# Chapter 1

# Conception et architecture du système

## 1.1 Introduction

Ce chapitre plonge au cœur de "IBLOCK", en explorant l'architecture et les choix de conception qui sous-tendent cette plateforme d'apprentissage de la programmation innovante. Nous commencerons par une vue d'ensemble de l'architecture générale du système, puis nous explorerons en détail les composants frontaux, l'organisation de la base de données, et les solutions techniques clés qui permettent la collaboration en temps réel et garantissent une expérience utilisateur optimale.

## 1.2 Architecture Générale

"IBLOCK" adopte une architecture client-serveur, où une application web front-end, développée avec Next.js, interagit avec un serveur back-end, développé avec NestJS, pour fournir les fonctionnalités de la plateforme. Cette séparation des préoccupations permet une meilleure modularité, maintenabilité et évolutivité.

Comme illustré dans la Figure 1.1, l'architecture d'IBLOCK suit un modèle client-serveur classique, avec une application web côté client (Next.js) qui communique avec un serveur d'API (NestJS) pour accéder aux données et aux fonctionnalités du backend. Le serveur d'API interagit avec une base de données MongoDB pour la persistance des données. De plus, une connexion WebSocket est utilisée pour la collaboration en temps réel, permettant aux utilisateurs de voir les modifications des autres en direct.

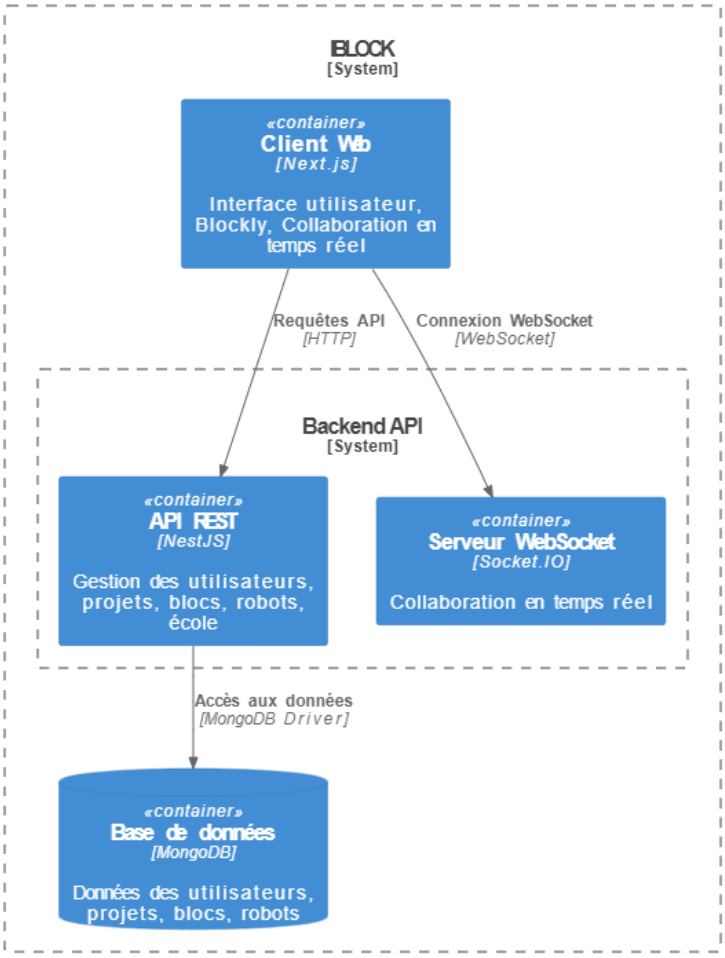


Figure 1.1: Architecture générale d'IBLOCK. Le client web, développé avec Next.js, interagit avec le backend via des requêtes API REST pour les opérations traditionnelles et via une connexion WebSocket pour la collaboration en temps réel.

## 1.3 Diagramme de Classe Global

Pour mieux comprendre les différentes entités qui composent le système "IBLOCK" et leurs relations, examinons le diagramme de classes global.

