# **1. PRESENTATION DU PROJET**

**1.1 OBJECTIFS ET ENJEUX**

Le projet **CESIZen** vise à concevoir une plateforme accessible au plus grand nombre, dédiée à la **gestion du stress** et à la **sensibilisation à la santé mentale**. L’objectif principal est de proposer des outils concrets pour aider les utilisateurs à mieux comprendre les mécanismes psychiques, tout en leur offrant des moyens simples et fiables pour améliorer leur bien-être mental au quotidien.

L’enjeu est double :

* **Préventif**, en favorisant l’éducation à la santé mentale grâce à des contenus pédagogiques validés ;
* **Pratique**, en donnant accès à des outils d’auto-évaluation et de suivi (comme le diagnostic de stress) afin d’agir directement sur les symptômes de stress ou d’anxiété.

**1.2 PUBLIC VISE ET ACTEURS IMPLIQUES**

La plateforme s’adresse à un **public large** :

* **Visiteurs anonymes**, qui peuvent consulter les contenus sans créer de compte ;
* **Utilisateurs enregistrés**, qui bénéficient de fonctionnalités supplémentaires comme la gestion de diagnostic de stress, l’interaction avec les contenus (commentaire, like …) et la création de contenus
* **Administrateurs**, chargés de la modération, de la gestion des utilisateurs et de l’administration des contenus.

Les parties prenantes du projet comprennent : les utilisateurs finaux, les développeurs, les administrateurs, les intervenants pédagogiques, ainsi que les professionnels de santé mentale qui pourraient valider les contenus ou outils proposés.

**2. PROTOTYPE FONCTIONNEL**

**2.1 DESCRIPTION GENERALE**

Le prototype de CESIZen prend la forme d’une application **web et mobile**, conçue selon une approche **"mobile first"**. Cette orientation garantit une **expérience utilisateur fluide sur smartphone**, tout en assurant une compatibilité optimale sur des supports plus larges comme les tablettes ou ordinateurs.

L’interface propose un accès à des **contenus pédagogiques**, des **diagnostics de stress interactifs**, ainsi qu’à un **espace personnel** permettant de gérer son profil ou son historique. Le projet repose sur une stack JavaScript complète (React/React Native côté front-end, Node.js côté back-end), ce qui assure cohérence et facilité de maintenance.

**2.2 APPROCHE MOBILE FIRST ET CONCEPTION DE L’INTERFACE**

La démarche **mobile first** a été intégrée dès les premières phases de conception. Cela a permis de prioriser la **lisibilité**, la **navigation intuitive** et la **réactivité** de l’interface sur petits écrans. Les maquettes graphiques et les composants ont été pensés pour être ergonomiques, accessibles et épurés, favorisant un usage régulier.

**2.3 FONCTIONNALITES DEVELOPPEES**

**2.3.1 GESTION DES COMPTES UTILISATEURS**

Cette fonctionnalité centrale garantit la **personnalisation** de l’expérience et la **sécurité** des données utilisateur. Elle comprend :

* La création de compte à partir d’un profil anonyme
* La gestion et modification des informations personnelles (en cours de développement)
* La réinitialisation de mot de passe via email sécurisé (en cours de développement)
* L’administration des comptes utilisateurs (création, suspension, suppression) par un profil administrateur (en cours de développement)

**2.3.2 GESTION DES CONTENUS INFORMATIFS**

L’application propose une base de contenus liés à la santé mentale :

* Accès aux menus et pages de contenus pédagogiques (articles, infographies, etc.) pour tous les profils ;
* Interface d’administration pour modifier ou enrichir ces contenus.

