Creating a data-sharing platform for displaying real-time air quality data from IoT devices involves both front-end and back-end development. In this response, I'll provide you with a high-level overview of how you can approach this project using web development technologies.  
  
Front-end: HTML, CSS, and JavaScript  
  
1. User Interface Design: Start by designing a user-friendly interface to display air quality data. You can use HTML for the structure, CSS for styling, and JavaScript for interactivity. Here's a simplified HTML template for your platform:  
  
```html  
<!DOCTYPE html>  
<html>  
<head>  
    <title>Real-Time Air Quality Data</title>  
    <link rel="stylesheet" type="text/css" href="styles.css">  
</head>  
<body>  
    <header>  
        <h1>Real-Time Air Quality Data</h1>  
    </header>  
    <main>  
        <div id="data-container">  
            <!-- Data from IoT devices will be displayed here -->  
        </div>  
    </main>  
    <script src="script.js"></script>  
</body>  
</html>  
```  
  
2. CSS Styling: Create a CSS file (styles.css) to style the interface, making it visually appealing and responsive.  
  
3. JavaScript: Write JavaScript code (script.js) to make your platform interactive. You can use AJAX or Fetch API to request data from the server and update the UI in real-time.  
  
```javascript  
// Use JavaScript to fetch real-time data from the server  
function fetchData() {  
    fetch('/api/air-quality')  
        .then(response => response.json())  
        .then(data => {  
            // Update the UI with the received data  
            document.getElementById('data-container').innerHTML = JSON.stringify(data, null, 2);  
        })  
        .catch(error => console.error('Error fetching data: ', error));  
}  
  
// Fetch data at regular intervals (e.g., every 5 seconds)  
setInterval(fetchData, 5000);  
```  
  
Back-end: Server and APIs  
  
1. Server Setup: You need a server to handle data from IoT devices and serve it to the platform. You can use Node.js with Express, Python with Flask, or any other web framework of your choice.  
  
2. Database: You'll need a database to store the air quality data sent by IoT devices. Choose a database system like MySQL, PostgreSQL, MongoDB, or Firebase, depending on your needs.  
  
3. API Development: Create APIs on your server to handle data from IoT devices and serve it to the platform. For example, in Express.js, you can define a route like this:  
  
```javascript  
const express = require('express');  
const app = express();  
  
// Sample API endpoint to receive data from IoT devices  
[app.post](http://app.post/)('/api/air-quality', (req, res) => {  
    const data = req.body; // Process and store data in the database  
    // Return a response if needed  
    res.status(200).send('Data received successfully');  
});  
  
app.listen(3000, () => {  
    console.log('Server is running on port 3000');  
});  
```  
  
4. IoT Device Integration: IoT devices should send air quality data to the server at regular intervals using APIs or MQTT (Message Queuing Telemetry Transport), depending on the protocol you choose.  
  
5. Database Integration: Store incoming data in your database for historical analysis and retrieval.  
  
6. Security: Ensure that your server and IoT devices communicate securely, using encryption and authentication mechanisms.  
  
7. Real-time Updates: Implement WebSocket or Server-Sent Events (SSE) to send real-time updates to connected clients (your front-end).  
  
8. Authentication and Authorization: Implement user authentication and authorization if you want to restrict access to certain data.  
  
9. Scalability: Consider the scalability of your system as you add more IoT devices and users.  
  
This is a simplified overview of the process. The actual implementation may require additional components and considerations, depending on the scale and complexity of project.