Chapter 1

Conception

Ce chapitre se focalise sur la conception détaillée d'iBlock, en explorant l'architecture logicielle, les interactions entre les composants, et les choix de conception clés qui sous-tendent la plateforme. Nous utiliserons des diagrammes de séquence pour illustrer les flux de communication et les processus impliqués dans les fonctionnalités principales d'iBlock.

1.1 Diagrammes de Séquence

Les diagrammes de séquence offrent une représentation visuelle des interactions entre les différents composants d'un système au fil du temps. Ils sont particulièrement utiles pour comprendre les flux de messages, les appels de fonctions, et la chronologie des événements dans un scénario donné.

1.1.1 Diagramme de Séquence pour l'Authentification

Ce diagramme illustre le processus d'authentification d'un utilisateur dans iBlock. Il met en évidence les interactions entre la page de connexion (Next.js), le contrôleur d'authentification (NestJS), le service d'authentification (NestJS), et la base de données (MongoDB).

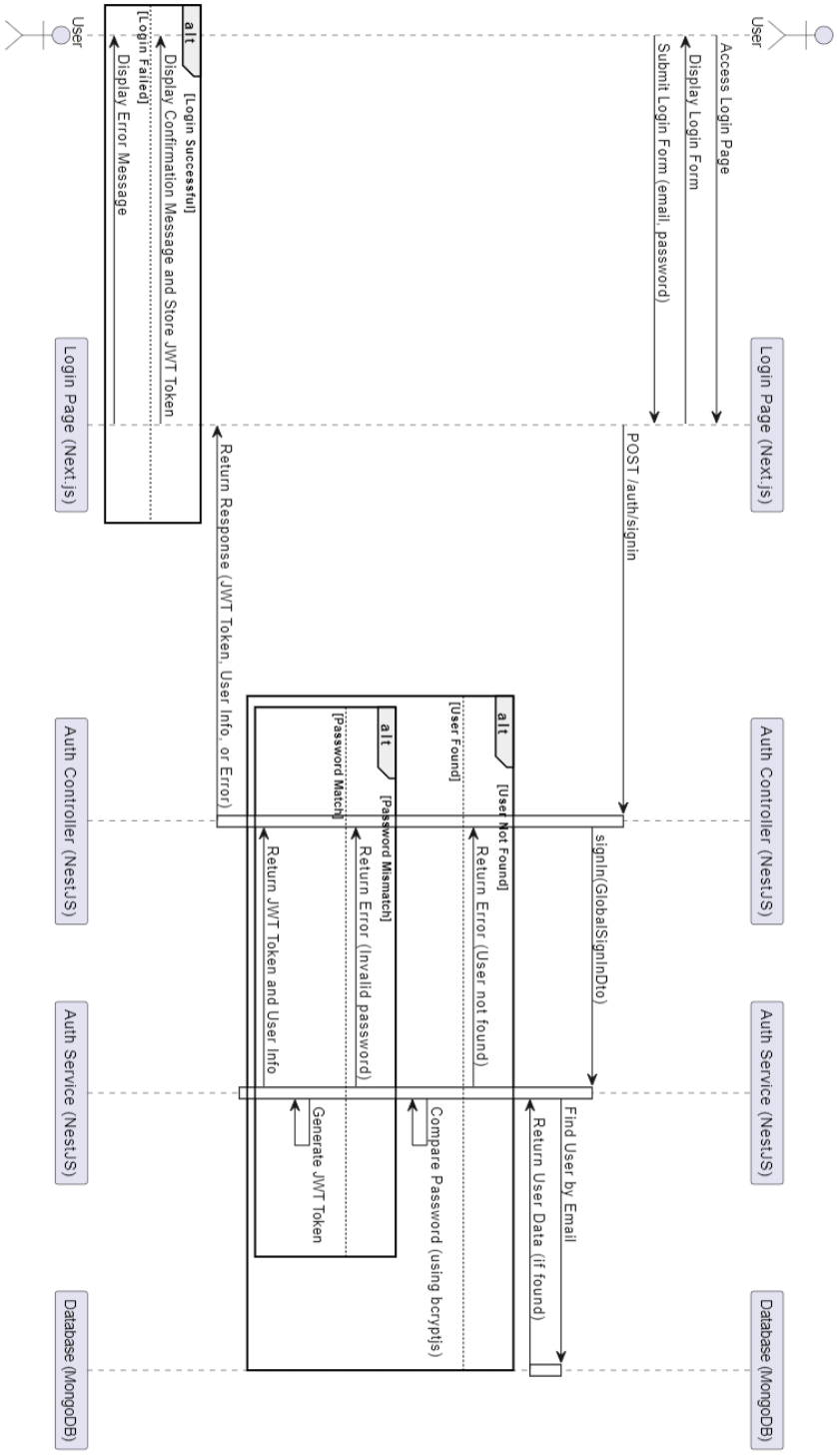

Figure 1.1: Diagramme de Séquence pour l'Authentification

Description du Diagramme:

L'utilisateur commence par accéder à la page de connexion d'iBlock, où il est invité à saisir ses informations d'identification via un formulaire de connexion. Après avoir soumis le formulaire avec son adresse e-mail et son mot de passe, la page de connexion envoie une requête POST au contrôleur d'authentification. Le contrôleur d'authentification délègue la validation des informations d'identification au service d'authentification. Ce service interroge la base de données MongoDB pour trouver l'utilisateur correspondant à l'adresse e-mail fournie. Si l'utilisateur est trouvé, le service d'authentification compare le mot de passe saisi avec le mot de passe haché stocké dans la base de données. En cas de correspondance, le service d'authentification génère un jeton JWT (JSON Web Token) pour l'utilisateur et le renvoie au contrôleur d'authentification, avec les informations utilisateur. Le contrôleur d'authentification transmet ensuite ces données à la page de connexion, qui stocke le jeton JWT dans le stockage local du navigateur et redirige l'utilisateur vers la page d'accueil d'iBlock. Ce processus garantit une authentification sécurisée en vérifiant les informations d'identification de l'utilisateur avant d'accorder l'accès à la plateforme.

1.1.2 Création d'un Projet

Ce diagramme de séquence détaille le processus de création d'un nouveau projet dans iBlock. Il met en lumière les interactions entre la page de création de projet (Next.js), le contrôleur de projet (NestJS), le service de projet (NestJS), et la base de données (MongoDB).

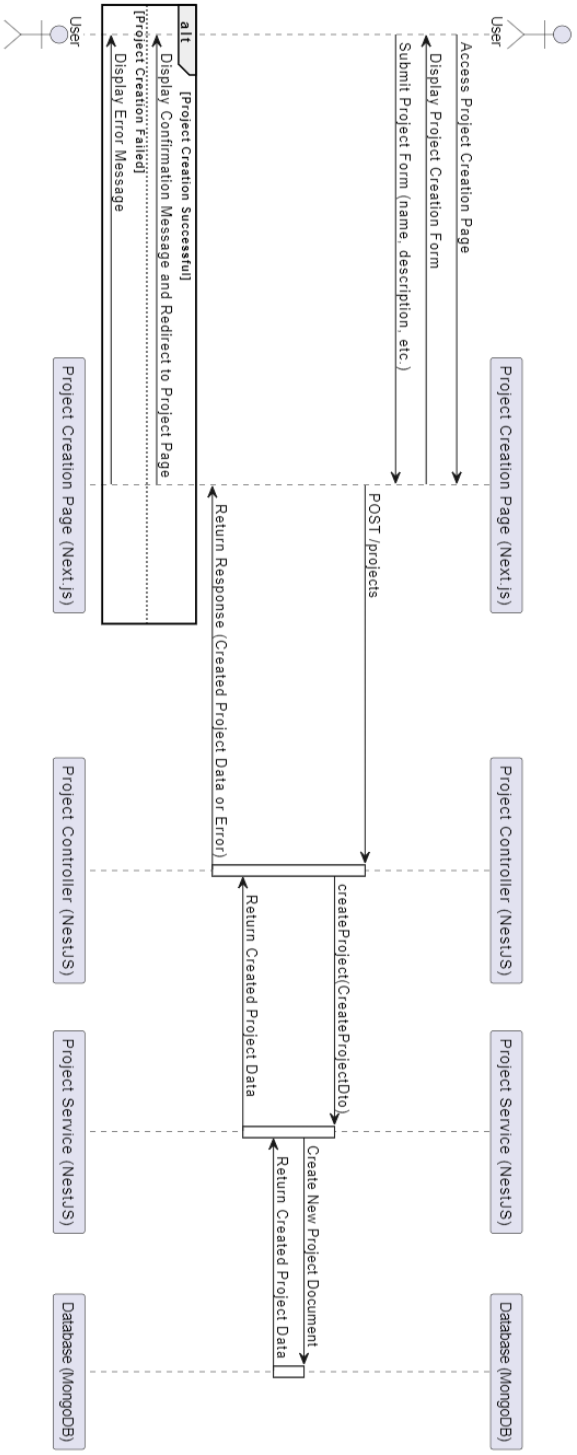

Figure 1.2: Diagramme de Séquence pour la Création d'un Projet

Description du Diagramme:

Pour créer un nouveau projet, l'utilisateur accède à la page de création de projet d'iBlock. La page affiche un formulaire permettant à l'utilisateur de saisir les informations du projet, telles que le nom, la description et autres détails pertinents. Une fois le formulaire soumis, la page de création de projet envoie une requête POST au contrôleur de projet. Le contrôleur de projet délègue la création du projet au service de projet. Le service de projet interagit avec la base de données MongoDB pour créer un nouveau document de projet contenant les informations fournies par l'utilisateur. La base de données renvoie ensuite les données du projet nouvellement créé au service de projet, qui les transmet au contrôleur de projet. Enfin, le contrôleur de projet renvoie une réponse à la page de création de projet, contenant les données du projet créé ou un message d'erreur en cas d'échec. La page de création de projet affiche alors un message de confirmation à l'utilisateur si la création du projet a réussi, ou un message d'erreur si la création a échoué. Ce processus garantit une gestion efficace de la création de projets, en assurant la persistance des données dans la base de données et en fournissant un retour d'information clair à l'utilisateur.

1.1.3 Sauvegarde des Données de Travail de l'Utilisateur

Ce diagramme de séquence illustre le processus de sauvegarde des données de travail de l'utilisateur dans iBlock, en se concentrant sur les interactions entre la page Blockly (Next.js), le contrôleur de projet (NestJS), le service de projet (NestJS), et la base de données (MongoDB).

