1. **Login system**
   1. A complete functioning login system for the user, with the ability to register an account or sign in with an existing account.
      1. The user should be able to create an account which saves across sessions.
      2. The user should be able to sign in with a username and password to an existing account.
      3. The user should be able to delete their account and remove it from the database.
      4. The user should be able to update their password information.
2. **User interface menu**
   1. A file menu is presented to the user which allows the user to work with custom files.
      1. The user should be able to open a fresh blank file.
      2. The user should be able to open a pre-existing project using the windows file explorer.
      3. The user should be able to see and quickly enter back into their most recent projects.
      4. The user should be able to close the application.
3. **Qbit simulation**
   1. The program should be able to accurately simulate quantum bits and the user should be able to interact with these.
      1. The program should contain to logic to build a probabilistic bit type, such as a qbit.
      2. The user should be able to create Qbit objects and interact with them.
      3. The user should be able to visually explore the Qbit objects they create through different GUI elements.
4. **Quantum gates**
   1. The program contains the logic for controlling quantum bits and stores matrix representations of quantum gates that can be used to execute various algorithms.
      1. The program should contain a collection of quantum logic gates saved in a machine friendly manner for easy recollection.
         1. The user should be able to retrieve the matrix-encoded quantum gates.
         2. The quantum gates should not be able to change during runtime.
         3. The program should appropriately retrieve these gates in an efficient manner whenever they are needed for a process.
            1. The program should be able to retrieve any given gate without the need for expensive searching.
            2. The program should be able to retrieve any given gate at any time it is necessary.
            3. The program should correctly retrieve any given gate needed for a specific procedure.
         4. The gates should be presented in an accessible way for developers.
            1. The gates should be stored internally in an easily readable form for any programmers going through the code.
            2. The gate identifiers should conform to standard constant styling guidelines.
      2. The program should be able to apply quantum gates to user specified Qbit objects.
         1. The program should contain a function to perform matrix multiplication to transform Qbit objects.
         2. The user should be able to decide which object they wish to apply a transformation too.
      3. The user should be able to construct quantum algorithms by applying a series of unary operations on Qbit objects.
         1. The user has the ability to pipe the output of one algorithm to another.
         2. The user can refer to previously used Qbit objects, which get modified rather than deleted.
5. **Probability view**
   1. The program contains the logic to display a real-time 2d graphic to the user representing the probability of finding a Qbit in a certain location.
      1. The graphic should operate in real time without any user interaction necessary.
      2. The graphic should be clear and descriptive.
         1. The graphic should contain a static legend that is centred around an appropriate value.
         2. The graphic should be appropriately labelled.
      3. The graphic should appropriately change depending on the user’s interaction with the Qbit objects powering it.
         1. The user should be able to change the underlying Qbit object and see a reflective and corresponding change in the graphic view.
6. **Particle view**
   1. The program contains the logic to display and control a 2d particle view to the user, showcasing some of the electromagnetic effects of Qbit objects as well as how these change after certain operations have been applied.
      1. The graphic should display independent of user input.
      2. The graphic should be interactive.
         1. The graphic should contain draggable particle objects, which allow the user to place them wherever they wish.
         2. The graphic should shift appropriately according to the location of any particle objects present.
         3. The graphic should not contain any undefined behaviour at the boundary points of the rendered window.
         4. The graphic should have the option to modify the particle objects being rendered on it.
      3. The user should be able to modify the graphic through commands.
         1. The user’s interactions with Qbit objects should be reflected in the render.
7. **Circuit diagram**
   1. The program should facilitate the construction of a circuit diagram object, representative of the current users’ command input during the current session.
      1. The program should provide the correct command string for creating a circuit diagram object.
         1. The program should automatically apply these commands and send them through to the end service.
         2. The program should parse out irrelevant, supplement or inexecutable commands.
      2. The program should generate the diagram, through an algorithmic approach of by retrieving from an exterior source.
         1. The program could generate a circuit diagram using a custom written algorithm stored in a python file in the program directory.
         2. The program could generate a circuit diagram by fetching a relevant one from online or through an API.
      3. The program should retrieve the diagram object and store it suitably.
         1. The program should be able to save a diagram object if generated algorithmically.
         2. The program should be able to download the diagram and store it locally if generated through a 3rd party tool.
      4. The program should be able to display the image to the user in any effective way.
         1. The program could store the image files in an easily accessible and obvious place, to be viewed inside windows.
         2. The program could use GUI interfaces to display the image to the user during runtime.
         3. The program could display the image to the user through the CLI, perhaps as an ascii representation.
8. **Code editor**
   1. The program should contain the functionality to accept user input and commands in an aesthetic and easy to use editor window that provides basic text edit capabilities.
      1. The program should be able to accept user input.
      2. The program should contain logic for running the user input.
      3. The program should be easy to navigate and contain clear functionality.
         1. The program should resemble other text editors that users may be familiar with.
         2. The program should contain appropriate menuing.
         3. The program should split features to make them atomic.
      4. The program should provide basic highlighting or other text differentiation to separate various elements of commands and make the text easier to read.
         1. The program could contain regex pattern matching for syntax highlighting.
         2. The program could use pythons built in functionality for highlighting text.
         3. The program should provide clear text entry points.
      5. The program should be able to save user input in the form of project files.
      6. The program should be able to open old project files.
      7. The program should contain basic clipboard functionality.
         1. The program should be able to copy text to the clipboard.
         2. The program should be able to cut text to the clipboard.
         3. The program should be able to paste text from the clipboard.
      8. The program should be easily exited from the editor window.
9. **Command interpretation**
   1. The program should contain the functionality to parse, interpret and run commands supplied by the user.
      1. The program should be able to lexically scan the users input and tokenise each element.
         1. The program should be able to label a token given a set of labels and labelling rules.
      2. The program should be able to parse out supplementary or irrelevant commands.
         1. The program should be able to recognise which commands don’t affect runtime and deal with these appropriately.
         2. The program should be able to remove, append or modify tokens easily.
      3. The program should be able to place the users’ commands into an appropriate data structure for efficient storage and retrieval.
         1. The program should make use of the tree data structure for effectively storing commands.
         2. The program should be able to retrieve data stored in tree data structures.
         3. The program should be able to traverse tree data structures in multiple ways.
      4. The program should be able to deal with the complex syntax surrounding objects and deal with this appropriately.
         1. The program should be able to pattern match objects to tokenise them.
         2. The program should be able to recognise the difference between a callable name and a callable’s parameters.
      5. The program should be to interpret the users’ commands and run these effectively.
         1. The program should be able to pipe outputs into appropriate other sections of the code to deal with specific input cases.
         2. The program should be able to recognise which functionality it needs to execute for which input it is given.
         3. The program should be able to execute commands in near real-time, without delay or command buildup.