Présenté par:

- Kevin CORONA
- Paul ARNAUD
- Bastien NIETO
- Valentin LEFEBVRE
- Nicolas MAUDUIT
- Flavio TAVERNIER

Le:

11 Décembre 2025

Table de matières :

1. Introduction

Le projet SkiOx représente une plateforme complète et intégrée de gestion et de secours destinée aux sportifs en montagne, notamment aux skieurs et aux sportifs en situation de haute altitude. Ce rapport technique couvre l'ensemble de la documentation relative à la fois au matériel (hardware) et au logiciel (software) du système, permettant une compréhension globale de l'architecture, du déploiement et de la maintenance du projet.

Le système SkiOx est conçu pour délivrer automatiquement de l'oxygène (ou, dans le prototype, une simulation en utilisant de l'eau) en fonction de paramètres physiologiques mesurés. L'objectif principal est d'augmenter les chances de survie d'un sportif pris dans une avalanche ou en situation de détresse respiratoire.

Le projet est hébergé sur la plateforme GitLab institutionnelle à l'adresse suivante : https://iut-git.unice.fr/skiox/SkiOx

2. Objet et objectifs du projet

2.1 Contexte

SkiOx est un système de secours portable conçu pour fonctionner dans les environnements de haute montagne. Le système intègre :

- Une unité électronique de mesure et de contrôle •

Des capteurs de paramètres physiologiques

- Une plateforme logicielle complète (backend et frontend) •

Des systèmes de communication sans fil

- Une infrastructure de gestion de données

2.2 Objectifs principaux

Les objectifs du projet SkiOx sont :

1. Fournir un système autonome capable de mesurer les paramètres physiologiques critiques (SpO_2, fréquence cardiaque, etc.)
2. Déclencher automatiquement la distribution d'oxygène en cas de détresse respiratoire
3. Communiquer les données en temps réel vers une plateforme centralisée pour supervision et archivage
4. Faciliter l'intervention des équipes de secours en montagne grâce à des données précises et en temps réel
5. Offrir une plateforme d'intégration IoT modulaire et extensible

2.3 Avertissement important

Le prototype décrit dans ce rapport utilise de l'eau et une bouteille squeezy pour simuler une bouteille d'oxygène sous pression.
Toute adaptation à une vraie bouteille d'oxygène doit impérativement être conçue, validée et installée par des professionnels qualifiés

, conformément aux normes et réglementations en vigueur sur les systèmes à oxygène sous pression. Ce document ne constitue pas une certification de sécurité médicale.

3. Architecture générale du système

3.1 Vue d'ensemble

Le système SkiOx suit une architecture modulaire et distribuée, typique des applications web full-stack modernes. Cette approche garantit une séparation claire des préoccupations, facilitant la scalabilité, la maintenance et l'évolution de chaque composant.

Composants principaux

Composant	Technologies	Description
Backend API	Node.js/Express, Docker	API RESTful implémentant la logique métier, gestion de la base de données et intégration des systèmes IoT
Base de données	MySQL, Docker Compose	Système de persistence des données avec interface d'administration phpMyAdmin
Interface utilisateur	Angular, TypeScript, PWA	Application front-end avec modules séparés pour les rôles de sauveteur et de sportif
Documentation	Swagger UI, Draw.io	Documentation API interactive et schémas d'architecture du système
Matériel	Arduino, capteurs IoT	Système portable d'acquisition et de contrôle

3.2 Principes de conception

L'architecture du système SkiOx repose sur les principes suivants :

- **Scalabilité** : Services indépendants communiquant principalement via protocole HTTP
- **Modularité** : Séparation claire des domaines fonctionnels
- **Résilience** : Chaque service peut être redémarré indépendamment
- **Testabilité** : Isolation des dépendances facilitant les tests unitaires et d'intégration
- **Extensibilité** : Architecture permettant l'intégration de nouveaux capteurs IoT et services

3.3 Schéma fonctionnel

Le système peut se décomposer en cinq sous-ensembles principaux :

1. **Sous-système de mesure** : Potentiomètres (simulation) OU capteur MAX30102 (réel) connectés aux entrées de l'Arduino
2. **Sous-système de commande** : Arduino Uno avec boutons de contrôle
3. **Sous-système d'affichage et d'indication** : Écran LCD I2C et LED RGB
4. **Sous-système de communication** : Module HC-05 connecté sur UART de l'Arduino
5. **Sous-système d'action** : Commande de l'électrovanne 12 V via module relais

4. Composants matériels

4.1 Principe de fonctionnement

Le système de secours se compose de plusieurs éléments distincts :

- Une unité électronique de mesure et de contrôle basée sur un Arduino Uno
- Un capteur de paramètres physiologiques (capteur MAX30102 de SpO_2 et fréquence cardiaque, complété par des potentiomètres de simulation)
- Un module de communication Bluetooth HC-05 pour transmettre les mesures à un système backend
- Un sous-système d'indication d'état (LED RGB et écran LCD I2C)
- Un bouton d'activation du secours et un bouton de sélection du mode
- Une électrovanne 12 V commandée par Arduino pour contrôler le débit de fluide
- Un module de relais 12 V et une batterie Li-ion 12,6 V – 3000 mAh pour l'alimentation
- Une bouteille squeezy connectée à l'électrovanne par des tubes pneumatiques