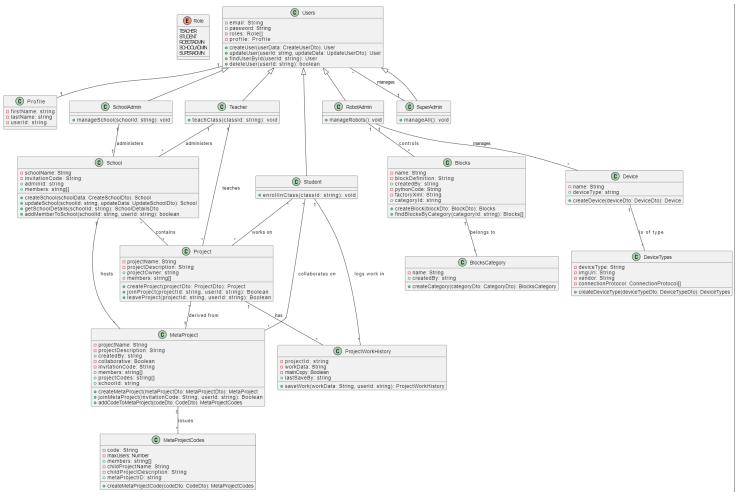


Figure 1.2: Diagramme de classe global d'IBLOCK. Il met en évidence les classes clés telles que les utilisateurs (avec leurs différents rôles), les écoles, les projets, les blocs de code, les dispositifs robotiques, et les méta-projets. Les relations entre ces classes, comme l'appartenance d'un projet à une école ou la participation d'un utilisateur à un projet, sont également clairement représentées.

## 1.4 Conception du Frontend (Next.js)

Le frontend d'IBLOCK est développé à l'aide de Next.js [4], un framework React pour la création d'applications web performantes et optimisées pour le SEO.

### 1.4.1 Fonctionnalités Clés

* **Rendu côté serveur (SSR) et génération de site statique (SSG) :** Next.js offre la flexibilité du rendu côté serveur et de la génération de site statique, améliorant ainsi les performances et le référencement. * **Routage simple et gestion des données :** Le système de routage basé sur les fichiers et les fonctions de récupération de données de Next.js simplifient la création d'applications web complexes. * **Intégration transparente avec React :** En tant que framework React, Next.js bénéficie de l'écosystème riche et mature de React, y compris des bibliothèques comme 'react-blockly' pour l'intégration de Blockly.

### 1.4.2 Intégration de Blockly

L'intégration de Blockly dans le frontend Next.js est réalisée à l'aide de la bibliothèque 'react-blockly' [6]. Cette bibliothèque fournit des composants React pour initialiser, afficher et interagir avec l'espace de travail Blockly. Blockly, étant une bibliothèque JavaScript, s'intègre parfaitement à l'environnement React de Next.js, permettant aux utilisateurs de créer des programmes visuellement en assemblant des blocs de code.

## 1.5 Architecture du Backend (NestJS)

Le backend d'IBLOCK est construit sur NestJS [5], un framework Node.js progressif pour la création d'applications serveur efficaces, fiables et évolutives. NestJS offre une architecture modulaire basée sur des modules, des contrôleurs et des services, ce qui permet une organisation claire du code et une meilleure maintenabilité.

### 1.5.1 Fonctionnalités Clés

NestJS offre un certain nombre de fonctionnalités clés qui en font un choix idéal pour le développement du backend d'IBLOCK :

* **Architecture modulaire :** NestJS encourage une architecture modulaire, ce qui facilite la division d'une application complexe en modules plus petits et réutilisables. * **Injection de dépendances :** L'injection de dépendances intégrée simplifie la gestion des dépendances et favorise la testabilité du code. * **Intégration TypeScript :** NestJS est conçu pour fonctionner de manière transparente avec TypeScript, offrant un typage statique et une meilleure maintenabilité du code.

### 1.5.2 Modules

Les modules sont les éléments constitutifs d'une application NestJS. Ils encapsulent un ensemble de fonctionnalités liées, telles que les contrôleurs, les services et autres composants. IBLOCK utilise des modules pour organiser son backend en domaines logiques, tels que la gestion des utilisateurs, la gestion des projets et la gestion des robots.

### 1.5.3 Contrôleurs

Les contrôleurs sont responsables de la gestion des requêtes entrantes et de la renvoi des réponses appropriées. Ils définissent les routes de l'API et gèrent la logique métier de haut niveau. Par exemple, un contrôleur 'UserController' pourrait gérer les requêtes liées à la création, la lecture, la mise à jour et la suppression des utilisateurs.

### 1.5.4 Services

Les services sont utilisés pour encapsuler la logique métier et les interactions avec la base de données. Ils sont injectés dans les contrôleurs, ce qui permet une séparation claire des préoccupations. Par exemple, un service 'UserService' pourrait gérer les opérations de création, de recherche et de mise à jour des utilisateurs dans la base de données.

