Erreur d\'initialisation:', error);
  }
}

```

Que fait ce code ?

1. `sequelize.sync({ force: true })` : Recrée toutes les tables
2. `bcrypt.hash()` : Crypte les mots de passe
3. Crée des données de test pour tous les modèles
4. Utilise des boucles `for` pour insérer les données

K. Démarrage du serveur (lignes 300-308)

javascript

```

app.listen(PORT, async () => {
  console.log(`🚀 Serveur Soakoja démarré sur http://localhost:${PORT}`);
  await initializeDatabase();
  console.log('📋 Comptes de test disponibles:');
  // ... affichage des comptes
});

```

Que fait ce code ?

1. Démarre le serveur sur le port spécifié
 2. Initialise la base de données
 3. Affiche les informations de connexion
-

Débogage étape par étape {#debogage}

1. Problèmes de démarrage

Erreur : "Cannot find module"

```
Error: Cannot find module 'express'
```

Solution :

```
bash  
# Réinstaller les dépendances  
npm install
```

Erreur : "Port already in use"

```
Error: listen EADDRINUSE: address already in use :::3000
```

Solutions :

```
bash  
# Option 1: Tuer le processus sur le port 3000  
npx kill-port 3000  
  
# Option 2: Changer le port  
set PORT=3001 && npm start
```

Erreur : "Database locked"

```
SQLITE_BUSY: database is locked
```

Solutions :

1. Fermer tous les programmes qui utilisent la base
2. Supprimer le fichier `database/soakoja.db`
3. Redémarrer l'application

2. Problèmes de connexion

Page de login ne s'affiche pas

Vérifications :

1. Le serveur est-il démarré ?

```
bash
```

Vérifier dans la console

 Serveur Soakoja démarré sur http://localhost:3000

2. L'URL est-elle correcte ? `http://localhost:3000`

3. Le port est-il libre ?

Impossible de se connecter avec admin/admin123

Débugger :

1. Vérifier que les données de test sont créées :

```
javascript
```

// Dans initializeDatabase(), ajouter des logs

`console.log('Création employé:', emp);`

2. Vérifier la table dans la base :

```
bash
```

Installer sqlite3 CLI

`npm install -g sqlite3`

Ouvrir la base

`sqlite3 database/soakoja.db`

Lister les employés

`SELECT * FROM Employes;`

3. Problème de cryptage mot de passe :

```
javascript
```

// Tester le hash

`const bcrypt = require('bcrypt');`

`bcrypt.compare('admin123', '$2b$10$...hash...').then(console.log);`

3. Erreurs de base de données

Table doesn't exist

SQLITE_ERROR: no such table: Employes

Solution :

javascript

```
// Dans server.js, changer force: true  
await sequelize.sync({ force: true, alter: true });
```

Foreign key constraint

SQLITE_CONSTRAINT: FOREIGN KEY constraint failed

Causes courantes :

1. Référence à un ID qui n'existe pas
2. Ordre de création des tables incorrect
3. Données de test invalides

Solution :

javascript

```
// Vérifier l'ordre de création  
await Employe.create(...); // D'abord les tables parent  
await Ouvrage.create(...);  
await Planning.create(...); // Ensuite les tables enfant
```

4. Erreurs de template

Template not found

Error: Failed to lookup view "login" in views directory

Solutions :

1. Vérifier que le dossier `views/` existe
2. Exécuter `node setup.js` pour créer la structure
3. Créer manuellement les fichiers manquants

Variables undefined dans template

```
ReferenceError: user is not defined
```

Solution :

```
javascript

// S'assurer de passer toutes les variables
res.render('template', {
  user: req.session.user,
  stats: stats,
  // ... autres variables nécessaires
});
```

5. Outils de débogage

Ajouter des logs de débogage

```
javascript

// Au début de chaque route
console.log('Route appelée:', req.path);
console.log("Utilisateur:", req.session.user);
console.log('Body:', req.body);

// Pour les erreurs
.catch(error => {
  console.error('Erreur détaillée:', error);
  res.status(500).send('Erreur serveur');
});
```

Utiliser nodemon pour le développement

```
bash

# Installer nodemon
npm install -g nodemon

# Démarrer avec recharge automatique
npm run dev
```

Debugger avec Chrome DevTools

```
bash
```

```
# Démarrer en mode debug  
node --inspect server.js
```

Puis ouvrir `chrome://inspect` dans Chrome

⚠️ Problèmes courants et solutions {#problemes}

1. L'application ne démarre pas

Symptômes :

- Erreur au lancement
- Page blanche
- Connexion impossible

Solutions étape par étape :

1. Vérifier Node.js :

```
bash  
  
node --version # Doit être >= 16  
npm --version # Doit être >= 8
```

2. Réinstaller proprement :

```
bash  
  
rm -rf node_modules/... # Supprimer dossier  
rm package-lock.json ... # Supprimer lock file  
npm install ..... # Réinstaller
```

3. Vérifier les permissions :

```
bash  
  
# Sur Windows  
npm config set cache --force  
  
# Sur Mac/Linux  
sudo npm install
```

2. Erreurs de base de données

Base corrompue :

```
bash
```

```
# Supprimer et recréer  
rm database/soakaja.db  
npm start
```

Tables manquantes :

```
javascript  
  
// Dans server.js, forcer la synchronisation  
await sequelize.sync({ force: true, alter: true });
```

Données incohérentes :

```
javascript  
  
// Ajouter validation dans les modèles  
nom: {  
    type: DataTypes.STRING,  
    allowNull: false,  
    validate: {  
        notEmpty: true,  
        len: [2, 50]  
    }  
}
```

3. Interface utilisateur

CSS ne se charge pas :

1. Vérifier que `public/css/style.css` existe
2. Exécuter `node setup.js`
3. Vérifier la route statique :

