Chapitre 1

Analyses et spécification des besoins

1.1 Introduction

Après avoir effectué une exploration approfondie du projet dans son ensemble, nous commençons par la phase d'analyse des besoins et des spécifications techniques. Dans ce deuxième chapitre, nous développons une étude fonctionnelle du projet basée sur les diagrammes de cas d'utilisation ainsi qu'une étude technique des technologies utilisées.

1.2 Étude fonctionnelle

L'étude fonctionnelle est une phase nécessaire pour expliquer le sujet et les tâches nécessaires à réaliser. Elle met en valeur la liste des acteurs, leurs tâches, et les interactions entre eux sous la forme des diagrammes UML.

1.2.1 Identification des acteurs de système

Dans notre projet centré sur l'apprentissage de la programmation avec des blocs pour contrôler des dispositifs, tels que le robot mBot, différents acteurs interviennent, chacun jouant un rôle déterminant :

Acteurs Principaux

- **Tuteur :** Ce rôle représente les enseignants ou les guides qui supervisent et soutiennent les apprenants dans leur apprentissage de la programmation par blocs pour contrôler le robot mBot.
- **Apprenant :** Les apprenants sont les utilisateurs finaux de la plateforme. Ils utilisent les outils de programmation par blocs pour créer des

programmes et contrôler le robot mBot dans le cadre de leurs activités d'apprentissage.

Acteurs Administratifs

- **Super Administrateur :** Ce rôle englobe l'entité ou l'organisation qui supervise l'ensemble de la plateforme. Le super administrateur a un contrôle total sur le système, y compris la gestion des utilisateurs et des paramètres globaux.
- **Administrateur Scolaire :** L'administrateur scolaire est responsable de la gestion des comptes des utilisateurs au sein de son établissement, de la surveillance des activités des apprenants et des tuteurs, ainsi que de l'accès aux rapports sur les performances des étudiants.
- **Administrateur du Robot :** Cet administrateur est chargé de la gestion des robots sur la plateforme. Il assure la disponibilité des robots pour les activités d'apprentissage, supervise les connexions matérielles, gère les mises à jour du firmware et peut ajouter de nouveaux dispositifs à contrôler. De plus, il a la capacité de créer des blocs personnalisés pour le contrôle de ces dispositifs, offrant ainsi une flexibilité accrue dans les activités d'apprentissage.

1.2.2 Identification des besoins fonctionnels

Création de Compte

Inscription :

- L'utilisateur peut s'inscrire sur la plateforme en fournissant son nom, son adresse e-mail et un mot de passe.
- L'utilisateur peut choisir son rôle lors de l'inscription, en sélectionnant soit le rôle d'apprenant, soit le rôle de tuteur.

Profil utilisateur

Fonctionnalités communes :

- **Accès au profil :** L'utilisateur peut accéder à son profil personnel pour gérer ses informations personnelles, où il peut créer des projets. Il a la possibilité de rejoindre une école, ce qui modifie son profil pour refléter son affiliation à cette école. Un bouton dans la barre de navigation lui permet de passer entre son profil personnel et son profil d'école.

Fonctionnalités spécifiques au rôle :

- **En tant que tuteur :**
  - **Gestion de projets personnels :** Le tuteur peut créer des projets personnels à partir de son profil. Les projets personnels peuvent être partagés avec d'autres utilisateurs via des invitations. Les utilisateurs invités peuvent accéder aux projets partagés et collaborer avec le tuteur.

  - **Gestion de projets d'école :** Le tuteur peut créer des méta-projets pour son école. Les méta-projets peuvent être collaboratifs ou non collaboratifs. Dans les méta-projets non collaboratifs, chaque participant crée son propre projet individuel sous le méta-projet. Dans les méta-projets collaboratifs, le tuteur peut créer des groupes de travail et partager le code avec chaque groupe. Le tuteur peut visualiser les projets des étudiants dans les méta-projets non collaboratifs et les projets des groupes dans les méta-projets collaboratifs. Le tuteur peut suivre les progrès des étudiants et fournir un soutien personnalisé.
- **En tant qu'apprenant :**
  - **Gestion de projets personnels :** L'apprenant peut créer des projets personnels à partir de son profil.
  - **Accès aux projets partagés :** L'apprenant peut rejoindre une école pour accéder aux projets partagés par son tuteur dans cette école.
  - **Rejoindre des projets d'école :** L'apprenant peut rejoindre des projets créés par son tuteur dans la même école.

