```
.
├── package-lock.json
├── package.json
├── src
│   ├── app.js
│   ├── config
│   │   └── db.js
│   ├── console.log
│   ├── controllers
│   │   ├── authController.js
│   │   ├── boardController.js
│   │   ├── noteController.js
│   │   └── productController.js
│   ├── middleware
│   │   └── authMiddleware.js
│   ├── models
│   │   ├── Board.js
│   │   ├── boardModel.js
│   │   └── productModel.js
│   ├── routes
│   │   ├── authRoutes.js
│   │   ├── boardRoutes.js
│   │   └── noteRoutes.js
│   ├── services
│   │   ├── authService.js
│   │   ├── boardServices
│   │   │   ├── createBoard.js
│   │   │   ├── deleteBoard.js
│   │   │   ├── fetchBoards.js
│   │   │   ├── getBoardById.js
│   │   │   └── updateBoard.js
│   │   ├── noteServices
│   │   │   └── noteService.js
│   │   └── productServices
│   │       ├── createProduct.js
│   │       ├── deleteProduct.js
│   │       ├── fetchProducts.js
│   │       ├── getProductById.js
│   │       └── updateProduct.js
│   └── testDb.js
└── testAssistantService.js
```

```
alessandrotornabene@4f34e342-5886-4f1a-ba7d-072eedb502eb lavagnetta-
backend % find src -type f -name "*.js" -exec echo -e
"\n\n==============================\nFile: {}
\n==============================\n" \; -exec cat {} \; -exec echo -e
"\n\n" \;

-e \n\n==============================\nFile: src/middleware/
authMiddleware.js\n==============================\n
// backend/middleware/authMiddleware.js

const jwt = require('jsonwebtoken');
const pool = require('../config/db');
const dotenv = require('dotenv');

dotenv.config();

/**
 * Middleware per autenticare le richieste.
 */
const authMiddleware = async (req, res, next) => {
    const authHeader = req.headers.authorization;

    if (!authHeader || !authHeader.startsWith('Bearer ')) {
        return res.status(401).json({ message: 'Autenticazione
richiesta.' });
    }

    const token = authHeader.split(' ')[1];

    try {
        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        const [rows] = await pool.execute('SELECT * FROM users WHERE id
= ?', [decoded.id]);

        if (rows.length === 0) {
            return res.status(401).json({ message: 'Utente non trovato.'
});
        }

        req.user = rows[0];
        console.log('Authenticated user:', req.user);
        next();
    } catch (error) {
        console.error('Errore di autenticazione:', error);
        res.status(401).json({ message: 'Token non valido.' });
    }
};

module.exports = authMiddleware;
-e \n\n
-e \n\n==============================\nFile: src/config/
db.js\n==============================\n
// src/config/db.js
const mysql = require('mysql2/promise');
const dotenv = require('dotenv');
require('dotenv').config();
```

```
// Carica le variabili d'ambiente dal file .env
dotenv.config();

// Log per verifica
console.log('DB_HOST:', process.env.DB_HOST);
console.log('DB_PORT:', process.env.DB_PORT);
console.log('DB_USER:', process.env.DB_USER);
console.log('DB_NAME:', process.env.DB_NAME);

// Creazione del pool di connessioni
const pool = mysql.createPool({
    host: process.env.DB_HOST,
    port: process.env.DB_PORT, // Porta separata
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    database: process.env.DB_NAME,
    waitForConnections: true,
    connectionLimit: 10,
    queueLimit: 0
});

module.exports = pool;
-e \n\n
-e \n\n==============================\nFile: src/models/
Board.js\n==============================\n
// src/models/Board.js

/*const mongoose = require('mongoose');

const ItemSchema = new mongoose.Schema({
    name: { type: String, required: true },
    bought: { type: Boolean, default: false },
});

const BoardSchema = new mongoose.Schema({
    name: { type: String, required: true },
    background: { type: String, default: 'blackboard.jpg' },
    userId: { type: String, required: true },
    items: [ItemSchema],
});

module.exports = mongoose.model('Board', BoardSchema);*/
-e \n\n
-e \n\n==============================\nFile: src/models/
boardModel.js\n==============================\n
// backend/models/boardModel.js

const pool = require('../config/db');

/**
 * Ottiene tutte le lavagnette di un utente.
 * @param {number} userId - ID dell'utente.
 * @returns {Array} - Array di lavagnette.
 */
```

```
const getBoardsByUserId = async (userId) => {
    const [rows] = await pool.execute('SELECT * FROM boards WHERE
user_id = ?', [userId]);
    return rows;
};

/**
 * Ottiene una lavagnetta per ID.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @returns {Object|null} - Lavagnetta trovata o null.
 */
const getBoardById = async (userId, boardId) => {
    const [rows] = await pool.execute('SELECT * FROM boards WHERE id = ?
AND user_id = ?', [boardId, userId]);
    return rows[0] || null;
};

/**
 * Crea una nuova lavagnetta.
 * @param {number} userId - ID dell'utente.
 * @param {string} name - Nome della lavagnetta.
 * @param {string} background - Sfondo della lavagnetta.
 * @param {boolean} isDefault - Indica se è la lavagnetta di default.
 * @returns {Object} - Lavagnetta creata.
 */
const createBoard = async (userId, name, background, isDefault = false)
=> {
    if (isDefault) {
        // Controlla se esiste già una lavagnetta di default
        const [existingDefault] = await pool.execute(
            'SELECT * FROM boards WHERE user_id = ? AND is_default = 1',
            [userId]
        );
        if (existingDefault.length > 0) {
            throw new Error('Esiste già una lavagnetta di default per
questo utente.');
        }
    }

    const [result] = await pool.execute(
        'INSERT INTO boards (user_id, name, background, is_default,
created_at, updated_at) VALUES (?, ?, ?, ?, NOW(), NOW())',
        [userId, name, background, isDefault ? 1 : 0]
    );
    const [board] = await pool.execute('SELECT * FROM boards WHERE id
= ?', [result.insertId]);
    return board[0];
};

/**
 * Aggiorna una lavagnetta esistente.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {Object} fieldsToUpdate - Campi da aggiornare.