## 1.6 Conception de la Base de Données (MongoDB)

MongoDB [1] a été choisi comme système de gestion de base de données (SGBD) pour IBLOCK. Il s'agit d'une base de données NoSQL orientée document, réputée pour sa flexibilité, son évolutivité et sa capacité à gérer des données semi-structurées. MongoDB stocke les données dans des documents JSON, ce qui offre une représentation intuitive et flexible des entités de l'application.

### 1.6.1 Avantages de MongoDB

Plusieurs facteurs ont motivé le choix de MongoDB pour IBLOCK :

* **Flexibilité du Schéma:** MongoDB offre une grande flexibilité en termes de schéma de données. Les documents au sein d'une même collection peuvent avoir des structures différentes, ce qui est idéal pour les applications dont les données évoluent rapidement. * **Évolutivité:** MongoDB est conçu pour l'évolutivité horizontale, ce qui signifie qu'il peut gérer de grandes quantités de données et un trafic élevé en ajoutant simplement des serveurs au cluster. * **Performance:** MongoDB offre d'excellentes performances pour les opérations de lecture et d'écriture, en particulier pour les applications qui nécessitent un accès rapide aux données. * **Intégration Facile:** MongoDB s'intègre facilement avec Node.js et NestJS, ce qui simplifie le développement et la maintenance du backend.

### 1.6.2 Structure de la Base de Données

La base de données d'IBLOCK est organisée en collections, chaque collection représentant un type d'entité du système. Voici les principales collections utilisées :

* **Utilisateurs:** Stocke les informations sur les utilisateurs, telles que le nom d'utilisateur, le mot de passe, le rôle et les informations de profil. * **Écoles:** Contient les informations sur les écoles, y compris le nom, l'adresse et les membres. * **Projets:** Stocke les détails des projets, tels que le nom, la description, le propriétaire et les membres. * **Blocs:** Contient les définitions des blocs de code personnalisés créés par les utilisateurs. * **Dispositifs:** Stocke les informations sur les dispositifs robotiques, y compris le type, le nom

et les paramètres de connexion. * **Méta-Projets:** Contient les informations sur les méta-projets, qui regroupent plusieurs projets liés.

## 1.7 Diagrammes de Séquence

Pour illustrer plus en détail les interactions entre les différents composants d'IBLOCK, nous allons utiliser des diagrammes de séquence. Ces diagrammes permettent de visualiser le flux des messages et les opérations effectuées lors de scénarios d'utilisation spécifiques.

### 1.7.1 Authentification

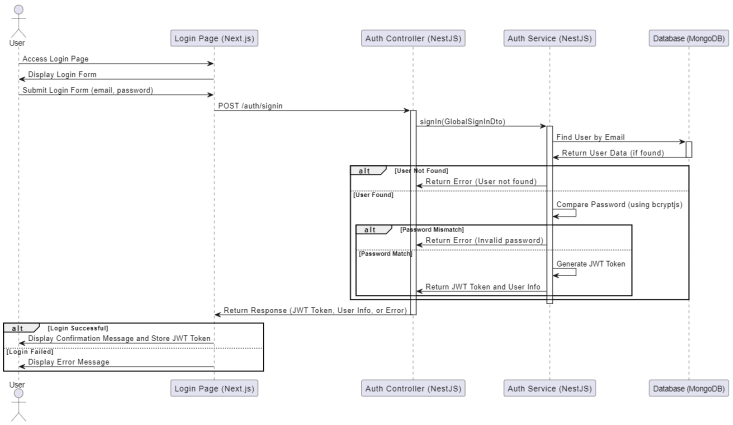


Figure 1.3: Diagramme de séquence pour l'authentification. L'utilisateur soumet ses identifiants de connexion via la page de connexion. Le serveur d'API vérifie ces identifiants par rapport à la base de données. Si l'authentification est réussie, un token JWT est généré et renvoyé à l'utilisateur, lui permettant d'accéder aux fonctionnalités protégées de la plateforme.