Rôles et autorisations

- **En tant que Super Administrateur :**
  - **Gestion des utilisateurs :** Le super administrateur peut créer, modifier et supprimer des comptes utilisateur. Il peut gérer les rôles et les autorisations des utilisateurs.
  - **Configuration du système :** Le super administrateur peut configurer les paramètres globaux de la plateforme.
  - **Surveillance du système :** Le super administrateur peut surveiller les activités de la plateforme.
- **En tant qu'Administrateur Scolaire :**
  - **Gestion des comptes utilisateurs :** L'administrateur scolaire peut créer, modifier et supprimer des comptes utilisateur au sein de son établissement. Il peut gérer les rôles et les autorisations des utilisateurs au niveau de l'école.
- **En tant qu'Administrateur du Robot :**
  - **Gestion des robots :** L'administrateur du robot est responsable de la gestion des robots sur la plateforme. Il assure la disponibilité des robots pour les activités d'apprentissage.
  - **Maintenance des robots :** L'administrateur du robot supervise les connexions matérielles, gère les mises à jour du firmware et assure le bon fonctionnement des robots.
  - **Personnalisation des blocs :** L'administrateur du robot peut créer des blocs personnalisés pour le contrôle des dispositifs, offrant ainsi une flexibilité accrue dans les activités d'apprentissage impliquant les robots.

1.2.3 Identification des besoins non fonctionnels

- **Sécurité :**
  - **Sécurité des données sensibles :** Assurer la protection des données personnelles des utilisateurs, notamment les informations d'identification et les données d'apprentissage, en utilisant un stockage sécurisé et des protocoles de transmission chiffrés.
  - **Sécurité des interactions robot-utilisateur :** Garantir que les interactions entre les utilisateurs et les robots, telles que l'envoi de commandes et la réception de données, sont sécurisées pour prévenir les attaques et les manipulations externes.
- **Performances :**
  - **Temps de réponse des commandes :** Assurer des temps de réponse rapides entre l'envoi de commandes depuis l'interface utilisateur et l'exécution des actions par le robot mBot, pour une expérience utilisateur réactive et fluide.
  - **Scalabilité des fonctionnalités :** Concevoir le système pour qu'il puisse prendre en charge de manière transparente l'ajout de nouvelles fonctionnalités d'apprentissage et de contrôle des robots sans compromettre les performances globales.
- **Utilisabilité :**
  - **Interface de programmation intuitive :** Développer une interface de programmation visuelle conviviale, basée sur des blocs logiques, permettant aux utilisateurs de créer des programmes pour contrôler les dispositifs sans avoir besoin de compétences en programmation textuelle.
- **Fiabilité :**
  - **Stabilité des connexions robot-utilisateur :** Assurer la stabilité et la fiabilité des connexions entre l'interface utilisateur et le robot mBot, en minimisant les interruptions de connexion et en fournissant des mécanismes de reconnexion automatique.
  - **Gestion des erreurs :** Mettre en place des mécanismes de gestion des erreurs robustes pour détecter, signaler et corriger les anomalies dans l'exécution des programmes et les interactions avec le robot mBot.
- **Performance du système :**
  - **Compatibilité multiplateforme :** Assurer la compatibilité du système avec différents périphériques et systèmes d'exploitation, y compris les ordinateurs de bureau, les tablettes et les smartphones, pour offrir une expérience utilisateur homogène sur toutes les plateformes.

1.3 Les diagrammes de cas d'utilisation

Le diagramme de cas d'utilisation décrit le comportement du système du point de vue de l'utilisateur. Il divise ses fonctionnalités en cas d'utilisation qui

permettent d'exprimer les besoins de ses utilisateurs, il est présenté par la figure 1 ci-dessous.