4.2 Fonctionnement logique

Le fonctionnement du système suit la séquence suivante :

1. Le système est mis sous tension ; l'Arduino initialise les capteurs, l'écran LCD, la LED RGB et la liaison Bluetooth
2. L'utilisateur sélectionne le mode « simulation » (potentiomètres) ou mode « capteur réel » (MAX30102) via un bouton dédié
3. Lorsqu'un sportif se retrouve en situation de détresse, le bouton de secours est pressé pour activer le système

4. En fonction des mesures (SpO_2, fréquence cardiaque, « niveau de batterie » simulant l'état énergétique) et de la logique logicielle, l'Arduino décide de déclencher l'électrovanne ou de rester fermée
5. Les données mesurées sont affichées sur l'écran LCD et envoyées via Bluetooth HC-05 au backend pour traitement et enregistrement
6. L'état global du système (ON/OFF) est indiqué par la LED RGB : rouge = système hors service, vert = système opérationnel

4.3 Liste complète des composants

4.3.1 Composants déjà implémentés

Microcontrôleur

- 1 × Arduino Uno (ou équivalent, 5 V, ATmega328P)

Interface utilisateur

- 1 × bouton poussoir principal (pull-down) pour l'activation du système de secours
- 1 × LED RGB (utilisation des canaux rouge et vert avec résistances de limitation)

Capteurs de simulation

- 3 × potentiomètres (valeur typique 10 kΩ) simulant :
  - le niveau de la batterie
  - le niveau d'oxygène
  - la fréquence cardiaque

Affichage

- 1 × écran LCD 16×2 (ou 20×4) avec module I2C (PCF8574 ou équivalent) pour affichage des valeurs et de l'état du système

Communication

- 1 × module Bluetooth HC-05 (interface série UART, 3,3–5 V) pour transmettre les données au backend

Signalisations diverses

- 2–3 × résistances pour LED (220–330 Ω typiquement)
- Divers fils de connexion (dupont mâle-mâle, mâle-femelle, etc.)
- Plaquette de prototypage (breadboard) ou carte PCB selon version

4.3.2 Composants à implémenter

Capteur physiologique réel

- 1 × module capteur MAX30102 (capteur SpO_2 + fréquence cardiaque, interface I^2C)

Sélecteur de mode

- 1 × deuxième bouton poussoir pour alterner entre mode simulation et mode capteur réel

Système d'injection fluide (simulation oxygène)

- 1 × électrovanne 12 V DC adaptée aux liquides (pour le prototype : eau)
- 1 × module relais 12 V (commande via GPIO Arduino, isolation galvanique fortement recommandée)
- 1 × batterie Li-ion 12,6 V – 3000 mAh (avec chargeur et 2 lignes de sortie)
- 1 × bouteille squeezy (bouteille souple) remplie d'eau pour simuler une bouteille d'oxygène sous pression
- Tubes de connexion (tuyaux pneumatiques nylon pour air/eau)

Mécanique et intégration

- Boîte / boîtier imprimé en 3D pour contenir l'Arduino, les modules électroniques, la batterie, les boutons, l'écran LCD et la LED RGB
- Passe-câbles et supports internes (impression 3D ou autres)

4.4 Câblage matériel

4.4.1 Entrées analogiques

- **A0** : potentiomètre 1 (niveau batterie simulé)
- **A1** : potentiomètre 2 (niveau d'oxygène simulé)
- **A2** : potentiomètre 3 (fréquence cardiaque simulée)

En mode « capteur réel », A0–A2 peuvent être ignorées logiciellement ou réutilisées pour d'autres mesures.

4.4.2 Bus I2C

- **SDA (A4 sur Arduino Uno)** → SDA LCD + SDA MAX30102
- **SCL (A5 sur Arduino Uno)** → SCL LCD + SCL MAX30102

Les deux modules partagent le même bus I2C ; leurs adresses I2C doivent être compatibles.

4.4.3 Liaisons numériques

- **D2** : bouton d'activation secours (pull-down)
- **D3** : bouton de sélection mode (simulation/capteur réel)
- **D4** : LED RGB – canal rouge (via résistance)
- **D5** : LED RGB – canal vert (via résistance)
- **D6** : commande du module relais (IN du relais 12 V)
- **D10** : RX Arduino ← TX HC-05 (via diviseur de tension si nécessaire pour 3,3 V)
- **D11** : TX Arduino → RX HC-05

4.4.4 Alimentation

5 V Arduino :

- Alimente l'Arduino, le LCD, le MAX30102, le HC-05, la LED RGB, les boutons, etc.
- Les courants restent modestes (quelques centaines de mA maximum)

12 V (batterie Li-ion 12,6 V) :

- Alimente l'électrovanne via module relais
- Le module relais est commandé en 5 V (côté Arduino), mais commute la ligne 12 V (côté puissance)

Séparation des masses :

La masse (GND) de la partie 5 V (Arduino) doit être reliée à la masse 12 V pour une référence commune des signaux de commande, tout en conservant l'isolation galvanique entre côté commande et côté puissance via le relais.