 * @returns {Object|null} - Lavagnetta aggiornata o null se non trovata.
```

```
 */
const updateBoard = async (userId, boardId, fieldsToUpdate) => {
    const fields = [];
    const values = [];

    if (fieldsToUpdate.name !== undefined) {
        fields.push('name = ?');
        values.push(fieldsToUpdate.name);
    }
    if (fieldsToUpdate.background !== undefined) {
        fields.push('background = ?');
        values.push(fieldsToUpdate.background);
    }
    if (fieldsToUpdate.items !== undefined) {
        fields.push('items = ?');
        values.push(JSON.stringify(fieldsToUpdate.items));
    }
    if (fieldsToUpdate.is_default !== undefined) {
        if (fieldsToUpdate.is_default) {
            // Se si imposta come default, rimuove lo status di default
da altre lavagnette
            await pool.execute(
                'UPDATE boards SET is_default = 0 WHERE user_id = ? AND
is_default = 1',
                [userId]
            );
        }
        fields.push('is_default = ?');
        values.push(fieldsToUpdate.is_default ? 1 : 0);
    }

    if (fields.length === 0) {
        return await getBoardById(userId, boardId);
    }

    values.push(boardId, userId);

    const [result] = await pool.execute(
        `UPDATE boards SET ${fields.join(', ')}, updated_at = NOW()
WHERE id = ? AND user_id = ?`,
        values
    );

    if (result.affectedRows === 0) {
        return null;
    }

    const [board] = await pool.execute('SELECT * FROM boards WHERE id
= ?', [boardId]);
    return board[0];
};

/**
 * Elimina una lavagnetta.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
```

```
 * @returns {boolean} - True se eliminata, false altrimenti.
 */
const deleteBoard = async (userId, boardId) => {
    const [defaultCheck] = await pool.execute('SELECT is_default FROM
boards WHERE id = ? AND user_id = ?', [boardId, userId]);
    if (defaultCheck.length > 0 && defaultCheck[0].is_default === 1) {
        throw new Error('La lavagnetta di default non può essere
eliminata.');
    }

    const [result] = await pool.execute('DELETE FROM boards WHERE id = ?
AND user_id = ?', [boardId, userId]);
    return result.affectedRows > 0;
};

/**
 * Aggiorna lo sfondo di una lavagnetta.
 * @param {number} boardId - ID della lavagnetta.
 * @param {string} background - Nuovo sfondo.
 * @returns {Object|null} - Lavagnetta aggiornata o null se non trovata.
 */
const updateBoardBackground = async (boardId, background) => {
    const [result] = await pool.execute(
        'UPDATE boards SET background = ?, updated_at = NOW() WHERE id =
?',
        [background, boardId]
    );

    if (result.affectedRows === 0) {
        return null;
    }

    const [board] = await pool.execute('SELECT * FROM boards WHERE id
= ?', [boardId]);
    return board[0];
};

/**
 * Aggiorna il nome di una lavagnetta.
 * @param {number} boardId - ID della lavagnetta.
 * @param {string} name - Nuovo nome.
 * @returns {Object|null} - Lavagnetta aggiornata o null se non trovata.
 */
const updateBoardName = async (boardId, name) => {
    const [result] = await pool.execute(
        'UPDATE boards SET name = ?, updated_at = NOW() WHERE id = ?',
        [name, boardId]
    );

    if (result.affectedRows === 0) {
        return null;
    }

    const [board] = await pool.execute('SELECT * FROM boards WHERE id
= ?', [boardId]);
    return board[0];
```

```
};

module.exports = {
    getBoardsByUserId,
    getBoardById,
    createBoard,
    updateBoard,
    deleteBoard,
    updateBoardBackground,
    updateBoardName
};
-e \n\n
-e \n\n==============================\nFile: src/models/
productModel.js\n==============================\n
// backend/models/productModel.js

const pool = require('../config/db');

/**
 * Ottiene tutti i prodotti di una lavagnetta per un utente.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @returns {Array} - Array di prodotti.
 */
const getProductsByBoardId = async (userId, boardId) => {
    const [rows] = await pool.execute(
        `SELECT p.* FROM products p
         INNER JOIN boards b ON p.board_id = b.id
         WHERE p.board_id = ? AND b.user_id = ?`,
        [boardId, userId]
    );
    return rows;
};

/**
 * Crea un nuovo prodotto in una lavagnetta.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {string} name - Nome del prodotto.
 * @returns {Object} - Prodotto creato.
 */
const createProduct = async (userId, boardId, name) => {
    // Verifica che la lavagnetta appartenga all'utente
    const [boardRows] = await pool.execute('SELECT * FROM boards WHERE
id = ? AND user_id = ?', [boardId, userId]);
    if (boardRows.length === 0) {
        throw new Error('Lavagnetta non trovata o accesso non
autorizzato.');
    }

    const [result] = await pool.execute(
        'INSERT INTO products (board_id, name, is_purchased, created_at,
updated_at) VALUES (?, ?, 0, NOW(), NOW())',
        [boardId, name]
    );
```

```
    const [product] = await pool.execute('SELECT * FROM products WHERE
id = ?', [result.insertId]);
    return product[0];
};

/**
 * Aggiorna un prodotto.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {number} productId - ID del prodotto.
 * @param {string} name - Nuovo nome del prodotto.
 * @param {boolean} is_purchased - Stato di acquisto.
 * @returns {Object} - Prodotto aggiornato.