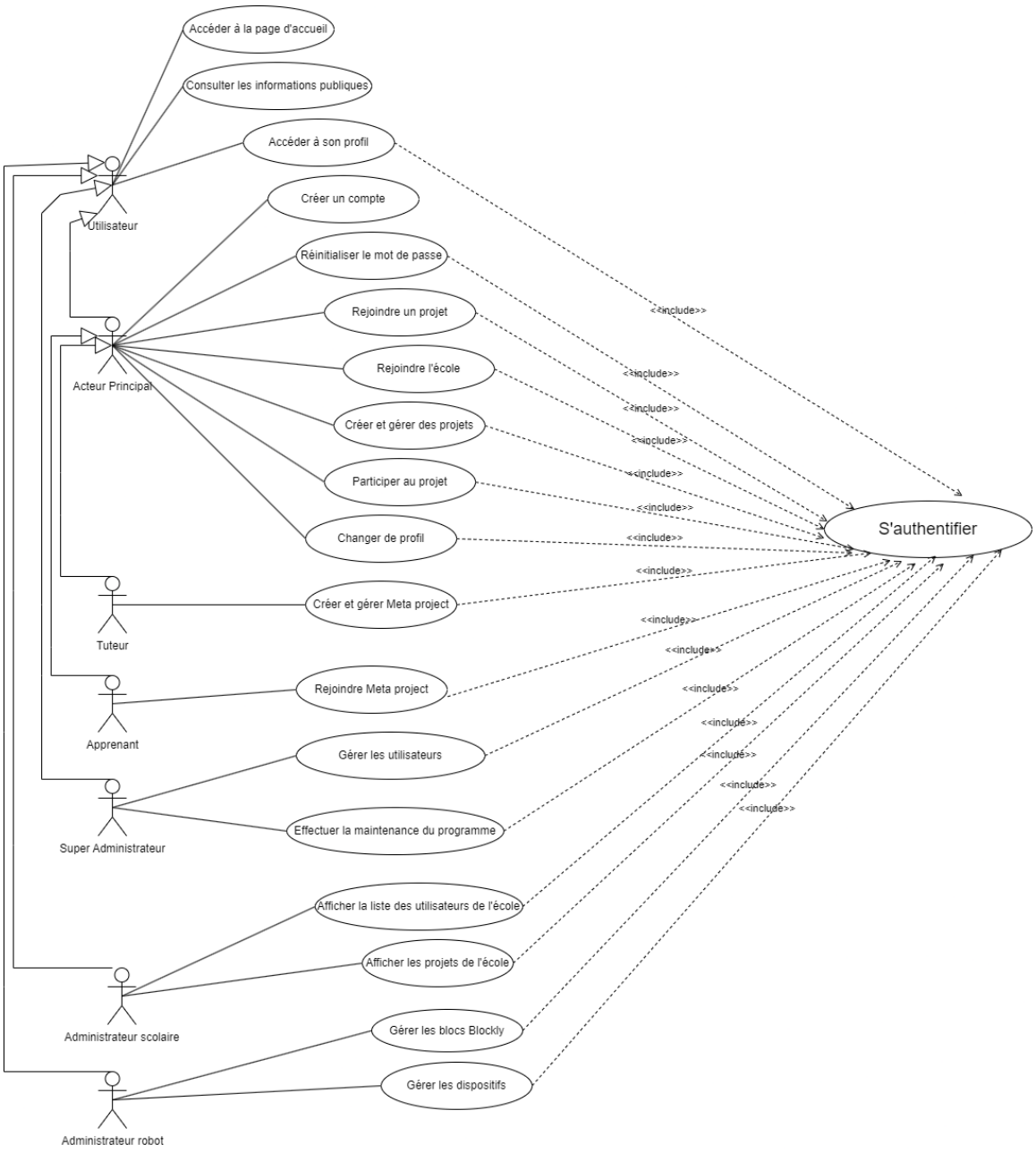

FIGURE 1.1 – Diagramme Global des Cas d'Utilisation

Créer et Gérer des Projets

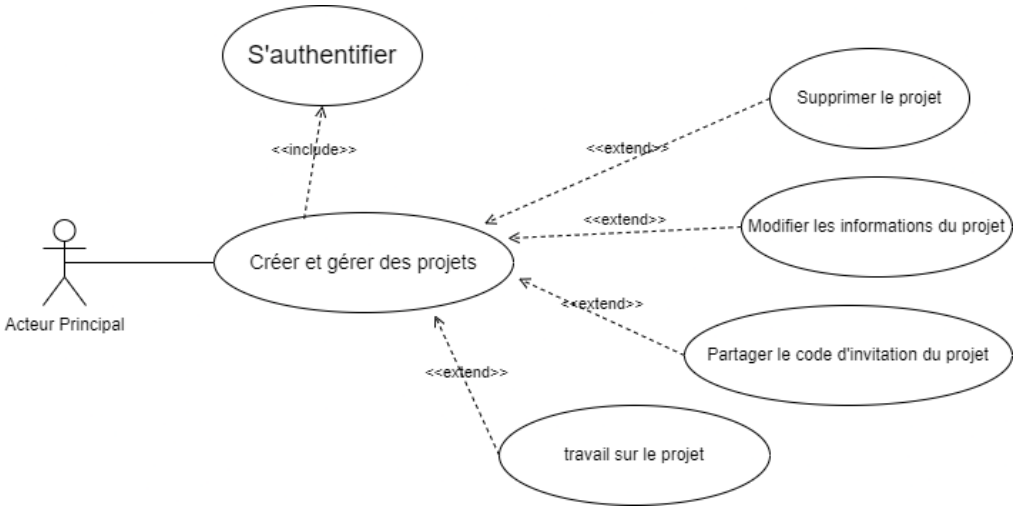

FIGURE 1.2 – Diagramme de Cas d'Utilisation pour Créer et Gérer des Projets

Description textuelle cas d'utilisation (CU) : Créer et Gérer des Projets

Acteur :	Acteur Principal
Pré-condition :	L'utilisateur est authentifié.
Enchaînement principal :	1. L'utilisateur accède à l'interface de création de projet. 2. L'utilisateur fournit les informations nécessaires pour créer un projet. 3. Le système valide les informations et crée le projet. 4. Le système affiche le projet nouvellement créé dans la liste des projets de l'utilisateur.
Post-condition :	Le projet est créé et visible dans la liste des projets de l'utilisateur.
Enchaînement alternatif :	— Si les informations fournies sont incorrectes, le système affiche un message d'erreur et demande à l'utilisateur de corriger les informations.