4.5 Intégration du capteur MAX30102

4.5.1 Description

Le MAX30102 est un capteur optique intégrant des LED rouge et infrarouge et un photodétecteur, destiné à mesurer la saturation en oxygène du sang (SpO_2) et la fréquence cardiaque. Il communique via le bus I^2C (SDA/SCL) et fonctionne sous 3,3–5 V.

4.5.2 Connexions électriques

Brochage typique du module :

- **VIN** → 3,3 V ou 5 V (selon module)
- **GND** → GND Arduino **SCL**
- → SCL Arduino (A5) **SDA**
- → SDA Arduino (A4)
- **INT (optionnel)** → entrée numérique Arduino (par ex. D7) pour gérer les interruptions

4.5.3 Intégration logicielle

En mode simulation, le code lit simplement les valeurs analogiques A0–A2. En mode capteur réel, le code :

1. Initialise le MAX30102 sur le bus I2C
2. Configure la fréquence d'échantillonnage, les courants de LED, etc.
3. Lit périodiquement les valeurs de SpO_2 et de fréquence cardiaque à partir du capteur
4. Met à jour les variables internes qui remplacent les valeurs issues des potentiomètres

Le reste de la logique (décision d'ouvrir ou non l'électrovanne) est identique entre les deux modes.

4.6 Module Bluetooth HC-05

4.6.1 Rôle

Le module HC-05 assure la communication entre l'Arduino et un système backend (ordinateur, smartphone, serveur, etc.) via Bluetooth série (profil SPP). Les valeurs de SpO_2, fréquence cardiaque, niveau de batterie et état du système y sont transmises à intervalle régulier.

4.6.2 Connexion

- **VCC** → 5 V Arduino
- **GND** → GND Arduino
- **TX** → RX Arduino (D10, éventuellement via un pont diviseur si nécessaire)
- **RX** → TX Arduino (D11)
- **EN/KEY (optionnel)** → pour passer en mode commande (AT)

4.6.3 Paramétrage

- **Nom par défaut** : « HC-05 »
- **Baud rate typique** : 9600 bauds
- Peut être modifié via commandes AT si besoin

4.7 Chaîne de puissance et gestion de l'électrovanne

4.7.1 Architecture de puissance

1. **Batterie Li-ion 12,6 V – 3000 mAh** : Fournit la tension nécessaire à l'électrovanne avec système de protection (BMS) adapté
2. **Module relais 12 V** : Bobine commandée en 5 V par l'Arduino (sortie D6), contacts de puissance commutant la ligne 12 V
3. **Électrovanne 12 V** : Fonctionnement typique normalement fermée (NC), s'ouvre lorsque la bobine est alimentée en 12 V

4.7.2 Logique de commande

Lorsque :

- le système est ON
- le bouton de secours est pressé
- et que les valeurs physiologiques dépassent des seuils critiques (SpO_2 faible, fréquence cardiaque anormale, etc.)

L'Arduino active la sortie D6 → le module relais ferme le contact → l'électrovanne 12 V est alimentée → le fluide (eau / oxygène simulé) circule depuis la bouteille.

La durée d'activation peut être :

- Temps fixe (ex. 5–10 s, configurable)

- Ou fonction de l'état du patient (plus long si les paramètres restent critiques)

4.8 Interface utilisateur matérielle

4.8.1 LED RGB (état du système)

- **Rouge** : système OFF ou en erreur
- **Vert** : système ON et opérationnel Implémentation

:

- D4 → canal rouge de la LED (avec résistance)
- D5 → canal vert de la LED (avec résistance)

4.8.2 Écran LCD I2C

Affichages utilisés :

- **Ligne 1** : SpO_2 (réelle ou simulée), Fréquence cardiaque (réelle ou simulée)
- **Ligne 2** : État du système : MODE: SIMU ou MODE: REAL Le

LCD est connecté en I2C et alimenté en 5 V.

4.8.3 Boutons

- **Bouton de secours (D2)** : Pression courte pour activation/désactivation du mode secours
- **Bouton de sélection de mode (D3)** : Pression courte pour basculer entre mode SIMULATION et mode CAPTEUR

4.9 Logique logicielle matérielle

4.9.1 Initialisation

1. Initialiser ports E/S (LED, relais, boutons)
2. Initialiser bus I2C
3. Initialiser écran LCD
4. Initialiser module Bluetooth HC-05 (Serial)
5. Initialiser MAX30102 (si disponible)
6. Afficher écran de démarrage
7. Mettre LED en rouge (système OFF)

4.9.2 Boucle principale (pseudo-code)

Lire bouton ON/OFF (D2)

- Si pressé : basculer état du système (ON ↔ OFF)
- Mettre à jour LED RGB (rouge si OFF, vert si ON)

Si système est ON :

- Lire bouton de mode (D4)
  - Si pressé : basculer mode (SIMULATION ↔ CAPTEUR)
- Lire bouton secours (D3)
  - Si pressé : marquer que secours est demandé
- Selon le mode :
  - SIMULATION : lire A0 (Bat), A1 (O2 simulé), A2 (HR)
  - CAPTEUR : lire données MAX30102 (SpO2, HR) ; A0 reste pour batterie
- Calculer indicateurs (niveaux, seuils)
- Si secours est demandé ET indicateurs au-delà des seuils critiques :
  - Activer électrovanne (GPIO D7 = HIGH) pendant durée prévue (ex. 5-10 s)
  - Afficher "Val:ON" sur LCD
- Sinon :
    - Désactiver électrovanne (GPIO D7 = LOW)
    - Afficher "Val:OFF" sur LCD
  - Mettre à jour affichage LCD (SpO2, HR, Bat, Mode, État Valve)
  - Envoyer données via Bluetooth HC-05 (série D8/D9)
  - Gérer temporisations, filtrage des mesures, etc.