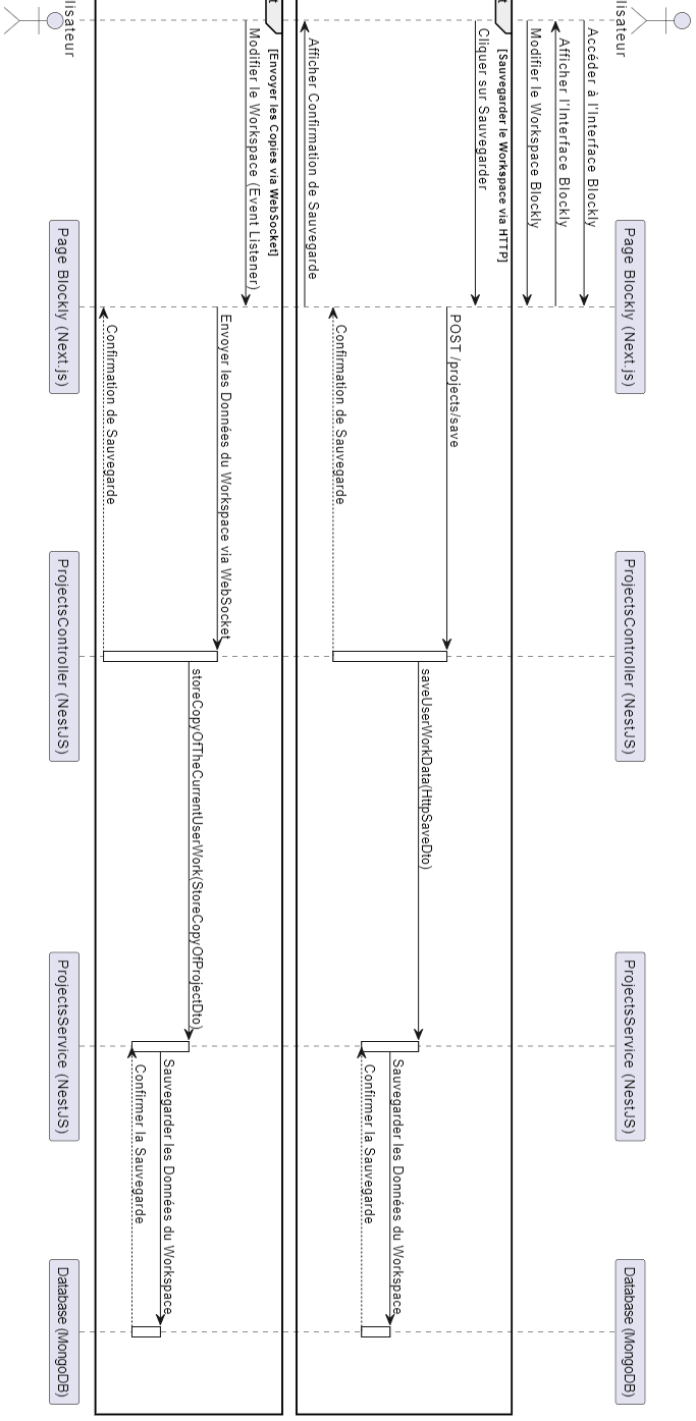

Figure 1.3: Diagramme de Séquence pour la Sauvegarde des Données de Travail de l'Utilisateur

Description du Diagramme:

Lorsqu'un utilisateur accède à l'interface Blockly sur la page Blockly d'iBlock, la page affiche l'interface permettant à l'utilisateur de modifier le workspace et de créer des programmes. Pour sauvegarder son travail, l'utilisateur clique sur le bouton "Sauvegarder". Cette action déclenche une requête POST vers le contrôleur de projet, qui à son tour appelle le service de projet pour sauvegarder les données du workspace. Le service de projet interagit avec la base de données MongoDB pour stocker les données du workspace. Une fois la sauvegarde effectuée, la base de données confirme l'opération au service de projet, qui renvoie la confirmation au contrôleur de projet. Enfin, le contrôleur de projet renvoie la confirmation à la page Blockly, qui affiche un message de confirmation à l'utilisateur.

En parallèle de la sauvegarde manuelle, iBlock utilise WebSocket pour un suivi en temps réel des modifications du workspace. Lorsqu'une modification est détectée dans le workspace Blockly, un écouteur d'événements déclenche l'envoi des données du workspace au contrôleur de projet via WebSocket. Le contrôleur de projet appelle ensuite le service de projet pour stocker une copie du travail de l'utilisateur. Le service de projet sauvegarde ces données dans la base de données MongoDB. La base de données confirme la sauvegarde au service de projet, qui renvoie la confirmation au contrôleur de projet, et finalement à la page Blockly.

Ce diagramme illustre deux méthodes de sauvegarde des données de travail de l'utilisateur : une sauvegarde manuelle via HTTP et une sauvegarde automatique via WebSocket pour un suivi en temps réel des modifications.

1.1.4 Espace de Travail Collaboratif

Ce diagramme de séquence illustre le fonctionnement de l'espace de travail collaboratif dans iBlock, mettant en évidence les interactions entre l'éditeur de workspace (Next.js), le contrôleur de projet (NestJS), le service de projet (NestJS), et la base de données (MongoDB).