TABLE 1.1 – Description textuelle du cas d'utilisation : Créer et Gérer des Projets

Créer et Gérer Meta Projet

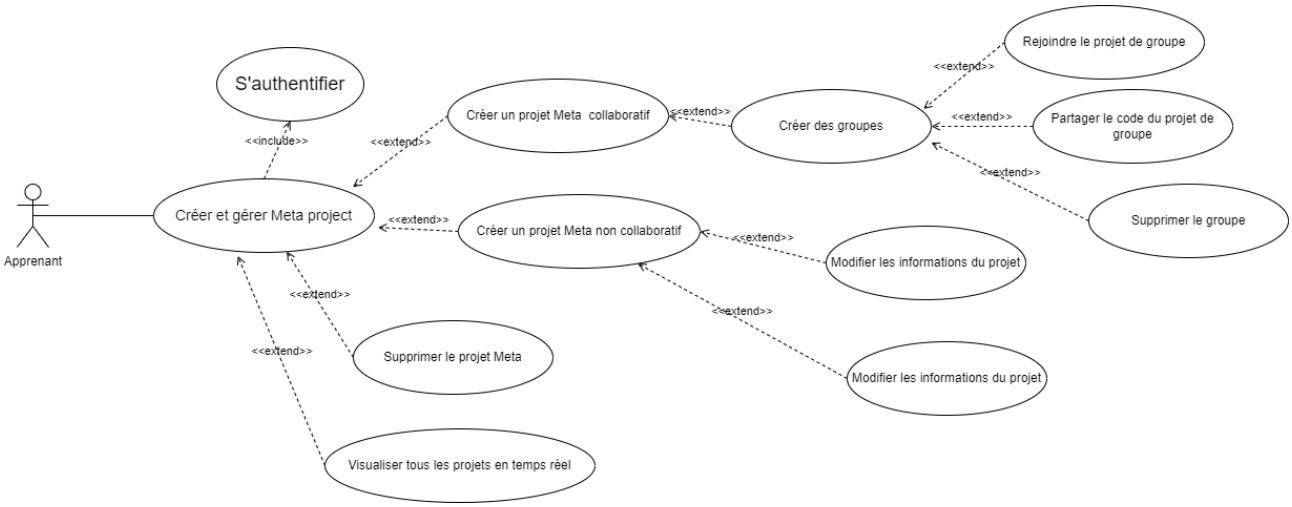

FIGURE 1.3 – Diagramme de Cas d'Utilisation pour Créer et Gérer Meta Projet

Description textuelle cas d'utilisation (CU) : Créer et Gérer Meta Projet

Acteur :	Apprenant
Pré-condition :	L'apprenant est authentifié.
Enchaînement principal :	1. L'apprenant accède à l'interface de création de meta projet. 2. L'apprenant fournit les informations nécessaires pour créer un meta projet. 3. Le système valide les informations et crée le meta projet. 4. Le système affiche le meta projet nouvellement créé dans la liste des projets de l'apprenant.
Post-condition :	Le meta projet est créé et visible dans la liste des projets de l'apprenant.
Enchaînement alternatif :	— Si les informations fournies sont incorrectes, le système affiche un message d'erreur et demande à l'apprenant de corriger les informations.