 */
const updateProduct = async (userId, boardId, productId, name,
is_purchased) => {
    // Verifica che la lavagnetta appartenga all'utente
    const [boardRows] = await pool.execute('SELECT * FROM boards WHERE
id = ? AND user_id = ?', [boardId, userId]);
    if (boardRows.length === 0) {
        throw new Error('Lavagnetta non trovata o accesso non
autorizzato.');
    }

    // Costruisci la query dinamicamente
    let query = 'UPDATE products SET updated_at = NOW()';
    const params = [];

    if (name !== undefined) {
        query += ', name = ?';
        params.push(name);
    }

    if (is_purchased !== undefined) {
        query += ', is_purchased = ?';
        params.push(is_purchased ? 1 : 0);
    }

    query += ' WHERE id = ? AND board_id = ?';
    params.push(productId, boardId);

    const [result] = await pool.execute(query, params);

    if (result.affectedRows === 0) {
        throw new Error('Prodotto non trovato o accesso non
autorizzato.');
    }

    const [product] = await pool.execute('SELECT * FROM products WHERE
id = ?', [productId]);
    return product[0];
};

/**
 * Elimina un prodotto.
 * @param {number} userId - ID dell'utente.
```

```
 * @param {number} boardId - ID della lavagnetta.
 * @param {number} productId - ID del prodotto.
 */
const deleteProduct = async (userId, boardId, productId) => {
    // Verifica che la lavagnetta appartenga all'utente
    const [boardRows] = await pool.execute('SELECT * FROM boards WHERE
id = ? AND user_id = ?', [boardId, userId]);
    if (boardRows.length === 0) {
        throw new Error('Lavagnetta non trovata o accesso non
autorizzato.');
    }

    const [result] = await pool.execute('DELETE FROM products WHERE id =
? AND board_id = ?', [productId, boardId]);

    if (result.affectedRows === 0) {
        throw new Error('Prodotto non trovato o accesso non
autorizzato.');
    }
};

module.exports = {
    getProductsByBoardId,
    createProduct,
    updateProduct,
    deleteProduct
};
-e \n\n
-e \n\n============================\nFile: src/
testDb.js\n============================\n
// src/testDb.js
const db = require('./config/db');

async function testConnection() {
    try {
        const connection = await db.getConnection();
        console.log('Connessione al database riuscita.');
        connection.release(); // Rilascia la connessione al pool
        process.exit(0);
    } catch (err) {
        console.error('Errore di connessione al database:', err);
        process.exit(1);
    }
}

testConnection();
-e \n\n
-e \n\n============================\nFile: src/controllers/
noteController.js\n============================\n
// src/controllers/noteController.js

const noteService = require('../services/noteServices/noteService');

/**
 * Recupera tutte le note per l'utente autenticato.
 */
```

```
exports.getNotes = async (req, res) => {
    try {
        console.log(`Recuperando le note per l'utente ID: ${req.user.id}
`);
        const notes = await noteService.getUserNotes(req.user.id);
        console.log(`Note recuperate: ${JSON.stringify(notes)}`);

        // Modifica qui: Invece di restituire solo l'array, restituisci
un oggetto con una proprietà 'notes'
        res.set('Cache-Control', 'no-store, no-cache, must-revalidate,
proxy-revalidate');
        res.set('Pragma', 'no-cache');
        res.set('Expires', '0');
        res.set('Surrogate-Control', 'no-store');
        res.status(200).json({ notes }); // Modificato da 'notes' a
'{ notes: notes }'
    } catch (error) {
        console.error('Errore nel recuperare le note:', error);
        res.status(500).json({ error: 'Errore nel recuperare le
note.' });
    }
};

/**
 * Crea una nuova nota per l'utente.
 */
exports.createNote = async (req, res) => {
    try {
        const { title, content, date } = req.body;
        console.log(`Creando una nuova nota per l'utente ID: $
{req.user.id}`);
        const newNote = await noteService.createNote(req.user.id, title,
content, date);
        console.log(`Nuova nota creata: ${JSON.stringify(newNote)}`);

        res.set('Cache-Control', 'no-store, no-cache, must-revalidate,
proxy-revalidate');
        res.set('Pragma', 'no-cache');
        res.set('Expires', '0');
        res.set('Surrogate-Control', 'no-store');
        res.status(201).json({ note: newNote }); // Modificato da
'newNote' a '{ note: newNote }'
    } catch (error) {
        console.error('Errore nella creazione della nota:', error);
        res.status(500).json({ error: 'Errore nella creazione della
nota.' });
    }
};

/**
 * Aggiorna una nota esistente.
 */
exports.updateNote = async (req, res) => {
    try {
        const { title, content, date } = req.body;
```

```
        console.log(`Aggiornando la nota ID: ${req.params.id} per
l'utente ID: ${req.user.id}`);
        const updatedNote = await noteService.updateNote(req.user.id,
req.params.id, title, content, date);
        if (updatedNote) {
            console.log(`Nota aggiornata: ${JSON.stringify(updatedNote)}
`);
            res.set('Cache-Control', 'no-store, no-cache, must-
revalidate, proxy-revalidate');
            res.set('Pragma', 'no-cache');
            res.set('Expires', '0');
            res.set('Surrogate-Control', 'no-store');
            res.status(200).json({ note: updatedNote }); // Modificato
da 'updatedNote' a '{ note: updatedNote }'
        } else {
            console.log('Nota non trovata per l\'aggiornamento.');
            res.status(404).json({ error: 'Nota non trovata.' });
        }
    } catch (error) {
        console.error('Errore nell\'aggiornamento della nota:', error);
        res.status(500).json({ error: 'Errore nell\'aggiornamento della
nota.' });
    }
};

/**
 * Elimina una nota.