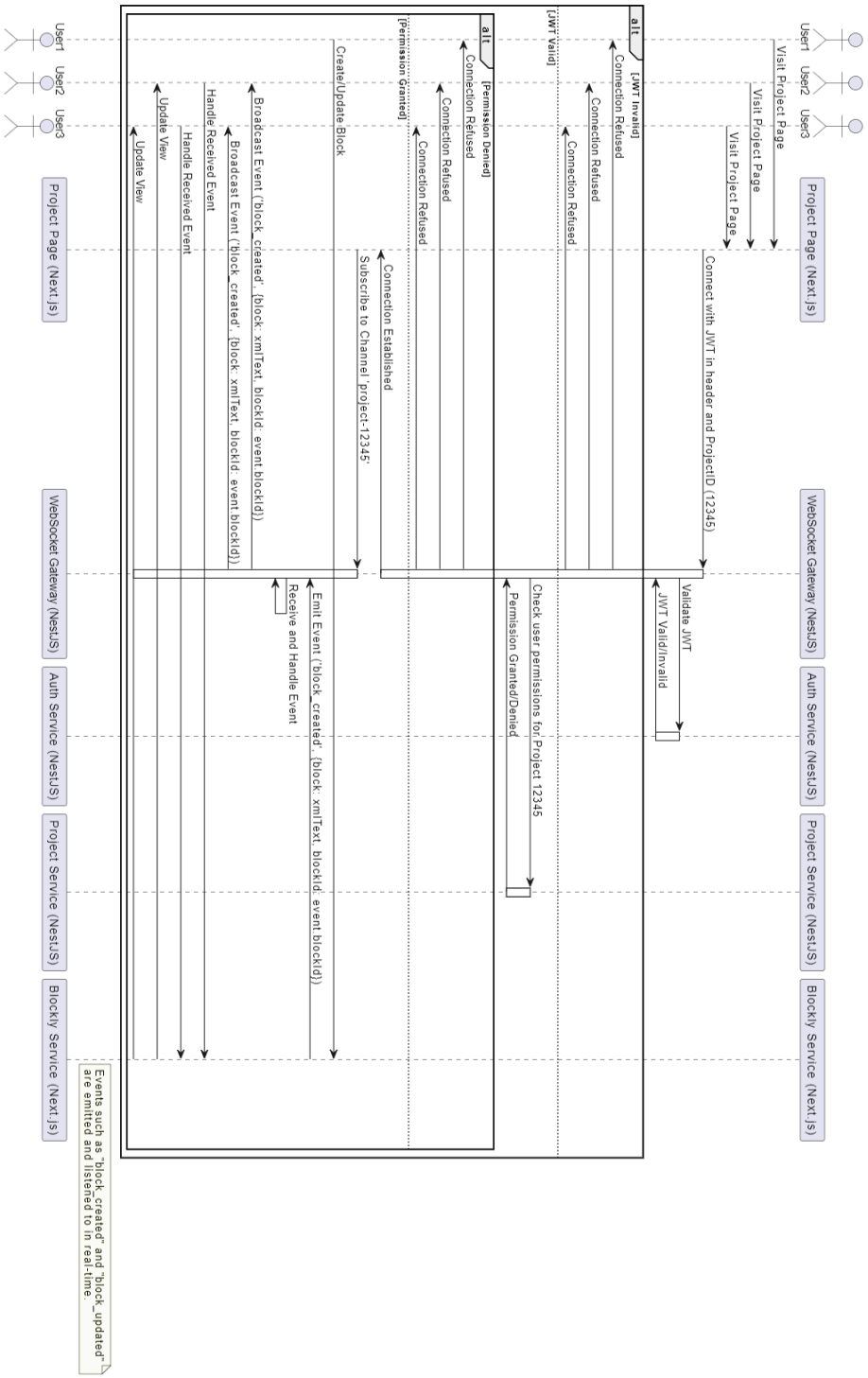

Figure 1.4: Diagramme de Séquence pour l'Espace de Travail Collaboratif

Description du Diagramme:

L'espace de travail collaboratif d'iBlock permet à plusieurs utilisateurs de travailler simultanément sur un même projet. Lorsqu'un utilisateur accède à l'interface Blockly de l'espace de travail collaboratif, l'éditeur de workspace affiche l'interface Blockly, permettant à l'utilisateur de modifier le workspace Blockly. Pour sauvegarder les modifications, l'utilisateur clique sur "Sauvegarder", ce qui déclenche une requête POST vers le contrôleur de projet. Le contrôleur de projet appelle le service de projet, qui sauvegarde les données du workspace dans la base de données MongoDB. La base de données confirme la sauvegarde, et la confirmation est renvoyée à l'utilisateur via l'éditeur de workspace.

En plus de la sauvegarde manuelle, iBlock utilise WebSocket pour synchroniser les modifications en temps réel entre les collaborateurs. Lorsqu'une modification est détectée dans le workspace, les données sont envoyées via WebSocket au contrôleur de projet. Le contrôleur de projet appelle le service de projet pour stocker une copie du travail, qui est ensuite sauvegardée dans la base de données. La base de données confirme la sauvegarde, et la confirmation est renvoyée à l'utilisateur via l'éditeur de workspace. Ce processus assure une collaboration fluide et une synchronisation en temps réel des modifications du workspace entre les utilisateurs.

1.1.5 Diagramme de Séquence pour le Suivi des Mises à Jour du Workspace Utilisant WebSocket pour l'enseignant

Ce diagramme de séquence illustre le processus de suivi des mises à jour du workspace par l'enseignant en utilisant WebSocket, mettant en évidence les interactions entre la page de suivi (Next.js), les contrôleurs et services de méta-projets et de projets (NestJS), le serveur WebSocket (NestJS), et la base de données (MongoDB).