TABLE 1.2 – Description textuelle du cas d'utilisation : Créer et Gérer Meta Projet

Gérer les utilisateurs

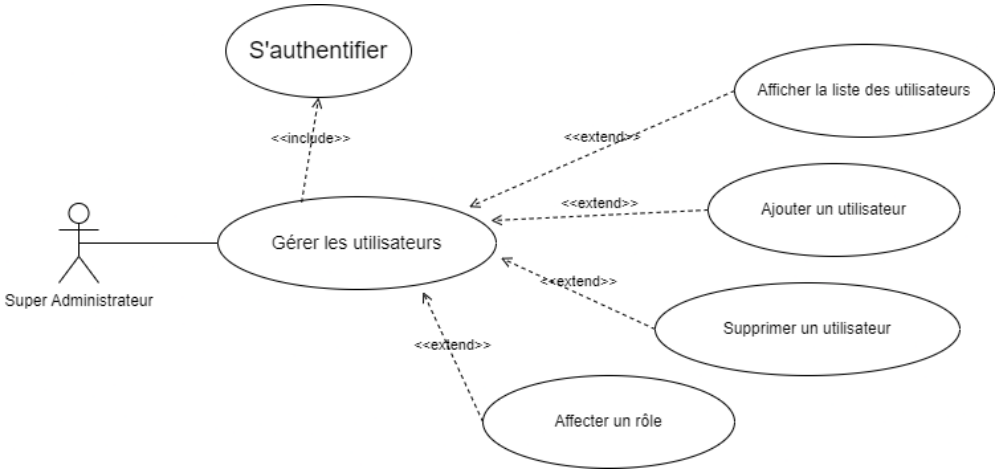

FIGURE 1.4 – Diagramme de Cas d'Utilisation pour Gérer les utilisateurs

Description textuelle cas d'utilisation (CU) : Gérer les utilisateurs

Acteur :	Super Administrateur
Pré-condition :	Le super administrateur est authentifié.
Enchaînement principal :	1. Le super administrateur accède à l'interface de gestion des utilisateurs. 2. Le super administrateur peut afficher la liste des utilisateurs. 3. Le super administrateur peut ajouter un nouvel utilisateur. 4. Le super administrateur peut supprimer un utilisateur existant. 5. Le super administrateur peut affecter un rôle à un utilisateur.
Post-condition :	Les modifications apportées aux utilisateurs sont enregistrées et visibles dans le système.
Enchaînement alternatif :	— Si les informations fournies sont incorrectes, le système affiche un message d'erreur et demande au super administrateur de corriger les informations.