 */
exports.deleteNote = async (req, res) => {
    try {
        console.log(`Eliminando la nota ID: ${req.params.id} per
l'utente ID: ${req.user.id}`);
        const result = await noteService.deleteNote(req.user.id,
req.params.id);
        if (result) {
            console.log('Nota eliminata con successo.');
            res.set('Cache-Control', 'no-store, no-cache, must-
revalidate, proxy-revalidate');
            res.set('Pragma', 'no-cache');
            res.set('Expires', '0');
            res.set('Surrogate-Control', 'no-store');
            res.status(200).json({ message: 'Nota eliminata con
successo.' });
        } else {
            console.log('Nota non trovata per l\'eliminazione.');
            res.status(404).json({ error: 'Nota non trovata.' });
        }
    } catch (error) {
        console.error('Errore nell\'eliminazione della nota:', error);
        res.status(500).json({ error: 'Errore nell\'eliminazione della
nota.' });
    }
};
-e \n\n
-e \n\n==============================\nFile: src/controllers/
boardController.js\n==============================\n
```

```
// src/controllers/boardController.js

const fetchBoardsService = require('../services/boardServices/
fetchBoards');
const getBoardByIdService = require('../services/boardServices/
getBoardById');
const createBoardService = require('../services/boardServices/
createBoard');
const updateBoardService = require('../services/boardServices/
updateBoard');
const deleteBoardService = require('../services/boardServices/
deleteBoard');

/**
 * Ottiene tutte le lavagnette per l'utente autenticato, eliminando
eventuali duplicati.
 */
exports.getBoards = async (req, res) => {
    try {
        console.log('req.user:', req.user); // Log per il debug
        let boards = await fetchBoardsService(req.user.id);

        // Rimuove duplicati in base all'ID delle lavagnette
        boards = boards.reduce((acc, current) => {
            const exists = acc.find(board => board.id === current.id);
            if (!exists) acc.push(current);
            return acc;
        }, []);

        res.status(200).json(boards);
    } catch (error) {
        console.error('Errore nel recuperare le lavagnette:', error);
        res.status(500).json({ error: error.message });
    }
};


/**
 * Ottiene una lavagnetta per ID.
 */
exports.getBoardById = async (req, res) => {
    try {
        const board = await getBoardByIdService(req.user.id,
req.params.id);
        if (board) {
            res.status(200).json(board);
        } else {
            res.status(404).json({ error: 'Lavagnetta non trovata.' });
        }
    } catch (error) {
        console.error('Errore nel recuperare la lavagnetta:', error);
        res.status(500).json({ error: error.message });
    }
};

/**
```

```
 * Crea una nuova lavagnetta.
 */
exports.createBoard = async (req, res) => {
    const { name, background } = req.body;
    try {
        const newBoard = await createBoardService(req.user.id, name,
background);
        res.status(201).json(newBoard);
    } catch (error) {
        console.error('Errore nella creazione della lavagnetta:',
error);
        res.status(500).json({ error: error.message });
    }
};

/**
 * Aggiorna una lavagnetta esistente.
 */
exports.updateBoard = async (req, res) => {
    const { name, background, items } = req.body;
    try {
        const fieldsToUpdate = {};
        if (name !== undefined) fieldsToUpdate.name = name;
        if (background !== undefined) fieldsToUpdate.background =
background;
        if (items !== undefined) fieldsToUpdate.items = items;

        const updatedBoard = await updateBoardService(req.user.id,
req.params.id, fieldsToUpdate);
        if (updatedBoard) {
            res.status(200).json(updatedBoard);
        } else {
            res.status(404).json({ error: 'Lavagnetta non trovata.' });
        }
    } catch (error) {
        console.error('Errore nell\'aggiornamento della lavagnetta:',
error);
        res.status(500).json({ error: error.message });
    }
};

/**
 * Elimina una lavagnetta.
 */
exports.deleteBoard = async (req, res) => {
    try {
        const result = await deleteBoardService(req.user.id,
req.params.id);
        if (result) {
            res.status(200).json({ message: 'Lavagnetta eliminata con
successo.' });
        } else {
            res.status(404).json({ error: 'Lavagnetta non trovata.' });
        }
    } catch (error) {
```

```
        console.error('Errore nell\'eliminazione della lavagnetta:',
error);
        res.status(500).json({ error: error.message });
    }
};
-e \n\n
-e \n\n==============================\nFile: src/controllers/
productController.js\n==============================\n
// backend/controllers/productController.js

const fetchProductsService = require('../services/productServices/
fetchProducts');
const createProductService = require('../services/productServices/
createProduct');
const updateProductService = require('../services/productServices/
updateProduct');
const deleteProductService = require('../services/productServices/
deleteProduct');

/**
 * Ottiene tutti i prodotti di una lavagnetta.
 */
exports.getProducts = async (req, res) => {
    const userId = req.user.id;
    const boardId = req.params.id;

    try {
        const products = await fetchProductsService(userId, boardId);
        res.json(products);
    } catch (error) {
        console.error('Errore nel recuperare i prodotti:', error);
        res.status(500).json({ error: 'Errore nel recuperare i
prodotti.' });
    }
};

/**
 * Crea un nuovo prodotto in una lavagnetta.
 */
exports.createProduct = async (req, res) => {
    const userId = req.user.id;
    const boardId = req.params.id;
    const { name } = req.body;

    if (!name) {
        return res.status(400).json({ error: 'Il nome del prodotto è
richiesto.' });
    }

    try {
        const product = await createProductService(userId, boardId,
name);
        res.status(201).json(product);
    } catch (error) {
        console.error('Errore nell\'aggiunta del prodotto:', error);
        res.status(500).json({ error: error.message });
```

```
    }
};

/**
 * Aggiorna un prodotto esistente.
 */
exports.updateProduct = async (req, res) => {
    const userId = req.user.id;
    const boardId = req.params.id;
    const productId = req.params.productId;
    const { name, is_purchased } = req.body;

    try {
        const updatedProduct = await updateProductService(userId,
boardId, productId, name, is_purchased);
        res.json(updatedProduct);
    } catch (error) {
        console.error('Errore nell\'aggiornamento del prodotto:',
error);
        res.status(500).json({ error: error.message });
    }
};

/**
 * Elimina un prodotto.