```
javascript  
  
app.use(express.static('public'));
```

Pages d'erreur 404 :

```
javascript
```

```
// Ajouter à la fin de server.js
app.use((req, res) => {
  res.status(404).send(` 
    <h1>Page non trouvée</h1>
    <p>La page ${req.path} n'existe pas.</p>
    <a href="/dashboard">Retour à l'accueil</a>
  `);
});
```

4. Problèmes de sécurité

Sessions qui expirent :

```
javascript

// Augmenter la durée des cookies
cookie: {
  maxAge: 7 * 24 * 60 * 60 * 1000, // 7 jours
  secure: false, // true si HTTPS
  httpOnly: true // Sécurité XSS
}
```

Mots de passe faibles :

```
javascript

// Ajouter validation
const validatePassword = (password) => {
  return password.length >= 8 &&
    /[A-Z]/.test(password) &&
    /[0-9]/.test(password);
};
```

🔧 Amélioration et maintenance {#maintenance}

1. Mode développement vs production

Fichier de configuration (config.js) :

```
javascript
```

```
module.exports = {
  development: {
    port: 3000,
    database: 'database/soakoja-dev.db',
    logging: true,
    debugMode: true
  },
  production: {
    port: process.env.PORT || 80,
    database: 'database/soakoja-prod.db',
    logging: false,
    debugMode: false
  }
};
```

Utilisation :

```
javascript
const env = process.env.NODE_ENV || 'development';
const config = require('./config')[env];
```

2. Logging avancé

Installation de winston :

```
bash
npm install winston
```

Configuration :

```
javascript
```

```

const winston = require('winston');

const logger = winston.createLogger({
  level: 'info',
  format: winston.format.json(),
  transports: [
    ... new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    ... new winston.transports.File({ filename: 'logs/combined.log' })
  ]
});

// Utilisation
logger.info('Utilisateur connecté', { userId: user.id });
logger.error('Erreur de connexion', { error: error.message });

```

3. Sauvegarde automatique

Script de sauvegarde (backup.js) :

```

javascript

const fs = require('fs');
const path = require('path');

function backupDatabase() {
  const date = new Date().toISOString().split('T')[0];
  const sourceDb = 'database/soakoja.db';
  const backupDb = `backups/soakoja-${date}.db`;

  if (fs.existsSync(sourceDb)) {
    fs.copyFileSync(sourceDb, backupDb);
    console.log(`✅ Sauvegarde créée: ${backupDb}`);
  }
}

// Exécuter quotidiennement
setInterval(backupDatabase, 24 * 60 * 60 * 1000);

```

4. Tests automatisés

Installation de Mocha :

```

bash

npm install --save-dev mocha chai supertest

```

Test d'exemple (test/auth.test.js) :

javascript

```
const request = require('supertest');
const app = require('../server');

describe('Authentication', () => {
  it('should login with valid credentials', (done) => {
    request(app)
      .post('/login')
      .send({ email: 'admin', password: 'admin123' })
      .expect(302) // Redirection après connexion
      .end(done);
  });

  it('should reject invalid credentials', (done) => {
    request(app)
      .post('/login')
      .send({ email: 'admin', password: 'wrongpass' })
      .expect(200) // Reste sur la page login
      .end(done);
  });
});
```

5. Performance et optimisation

Mise en cache :

javascript

```

const NodeCache = require('node-cache');
const cache = new NodeCache({ stdTTL: 600 });// 10 minutes

app.get('/dashboard', requireAuth, async (req, res) => {
  let stats = cache.get('dashboard-stats');

  if (!stats) {
    stats = {
      ouvrages_actifs: await Ouvrage.count({ where: { statut: 'actif' } }),
      // ... autres stats
    };
    cache.set('dashboard-stats', stats);
  }

  res.render('dashboard', { stats });
});

```

Compression :

bash

```
npm install compression
```

javascript

```

const compression = require('compression');
app.use(compression());

```

6. Monitoring et alertes

Surveillance de l'application :

javascript

```

// Health check endpoint
app.get('/health', (req, res) => {
  res.json({
    status: 'OK',
    timestamp: new Date().toISOString(),
    uptime: process.uptime(),
    memory: process.memoryUsage()
  });
});

```

Alertes par email (avec nodemailer) :

javascript

```
const nodemailer = require('nodemailer');

const sendAlert = async (message) => {
  const transporter = nodemailer.createTransport({
    // Configuration SMTP
  });

  await transporter.sendMail({
    to: 'admin@soakoja.mg',
    subject: 'Alerte Soakoja',
    text: message
  });
}
```

Ressources supplémentaires

Documentation officielle :

- [Node.js](#)
- [Express.js](#)
- [Sequelize](#)
- [SQLite](#)

Outils utiles :

- **DB Browser for SQLite** : Interface graphique pour SQLite
- **Postman** : Test des API
- **VS Code** : Éditeur recommandé
- **Git** : Contrôle de version

Commandes utiles :

bash

```
# Vérifier les processus Node.js
```

```
ps aux | grep node
```

```
# Voir les logs en temps réel
```

```
tail -f logs/combined.log
```

```
# Nettoyer le cache npm
```

```
npm cache clean --force
```

```
# Mettre à jour les dépendances
```

```
npm update
```

🎯 Prochaines étapes

1. **Maîtriser les bases** : Comprendre chaque section du code
2. **Personnaliser** : Adapter l'application à vos besoins
3. **Sécuriser** : Ajouter HTTPS, validation renforcée
4. **Optimiser** : Cache, compression, monitoring
5. **Déployer** : Mettre en production sur un serveur

Ce guide évoluera avec votre application. N'hésitez pas à l'enrichir au fur et à mesure de votre apprentissage !