Sinon (système OFF) :

- LED RGB rouge

- Électrovanne fermée
- Affichage LCD peut montrer "SYSTEM OFF" ou rester fig

4.10 Procédure d'assemblage matériel

4.10.1 Étape 1 – Préparation des composants

1. Vérifier la présence de tous les composants
2. Préparer les outils : fer à souder, tournevis, coupe-câble, pince à dénuder, multimètre, imprimante 3D si nécessaire

4.10.2 Étape 2 - Câblage basse tension (5 V)

1. Monter l'Arduino et les modules sur une breadboard ou un PCB
2. Réaliser les connexions 5 V et GND communes
3. Connecter les signaux SDA/SCL, TX/RX, entrées analogiques, sorties LED/relais, boutons

4.10.3 Étape 3 – Câblage puissance 12 V

1. Connecter la batterie 12,6 V au module relais
2. Relier la sortie du relais à l'électrovanne
3. Relier l'autre borne de l'électrovanne au GND 12 V
4. Vérifier les connexions avec un multimètre hors tension

4.10.4 Étape 4 – Programmation de l'Arduino

1. Charger le firmware sur l'Arduino via l'IDE Arduino
2. Vérifier la lecture des potentiomètres et/ou du MAX30102
3. Vérifier l'affichage sur le LCD et la communication Bluetooth
4. Vérifier la commande de la LED RGB et l'activation du relais

4.10.5 Étape 5 – Intégration dans le boîtier

1. Imprimer le boîtier 3D ou utiliser un boîtier standard

2. Fixer les composants à l'intérieur
3. Sortir les tuyaux pneumatiques vers l'extérieur et connecter à la bouteille squeezy
4. Fermer le boîtier et effectuer un test fonctionnel complet

4.11 Considérations mécaniques

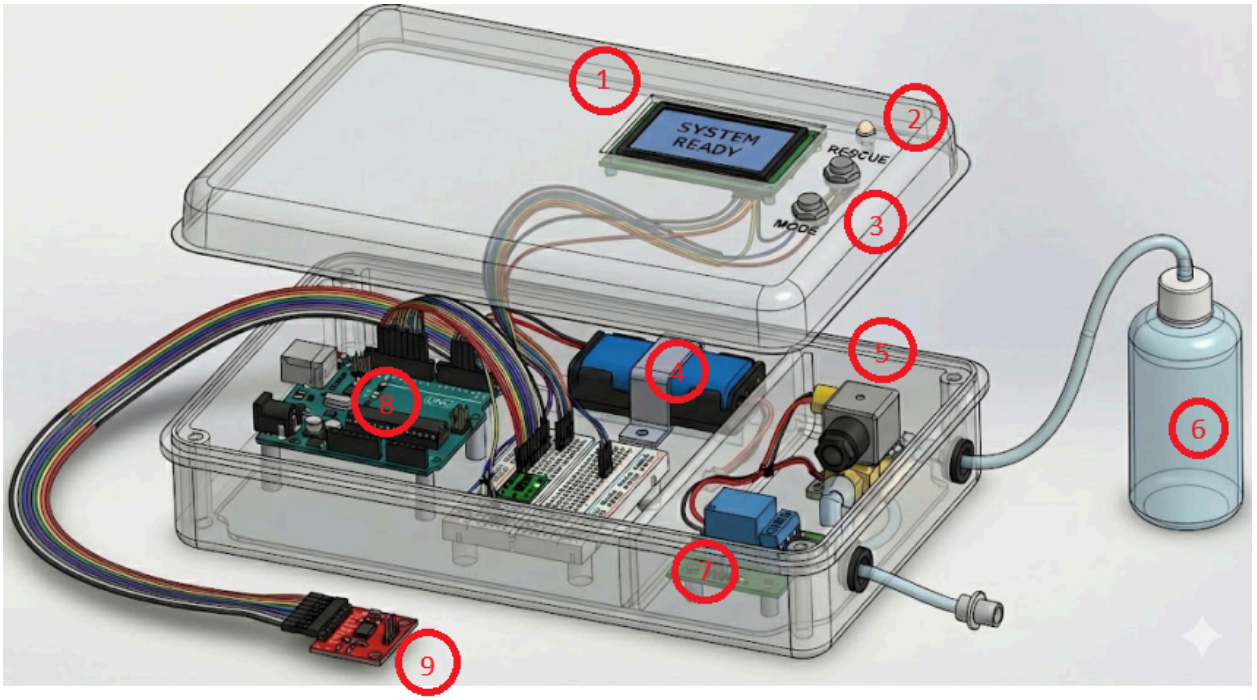
4.11.1 Objectifs du boîtier

- Protéger les composants électroniques des chocs et de l'humidité
- Regrouper dans un même boîtier : Arduino, modules, écran LCD, LED RGB, boutons, batterie
- Offrir une interface claire et ergonomique pour l'utilisateur final