TABLE 1.3 – Description textuelle du cas d'utilisation : Gérer les utilisateurs

Gérer les blocs

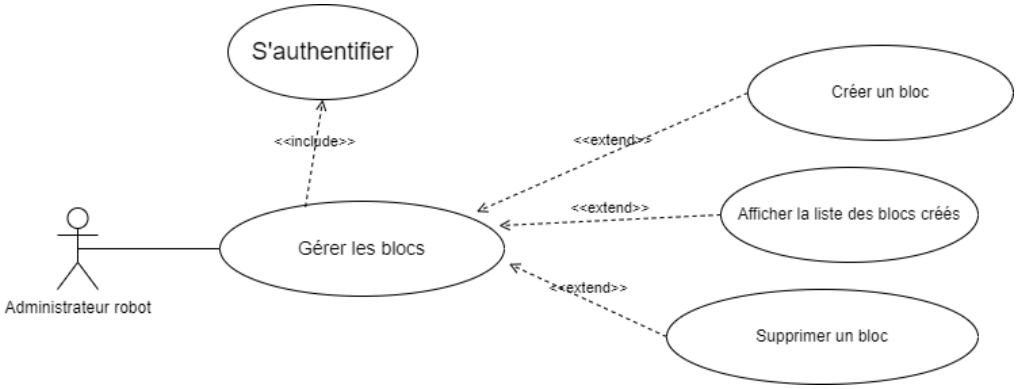

FIGURE 1.5 – Diagramme de Cas d'Utilisation pour Gérer les blocs

Description textuelle cas d'utilisation (CU) : Gérer les blocs

Acteur :	Administrateur robot
Pré-condition :	L'administrateur robot est authentifié.
Enchaînement principal :	1. L'administrateur robot accède à l'interface de gestion des blocs. 2. L'administrateur robot peut créer un nouveau bloc. 3. L'administrateur robot peut afficher la liste des blocs créés. 4. L'administrateur robot peut supprimer un bloc existant.
Post-condition :	Les modifications apportées aux blocs sont enregistrées et visibles dans le système.
Enchaînement alternatif :	— Si les informations fournies sont incorrectes, le système affiche un message d'erreur et demande à l'administrateur robot de corriger les informations.

TABLE 1.4 – Description textuelle du cas d'utilisation : Gérer les blocs

Gérer les dispositifs

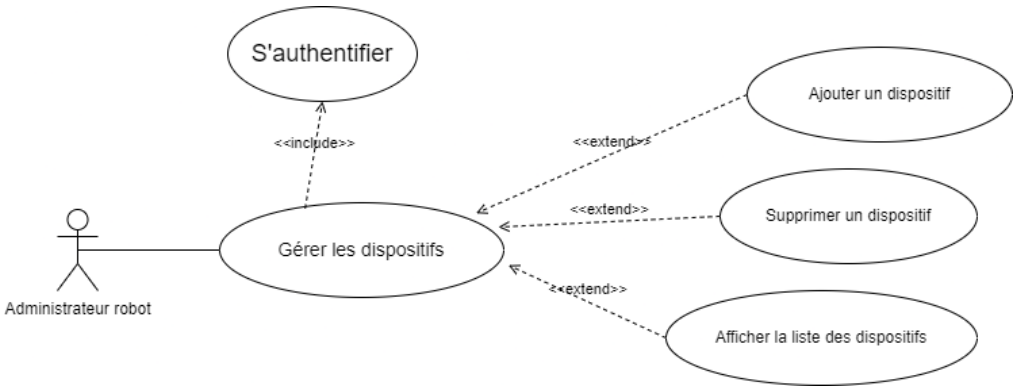

FIGURE 1.6 – Diagramme de Cas d'Utilisation pour Gérer les dispositifs

Description textuelle cas d'utilisation (CU) : Gérer les dispositifs

Acteur :	Administrateur robot
Pré-condition :	L'administrateur robot est authentifié.
Enchaînement principal :	1. L'administrateur robot accède à l'interface de gestion des dispositifs. 2. L'administrateur robot peut ajouter un nouveau dispositif. 3. L'administrateur robot peut afficher la liste des dispositifs. 4. L'administrateur robot peut supprimer un dispositif existant.
Post-condition :	Les modifications apportées aux dispositifs sont enregistrées et visibles dans le système.
Enchaînement alternatif :	— Si les informations fournies sont incorrectes, le système affiche un message d'erreur et demande à l'administrateur robot de corriger les informations.

TABLE 1.5 – Description textuelle du cas d'utilisation : Gérer les dispositifs

1.4 Étude Technique

Cette section de l'étude technique se concentre sur les technologies déployées pour le développement et l'implémentation d'iBlock, offrant une compréhension

approfondie de son architecture et de son fonctionnement.

1.4.1 Architecture Physique

L'architecture physique d'iBlock combine à la fois des éléments matériels et logiciels pour créer une plateforme robuste et interactive d'apprentissage de la programmation. Elle s'appuie sur des serveurs cloud pour héberger les services backend, des dispositifs robotiques tels que le robot mBot pour l'interaction physique, et des appareils utilisateurs variés (ordinateurs, tablettes, smartphones) pour accéder à l'interface web.

FIGURE 1.7 – Architecture Physique d'iBlock, illustrant la combinaison des serveurs cloud, des dispositifs robotiques, et des appareils utilisateurs.

1.4.2 Côté Client

Le côté client d'iBlock est développé avec Next.js, un framework React qui permet de créer des applications web performantes et optimisées pour le référencement. Nous avons choisi Next.js pour sa capacité à gérer le rendu côté serveur (SSR), ce qui améliore les performances et le SEO de l'application. De plus, Next.js offre un système de routage flexible et facilite la récupération des données depuis notre API backend. L'intégration de Blockly, notre éditeur de programmation visuelle, s'est faite de manière transparente grâce à l'utilisation de composants React au sein de Next.js.