**2.3.3 Module diagnostic du stress**

Ce module basé sur le questionnaire Holmes-Rahe ajoute une dimension interactive :

* Questionnaire dynamique de diagnostic de stress
* Calcul d’un score personnalisé en fonction des réponses
* Affichage d’un résultat avec des conseils adaptés
* Interface administrateur pour modifier le questionnaire ou les résultats affichés (en cours )

**3. CHOIX TECHNOLOGIQUES ET JUSTIFICATIONS**

**3.1 STACK RETENUE**

|  |  |
| --- | --- |
| Front-end mobile | React Native |
| Back-end | Node.js / Express |
| ORM | Sequelize |
| Authentification | JWT (JSON Web Token) |

Cette stack a été choisie pour plusieurs raisons :

* **Alignement avec la formation** : Elle correspond aux technologies enseignées durant les interventions, ce qui a permis un gain de temps significatif au développement.
* **Cohérence technique** : L’usage exclusif de JavaScript sur l’ensemble du projet (front et back) facilite la lisibilité, la maintenance et le partage de compétences entre développeurs.
* **Performance mobile** : React Native permet de créer des applications mobiles natives, légères et réactives.
* **Sécurité et ORM** : Sequelize apporte une couche d’abstraction facilitant les manipulations de la base de données tout en limitant les risques de vulnérabilités (ex. : injections SQL).
* **Écosystème riche** : Node.js et React bénéficient d’une documentation abondante et d’une forte communauté, ce qui réduit les obstacles techniques.

**3.2 COMPARATIF DES SOLUTIONS**

**Alternatives envisagées**

| **Critère** | **React Native + Node.js** | **Ionic + Laravel** | **Flask + Front séparé** |
| --- | --- | --- | --- |
| Mobile natif | Oui | Non (hybride) | Non |
| Rapidité de développement | Élevée | Bonne | Moyenne |
| Maintenance | Simple (JS partout) | Plus complexe | Complexe |
| Sécurité | JWT | Sessions/JWT | JWT |
| Communauté / support | Très large | Large | Moyenne |
| Évolutivité | Excellente | Correcte | Bonne |

La solution **React Native + Node.js** a donc été retenue pour sa **performance mobile**, sa **cohérence technique**, et sa **capacité à évoluer rapidement**. Les autres solutions, bien qu’intéressantes, comportaient des freins (hybridation, langages distincts, complexité accrue).

**4. ARCHITECTURE LOGICIELLE**

L’architecture du projet **CESIZen** repose sur une **séparation stricte entre le frontend et le backend**, suivant les principes de modularité, d’isolabilité des responsabilités et de maintenabilité. Cette structuration a été pensée pour faciliter l’évolution du projet dans le temps, tout en assurant la lisibilité du code même dans un contexte de développement individuel.

**4.1 BACKEND (NODE.JS + EXPRESS + SEQUELIZE)**

**Organisation générale**

Le backend est structuré selon une architecture **MVC (Model-View-Controller)** avec une couche middleware pour la gestion de l’authentification et une séparation claire entre les rôles de chaque fichier.

Description des responsabilités :

* **Models** : définissent la structure des entités manipulées par Sequelize (User, Content, Diagnostic, etc.). Chaque modèle correspond à une table de la base de données, avec ses associations et contraintes.
* **Controllers** : regroupent la logique métier propre à chaque domaine fonctionnel. Ils orchestrent les appels aux modèles, valident les données et assurent les réponses API.
* **Routes** : connectent les endpoints exposés à leurs contrôleurs respectifs. Elles centralisent l’organisation de l’API REST.
* **Middleware** : assurent des fonctions transverses comme la vérification du token JWT pour sécuriser les routes protégées.
* **Config** : contient la configuration de la base de données et des variables d’environnement.

**Pourquoi cette organisation ?**

* **Lisibilité** : Chaque responsabilité est isolée dans un dossier spécifique. Cela facilite le repérage des fonctions à modifier ou à corriger.
* **Évolutivité** : Ajouter une nouvelle fonctionnalité (ex. : gestion de notifications) peut se faire en créant un modèle, un contrôleur et une route dédiés, sans impacter le reste du code.
* **Testabilité** : L’architecture modulaire permet de tester chaque composant indépendamment (ex. : un contrôleur sans avoir besoin de la BDD réelle).
* **Sécurité** : Les règles d’accès sont gérées dans les middlewares, ce qui limite les erreurs et centralise le contrôle.