4.11.2 Recommandations

- Concevoir un boîtier en deux parties (couvercle + base) pour faciliter l'accès • Prévoir des ouvertures pour l'écran LCD, la LED RGB, les boutons, les connecteurs de charge, le passage des tuyaux
- Intégrer des supports internes pour fixer la carte Arduino, le module relais, la batterie
- Utiliser des vis ou clips pour fermer le boîtier

4.12 Schéma du projet



1. Écran LCD avec Module I2C
2. LED RGB
3. Boutons (Pull-down)
4. Batterie Li-ion (Alimentation du système) 12 V. 3000 mpA
5. Électrovanne 12 V.
6. Bouteille Squeeze
7. Module relais 12 V.
8. Arduino UNO
9. Capteur MAX30102

5. Architecture logicielle

5.1 Stack technologique

L'environnement de développement et de déploiement du projet SkiOx repose sur les technologies suivantes :

- **Node.js (version LTS)** : Environnement d'exécution JavaScript côté serveur
- **npm (Node Package Manager)** : Gestionnaire de dépendances
- **Docker Desktop** : Orchestration et gestion des conteneurs pour la base de données
- **Angular** : Framework front-end moderne
- **MySQL** : Base de données relationnelle
- **Express.js** : Framework web pour Node.js
- **Swagger UI** : Documentation API interactive

5.2 Architecture détaillée

5.2.1 Services et ports

L'architecture full-stack de SkiOx nécessite l'initialisation simultanée de quatre services distincts :

Service	Adresse	Port	Description
Application Web	http://localhost:4200	4200	Interface utilisateur pour sportifs et sauveteurs
API Backend	http://localhost:3000	3000	Point d'entrée des requêtes RESTful
Administration Base de Données	http://localhost:8080	8080	Gestion visuelle via phpMyAdmin
Documentation API	http://localhost:3001/api-docs	3001	Documentation interactive Swagger UI

5.2.2 Infrastructure de données (Service Backend)

La couche de persistence est gérée par conteneurisation Docker. L'exécution de la commande suivante, depuis le répertoire Back/, initialise l'instance de base de données MySQL et la console d'administration phpMyAdmin :

```
docker-compose up
```

Cette commande effectue automatiquement les opérations suivantes :

- Initialisation du serveur MySQL avec le schéma défini dans les scripts SQL • Chargement des données de test (seeds) dans la base de données
- Déploiement de l'interface d'administration phpMyAdmin

5.2.3 Serveur API (Backend)

Le cœur logique du système est fourni par une API RESTful développée avec Node.js et Express. Pour lancer le serveur API en mode développement, exécuter depuis le répertoire API/ :

```
npm run dev
```

Cette configuration active le rechargement automatique du code (hot-reload), facilitant l'itération rapide lors du développement. Pour un environnement de production, utiliser :

```
npm start
```

5.2.4 Interface utilisateur (Frontend)

L'application front-end, développée avec le framework Angular et architecturée comme une Progressive Web Application (PWA), est lancée depuis le répertoire IHM/ :

```
npm start
```

Cette commande compile dynamiquement les modules Angular et lance le serveur de développement intégré.

5.2.5 Documentation technique interactive

La documentation interactive de l'API, générée avec Swagger UI, est déployée via la commande suivante, depuis le répertoire swaggerui/ :

```
node app.js
```

5.3 Organisation du code

Le code est organisé selon une hiérarchie de répertoires conçue pour faciliter la navigation, la maintenance et la modularité :

Back/ : Ce répertoire regroupe toute la logique côté serveur. Il contient le fichier docker-compose.yml qui permet de lancer l'environnement complet (Base de données et API). Le sous-dossier API/src/iot/ est critique pour le projet car il gère l'interface avec le module Bluetooth HC-05 et le parsing des données physiologiques.

IHM/ : Contient l'application web Angular. La présence du fichier ngsw-config.json confirme l'architecture de type *Progressive Web App* (PWA), permettant une utilisation hors-ligne partielle pour les sauveteurs.

Docs/swaggerui/ : Héberge une mini-application Node.js indépendante servant à exposer la documentation de l'API générée via Swagger, facilitant les tests sans passer par l'interface graphique principale.

Hard/ : Regroupe les sources relatives à la programmation du microcontrôleur Arduino et la gestion des capteurs physiques (MAX30102).