FIGURE 1.8 – Architecture Côté Client d'iBlock, mettant en évidence l'utilisation de Next.js, React, HTML, CSS, et JavaScript.

Blockly

Blockly est un éditeur de programmation visuelle basé sur le web qui permet aux utilisateurs de créer des programmes en assemblant des blocs graphiques. Nous avons intégré Blockly dans iBlock pour offrir une interface de programmation intuitive et accessible aux débutants. Blockly nous permet de créer des blocs personnalisés pour contrôler le robot mBot, de générer du code à partir des blocs assemblés par l'utilisateur, et de personnaliser la boîte à outils avec les blocs pertinents pour notre projet.

1.4.3 Côté Serveur

Le backend d'iBlock est construit avec NestJS, un framework Node.js progressif qui offre une architecture modulaire et évolutive. NestJS nous permet de créer des API REST robustes et sécurisées. Nous avons défini des routes API à l'aide de contrôleurs NestJS pour gérer les requêtes provenant du frontend.

Le code backend est organisé en services et modules pour une meilleure maintenabilité. NestJS interagit avec MongoDB, notre base de données NoSQL, pour stocker et récupérer les données utilisateur, les projets, et les configurations des robots.

FIGURE 1.9 – Architecture Côté Serveur d'iBlock, illustrant l'utilisation de NestJS et MongoDB.

MongoDB

iBlock utilise MongoDB, une base de données NoSQL orientée document, pour la persistance des données. MongoDB est un choix approprié pour notre projet car il offre une grande flexibilité dans la structure des données et une excellente scalabilité. Nous stockons les données utilisateur, les projets, et les configurations des robots dans des collections MongoDB. NestJS utilise un pilote MongoDB pour interagir avec la base de données et effectuer des opérations CRUD.

1.4.4 La Sécurité

La sécurité est un pilier fondamental dans la conception et le déploiement d'iBlock, assurant la protection des données des utilisateurs et le maintien d'un environnement d'apprentissage sûr et fiable. Voici les stratégies clés mises en œuvre pour garantir la sécurité au sein du système iBlock :

Authentification et Autorisation L'authentification des utilisateurs dans iBlock est réalisée à l'aide de JSON Web Tokens (JWT). Lorsqu'un utilisateur se connecte avec succès, le backend génère un JWT qui est ensuite stocké dans un cookie sécurisé sur le navigateur de l'utilisateur. Ce JWT est envoyé avec chaque requête subséquente à l'API, permettant au backend de vérifier l'identité de l'utilisateur et d'autoriser ses actions. Les JWT ont une durée de vie limitée, après quoi l'utilisateur doit se reconnecter.

Chiffrement des Données Les mots de passe des utilisateurs sont cryptés à l'aide de l'algorithme bcrypt avant d'être stockés dans la base de données MongoDB. Bcrypt est un algorithme de hachage à sens unique qui rend impossible la récupération du mot de passe original à partir du hachage. Nous utilisons également le cryptage AES pour protéger d'autres données sensibles, telles que les informations personnelles des utilisateurs.

Sécurité au Niveau de l'API Notre API NestJS est protégée contre les attaques courantes telles que l'injection SQL, le cross-site scripting (XSS) et le cross-site request forgery (CSRF). Nous utilisons des requêtes paramétrées pour prévenir l'injection SQL et nous validons et nettoyons les entrées utilisateur pour

éviter les attaques XSS. Des jetons CSRF sont également utilisés pour protéger contre les attaques CSRF.