**4.2 FRONTEND MOBILE (REACT NATIVE)**

**Organisation modulaire**

Le frontend mobile suit une structure modulaire centrée autour de l’idée de **composants réutilisables** et d’une gestion d’état centralisée.

Composants clés :

* **Screens** : Chaque écran représente une vue complète (ex. : LoginScreen, HomeScreen). Ils consomment des composants, gèrent les états locaux et interagissent avec le store.
* **Components** : Ensemble de composants UI encapsulés, responsables de l’affichage (ex. : ContentCard, UserAvatar).
* **Redux store** : Centralise les états partagés de l’application (utilisateur connecté, contenu, diagnostic, interactions).
* **API** : Les appels au backend sont centralisés dans apiClient.js et ses services spécifiques, ce qui facilite leur maintenance et leur test.

Justification des choix :

* **Séparation des préoccupations** : La logique d’interface, la gestion des données, et l’appel aux services API sont chacun isolés.
* **Réutilisabilité** : Les composants peuvent être utilisés dans plusieurs écrans sans duplication de code.
* **Testabilité** : L’organisation permet de tester les composants, les hooks et les appels API de manière indépendante.
* **Scalabilité** : L’ajout d’une nouvelle fonctionnalité (ex. : système de notifications) peut se faire sans altérer la structure existante.

**4.3 POURQUOI CETTE ARCHITECTURE ?**

Bien qu’il s’agisse d’un **projet individuel**, j’ai fait le choix d’une architecture professionnelle, claire et modulaire pour plusieurs raisons :

* **Anticiper l’évolution du projet** : Le projet pourrait être étoffé ou repris par d'autres développeurs à l’avenir.
* **Soutenir une logique de production** : Une bonne organisation facilite le débogage, les tests, la documentation, et la réutilisation de code.
* **Approche DevOps-ready** : L’architecture actuelle s’intègre facilement dans un pipeline CI/CD, et permet une automatisation des tests, du linting ou du déploiement à terme.
* **Formation et professionnalisation** : Adopter une structure rigoureuse est une opportunité de mettre en application les bonnes pratiques professionnelles, même en contexte non collaboratif.

**5. TESTS ET VALIDATION**

**5.1 TESTS BACKEND**

**5.1.1 APPROCHE DE TEST**

La stratégie de test repose sur une approche **pyramidale** structurée autour de trois niveaux :

1. **Tests unitaires** : Vérification des fonctions/méthodes indépendantes, notamment au niveau des contrôleurs.
2. **Tests d’intégration** : Vérification de l’interaction entre les modules, incluant API, contrôleurs et base de données.
3. **Tests End-to-End (E2E)** : Simulations de parcours utilisateur réels pour valider l’ensemble de l’application.

**5.1.2 OUTILS UTILISES**

* **Jest** : Framework de test principal.
* **Supertest** : Test des routes HTTP.
* **Mocks/Stubs** : Simulation des dépendances.
* **Base de données de test dédiée** : Isolation des tests.

**5.1.3 ORGANISATION**

*/tests*

*/unit*

*/integration*

*/e2e*

Les tests sont organisés en dossiers distincts, en fonction du niveau testé. Les fichiers suivent une nomenclature claire pour chaque module ou fonctionnalité.