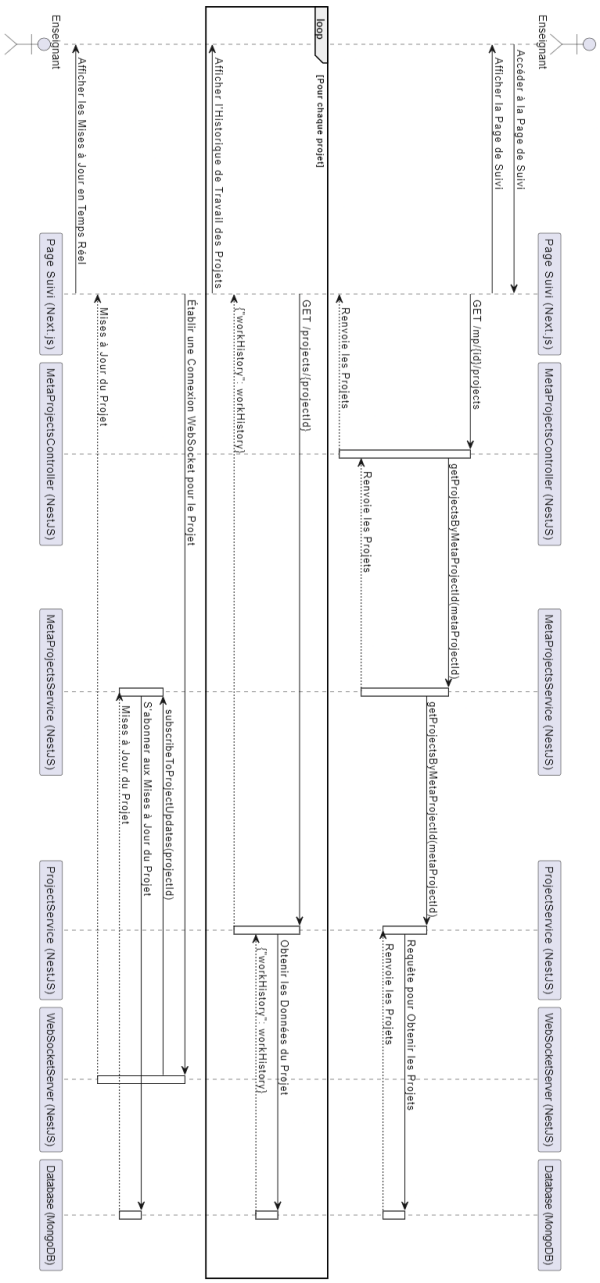

Figure 1.5: Diagramme de Séquence pour le Suivi des Mises à Jour du Workspace Utilisant WebSocket pour l'enseignant

Description du Diagramme:
L'enseignant peut suivre les mises à jour du workspace des étudiants en

temps réel grâce à la fonctionnalité de suivi d'iBlock. L'enseignant accède à la page de suivi des projets, qui affiche la liste des projets. La page de suivi récupère la liste des projets depuis le contrôleur de méta-projets. Pour chaque projet, la page de suivi récupère les données du projet, y compris l'historique de travail, depuis le service de projet. L'historique de travail des projets est ensuite affiché à l'enseignant.

Pour permettre un suivi en temps réel, la page de suivi établit une connexion WebSocket avec le serveur WebSocket pour chaque projet. Le serveur WebSocket s'abonne aux mises à jour du projet dans la base de données MongoDB. Lorsque des mises à jour sont effectuées sur un projet, la base de données les envoie au serveur WebSocket, qui les transmet à la page de suivi. La page de suivi affiche alors les mises à jour en temps réel à l'enseignant, offrant une vue actualisée du progrès des étudiants. Ce mécanisme de suivi en temps réel, rendu possible par WebSocket, facilite la supervision et l'accompagnement des étudiants par l'enseignant.

1.2 Conclusion

Ce chapitre a exploré la conception d'iBlock à travers des diagrammes de séquence, illustrant les interactions entre les composants frontend et backend, la base de données, et les utilisateurs. Ces diagrammes mettent en évidence les flux de communication et les processus clés pour l'authentification, la création de projets, la sauvegarde des données, la collaboration, et le suivi des progrès des étudiants. La conception d'iBlock s'articule autour d'une architecture logicielle robuste et modulaire, facilitant la maintenance, l'évolution, et l'intégration de nouvelles fonctionnalités.