Code Layout:

1. **Consistent Indentation:** Use a consistent and clear indentation style, such as spaces or tabs, and stick to it throughout your codebase.

2. **Naming Conventions:** Follow a consistent naming convention for variables, functions, and classes. Use descriptive names to make your code self-explanatory.

3. **Modularization:** Organize your code into modules or classes based on functionality. Each module should have a clear and specific purpose.

4. **Comments and Documentation:** Add comments to explain complex logic or to provide context for other developers. Consider using documentation tools like JSDoc or Doxygen for documenting functions and classes.

5. **File Structure:** Maintain a well-structured file hierarchy. Group related files and folders, and use meaningful directory names.

Readability:

6. **Consistent Code Style:** Enforce a consistent code style across your project by using linters or coding standards like PEP 8 for Python, ESLint for JavaScript, or PSR standards for PHP.

7. **Whitespace:** Use whitespace effectively to improve readability. Avoid code blocks that are too dense and use blank lines to separate sections logically.

8. **Avoid Deep Nesting:** Limit the depth of nested code blocks. Deeply nested code can be hard to follow and debug.

9. **Use Clear Variable Names:** Choose descriptive and meaningful variable names that convey their purpose.

10. **Avoid Overly Long Functions:** Break down long functions into smaller, reusable ones that perform a single, well-defined task.

Reusability:

11. **Modular Components:** Design your CRM application with modular components that can be reused across the application. This reduces redundancy and improves maintainability.

12. **Separation of Concerns:** Apply the principle of separation of concerns to isolate different aspects of your application (e.g., UI, business logic, data access).

13. **APIs and Interfaces:** Develop APIs and interfaces to expose functionality that can be reused by other parts of the application or external systems.

14. **Utilize Design Patterns:** Implement design patterns (e.g., Singleton, Factory, Observer) to create reusable, well-structured code.

15. **Version Control:** Use version control systems like Git to manage and track changes in your codebase. This facilitates code reuse and collaboration among developers.

16. **Testing:** Implement unit and integration testing to validate the functionality of your code. Test suites can help ensure that changes don't break existing functionality.

17. **Dependency Management:** Use dependency management tools (e.g., npm, Composer) to manage external libraries and packages. This ensures you're using up-to-date, well-maintained components.

18. **API Integration:** When integrating with external services or APIs, create reusable modules or wrappers that can be used for different parts of the application.

19. **Code Reviews:** Conduct code reviews to maintain code quality and promote reusability. Identify opportunities for refactoring and improvements.