 */
exports.deleteProduct = async (req, res) => {
    const userId = req.user.id;
    const boardId = req.params.id;
    const productId = req.params.productId;

    try {
        await deleteProductService(userId, boardId, productId);
        res.json({ message: 'Prodotto eliminato con successo.' });
    } catch (error) {
        console.error('Errore nella cancellazione del prodotto:',
error);
        res.status(500).json({ error: error.message });
    }
};
-e \n\n
-e \n\n==============================\nFile: src/controllers/
authController.js\n==============================\n
// backend/controllers/authController.js

const { registerUser, loginUser, renewToken } = require('../services/
authService');
const { validationResult } = require('express-validator');

/**
 * Controller per la registrazione di un utente.
 */
const register = async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
```

```
        return res.status(400).json({ success: false, errors:
errors.array() });
    }

    const { email, password, username } = req.body;

    try {
        const result = await registerUser(email, password, username);
        if (result.success) {
            res.json({ success: true, token: result.token, message:
result.message });
        } else {
            res.status(400).json({ success: false, message:
result.message });
        }
    } catch (error) {
        console.error('Errore nel controller di registrazione:', error);
        res.status(500).json({ success: false, message: 'Errore server
durante la registrazione.' });
    }
};

/**
 * Controller per il login di un utente.
 */
const login = async (req, res) => {
    const errors = validationResult(req);
    if (!errors.isEmpty()) {
        return res.status(400).json({ success: false, errors:
errors.array() });
    }

    const { email, password } = req.body;

    try {
        const result = await loginUser(email, password);
        if (result.success) {
            res.json({ success: true, token: result.token, message:
result.message });
        } else {
            res.status(400).json({ success: false, message:
result.message });
        }
    } catch (error) {
        console.error('Errore nel controller di login:', error);
        res.status(500).json({ success: false, message: 'Errore server
durante il login.' });
    }
};

/**
 * Controller per rinnovare un token JWT.
 */
const renewTokenController = async (req, res) => {
    try {
        const userId = req.user.id;
```

```
        const newToken = renewToken(userId);
        res.json({ success: true, token: newToken });
    } catch (error) {
        console.error('Errore nel controller di rinnovo token:', error);
        res.status(500).json({ success: false, message: 'Errore server
durante il rinnovo del token.' });
    }
};

module.exports = {
    register,
    login,
    renewToken: renewTokenController // Aggiunto
};
-e \n\n
-e \n\n============================\nFile: src/routes/
authRoutes.js\n============================\n
// backend/routes/authRoutes.js

const express = require('express');
const router = express.Router();
const { check } = require('express-validator');
const authController = require('../controllers/authController');
const boardRoutes = require('./boardRoutes'); // Rotte per le lavagnette
const noteRoutes = require('./noteRoutes'); // Rotte per le note
const authMiddleware = require('../middleware/authMiddleware'); //
Middleware

// Rotta per registrare un nuovo utente
router.post(
    '/register',
    [
        check('username', 'Il nome è richiesto.').not().isEmpty(),
        check('email', 'Inserisci un email valida.').isEmail(),
        check('password', 'La password deve avere almeno 6
caratteri.').isLength({ min: 6 })
    ],
    authController.register
);

// Rotta per il login
router.post(
    '/login',
    [
        check('email', 'Inserisci un email valida.').isEmail(),
        check('password', 'La password è richiesta.').exists()
    ],
    authController.login
);

// Rotta per il rinnovo del token
router.post('/renew-token', authMiddleware,
authController.renewToken); // Aggiunto

// Monta le rotte delle lavagnette sotto /api/auth/boards
router.use('/boards', boardRoutes);
```

```
// Monta le rotte delle note sotto /api/auth/notes
router.use('/notes', noteRoutes);

module.exports = router;
-e \n\n
-e \n\n==============================\nFile: src/routes/
noteRoutes.js\n==============================\n
// src/routes/noteRoutes.js

const express = require('express');
const router = express.Router();
const noteController = require('../controllers/noteController');
const authMiddleware = require('../middleware/authMiddleware');

// Applica il middleware di autenticazione a tutte le rotte
router.use(authMiddleware);

// Rotte per le note
router.get('/', noteController.getNotes); // GET /api/auth/notes
router.post('/', noteController.createNote); // POST /api/auth/notes
router.put('/:id', noteController.updateNote); // PUT /api/auth/
notes/:id
router.delete('/:id', noteController.deleteNote); // DELETE /api/auth/
notes/:id

module.exports = router;
-e \n\n
-e \n\n==============================\nFile: src/routes/
boardRoutes.js\n==============================\n
// src/routes/boardRoutes.js

const express = require('express');
const router = express.Router();
const boardController = require('../controllers/boardController');
const productController = require('../controllers/productController');
const authMiddleware = require('../middleware/authMiddleware');

// Applica il middleware di autenticazione a tutte le rotte
router.use(authMiddleware);

// Rotte per le lavagnette
router.get('/', boardController.getBoards); // GET /api/auth/boards
router.get('/:id', boardController.getBoardById); // GET /api/auth/
boards/:id
router.post('/', boardController.createBoard); // POST /api/auth/boards
router.put('/:id', boardController.updateBoard); // PUT /api/auth/
boards/:id
router.delete('/:id', boardController.deleteBoard); // DELETE /api/auth/
boards/:id

// Rotte per i prodotti all'interno di una lavagnetta
router.get('/:id/products', productController.getProducts); // GET /api/
auth/boards/:id/products
router.post('/:id/products', productController.createProduct); // POST /
api/auth/boards/:id/products
```

```
router.put('/:id/products/:productId', productController.updateProduct);
// PUT /api/auth/boards/:id/products/:productId
router.delete('/:id/products/:productId',
productController.deleteProduct); // DELETE /api/auth/boards/:id/
products/:productId

module.exports = router;
-e \n\n
-e \n\n==============================\nFile: src/
app.js\n==============================\n
// backend/app.js

const express = require('express');
const cors = require('cors');
const morgan = require('morgan');
const authRoutes = require('./routes/authRoutes');
const noteRoutes = require('./routes/noteRoutes');
const boardRoutes = require('./routes/boardRoutes');
const bodyParser = require('body-parser');
const dotenv = require('dotenv');

// Load environment variables from .env file
dotenv.config();

const app = express();
const PORT = process.env.PORT || 3000;


// CORS configuration
const corsOptions = {
    origin: '*', // Allow all origins (for testing purposes)
    methods: ['GET', 'POST', 'PUT', 'DELETE'],
    allowedHeaders: ['Content-Type', 'Authorization'],
    credentials: true,
};

// Logging middleware
app.use(morgan('combined'));

// CORS middleware
app.use(cors(corsOptions));

// Body parsing middleware
app.use(bodyParser.json());

// Authentication routes
app.use('/api/auth', authRoutes);

// Protected routes (require authentication)
app.use('/api/notes', noteRoutes);
app.use('/api/boards', boardRoutes);

// Base route
app.get('/api/ping', (req, res) => {
    res.send('Pong');
});
```

```
// Error handling middleware
app.use((err, req, res, next) => {
    console.error(err.stack);
    res.status(500).send('Something went wrong!');
});

// Start the server
app.listen(PORT, '0.0.0.0', () => {
    console.log(`Server running on port ${PORT}`);
});
-e \n\n
-e \n\n==============================\nFile: src/services/noteServices/
noteService.js\n==============================\n
// src/services/noteServices/noteService.js

const db = require('../../config/db');

/**
 * Recupera tutte le note dell'utente.