### 1.7.2 Création d'un Projet

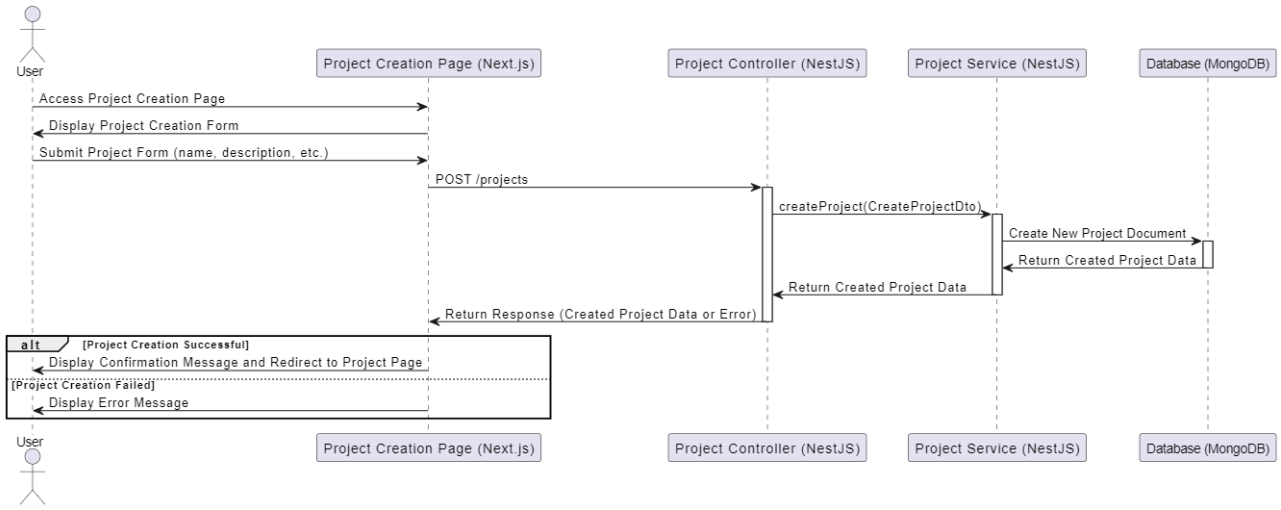


Figure 1.4: Diagramme de séquence pour la création d'un projet. L'utilisateur, via la page de création de projet, soumet les détails du projet. Le serveur d'API valide ces informations, crée un nouveau document de projet dans la base de données, et renvoie les détails du projet créé à l'utilisateur.

### 1.7.3 Suivi des Mises à Jour du Workspace

Ce diagramme de séquence illustre comment l'enseignant peut suivre les mises à jour du workspace de ses étudiants en temps réel, mettant en évidence la capacité d'IBLOCK à fournir un suivi personnalisé et une assistance en temps opportun.

### 1.7.4 Fonctionnement du Tableau de Bord Collaboratif

Ce diagramme de séquence illustre le fonctionnement du tableau de bord collaboratif d'IBLOCK, mettant en évidence les deux mécanismes d'enregistrement des données : l'enregistrement automatique via WebSocket et l'enregistrement manuel via une requête HTTP.

### 1.7.5 Envoi de Commandes au Robot

Ce diagramme de séquence illustre comment les commandes générées à partir des blocs Blockly sont envoyées au robot via WebSocket et un serveur Python dédié.

