Rules and Solutions

1. **State Management:**
   - **Issue:** The use of StatefulWidget and manual state management in `TaskListScreen` could lead to complex and error-prone code as the application grows.
   - **Impact:** Reduced code maintainability and potential bugs related to state changes.
2. **Data Fetching:**
   - **Issue:** Fetching data in `initState` and using `FutureBuilder` in `TimeManagementApp`.
   - **Impact:** Unpredictable behavior and potential issues with asynchronous data fetching.
3. **Code Duplication:**
   - **Issue:** Date and time picker logic is duplicated in `_selectDate`, `_selectTimeFrom`, and `_selectTimeTo`.
   - **Impact:** Redundant code increases the risk of errors and makes maintenance challenging.
4. **UI Logic in AlertDialog:**
   - **Issue:** The UI creation logic for creating a new task is embedded within `_showAddTaskDialog`.
   - **Impact:** Code in AlertDialog becomes hard to maintain and understand.
5. **Hard-Coded Strings:**
   - **Issue:** There are hard-coded strings throughout the codebase.
   - **Impact:** Reduced code maintainability and increased chances of introducing errors during updates.
6. **Error Handling:**
   - **Issue:** Lack of explicit error handling in data fetching and Firestore operations.

- **Impact:** Potential for unhandled errors leading to unexpected behavior.

7. **Responsibilities in Widgets:**
   - **Issue:** Some widgets, like `TaskSearchDelegate` and `TaskListScreen`, handle both UI rendering and logic.
   - **Impact:** Violation of the Single Responsibility Principle, making the code less modular.

Refactoring and Design Patterns:

1. **State Management:**
   - **Refactoring:** Implement Provider, Riverpod, or Bloc for more organized and scalable state management.
2. **Data Fetching:**
   - **Refactoring:** Use the Repository pattern to move data fetching logic to a dedicated service class.
3. **Code Duplication:**
   - **Refactoring:** Extract date and time picker logic into a separate helper function to avoid redundancy; consider using the Strategy pattern.
4. **UI Logic in AlertDialog:**
   - **Refactoring:** Create a separate widget for the task creation dialog to encapsulate UI logic and improve readability; consider using the Builder pattern.
5. **Hard-Coded Strings:**
   - **Refactoring:** Define constants or use enums for strings; consider using the Strategy pattern for string handling.
6. **Error Handling:**
   - **Refactoring:** Implement the Strategy pattern for error handling, providing a consistent and modular approach.
7. **Responsibilities in Widgets:**
   - **Refactoring:** Separate UI rendering and logic into distinct classes or methods, adhering to the Single Responsibility Principle.