 * @param {number} userId - ID dell'utente.
 * @returns {Array} - Array di note.
 */
const getUserNotes = async (userId) => {
    try {
        console.log(`Eseguendo la query per recuperare le note
dell'utente ID: ${userId}`);
        const [rows] = await db.execute(
            'SELECT id, title, content, date, created_at, updated_at
FROM notes WHERE user_id = ? ORDER BY date DESC',
            [userId]
        );
        console.log(`Risultato della query: ${JSON.stringify(rows)}`);
        return rows;
    } catch (error) {
        console.error('Errore nel recuperare le note dal database:',
error);
        throw error;
    }
};

/**
 * Crea una nuova nota per l'utente.
 * @param {number} userId - ID dell'utente.
 * @param {string} title - Titolo della nota.
 * @param {string} content - Contenuto della nota.
 * @param {string} date - Data della nota (YYYY-MM-DD).
 * @returns {Object} - La nota creata.
 */
const createNote = async (userId, title, content, date) => {
    try {
        console.log(`Inserendo una nuova nota per l'utente ID: ${userId}
`);
        const [result] = await db.execute(
            'INSERT INTO notes (user_id, title, content, date,
created_at, updated_at) VALUES (?, ?, ?, ?, NOW(), NOW())',
```

```
            [userId, title, content, date]
        );
        console.log(`ID della nuova nota inserita: ${result.insertId}`);
        const [rows] = await db.execute(
            'SELECT id, title, content, date, created_at, updated_at
FROM notes WHERE id = ?',
            [result.insertId]
        );
        console.log(`Nuova nota recuperata: ${JSON.stringify(rows[0])}
`);
        return rows[0];
    } catch (error) {
        console.error('Errore nella creazione della nota nel database:',
error);
        throw error;
    }
};

/**
 * Aggiorna una nota esistente.
 * @param {number} userId - ID dell'utente.
 * @param {number} noteId - ID della nota.
 * @param {string} title - Nuovo titolo.
 * @param {string} content - Nuovo contenuto.
 * @param {string} date - Nuova data (YYYY-MM-DD).
 * @returns {Object|null} - Nota aggiornata o null se non trovata.
 */
const updateNote = async (userId, noteId, title, content, date) => {
    try {
        console.log(`Aggiornando la nota ID: ${noteId} per l'utente ID:
${userId}`);
        const [result] = await db.execute(
            'UPDATE notes SET title = ?, content = ?, date = ?,
updated_at = NOW() WHERE id = ? AND user_id = ?',
            [title, content, date, noteId, userId]
        );

        if (result.affectedRows === 0) {
            console.log('Nessuna nota aggiornata. Nota non trovata o non
appartiene all\'utente.');
            return null; // Nota non trovata o non appartiene all'utente
        }

        const [rows] = await db.execute(
            'SELECT id, title, content, date, created_at, updated_at
FROM notes WHERE id = ?',
            [noteId]
        );
        console.log(`Nota aggiornata recuperata: $
{JSON.stringify(rows[0])}`);
        return rows[0];
    } catch (error) {
        console.error('Errore nell\'aggiornamento della nota nel
database:', error);
        throw error;
    }
```

```
};

/**
 * Elimina una nota.
 * @param {number} userId - ID dell'utente.
 * @param {number} noteId - ID della nota.
 * @returns {boolean} - True se eliminata, false altrimenti.
 */
const deleteNote = async (userId, noteId) => {
    try {
        console.log(`Eliminando la nota ID: ${noteId} per l'utente ID: $
{userId}`);
        const [result] = await db.execute(
            'DELETE FROM notes WHERE id = ? AND user_id = ?',
            [noteId, userId]
        );
        console.log(`Numero di righe eliminate: ${result.affectedRows}
`);
        return result.affectedRows > 0;
    } catch (error) {
        console.error('Errore nell\'eliminazione della nota nel
database:', error);
        throw error;
    }
};

module.exports = {
    getUserNotes,
    createNote,
    updateNote,
    deleteNote
};
-e \n\n
-e \n\n==============================\nFile: src/services/boardServices/
deleteBoard.js\n==============================\n
// backend/services/boardServices/deleteBoard.js

const boardModel = require('../../models/boardModel');

/**
 * Servizio per eliminare una lavagnetta.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @returns {boolean} - True se eliminata, false altrimenti.