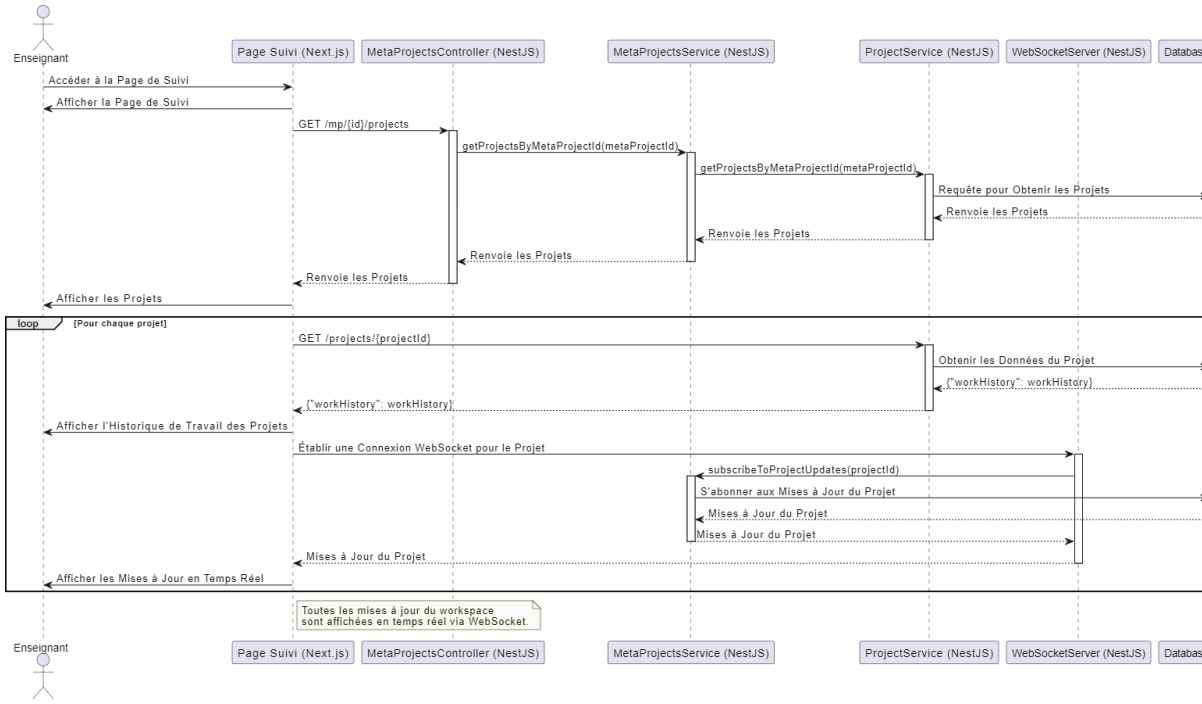


Figure 1.5: Diagramme de séquence pour le suivi des mises à jour du workspace par l'enseignant. Lorsqu'un étudiant modifie son workspace, le client web envoie une notification au serveur d'API via WebSocket. Le serveur d'API diffuse ensuite cette mise à jour à tous les clients connectés qui sont autorisés à visualiser le projet, y compris l'enseignant. L'enseignant peut ainsi visualiser en temps réel les progrès de ses étudiants sur leurs projets respectifs.

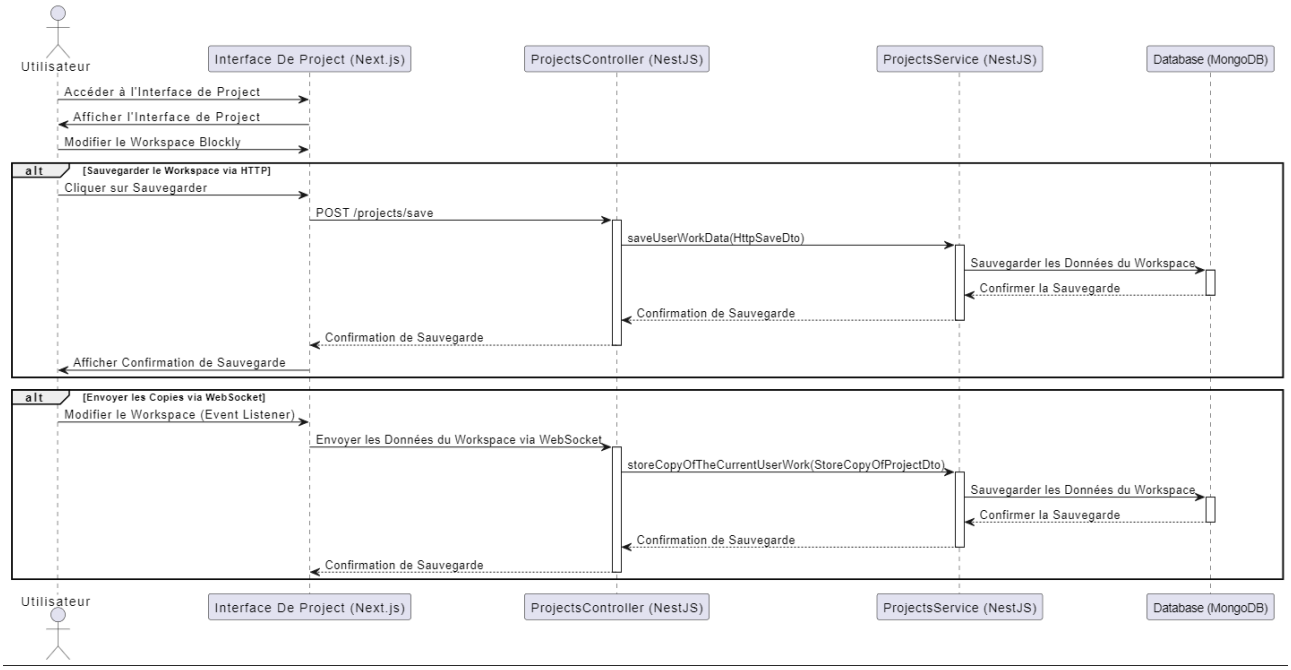


Figure 1.6: Diagramme de séquence pour le fonctionnement du tableau de bord collaboratif. Ce diagramme montre comment les modifications du workspace sont envoyées au serveur via WebSocket et comment les utilisateurs peuvent sauvegarder leur travail manuellement via une requête HTTP. Le diagramme met également en évidence la sauvegarde automatique des données du workspace à intervalles réguliers via WebSocket.