**5.1.4 EXTRAITS DU PLAN DE TEST**

| ID | Scénario | Vérification | Prérequis | Résultat attendu |
| --- | --- | --- | --- | --- |
| TU-001 | Ajout d'un like sur un contenu | Interaction créée correctement en base de données | Modèle Content et ContentInteraction mockés | Status 201, message confirmant l'ajout du like, interaction créée avec userId, contentId et type corrects |
| TU-002 | Suppression d'un like existant | Interaction supprimée correctement | Modèle Content et ContentInteraction mockés avec une interaction existante | Status 200, message confirmant la suppression, méthode destroy appelée |
| TU-003 | Tentative d'interaction avec un contenu inexistant | Erreur 404 générée | Modèle Content mockés pour retourner null | Status 404, message d'erreur "Contenu non trouvé" |
| TU-004 | Récupération des statistiques d'interactions | Comptage correct des différents types d'interactions | Modèle Content et ContentInteraction mockés | Status 200, statistiques correctes pour chaque type d'interaction |
| TU-005 | Suppression d'un diagnostic et ses interactions | Le diagnostic et les interactions associées sont supprimés | Modèles Diagnostic et DiagnosticInteraction mockés | Status 200, message de confirmation, interactions et diagnostic supprimés |
| TU-006 | Validation token JWT valide | Middleware autorise l'accès | JWT et modèle User mockés | Middleware autorise l'accès, req.user contient les informations utilisateur |
| TU-007 | Validation token JWT manquant | Middleware refuse l'accès | Requête sans token | Status 401, message d'erreur approprié |
| TU-008 | Création de contenu | Contenu créé en base de données | Modèle Content mocké | Status 201, données du contenu correctes, userId correct |
| TU-009 | Récupération d'un contenu spécifique | Contenu et statistiques retournés | Modèles Content et ContentInteraction mockés | Status 200, contenu correct avec statistiques |
| TI-001 | Inscription d'un nouvel utilisateur | Utilisateur créé avec mot de passe haché | Base de données test vide | Status 201, token généré, utilisateur créé en BDD |
| TI-002 | Connexion utilisateur | Authentification réussie, token généré | Utilisateur existant en BDD | Status 200, token valide généré, informations utilisateur retournées sans mot de passe |
| TI-003 | Création d'un diagnostic | Diagnostic créé avec score et niveau de risque calculés | Utilisateur authentifié | Status 201, diagnostic créé avec score et riskLevel corrects |
| TI-004 | CRUD complet sur les contenus | Opérations Create, Read, Update, Delete fonctionnelles | Utilisateur authentifié | Toutes les opérations réussissent avec les bons statuts HTTP |
| TI-005 | Récupération des statistiques d'un contenu | Statistiques calculées correctement | Contenu avec interactions | Status 200, statistiques correctes (likes, dislikes, vues, favoris) |
| TI-006 | Hachage mot de passe dans le modèle User | Hook beforeCreate fonctionne | Base de données test | Mot de passe haché différent de l'original mais validant bcrypt.compare |
| TI-007 | Vérification méthode checkPassword | Validation correcte des mots de passe | Utilisateur existant | Retourne true pour le bon mot de passe, false pour le mauvais |
| E2E-001 | Parcours utilisateur complet | Flux complet d'utilisation | Application déployée en test | Toutes les étapes réussissent avec les bons statuts HTTP |
| E2E-002 | Tests de non-régression | Les fonctionnalités critiques continuent de fonctionner | Mocks pour simuler les réponses | Toutes les opérations réussissent, comportement conforme |