 */
const deleteBoardService = async (userId, boardId) => {
    try {
        const result = await boardModel.deleteBoard(userId, boardId);
        return result;
    } catch (error) {
        throw new Error(error.message);
    }
};

module.exports = deleteBoardService;
-e \n\n
```

```
-e \n\n==============================\nFile: src/services/boardServices/
fetchBoards.js\n==============================\n
// backend/services/boardServices/fetchBoards.js

const boardModel = require('../../models/boardModel');

/**
 * Servizio per recuperare tutte le lavagnette di un utente.
 * @param {number} userId - ID dell'utente.
 * @returns {Array} - Array di lavagnette.
 */
const fetchBoards = async (userId) => {
    try {
        const boards = await boardModel.getBoardsByUserId(userId);
        return boards;
    } catch (error) {
        throw new Error('Errore nel recuperare le lavagnette.');
    }
};

module.exports = fetchBoards;
-e \n\n
-e \n\n==============================\nFile: src/services/boardServices/
updateBoard.js\n==============================\n
// src/services/boardServices/updateBoard.js

const db = require('../../config/db');

/**
 * Aggiorna una lavagnetta con i campi forniti.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {Object} fieldsToUpdate - Campi da aggiornare (name,
background, items).
 * @returns {Object} - Lavagnetta aggiornata.
 */
const updateBoard = async (userId, boardId, fieldsToUpdate) => {
    try {
        console.log('Parametri di updateBoard:', { userId, boardId,
fieldsToUpdate });

        let setClause = '';
        const values = [];

        if (fieldsToUpdate.name !== undefined) {
            setClause += 'name = ?, ';
            values.push(fieldsToUpdate.name);
        }
        if (fieldsToUpdate.background !== undefined) {
            setClause += 'background = ?, ';
            values.push(fieldsToUpdate.background);
        }
        if (fieldsToUpdate.items !== undefined) {
            setClause += 'items = ?, ';
            values.push(JSON.stringify(fieldsToUpdate.items));
        }
```

```
        if (setClause === '') {
            throw new Error('Nessun campo da aggiornare.');
        }

        // Rimuove la virgola finale e lo spazio
        setClause = setClause.slice(0, -2);

        const query = `UPDATE boards SET ${setClause}, updated_at =
NOW() WHERE id = ? AND user_id = ?`;
        values.push(boardId, userId);

        console.log('Eseguendo la query:', query, values);

        const [result] = await db.execute(query, values);

        console.log('Risultato della query UPDATE:', result);

        if (result.affectedRows === 0) {
            throw new Error('Lavagnetta non trovata o non aggiornata.');
        }

        const [updatedBoard] = await db.execute('SELECT * FROM boards
WHERE id = ?', [boardId]);
        console.log('Lavagnetta aggiornata:', updatedBoard[0]);

        return updatedBoard[0];
    } catch (error) {
        console.error('Errore nell\'aggiornamento della lavagnetta:',
error);
        throw new Error(error.message || 'Errore nell\'aggiornamento
della lavagnetta.');
    }
};

module.exports = updateBoard;
-e \n\n
-e \n\n==============================\nFile: src/services/boardServices/
getBoardById.js\n==============================\n
// backend/services/boardServices/getBoardById.js

const boardModel = require('../../models/boardModel');

/**
 * Servizio per recuperare una lavagnetta per ID.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @returns {Object|null} - Lavagnetta trovata o null.
 */
const getBoardById = async (userId, boardId) => {
    try {
        const board = await boardModel.getBoardById(userId, boardId);
        return board;
    } catch (error) {
        throw new Error('Errore nel recuperare la lavagnetta.');
    }
```

```
};

module.exports = getBoardById;
-e \n\n
-e \n\n==============================\nFile: src/services/boardServices/
createBoard.js\n==============================\n
// backend/services/boardServices/createBoard.js

const boardModel = require('../../models/boardModel');

/**
 * Servizio per creare una nuova lavagnetta.
 * @param {number} userId - ID dell'utente.
 * @param {string} name - Nome della lavagnetta.
 * @param {string} background - Sfondo della lavagnetta.
 * @returns {Object} - Lavagnetta creata.
 */
const createBoard = async (userId, name, background) => {
    try {
        // Se l'utente non ha lavagnette, crea una lavagnetta di default
        const boards = await boardModel.getBoardsByUserId(userId);
        let isDefault = false;
        if (boards.length === 0) {
            isDefault = true;
        }

        const newBoard = await boardModel.createBoard(userId, name,
background, isDefault);
        return newBoard;
    } catch (error) {
        throw new Error('Errore nella creazione della lavagnetta.');
    }
};

module.exports = createBoard;
-e \n\n
-e \n\n==============================\nFile: src/services/
productServices/deleteProduct.js\n==============================\n
// backend/services/productServices/deleteProduct.js

const productModel = require('../../models/productModel');

/**
 * Servizio per eliminare un prodotto.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {number} productId - ID del prodotto.
 * @returns {boolean} - True se eliminato, false altrimenti.
 */
const deleteProduct = async (userId, boardId, productId) => {
    try {
        await productModel.deleteProduct(userId, boardId, productId);
        return true;
    } catch (error) {
        throw new Error(error.message);
    }
```

```
};

module.exports = deleteProduct;
-e \n\n
-e \n\n==============================\nFile: src/services/
productServices/getProductById.js\n==============================\n
// backend/services/productServices/getProductById.js

const productModel = require('../../models/productModel');

/**
 * Servizio per recuperare un prodotto per ID.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {number} productId - ID del prodotto.
 * @returns {Object|null} - Prodotto trovato o null.
 */
const getProductById = async (userId, boardId, productId) => {
    try {
        // Implementa se necessario
        const products = await productModel.getProductsByBoardId(userId,
boardId);
        const product = products.find(p => p.id === productId);
        return product || null;
    } catch (error) {
        throw new Error('Errore nel recuperare il prodotto.');
    }
};

module.exports = getProductById;
-e \n\n
-e \n\n==============================\nFile: src/services/
productServices/updateProduct.js\n==============================\n
// backend/services/productServices/updateProduct.js

const productModel = require('../../models/productModel');

/**
 * Servizio per aggiornare un prodotto esistente.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {number} productId - ID del prodotto.
