1. **Set Up the Project Environment**
2. 2. **Backend Setup**
3. Connect Backend to Database
4. Front-End Setup
5. Integrate Control Objects
6. **6. Implement Additional Features**
7. **User Authentication**: Create routes for login and register. Use the User entity and control object to handle login logic.
8. **Alerts**: Use your alert control object to create weather alerts and notify users via the UI.
9. **Dashboard Customization**: Implement dashboard settings where users can adjust how the weather data is displayed.
10. **7. Testing the Website**
11. Test each feature as you build it by running the Flask app and interacting with the forms.
12. Use Postman or cURL to simulate API requests and check if the backend routes are working as expected.

**Steps Summary:**

1. **Set Up the Project** (Choose Flask or Django).
2. **Design Frontend Pages** (Create forms, buttons, etc. in HTML/CSS).
3. **Implement Backend Logic** (Using Python, Flask/Django for control logic and database handling).
4. **Integrate Use Cases** (Implement logic for each use case like WeatherDataControl, UserControl).
5. **Add Database Connection** (SQLAlchemy or Django ORM).
6. **Testing and Debugging** (Interact with your site and ensure everything is working).

**3. Develop HTML Templates**

Create HTML templates in the templates/ directory. These templates will be used by the Flask routes defined in your app.py. Here’s a quick guide:

1. **Base Template (base.html)**: Create a base template that includes common elements like navigation, header, and footer.
2. **Other Templates**: Develop individual templates for login, viewing weather data, feedback submission, data export, alerts, etc. Use the basic Jinja2 templating engine to display data passed from your Flask views.

**4. Integrate and Test Your Application**

1. **Run Your Flask Application**: Ensure all routes in app.py are functioning as intended. Start the Flask server:
2. **Test Each Use Case**:
   * Navigate through each route on the web application.
   * Make sure data is correctly displayed and interactions such as login, alert creation, feedback submission, etc., work as expected.
3. **Debug Issues**:
   * Use the browser’s developer console to check for frontend errors.
   * Use the Flask log output to identify and resolve backend errors.

**5. Write Unit Tests**

1. **Create Test Scripts** in the tests/ directory for each control object.
2. **Run Tests** using pytest:
3. **Cover All Use Cases**: Ensure you test each control object to validate all interactions, including error handling and edge cases.

**6. Deployment**

1. **Set Up a Production Server**:
   * Choose a cloud provider (e.g., AWS, DigitalOcean, Heroku).
   * Set up the server with necessary software like Nginx, Python, MySQL, and Gunicorn.
2. **Configure Your Server**:
   * Set up Gunicorn to run your Flask app.
   * Configure Nginx as a reverse proxy to forward traffic to Gunicorn.
3. **Deploy Your App**:
   * Clone your project to the server.
   * Start Gunicorn and ensure it runs on boot (use systemd or another service manager).
4. **Secure Your Application**:
   * Set up HTTPS with SSL (use Let’s Encrypt for free SSL certificates).
   * Secure your MySQL database by restricting access.

**7. Monitor and Maintain**

1. **Set Up Monitoring**:
   * Use tools like UptimeRobot, New Relic, or similar to monitor your application’s performance and uptime.
2. **Backups**:
   * Regularly back up your MySQL database and application files.
3. **Maintenance**:
   * Regularly update dependencies and security patches.

**Conclusion**

Following these steps will ensure that your weather website is not only built and tested properly but also deployed securely in a production environment. Each step involves integrating your objects with the Flask application, ensuring the database is correctly set up, and maintaining robust testing to handle all use cases. Let me know if you need specific help or further elaboration on any of these steps!