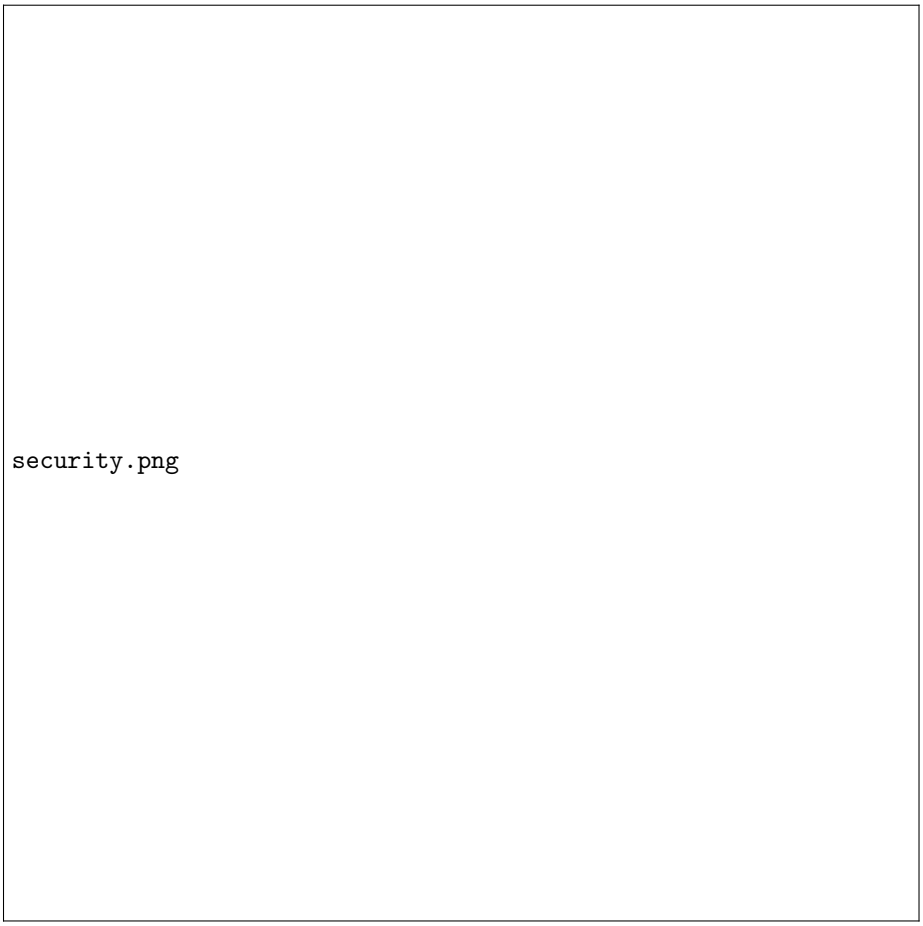

FIGURE 1.10 – Stratégies de Sécurité d'iBlock

La mise en œuvre de ces mesures de sécurité souligne l'engagement d'iBlock à fournir une plateforme d'apprentissage non seulement innovante et engageante mais également sûre et fiable pour tous ses utilisateurs.

1.5 Conclusion

Ce chapitre a offert une exploration approfondie des besoins fonctionnels et non fonctionnels, des acteurs impliqués, ainsi que de l'architecture technique d'iBlock, une plateforme novatrice dédiée à l'apprentissage de la programmation

à travers des blocs pour contrôler des dispositifs tels que le robot mBot. Grâce à une étude fonctionnelle détaillée, nous avons pu identifier clairement les rôles et les interactions des différents acteurs avec le système, soulignant l'importance d'une interface intuitive et accessible pour faciliter l'engagement des apprenants et des tuteurs dans l'environnement d'apprentissage.

L'analyse technique a mis en lumière les choix technologiques qui sous-tendent iBlock. Le côté client, développé avec Next.js, tire parti du rendu côté serveur (SSR) pour des performances optimales et un meilleur référencement. L'intégration de Blockly, un éditeur de programmation visuelle, offre une interface conviviale pour la création de programmes par blocs. Côté serveur, NestJS fournit une architecture robuste et modulaire pour la création d'API REST, tandis que MongoDB assure une gestion flexible et évolutive des données.

La sécurité d'iBlock est assurée par une combinaison de mécanismes performants. L'authentification basée sur JWT garantit la sécurité des sessions utilisateur, tandis que le cryptage bcrypt protège les mots de passe et le cryptage AES sécurise les données sensibles. L'API NestJS est protégée contre les attaques courantes grâce à des mesures telles que les requêtes paramétrées et les jetons CSRF.

En résumé, ce chapitre a établi une base solide pour la conception et le déploiement réussis d'iBlock, en alignant les besoins pédagogiques avec des solutions techniques avancées. L'engagement envers une expérience d'apprentissage sécurisée, interactive et enrichissante est clairement manifesté à travers les choix de conception et d'architecture du système. Alors que iBlock continue d'évoluer, il se positionne comme un outil éducatif clé dans le domaine de la programmation robotique, promettant de transformer la manière dont les concepts de programmation sont enseignés et appris.