Structure simplifié de l'arborescence du code :

```
Skiox/
├── Back/
│   ├── API/
│   │   ├── src/
│   │   │   ├── config/
│   │   │   ├── controllers/
│   │   │   ├── database/
│   │   │   ├── iot/
│   │   │   ├── middleware/
│   │   │   └── routes/
│   │   ├── app.js
│   │   └── index.js
│   └── BDD/
│       └── init/
├── Docs/
│   ├── swaggerui/
│   └── SchemaProjet.drawio
├── Hard/
├── IHM/
│   ├── public/
│   ├── src/
│   ├── angular.json
│   └── ngsw-config.json
└── README.md
```

5.4 Conventions de codage

Le projet respecte les conventions suivantes :

- **Langage côté serveur** : JavaScript/Node.js avec support TypeScript
- **Langage côté client** : TypeScript avec framework Angular
- **Linting** : ESLint pour garantir la cohérence du code **Gestion**
- **des dépendances** : npm pour tous les modules **Conventions**
- **de nommage** :
  - camelCase pour JavaScript et TypeScript
  - snake_case pour SQL
- **Documentation** : JSDoc pour les fonctions publiques

5.5 Composants logiciels principaux

5.5.1 Backend API

L'API RESTful est le cœur du système logiciel. Elle assure :

- La gestion des données physiologiques reçues des appareils IoT •

Le stockage et la persistence des données

- La fourniture d'endpoints pour l'interface utilisateur •

L'intégration avec les systèmes de communication

Répertoire structure :

- **controllers/** : Logique métier (gestion de base de données, santé du système, intégration IoT)
- **routes/** : Définition des endpoints RESTful
- **config/** : Configuration de l'application
- **middleware/** : Middleware Express (authentification, validation)
- **database/** : Modules de gestion de base de données
- **iot/** : Modules d'intégration des systèmes IoT

5.5.2 Base de données MySQL

La base de données gère :

- Les informations des utilisateurs (sportifs et sauveteurs) •

Les données physiologiques en historique

- Les événements d'activation du système
- Les métadonnées de localisation et conditions environnementales Le

schéma SQL est versionné dans BDD/init/schema.sql.

5.5.3 Interface utilisateur Angular

L'application Angular est architecturée avec deux modules principaux :

- **Module Sauveteur** : Interface spécifique pour les équipes de secours
  - Vue des sportifs inscrits
  - Suivi en temps réel de l'état physiologique
  - Historique des interventions
  - Planification des opérations de secours
- **Module Sportif** : Interface pour les utilisateurs finaux
  - Configuration du système personnel
  - Suivi des paramètres physiologiques
  - Alertes de détresse
  - Historique des données

5.5.4 Documentation interactive Swagger UI

La documentation API est générée automatiquement via Swagger UI, fournissant :

- Description de tous les endpoints disponibles
- Schémas de requête/réponse
- Exemples d'utilisation
- Interface de test interactive

6. Intégration des systèmes IoT

6.1 Architecture du système IoT

L'application SkiOx intègre des systèmes IoT pour la collecte de données physiologiques et biométriques. Le module backend responsable de cette intégration est situé dans **Back/API/src/iot/**.

6.2 Flux de données IoT

Les données collectées par les capteurs IoT :

1. **Transmission** : Sont transmises vers l'API backend via protocoles de communication dédiés (Bluetooth série, HTTP)
2. **Traitement** : Sont traitées et validées par la logique métier (vérification des seuils, détection d'anomalies)
3. **Stockage** : Sont stockées dans la base de données MySQL pour archivage et historique
4. **Disponibilité** : Sont rendues disponibles à l'interface utilisateur via les endpoints RESTful

6.3 Sécurité et fiabilité de la communication IoT

Les communications IoT implémentent :

- **Validation des données entrantes** : Vérification de la cohérence et plausibilité des mesures
- **Gestion des erreurs de transmission** : Mécanismes de retry et reconnexion
- **Authentification et chiffrement** : Protection des données sensibles en transit

- **Logging et monitoring** : Enregistrement des anomalies et des défaillances
- **Seuils d'alerte** : Détection automatique de conditions critiques

6.4 Module d'intégration IoT (Back/API/src/iot/)

Le module IoT gère :

- La réception et le parsing des trames de données du dispositif matériel •

La validation des paramètres physiologiques reçus

- La détermination des seuils critiques et des conditions d'alerte
- L'interaction avec le contrôleur de backend pour persistance
- La génération des événements pour notification aux sauveteurs

Points d'entrée principaux :

- Réception via Bluetooth HC-05 des données du système matériel
- Endpoints API pour consultation historique et données en temps réel •

Webhooks pour notification en temps réel aux applications clients

7. Gestion de version et workflow de développement

7.1 Modèle de branching (GitFlow)

Le projet SkiOx adopte un workflow inspiré de GitFlow, qui structure le développement et les déploiements selon le tableau suivant :

Branche	Rôle	Description
main	Production	Version stable utilisée pour les déploiements en production
dev	Développement	Intégration des nouvelles fonctionnalités et corrections avant release
feature/xxx	Développement des fonctionnalités	Branches temporaires créées à partir de dev (ex: feature/6-implementation-navbar)

Avantages du workflow :

- Séparation claire entre le développement actif (dev) et la version stable (main) •

Facilite la gestion des versions et des releases

- Permet de préparer des hotfixes sans interrompre le développement principal

7.2 Configuration Git

7.2.1 Configuration de .gitignore

Le projet utilise des fichiers .gitignore au niveau racine et dans les sous-dossiers pour exclure automatiquement les fichiers non essentiels :

Fichiers exclus :

- node_modules/ : Dépendances du projet (reconstruites via npm install) •

.env et .env.local : Variables d'environnement sensibles

- /dist/, /build/ : Artefacts de compilation
- .vscode/, .idea/ : Fichiers de configuration d'IDE
- Fichiers temporaires du système (.DS_Store, Thumbs.db, etc.)

