```javascript  
// backend/server.js  
const express = require('express');  
const cors = require('cors');  
const db = require('./db'); // Make sure db.js is in the same directory  
const bcrypt = require('bcrypt'); // Make sure to install: npm install bcrypt  
  
const app = express();  
const PORT = process.env.PORT || 3000;  
const SALT\_ROUNDS = 10; // For bcrypt hashing  
  
// --- Middleware ---  
// Enable CORS for requests from your frontend origin (adjust the origin URL as needed!)  
const corsOptions = {  
  origin: '[http://127.0.0.1:5500](http://127.0.0.1:5500/)', // Example: Common default for VS Code Live Server  
  optionsSuccessStatus: 200  
};  
app.use(cors(corsOptions));  
app.use(express.json()); // Parse JSON bodies sent in POST requests  
  
// --- API Routes ---  
  
// 1. Signup endpoint  
[app.post](http://app.post/)('/api/signup', async (req, res) => {  
    const { username, email, password } = req.body;  
  
    // Basic validation  
    if (!username || !email || !password) {  
        return res.status(400).json({ error: 'Username, email, and password are required.' });  
    }  
  
    try {  
        // Check if user already exists (by username or email)  
        db.get(`SELECT id FROM users WHERE username = ? OR email = ?`, [username, email], (err, row) => {  
            if (err) {  
                console.error('Database error during user existence check:', err.message);  
                return res.status(500).json({ error: 'Internal server error during signup check.' });  
            }  
            if (row) {  
                return res.status(409).json({ error: 'Username or email already exists.' });  
            }  
  
            // Hash the password using bcrypt  
            bcrypt.hash(password, SALT\_ROUNDS, (err, hash) => {  
                if (err) {  
                    console.error('Error hashing password:', err);  
                    return res.status(500).json({ error: 'Error processing password.' });  
                }  
  
                // Insert the new user into the SQLite database  
                const stmt = db.prepare(`INSERT INTO users (username, email, password\_hash) VALUES (?, ?, ?)`);  
                stmt.run([username, email, hash], function(err) {  
                    if (err) {  
                        console.error('Database error during user creation:', err.message);  
                        // Specific constraint errors (e.g., duplicate username/email not caught here due to prior check,  
                        // but good practice to handle unexpected DB errors)  
                        return res.status(500).json({ error: 'Failed to create user.' });  
                    }  
                    // Send success response. Status 201 means 'Created'.  
                    res.status(201).json({ message: 'User created successfully.' });  
                    // Finalize the prepared statement  
                    stmt.finalize();  
                });  
            });  
        });  
    } catch (error) {  
        console.error('Unexpected error during signup:', error);  
        res.status(500).json({ error: 'An unexpected error occurred.' });  
    }  
});  
  
// 2. Login endpoint  
[app.post](http://app.post/)('/api/login', (req, res) => {  
    const { username, password } = req.body;  
  
    // Basic validation  
    if (!username || !password) {  
        return res.status(400).json({ error: 'Username and password are required.' });  
    }  
  
    // Query the database for the user by username  
    db.get(`SELECT id, username, password\_hash FROM users WHERE username = ?`, [username], (err, userRow) => {  
        if (err) {  
            console.error('Database error during login lookup:', err.message);  
            return res.status(500).json({ error: 'Internal server error during login.' });  
        }  
        // If no user found with that username  
        if (!userRow) {  
             // Security: Don't reveal if username or password was wrong  
            return res.status(401).json({ error: 'Invalid credentials.' });  
        }  
  
        // Use bcrypt to compare the provided password with the stored hash  
        bcrypt.compare(password, userRow.password\_hash, (err, result) => {  
            if (err) {  
                console.error('Error comparing passwords:', err);  
                return res.status(500).json({ error: 'Error verifying password.' });  
            }  
            // If the password matches the hash  
            if (result) {  
                // Successful login  
                // IMPORTANT: Never send the password hash back to the client!  
                // In a full app, you would create a session or issue a JWT here.  
                // For now, we just send a success message and basic user info.  
                return res.json({  
                    message: 'Login successful',  
                    user: { id: userRow.id, username: userRow.username }  
                    // Add other non-sensitive user data you want to send initially if needed  
                });  
            } else {  
                // Password did not match  
                return res.status(401).json({ error: 'Invalid credentials.' });  
            }  
        });  
    });  
});  
  
// 3. Get total signups endpoint  
app.get('/api/metrics/total-signups', (req, res) => {  
    // Execute a SQL query to count all rows in the users table  
    db.get(`SELECT COUNT(\*) as total FROM users`, [], (err, row) => {  
        if (err) {  
            console.error('Database error fetching total signups:', err.message);  
            // Return a 500 Internal Server Error status with an error message  
            return res.status(500).json({ error: 'Failed to retrieve signup count.' });  
        }  
        // Send the count back as JSON. row.total holds the result of COUNT(\*)  
        res.json({ totalSignups: row.total });  
    });  
});  
  
// --- Graceful Shutdown ---  
// Ensure the database connection is closed properly when the Node.js process is terminated  
process.on('SIGINT', () => {  
    db.close((err) => {  
        if (err) {  
            console.error('Error closing database connection:', err.message);  
        } else {  
            console.log('Database connection closed successfully.');  
        }  
        // Exit the Node.js process  
        process.exit(0);  
    });  
});  
  
// --- Start the Server ---  
// Tell the Express app to listen for incoming requests on the specified PORT  
app.listen(PORT, () => {  
    // Log messages to the console when the server starts successfully  
    console.log(`EcoQuest Backend Server is now running and listening on http://localhost:${PORT}`);  
    console.log(`API endpoints are available under http://localhost:${PORT}/api`);  
    // Example endpoints:  
    // POST <http://localhost:3000/api/signup>  
    // POST <http://localhost:3000/api/login>  
    // GET  <http://localhost:3000/api/metrics/total-signups>  
});  
  
```

|  |  |
| --- | --- |
| https://ssl.gstatic.com/ui/v1/icons/mail/no_photo.png |  |