**5.1.5 RESULTATS ET ANALYSE**

| **ID** | **Résultat** | **Analyse** | **Commentaire** |
| --- | --- | --- | --- |
| TU-001 | SUCCÈS | Les interactions sur les contenus fonctionnent correctement | Le contrôleur interactionController gère correctement l'ajout d'une interaction |
| TU-002 | SUCCÈS | La suppression d'interactions existantes fonctionne | Le mécanisme toggle (ajout/suppression) fonctionne correctement |
| TU-003 | SUCCÈS | Le contrôleur retourne bien 404 pour les contenus inexistants | La vérification d'existence des ressources est robuste |
| TU-004 | SUCCÈS | Les statistiques sont correctement calculées | Le comptage des différents types d'interactions fonctionne |
| TU-005 | SUCCÈS | La suppression en cascade fonctionne correctement | Les diagnostics et leurs interactions associées sont supprimés |
| TU-006 | ÉCHEC | Le test s'attend à une structure utilisateur différente de celle réelle | Différence entre les champs attendus et réels dans req.user (email, role) |
| TU-007 | ÉCHEC | Messages d'erreur différents entre l'attente et l'implémentation | Le test attend "Accès non autorisé" mais reçoit "Non autorisé: Token manquant" |
| TU-008 | ÉCHEC | Le mock Content.create est appelé avec des paramètres différents | Problème avec les champs mediaUrl et tags qui sont absents dans l'implémentation réelle |
| TU-009 | SUCCÈS | Récupération de contenu et de ses statistiques fonctionne | Le contrôleur getContent agrège correctement les statistiques |
| TI-001 | SUCCÈS | L'inscription utilisateur fonctionne correctement | L'utilisateur est créé en BDD avec un mot de passe correctement haché |
| TI-002 | SUCCÈS | L'authentification fonctionne correctement | Le token JWT est correctement généré et les informations utilisateur retournées |
| TI-003 | ÉCHEC | Erreur de contrainte de clé étrangère | Le diagnostic ne peut pas être créé car la référence à l'utilisateur est invalide |
| TI-004 | ÉCHEC | Opérations DELETE et GET échouent avec statut 401 | Problème d'authentification, le token n'est pas correctement vérifié |
| TI-005 | ÉCHEC | Erreur 404 lors de la récupération du contenu | Le contenu n'est pas trouvé, problème de création préalable |
| TI-006 | SUCCÈS | Le hook beforeCreate hache correctement les mots de passe | Le modèle User gère bien la sécurité des mots de passe |
| TI-007 | SUCCÈS | La méthode checkPassword valide correctement les mots de passe | La vérification de mot de passe est robuste |
| E2E-001 | ÉCHEC | Erreur lors de la création du diagnostic (statut 500) | Problème de clé étrangère dans la base de données |
| E2E-002 | ÉCHEC | La connexion échoue avec statut 401 | Le mock de l'utilisateur ne fonctionne pas correctement, problèmes d'authentification |

**6. PLAN D'INSTALLATION DU PROJET CESIZEN**

**6.1 PREREQUIS**

Avant de commencer, assurez-vous d'avoir installé :

* **Node.js** (version 18 ou supérieure recommandée)
* **npm** (installé avec Node.js)
* **Expo CLI** : npm install -g expo-cli
* **MySQL** (pour la base de données)
* **Git** (pour cloner le dépôt)

**6.2 CLONAGE DU DEPOT**

Clonez le projet avec :

git clone https://github.com/Mekkilangelo/Cesizen.git

cd Cesizen

**6.3 CONFIGURATION DU BACK-END**

1. **Installation des dépendances** :
2. cd server
3. npm install
4. **Configuration de la base de données** :
   * Créez une base de données MySQL nommée cesizen.
   * Ajoutez un fichier .env à la racine de server :
   * DB\_HOST=localhost
   * DB\_USER=your\_mysql\_user
   * DB\_PASSWORD=your\_mysql\_password
   * DB\_NAME=cesizen
   * JWT\_SECRET=your\_jwt\_secret
   * PORT=5000
5. **Démarrage du serveur** :
   * En production : npm start
   * En développement : npm run dev

**6.4 CONFIGURATION DU FRONT-END MOBILE**

1. **Installation des dépendances** :
2. cd ../mobile
3. npm install
4. **Lancement de l’application mobile** :
5. npm start

Ouvrez Expo DevTools pour lancer l’application sur un émulateur ou téléphone avec **Expo Go**.

1. **Configuration de l’API** :

Vérifiez que le front-end pointe vers l’URL de l'API (votre back-end) dans les fichiers de configuration appropriés du dossier mobile.

**6.5 EXECUTION DES TESTS**

1. **Back-end** :
2. cd server
3. npm test
4. **Front-end mobile** :
5. cd mobile
6. npm test

Pour les tests end-to-end Cypress :

npm run cy:open