DBManager Class:

- The database is capable of storing both the records and the profiles
- For the function addSessionRecord(), instead of adding all the session variables in the parameters of the function (like the denas project), I decided to instead add the entire session object. This is because sessionRecord has many variables, and adding them all as parameters would have been tedious.
- The function addProfile() takes in the profile's variables and not the Profile object itself. This is because compared to sessionRecord, the Profile class has less member variables, allowing the parameters to contain all the variables without looking messy
- The heartRateVariation and heartRateTimes variables of the record database are of type TEXT. This is because sql does not accept variables of type QVector<double>. I decided instead to convert these variables into a string, that each contained the respective values followed by a comma.

MainWindow UI Class:

- The visual aspect of the session has been made to replicate that seen in the specification; it includes a ui element for battery level, heart rate connectivity, coherence, time, achievement score, breath pace, and a graph.
- The device has also been made to match the specs; it reuses buttons and other assets from the project provided on the course bright space as a method to reduce work.
- Coherence level has been changed to 1 bar as it is easy enough to change the color of the light during runtime. In a real world application 3 lights may be needed to match hardware restrictions.
- The "Admin" tab is also set to match that of the provided Denas project; it is made to allow access to key features with relative ease during run time.
- The test data selection is an additional function that allows the user to change the input type during runtime as there is no hardware to allow real input.

MainWindow Class:

- The class contains 2 menu objects, this is useful because it allows us to return to the root of the program by simply replacing the working menu with the address for the root and updating the ui.
- The two booleans power status and hr contact allow us to keep the system in the correct mode while processes run separately, such as during a session.
- While the session is not in progress we set the value of the session to Null; this allows us to determine whether or not a session is in progress and act accordingly when changing our current state.

Session Class:

- The Session receives the heartbeat with the time when receiving a heartbeat because QTImer is very imprecise with its timing, which would certainly have a noticeable impact on the output of the program

- It is separate from SessionRecord because it has member functions that work on its members during which could affect their values in the class if misused

Session Record Class

- The time and heartbeat variations are stored in QVector<double>* because that is what is required for the graphing function

Menu Class:

- A very simple menu class based on the Denas project that acts as a tree, each menu has a root/parent and can contain children.
- Each instance is also associated with a name and the name of its children that are used by the ui to show the flow of the menu.
- During runtime the user can delete menu items to clear the history. Since new items are added after each session, we must have the ability to clear them to avoid selecting a non existing record.

Profile Class:

- A very simple class that holds 3 pieces of information. 2 of these, breath pace and battery level, are used during runtime while the other is used to save and retrieve that information between sessions.

Plot Class:

- A prebuilt class taken from the link provided in the class discord that allows us to create and update a graph element in the ui during runtime.

Traceability Matrix

The traceability matrix shows the requirements, the related use cases, the classes that fulfills those use cases, tests that show the classes for those use cases work as intended and the description of the classes and their requirements. This documents the tests runs and results.

Class Diagram, Use Case Diagram and Sequence Diagrams:

The class diagrams, use case diagrams and sequence diagrams were made in an efficient yet simple way. The class diagram describes the information structures used between the classes, the sequence diagrams show the interactions between the instances of classes, actors, subsystems etc. The use case diagram shows user requirements and the relationship between the uses and the requirements.