To build a parser that can check the format of Passenger Name Lists (PNLs) and identify errors, you can follow these steps:

1. **Define PNL Format Rules**: Start by understanding the specific PNL format or formats you want to support. Different airlines or systems may have variations in their PNL formats, so you need to define the rules for each one.
2. **Choose a Programming Language**: Select a programming language that you are comfortable with and that is suitable for text parsing. Python is often a good choice for this task due to its robust text processing libraries.
3. **Text Parsing**: Write code to parse PNL data. Use regular expressions or text processing libraries to extract relevant information, such as flight details, passenger names, and travel document information.
4. **Validation Rules**: Define validation rules for each component of the PNL. This includes rules for flight information, passenger names, document numbers, and other relevant data. Common validations include checking for correct date formats, valid airport codes, and passport number formats.
5. **Error Detection**: Implement error detection logic in your parser. When the parser encounters data that doesn't conform to the defined format or validation rules, it should flag it as an error. Record the nature of the error and its location within the PNL.
6. **Error Reporting**: Develop a system for reporting errors to the user. You can create a report that lists the errors found in the PNL, providing detailed information about each error and its location in the PNL.
7. **Testing**: Test your PNL parser with various PNL samples, including both valid and invalid ones. Ensure that it correctly identifies errors and provides accurate error reports.
8. **User Interface**: Create a user-friendly interface for users to input or upload PNL data for validation. This can be a web-based form or a desktop application, depending on your preferences and target audience.
9. **Documentation**: Document how to use your PNL parser, including input requirements and error messages. Make the documentation accessible to users.
10. **Deployment**: Deploy your PNL parser as a tool that users can access. Depending on your choice, this can be a web application, a standalone desktop application, or a command-line tool.
11. **Maintenance**: Regularly update and maintain your PNL parser to accommodate changes in PNL formats or to fix any issues that may arise. Keep track of industry standards and adapt your parser accordingly.
12. **User Support**: Provide support for users who encounter issues or have questions about your PNL parser. Consider setting up a support system or FAQ section if needed.

Remember that the complexity of your PNL parser will depend on the specific PNL formats you want to support and the range of validation rules you need to implement. Collaboration with industry experts or access to PNL format specifications can be invaluable in ensuring that your parser accurately checks PNL data.