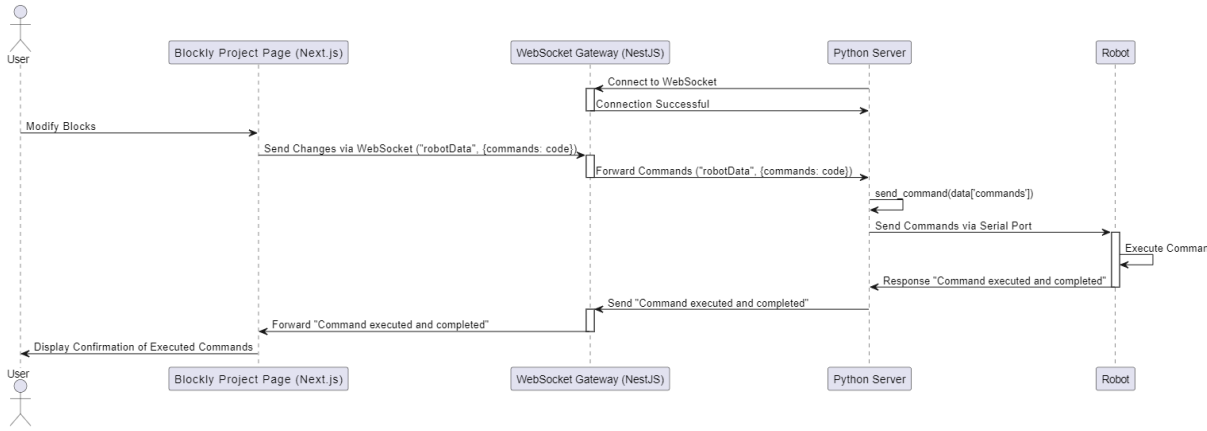


Figure 1.7: Diagramme de séquence pour l'envoi de commandes au robot. Lorsqu'un utilisateur modifie les blocs de code Blockly, le client web envoie les commandes mises à jour au serveur d'API via WebSocket. Le serveur d'API transmet ces commandes à un serveur Python dédié, qui est responsable de la communication avec le robot via le port série. Le serveur Python envoie les commandes au robot et reçoit les réponses du robot, qui sont ensuite renvoyées au client web via le serveur d'API et WebSocket.

## 1.8 Conclusion

Cette partie a exploré en détail l'architecture et la conception du backend d'IBLOCK, ainsi que le rôle crucial de la base de données MongoDB dans le fonctionnement du système. Nous avons vu comment NestJS, avec son architecture modulaire basée sur des modules, des contrôleurs et des services, offre un cadre robuste et maintenable pour le développement du backend.

Le choix de MongoDB, une base de données NoSQL orientée document, s'est avéré judicieux en raison de sa flexibilité, de son évolutivité et de sa capacité à gérer les données semi-structurées inhérentes à IBLOCK. La structure de la base de données, organisée en collections représentant les différentes entités du système, assure une gestion efficace et intuitive des données.

Les diagrammes de séquence ont permis de visualiser clairement les interactions complexes entre les différents composants d'IBLOCK, mettant en lumière le fonctionnement du système, de l'authentification des utilisateurs à l'envoi de commandes aux robots, en passant par la collaboration en temps réel sur le tableau de bord.

La combinaison de ces technologies et choix de conception confère à IBLOCK une base solide pour offrir une expérience d'apprentissage de la programmation collaborative, engageante et efficace. La partie suivante se penchera sur la mise en œuvre concrète de ces composants et fonctionnalités.

# Bibliography


[1] Chodorow, K., Dirolf, M. (2013). MongoDB: the definitive guide. O'Reilly Media, Inc.

[2] Fielding, R. T. (2000). Architectural styles and the design of network-based software architectures (Doctoral dissertation, University of California, Irvine).

[3] Fowler, M. (2002). Patterns of enterprise application architecture. Addison-Wesley Professional.

[4] Next.js Documentation. Retrieved from https://nextjs.org/docs

[5] NestJS Documentation. Retrieved from https://docs.nestjs.com/

[6] React Blockly. Retrieved from https://www.npmjs.com/package/react-blockly

[7] Socket.IO Documentation. Retrieved from https://socket.io/docs/