7.2.2 Pipeline CI/CD

Le fichier .gitlab-ci.yml à la racine du projet configure les pipelines automatisés :

- **Tests** : Exécution automatique des suites de tests
- **Build** : Compilation et bundling du code
  - **Déploiement** : Déploiement automatique sur l'environnement approprié

7.3 Processus de contribution

7.3.1 Préparation de l'environnement de développement

Étape 1 : Clonage du dépôt

```
git clone https://iut-git.unice.fr/skiox/SkiOx.git cd SkiOx
```

Étape 2 : Installation des dépendances

Installation des dépendances du backend

```
cd Back/API npm install
```

Installation des dépendances du frontend

```
cd ../../IHM npm install cd ..
```

Étape 3 : Gestion des branches

Pour contribuer au projet, suivre le workflow de gestion de branches :

Travailler à partir de la branche dev pour les nouvelles fonctionnalités

```
git checkout dev git pull origin dev
```

Créer une branche feature dédiée selon la nomenclature

```
git checkout -b feature/description-fonctionnalite
```

7.3.2 Développement et validation

Implémentation des modifications :

1. Rédiger un code lisible et bien commenté
2. Respecter les conventions de nommage (camelCase pour JavaScript, snake_case pour SQL)
3. Documenter les fonctions publiques avec des commentaires JSDoc

Tests locaux :

Tests du backend (Node.js)

```
cd Back/API npm test
```

Tests du frontend (Angular)

```
cd IHM npm test
```

Commits :

Effectuer des commits réguliers avec des messages explicites et concis :

```
git commit -m "Fonctionnalité: implémentation de la nouvelle
API de mesure" git commit -m "Correction: résolution du bug
dans la validation des données"
```

Chaque commit doit représenter une unité logique de travail distinct.

Sécurité :

- Respecter strictement le fichier .gitignore à tous les niveaux du projet
- Ne jamais committer de fichiers sensibles (node_modules/, .env, .env.local, clés d'API)

7.3.3 Soumission et révision des modifications

Envoi vers le dépôt distant :

```
git push origin feature/description-fonctionnalite
```

Création d'une Merge Request (MR) :

1. Accéder à la plateforme GitLab et naviguer vers l'onglet « Merge Requests »
2. Cliquer sur « New Merge Request »
3. Sélectionner la branche source (feature/...) et la branche cible (dev par défaut, ou main pour les corrections critiques)
4. Remplir le titre et la description avec les informations pertinentes :
    - Résumé des changements

    - Justification des modifications
    - Impact sur d'autres composants si applicable
5. Assigner des reviewers selon le composant modifié
6. Soumettre la Merge Request pour révision

7.4 Processus des Merge Requests

Les Merge Requests sont soumises à un processus de révision structuré garantissant la qualité et la cohérence du code avant fusion.

7.4.1 Reviewers par domaine

Domaine	Reviewers	Vérification
Frontend (IHM)	Nicolas, Flavio	Revue technique du code Frontend
Frontend (Sécurité/Architecture)	Valentin	Vérification de la sécurité et des aspects architecturaux
Backend API	Bastien, Valentin	Revue technique et revue mutuelle

7.4.2 Processus de révision

1. Le reviewer examine le code et les changements proposés
2. Demande des clarifications ou des modifications si nécessaire
3. Approuve ou rejette la Merge Request
4. Une fois approuvée, la branche est fusionnée dans la branche cible
5. La branche feature est supprimée après fusion

8. Déploiement et utilisation

8.1 Démarrage complet du système

Pour démarrer l'ensemble du système SkiOx, exécuter les commandes suivantes dans des terminaux séparés :

Terminal 1 – Infrastructure de données :

```
cd SkiOx/Back docker-compose up
```

Terminal 2 – API Backend :

```
cd SkiOx/Back/API npm run dev
```

Terminal 3 – Frontend :

```
cd SkiOx/IHM npm start
```

Terminal 4 – Documentation API :

```
cd SkiOx/swaggerui node app.js
```

8.2 Vérification du déploiement

Une fois tous les services initialisés, vérifier que l'écosystème complet est accessible :

- **Application Web** : http://localhost:4200

- **API Backend** : http://localhost:3000
- **Administration BDD** : http://localhost:8080
- **Documentation API** : http://localhost:3001/api-docs

8.3 Utilisation du système

8.3.1 Pour les sportifs

1. Accéder à l'interface web à http://localhost:4200
2. S'authentifier avec ses identifiants
3. Configurer le dispositif matériel SkiOx associé
4. Vérifier que les paramètres physiologiques sont correctement affichés
5. En cas d'urgence, actionner le bouton de secours sur le dispositif

8.3.2 Pour les sauveteurs

1. Accéder à l'interface web à http://localhost:4200
2. S'authentifier avec ses identifiants de sauveteur
3. Consulter la liste des sportifs et leurs états physiologiques en temps réel
4. Recevoir des notifications en cas d'activation d'urgence
5. Accéder à l'historique des interventions et des données physiologiques

8.3.3 Pour les administrateurs

1. Accéder à phpMyAdmin à http://localhost:8080
2. Utiliser les identifiants MySQL configurés
3. Gérer la base de données directement si nécessaire
4. Consulter la documentation API à http://localhost:3001/api-docs