 * @param {string} name - Nuovo nome del prodotto.
 * @param {boolean} is_purchased - Stato di acquisto.
 * @returns {Object} - Prodotto aggiornato.
 */
const updateProduct = async (userId, boardId, productId, name,
is_purchased) => {
    try {
        const updatedProduct = await productModel.updateProduct(userId,
boardId, productId, name, is_purchased);
        return updatedProduct;
    } catch (error) {
        throw new Error(error.message);
    }
};
```

```
module.exports = updateProduct;
-e \n\n
-e \n\n==============================\nFile: src/services/
productServices/fetchProducts.js\n==============================\n
// backend/services/productServices/fetchProducts.js

const productModel = require('../../models/productModel');

/**
 * Servizio per recuperare tutti i prodotti di una lavagnetta.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @returns {Array} - Array di prodotti.
 */
const fetchProducts = async (userId, boardId) => {
    try {
        const products = await productModel.getProductsByBoardId(userId,
boardId);
        return products;
    } catch (error) {
        throw new Error('Errore nel recuperare i prodotti.');
    }
};

module.exports = fetchProducts;
-e \n\n
-e \n\n==============================\nFile: src/services/
productServices/createProduct.js\n==============================\n
// backend/services/productServices/createProduct.js

const productModel = require('../../models/productModel');

/**
 * Servizio per creare un nuovo prodotto.
 * @param {number} userId - ID dell'utente.
 * @param {number} boardId - ID della lavagnetta.
 * @param {string} name - Nome del prodotto.
 * @returns {Object} - Prodotto creato.
 */
const createProduct = async (userId, boardId, name) => {
    try {
        const product = await productModel.createProduct(userId,
boardId, name);
        return product;
    } catch (error) {
        throw new Error(error.message);
    }
};

module.exports = createProduct;
-e \n\n
-e \n\n==============================\nFile: src/services/
authService.js\n==============================\n
// src/services/authService.js
```

```
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const db = require('../config/db'); // Assicurati che il percorso sia
corretto

const saltRounds = 10;

/**
 * Registra un nuovo utente.
 * @param {string} email - Email dell'utente.
 * @param {string} password - Password dell'utente.
 * @param {string} username - Username dell'utente.
 * @returns {Object} - Risultato della registrazione.
 */
const registerUser = async (email, password, username) => {
    try {
        // Verifica se l'utente esiste già
        const [existingUser] = await db.execute('SELECT id FROM users
WHERE email = ?', [email]);
        if (existingUser.length > 0) {
            return { success: false, message: 'Email già in uso.' };
        }

        // Hash della password
        const hashedPassword = await bcrypt.hash(password, saltRounds);

        // Inserisce l'utente nel database
        const [result] = await db.execute('INSERT INTO users (email,
password, username) VALUES (?, ?, ?)', [email, hashedPassword,
username]);
        const userId = result.insertId;

        // Log del username durante la registrazione
        console.log('Username durante la registrazione:', username);

        // Crea un token JWT includendo email
        const token = jwt.sign({ id: userId, username, email },
process.env.JWT_SECRET, { expiresIn: '1h' });

        // Logging della registrazione nel database
        await db.execute('INSERT INTO interactions (user_id,
interaction, response) VALUES (?, ?, ?)', [userId, 'Register',
'Success']);

        return { success: true, message: 'Registrazione riuscita.',
token };
    } catch (error) {
        console.error('Errore nel servizio di registrazione:', error);
        // Logging dell'errore nel database
        await db.execute('INSERT INTO interactions (user_id,
interaction, response) VALUES (?, ?, ?)', [null, 'Register', `Errore: $
{error.message}`]);
        return { success: false, message: 'Errore durante la
registrazione.' };
    }
};
```

```
/**
 * Effettua il login dell'utente.
 * @param {string} email - Email dell'utente.
 * @param {string} password - Password dell'utente.
 * @returns {Object} - Risultato del login.
 */
const loginUser = async (email, password) => {
    try {
        // Trova l'utente nel database
        const [rows] = await db.execute('SELECT id, password, username,
email FROM users WHERE email = ?', [email]);

        if (rows.length === 0) {
            // Logging del tentativo di login fallito
            await db.execute('INSERT INTO interactions (user_id,
interaction, response) VALUES (?, ?, ?)', [null, 'Login', 'Credenziali
non valide']);
            return { success: false, message: 'Credenziali non
valide.' };
        }

        const user = rows[0];

        // Log del username durante il login
        console.log('Username durante il login:', user.username);

        // Confronta le password
        const match = await bcrypt.compare(password, user.password);

        if (!match) {
            // Logging del tentativo di login fallito
            await db.execute('INSERT INTO interactions (user_id,
interaction, response) VALUES (?, ?, ?)', [user.id, 'Login',
'Credenziali non valide']);
            return { success: false, message: 'Credenziali non
valide.' };
        }

        // Crea un token JWT includendo email
        const token = jwt.sign({ id: user.id, username: user.username,
email: user.email }, process.env.JWT_SECRET, { expiresIn: '1h' });

        // Logging del login riuscito
        await db.execute('INSERT INTO interactions (user_id,
interaction, response) VALUES (?, ?, ?)', [user.id, 'Login',
'Success']);

        return { success: true, message: 'Login riuscito.', token };
    } catch (error) {
        console.error('Errore nel servizio di login:', error);
        // Logging dell'errore nel database
        await db.execute('INSERT INTO interactions (user_id,
interaction, response) VALUES (?, ?, ?)', [null, 'Login', `Errore: $
{error.message}`]);
        return { success: false, message: 'Errore durante il login.' };
```

```
    }
};

module.exports = { registerUser, loginUser };
-e \n\n
alessandrotornabene@4f34e342-5886-4f1a-ba7d-072eedb502eb lavagnetta-
backend %
```