Traceability Matrix

ID	Requirement	Related Use Case	Fulfilled By	Test	Description
1	Wave analysis and simulation must occur, resulting in a dominant frequency	Use Case 1	analysisWave.cpp, WaveSimulator.cpp	Observe the output of getBaseline()	The AnalysisWave class, through getBaseline(), simulates the process of analyzing a wave, ultimately providing a dominant frequency. This simulation includes generating a wave and assigning a random frequency within a specified band, which imitates real-world EEG signal processing.
2	Frequency bands for the simulation must be defined and utilized across all sessions	Use Case 1	frequencyBands.h, session.cpp	Review frequency band usage in session instantiation	FrequencyBand definitions are declared in frequencyBands.h, which the Session class utilizes to randomly select a frequency band for the session, ensuring consistency across the simulation of 21 electrodes. This is critical for the authenticity of the treatment simulation.
3	Application's main control flow is correctly initialized	Use Case 1, Use Case 2	main.cpp	Run the application	The primary execution flow is defined in main.cpp, where the QApplication is initiated, and the MainWindow is instantiated and displayed. This file orchestrates the application's lifeline from ignition to termination.
4	The treatment delivery simulation must respect frequency parameters	Use Case 1	site.cpp, session.cpp	Observe the debug output for treatment frequency	Site::deliverTreatment in site.cpp takes in a treatment frequency and logs it, representing the application of a treatment at a certain frequency. This is a core part of simulating therapy sessions where different frequencies might be used.

5	Simulation must generate wave frequencies within a random but constrained range	Use Case 1	waveSimulator.cpp	Run the simulator and monitor frequency generation	WaveSimulator::generateRandomFrequency() in waveSimulator.cpp is responsible for producing a random frequency within the band range defined for a session. This randomness introduces variability, simulating real-world scenarios within controlled parameters.
6	The software must efficiently manage session controls, including starting, pausing, resuming, and stopping sessions, ensuring robust handling of session states and transitions.	Use Case 1	MainWindow.cpp, session.cpp	Conduct tests to validate the correct execution of session controls under various operational conditions, including transitions between different session states.	The device's session management functions include initiating, pausing, resuming, and stopping sessions, ensuring smooth transitions and reliable operations under various user scenarios. These functions handle the full lifecycle of a session, from start to finish, maintaining session integrity and user control.
7	The device must accurately initialize, display, and update the current date and time throughout its operation, ensuring that all session data are correctly timestamped and that system functions relying on the date/time are properly synchronized.	Use Case 1	MainWindow.cpp	Verify the correct display of initial date and time upon device start-up and monitor the update accuracy every second. Additionally, test visibility toggles for date and time UI components.	The device initializes and continuously updates the current date and time, starting with system initialization where the current date and time are set and then hidden from the user interface to simplify interactions. A timer then increments the displayed time every second to ensure ongoing accuracy during operation.
8	The device must manage battery levels effectively, providing warnings when the battery is low, and shutting down when depleted to prevent data loss and hardware damage.	Use Case 1, Use Case 2	MainWindow.cpp	Simulate various battery levels to observe warning displays and automatic shutdown behavior. Test pause and resume functionality under different battery conditions.	The device's battery management functions include monitoring battery levels, updating UI indicators, and controlling device operations based on battery status. Critical actions like automatic shutdown when the battery is depleted and visual warnings for low battery are managed to ensure device reliability and safety.

9	The device must provide visual feedback through color-coded light indicators, ensuring clear communication of system states such as active, standby, and alert conditions.	Use Case 1	MainWindow.cpp	Check the color transitions and flashing mechanisms for each light indicator under various system states.	These functions control the visual feedback system of the device, using different colors to indicate specific states. lightOn and lightOff control the color and default state of the lights. The flashing functions for red and green lights are managed by timers, starting and stopping the flash to signal critical alerts or active processes, while the blue light functions indicate operational readiness.
10	The system must provide an intuitive and responsive menu selection interface, allowing users to navigate through options and select functions seamlessly.	Use Case 1	Use Case 1	MainWindow.cpp	The device's menu management involves showMenuSelection, closeMenuSelection, showMenu, and doubleClickMenu functions, which facilitate the opening, closing, and interaction with the menu. These functions ensure a user-friendly navigation experience, allowing for responsive selection and execution of menu options.