9. Considérations de sécurité

9.1 Prototype à eau

Pour le prototype actuel :

- Utiliser uniquement de l'eau dans la bouteille squeezy
- S'assurer que les tuyaux et l'électrovanne sont compatibles avec l'eau à faible pression
- Protéger les éléments électroniques de toute fuite ou condensation

9.2 Évolution future vers une vraie bouteille d'oxygène

Si le système est ultérieurement connecté à une bouteille d'oxygène :

- Utiliser uniquement des composants qualifiés pour le service oxygène (tuyaux, électrovannes, joints, lubrifiants, etc.)
- Respecter les consignes des organismes spécialisés (EIGA, CGA, etc.) pour :
    - la propreté (absence d'huile/graisse)
    - le choix des matériaux (EPDM, Viton, PTFE, inox, etc.)
    - la pression de fonctionnement
    - les procédures de montage et de maintenance
- Faire valider la conception par des ingénieurs et professionnels de la sécurité des gaz sous pression

> **AVERTISSEMENT CRITIQUE**
>
> Une mauvaise conception ou installation d'un système à oxygène sous pression peut provoquer feu, explosion, blessures graves ou décès. Ne jamais connecter le prototype tel qu'il est décrit directement à une bouteille d'oxygène réelle.

9.3 Sécurité informatique

9.3.1 Protection des données sensibles

- Les variables d'environnement (clés d'API, mots de passe) sont stockées dans .env et .env.local
- Ces fichiers sont exclus du dépôt Git via .gitignore
- Les données physiologiques des utilisateurs sont chiffrées en transit •

Authentification robuste avec tokens JWT ou sessions sécurisées

9.3.2 Validation des données

- Toutes les entrées utilisateur sont validées et sanitisées •

Les requêtes API incluent des vérifications de cohérence •

Les données IoT sont validées avant persistance

9.3.3 Accès et autorisations

- Contrôle d'accès basé sur les rôles (RBAC) : sportif vs sauveteur
- Chaque utilisateur n'accède qu'à ses propres données ou à celles autorisées •

Audit logging de tous les accès sensibles

10. Documentation et support

10.1 Ressources disponibles

La documentation technique du projet est disponible via :

- **Swagger UI** : http://localhost:3001/api-docs (documentation API interactive)
- **README.md** : Guide général et instructions de démarrage
- **Schémas d'architecture** : Diagrammes Draw.io dans le répertoire Docs/

10.2 Résolution de problèmes

Pour toute question ou problème :

1. Consulter la documentation existante (README, Swagger UI)
2. Vérifier les issues ouvertes sur GitLab
3. Contacter l'équipe de développement via les canaux appropriés

10.3 Contacts

- Pour les questions de développement backend : Bastien, Valentin •

Pour les questions de développement frontend : Nicolas, Flavio

- Pour les questions de sécurité et architecture : Valentin
- Pour les questions matérielles : Kevin CORONA, Paul ARNAUD

11. Conclusion

Le projet SkiOx représente une solution complète et intégrée pour la gestion et le secours des sportifs en montagne. Cette documentation technique couvre l'ensemble des aspects du système, du matériel électronique à l'infrastructure logicielle, en passant par les processus de développement et de déploiement.

Points clés

1. **Architecture modulaire** : Séparation claire entre composants matériel et logiciel
2. **Scalabilité** : Services indépendants facilitant l'évolution future
3. **Intégration IoT** : Connexion directe entre capteurs matériels et plateforme logicielle
4. **Sécurité** : Multiples couches de protection pour les données sensibles
5. **Maintenabilité** : Code bien organisé, documenté et versionné

Recommandations futures

- Implémenter des tests d'intégration complets entre composants matériel et logiciel
- Déployer un système de monitoring et alertes pour les sauveteurs
- Évaluer l'intégration avec d'autres capteurs physiologiques (EMG, EEG, etc.)
- Conformer le système aux normes médicales et de sécurité applicables avant utilisation réelle
- Envisager le déploiement en environnement de production sur infrastructure cloud (Azure, AWS)

Prochaines étapes

1. Validation complète du prototype avec données réelles
2. Évaluation des performances du système en conditions réelles de montagne
3. Formation des sauveteurs à l'utilisation du système
4. Amélioration continue basée sur les retours d'expérience
5. Certification et approvals réglementaires pour déploiement opérationnel

Document rédigé par :

Kevin CORONA

Paul ARNAUD

Bastien NIETO

Valentin LEFEVBRE

Nicolas MAUDUIT

Flavio TAVERNIER

Date : 12/12/2025

Version : 1.0

Status : Complet

Références

[1] SunFounder. MAX30102 Heart Rate Sensor Module – Specifications
[2] ElectronicWings. Bluetooth Module HC-05 – Pinout & AT Commands
[3] Valmet. Safety Guidelines for Oxygen Valves
[4] Analog Devices. MAX30102 Datasheet – Pulse Oximeter and Heart-Rate Sensor
[5] Components101. HC-05 Bluetooth Module – Specifications
[6] Oxidation Technologies. Oxygen Safety – Compatible Materials
[7] GitFlow Documentation
[8] Angular Framework Documentation
[9] Node.js and Express Documentation
[10] Docker and Docker Compose Documentation